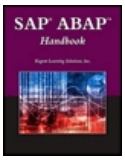


SAP ABAP Handbook

by Kogent Learning Solutions, Inc.

Chapter 12: Forms in MySAP ERP: SAPscript and SAP Smart Forms

Overview

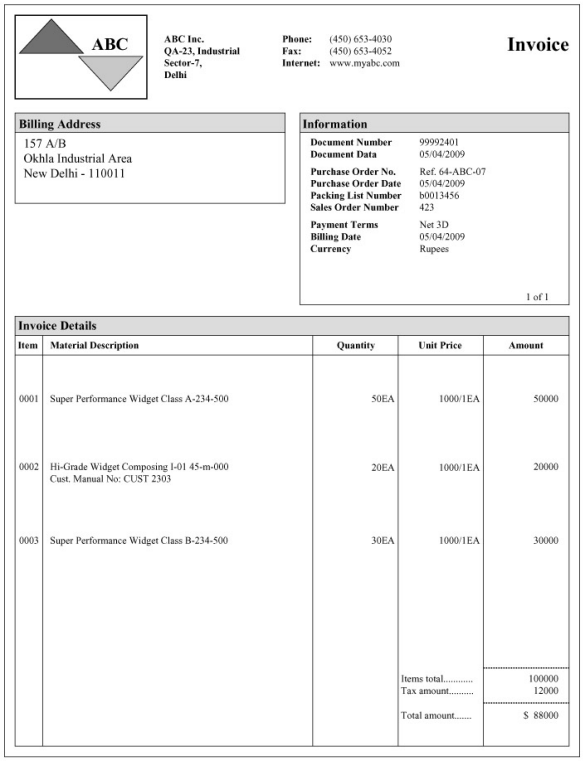
A business form is used as a standard format to document various processes in a business. Examples of business forms that companies routinely maintain to record the business processes are invoices, credit memos, and delivery notes. The mySAP ERP system provides two SAP programming tools, SAPscript and SAP Smart Forms, to design, build, and print business forms. SAPscript was the only tool in versions of SAP R/3 before release 4.6 that allowed you to use built-in forms as templates to create the required business forms. SAP Smart Forms is an alternative tool used to design, build, and print business forms in an SAP system. The SAPscript tool cannot be used to print multiple forms simultaneously. To overcome this drawback, you can use the SAP Smart Forms tool, which supports mass printing. In other words, you can design and print multiple forms simultaneously by using the SAP Smart Forms tool. This tool provides the facility to print and send documents through e-mail, fax, and the Internet, and was first shipped with SAP R/3 4.6.

SAPscript has two important components, a print program and a layout set. A print program is a sequence of function modules such as `OPEN_FORM`, `START_FORM`, and `WRITE_FORM`. The print program is used to retrieve the data of a form and call the layout set objects, which are called windows. In SAPscript, an ABAP program populates the contents of a window through a function call given by the print program. SAP Smart Forms, on the other hand, has three important components: a print program manually, a layout set, and a function module. Unlike SAPscript, where you have to activate a layout set for a form manually, the layout set for the form is generated automatically in SAP Smart Forms. This automatic generation of a layout set creates a standard callable SAP function module, which reduces the time to generate the output. This is because when a print program of Smart Forms calls a form, the form itself takes over the task of generating the output (using the generated function module), without the involvement of the print program.

In this chapter, you learn about the SAPscript and SAP Smart Forms tools in detail, including the structure and components of an SAPscript form, the print programs used to print the forms, and the various function modules, control commands, and symbols used in SAPscript. This chapter also discusses the SAP Smart Forms tool and its advantages, the differences between an SAPscript form and a Smart Form, and how to migrate an SAPscript form to a Smart Form.

Exploring the SAPscript Tool

The SAPscript tool of the mySAP ERP system is used to build and manage business forms, such as invoices and purchase orders. The SAPscript tool provides numerous templates that simplify the designing of a business form. Figure 12.1 shows an example of an invoice form created by using the SAPscript tool:



ABC

ABC Inc.
QA-23, Industrial
Sector-7,
Delhi

Phone: (450) 653-4030
Fax: (450) 653-4052
Internet: www.myabc.com

Invoice

Billing Address

157 A/B
Okhla Industrial Area
New Delhi - 110011

Information

Document Number	99992401
Document Data	05/04/2009
Purchase Order No.	Ref. 64-ABC-07
Purchase Order Date	05/04/2009
Packing List Number	b0013456
Sales Order Number	423
Payment Terms	Net 3D
Billing Date	05/04/2009
Currency	Rupees

1 of 1

Invoice Details

Item	Material Description	Quantity	Unit Price	Amount
0001	Super Performance Widget Class A-234-500	50EA	1000/1EA	50000
0002	Hi-Grade Widget Composing I-01 45-m-000 Cust. Manual No: CUST 2303	20EA	1000/1EA	20000
0003	Super Performance Widget Class B-234-500	30EA	1000/1EA	30000
			Items total.....	100000
			Tax amount.....	12000
			Total amount.....	$ 88000

Figure 12.1: Example of an invoice form in SAPscript

The mySAP ERP system comes with standard SAPscript forms that are delivered with the SAP standard client (usually referred to as client 000). Examples of standard SAPscript forms delivered with client 000 are described in Table 12.1:

Table 12.1: Examples of standard SAPscript forms

Form Description	Form Name
Sales Order Confirmation	RVORDER01
Packing List	RVDELNOTE
Invoice	RVINVOICE01
Purchase Order	MEDRUCK
Prenumbered Check	F110_PRENUM_CHCK

In the forthcoming sections, we describe the following:

- Components of the SAPscript tool
- Structure of an SAPscript form
- Managing tools
- Accessing PC Editor
- Accessing the Form Painter tool
- Form subobjects
- The SAPscript runtime environment
- The print program
- SAPscript function modules
- Controlling SAPscript forms

- SAPscript control commands
- SAPscript symbols

Components of the SAPscript Tool

The following is a brief description of the various components of the SAPscript tool:

- **Editor—** Edit s the text in an SAPscript form. The transaction of an application automatically calls this editor if you need to maintain texts related to the application.
- **Styles and Forms—** Define and print the style and layout of SAPscript forms.
- **Composer or Form Processor—** Acts as a central output module to prepare final layout and text for an output device by including styles, various formatting options, and the respective text.
- **Programming Interface—** Allows you to include SAPscript components into ABAP programs and control the output of forms from the programs.
- **Database Tables—** Store texts, styles, and forms .

Figure 12.2 graphically represents the components of the SAPscript tool:

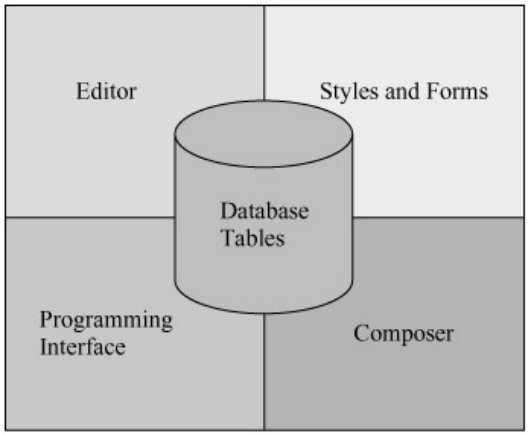


Figure 12.2: Components of the SAPscript tool

Structure of an SAPscript Form

The structure of an SAPscript form comprises two main components:

- **Content—** Can be either text (for example, the business data for an invoice) or graphics (for example, the company logo).
- **Layout—** Defined by a set of windows in which the content of the form appears.

Figure 12.3 shows the structure of an SAPscript form (an invoice):

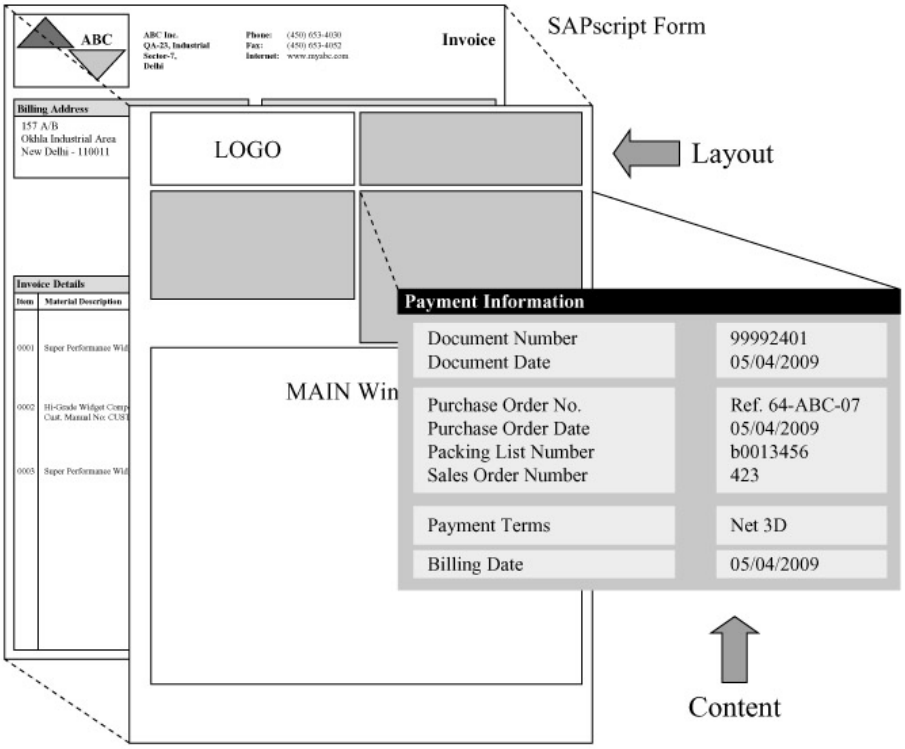


Figure 12.3: Structure of an SAPscript form

Figure 12.3 shows the layout and content of an SAPscript invoice. The layout is defined by a set of windows: the Logo window, Address window, Billing Address window, Information window, and Main window. These windows contain details of the invoice, as shown in Figure 12.4:

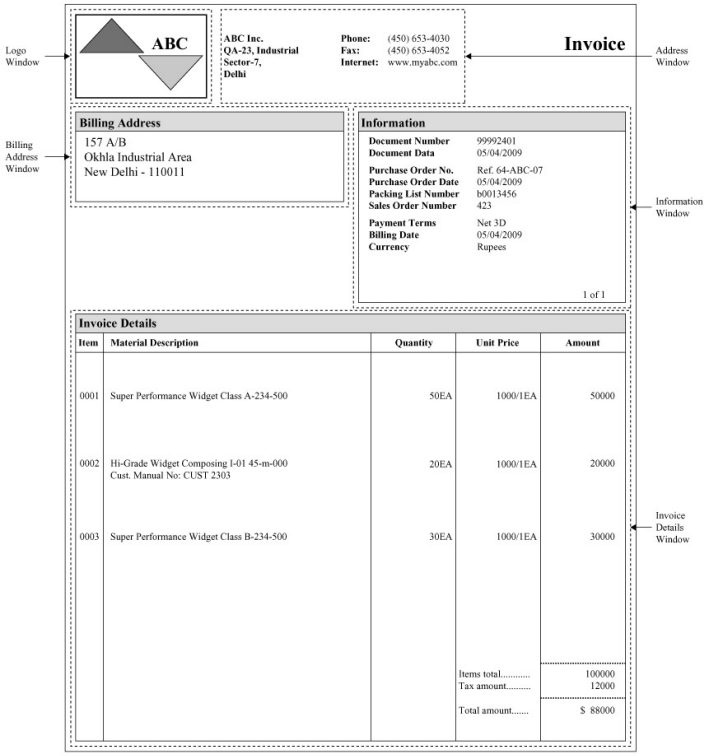


Figure 12.4: Windows in an SAPscript invoice form

Now, perform the following steps to access an SAPscript form:

1. In the SAP `Easy Access` screen, select SAP `menu > Tools > Form Printout > SAPscript`, as shown in

Figure 12.5:

2. Double-click the `Form` option in the `SAPscript` folder (Figure 12.5). The request screen of Form Painter appears, as shown in Figure 12.6:

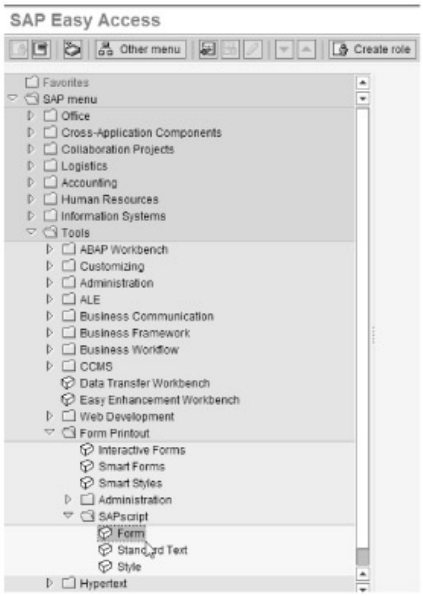


Figure 12.5: Selecting the form option in the SAPscript folder

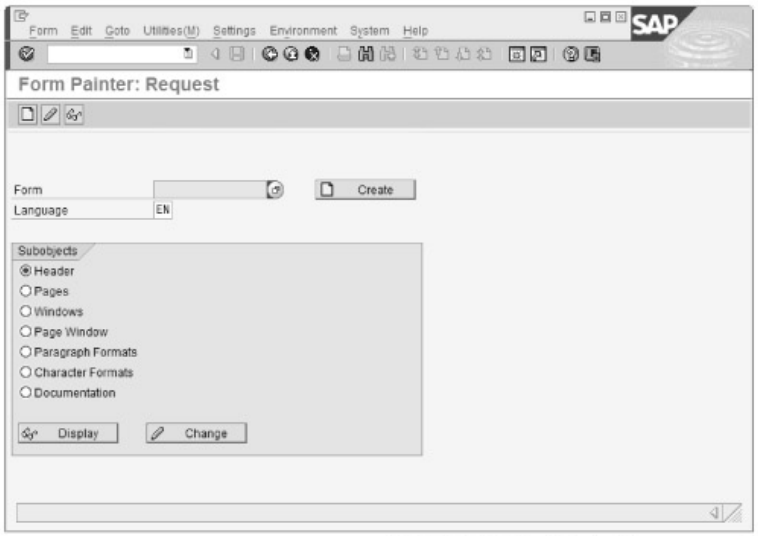


Figure 12.6: The form painter: request screen

Note An alternative way to open the `Form Painter: Request` screen is to write the `SE71` transaction code in the `Command` field and then click the `Enter` icon or press the ENTER key.

Managing Tools

The task of managing forms typically involves managing the layout and content. The SAP system provides the following tools to manage SAPscript forms:

- **PC Editor (or Graphical PC Editor)—** Manages the content of an SAPscript form. It is a graphical text-based tool.

- **Form Painter (or Graphical Form Painter)—** Manages the design and layout of an SAPscript form. It is a graphical tool.

Now, let's learn how to access and use these tools to manage the content of an SAPscript form.

Accessing PC Editor

In this section, we explain how to manage the content of an SAPscript form by using the PC Editor tool. In this case, we use the `RVORDER01` form, which is a predefined form in the mySAP ERP system.

Now, perform the following steps to display the RVORDER01 form by using the PC Editor tool:

1. In the `Form Painter: Request` screen (see Figure 12.6), enter a name for the form in the `Form` field, for example, RVORDER01. In addition, ensure that the `Language` field has EN as the default value.

2. In the `Subobjects` group box, select a radio button for any one of the following options: `Header`, `Pages`, `Windows`, `Page Window`, `Paragraph Formats`, `Character Formats`, and `Documentation`. Table 12.2 provides a brief description of these options:

Table 12.2: List of SAPscript subobjects

Subobject	Description
Header	Contains global data of an SAPscript form, such as its page format, page orientation, or the initially used font. It also includes basic information about the form, such as its name, description, and status (that is, whether the form is active or not).
Pages	Defines the pages included in an SAPscript form. A page can have attributes, such as the name of the next page and the type of page numbering. Note that the settings of the pages of a form, which are specified by a user, are maintained throughout the form.
Windows	Represents the logical units that do not have a physical position on a page, such as an address or logo window. A window name should reflect the text displayed in the window (for example, the address window would contain the address of a company or an organization). Every window is assigned a window type, such as a constant window, a variable window, or a main window. Unlike other windows, the text `Continuous` appears in the main window, which signifies that this window contains several pages.
Page Window	Describes the position and size of a window on a specific page of an SAPscript form.
Paragraph Formats	Specifies the font, tab, and outline information for the paragraphs in an SAPscript form.
Character Formats	Specifies information of the font in the paragraphs of an SAPscript form.
Documentation	Consists of technical documentation about the components of an SAPscript form, such as its pages and windows.

Note The preceding components are discussed in detail later in this chapter.

Figure 12.7 shows the `Header` radio button selected in the `Form Painter: Request` screen:

3. Click the `Display` or `Change` button on the `Form Painter: Request` screen. The `Display` button is used to open an SAPscript form in the display mode so that it can be read. The `Change` button is used to open the same form in the change mode so that it can be modified. In this case, we click the `Display` button (see Figure 12.7).

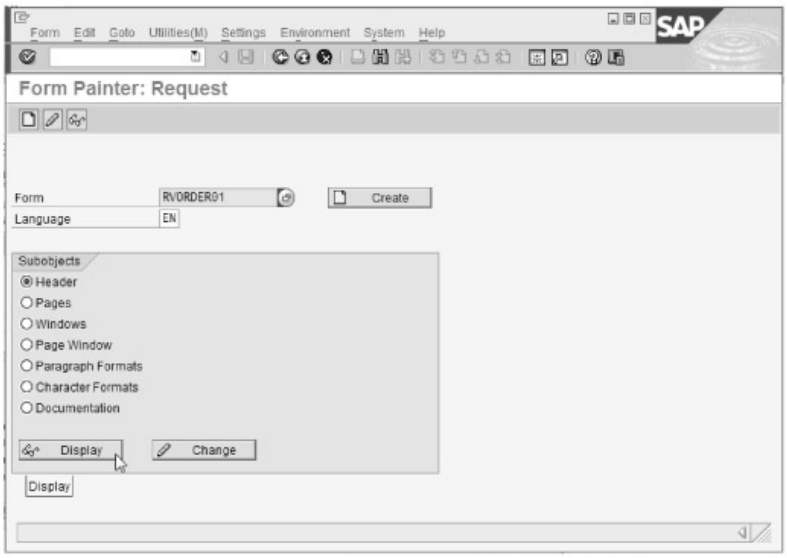


Figure 12.7: Selecting the header radio button

Note If your SAP client number is other than 000, the Information message box informs you that the `RVORDER01` form language EN is not available in client XXX. In this message, XXX stands for the client number of your SAP system.

Clicking the `Display` button displays the `Form: Display Header: RVORDER01` screen, as shown in Figure 12.8:

Figure 12.8 displays the information stored in the `Administrative Data` control of the `RVORDER01` form, which includes administrative information related to the form, such as the name, functionality, status, creation and modification dates, and language-related information, such as the current and original languages of the form.

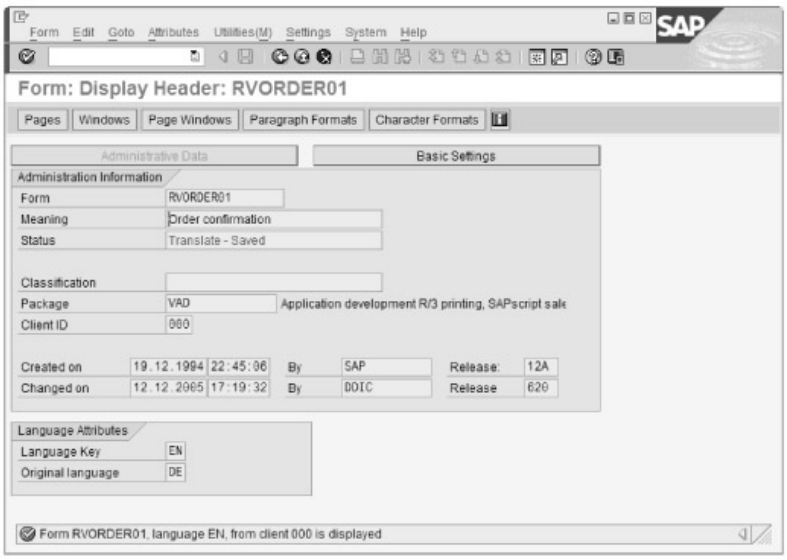


Figure 12.8: The form: display header: RVORDER01 screen

The `Basic Settings` control includes page-related information, such as page orientation, format, name, and default text formatting values pertaining to the paragraph, tab stop position, and font family.

Note To know more about administrative and page-related information of a form, refer to the "Form Subobjects" section of this chapter.

Now let's learn how to activate the Form Painter tool. Remember that, unlike the PC Editor tool, which is activated by default, you need to activate the Form Painter tool to access it.

Activating the Form Painter Tool

The Form Painter tool provides the graphical layout of an SAPscript form as well as the various functionalities to manipulate the form. Before using the Form Painter tool, you need to activate it by performing the following steps:

1. Open the `Form Painter: Request` screen by either navigating through the `SAP menu` or by using the `SE71` transaction code.
2. In the `Form Painter: Request` screen, select `Settings > Form Painter`, as shown in Figure 12.9:

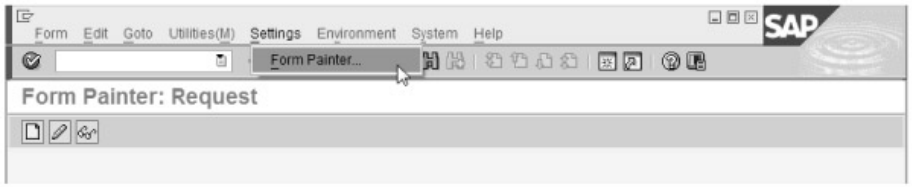


Figure 12.9: Displaying the form painter option

The `User-Specific Setting` dialog box appears.

3. In the `User-Specific Setting` dialog box, select the `Graphical Form Painter` check box in the `Form Painter` group box. In addition, ensure that the `Graphical PC editor` check box is selected by default, as shown in Figure 12.10:
4. Click the `Transfer` icon or press the ENTER key to use the settings specified in the `User- Specific Settings` dialog box for creating an SAPscript form.

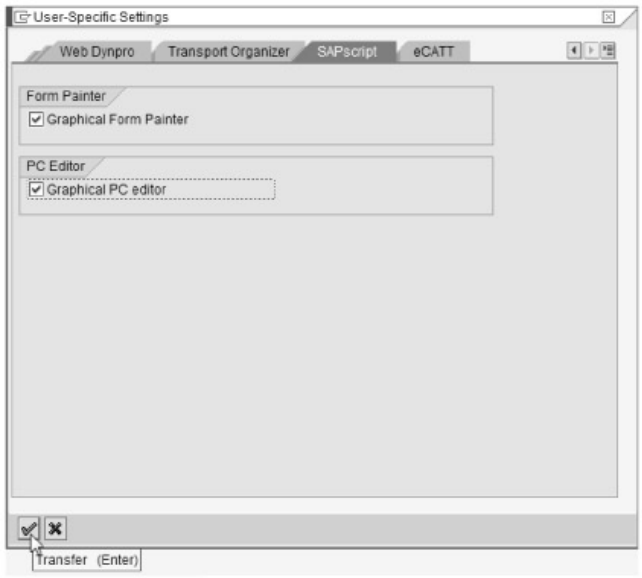


Figure 12.10: The user-specific setting dialog box

Note that the `Form Painter: Request` screen contains a different set of options in the `Subobjects` group box from those shown in Figure 12.7. Now, the `Subobjects` group box contains the `Header`, `Page Layout`, `Paragraph Formats`, `Character Formats`, and `Documentation` radio buttons.

Accessing the Form Painter Tool

Form Painter is a graphical tool used to design and set the layout of an SAPscript form. In this section, we learn how to create an invoice form after copying its basic layout structure from a standard SAPscript form, `RVINVOICE01`, and display its layout by accessing the Form Painter tool. Perform the following steps to do so:

1. Open the Form Painter: Request screen either by navigating the SAP menu or by using the SE71 transaction code.
2. In the Form Painter: Request screen, enter a name and language for an SAPscript form in the Form and Language fields, respectively. In this case, we have entered "RVINVOICE01" and "EN," respectively, in these fields (see Figure 12.11).
3. Select the Page Layout radio button in the Subobjects group box, as shown in Figure 12.11:
4. Next, select Utilities > Copy from Client to create a copy of the RVINVOICE01 form, as shown in Figure 12.12:

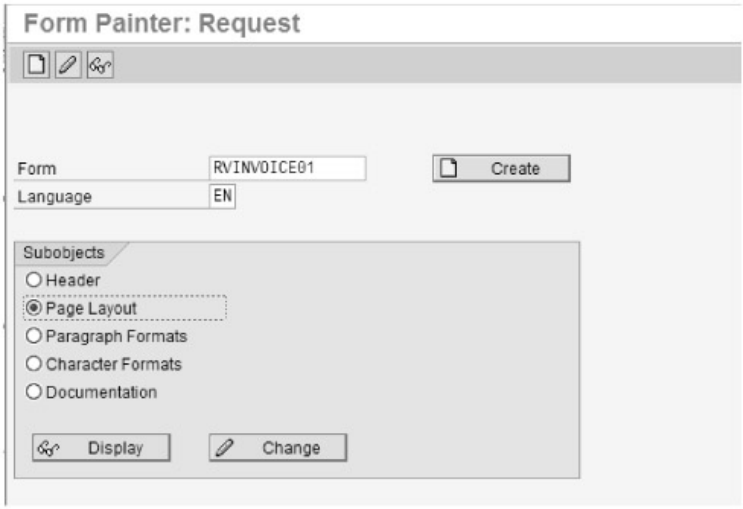


Figure 12.11: Entering field values in the form painter: request screen

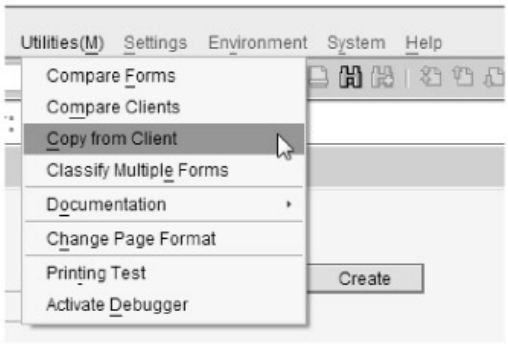


Figure 12.12: Selecting the copy from client option

The Copy Forms Between Clients screen appears.

5. In the Copy Forms Between Clients screen, enter the original name of the form, "RVINVOICE01," in the Form Name field, the number of the source client, "000," in the Source Client field, and the name of the target form, "ZINVOICE02," in the Target Form field. Ensure that other settings remain unchanged, as shown in Figure 12.13:
6. Next, click the Execute icon in the Copy Forms Between Clients screen (Figure 12.13). The Create Object Directory Entry dialog box appears.
7. In the Create Object Directory Entry dialog box, enter the package name, ZKOG_PCKG, in the Package field and click the Save icon. The Prompt for local Workbench request dialog box appears, as shown in Figure 12.14:
8. In the Prompt for local Workbench request dialog box, click the Continue icon. The ZINVOICE02

form is copied from the RVINVOICE01 form and displayed in the `Copy Forms Between Clients` screen, as shown in Figure 12.15:

9. Now, click the `Back` icon twice and navigate back to the `Form Painter: Request` screen, which contains the name of the copied form (ZINVOICE02), as shown in Figure 12.16:

10. Now, click the `Display` or `Change` button. In our case, we click the `Change` button (Figure 12.16). The `Form ZINVOICE02: Layout of Page FIRST` window and the `Form: Change Page Layout: ZINVOICE02` screen appear, as shown in Figure 12.17:

Figure 12.13: Specifying the fields in the copy forms between clients screen

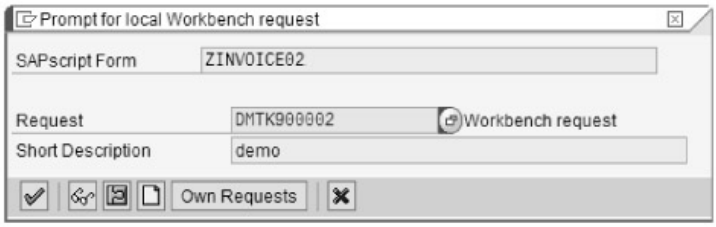


Figure 12.14: The prompt for local workbench request dialog box

Figure 12.15: Copying a form between two clients

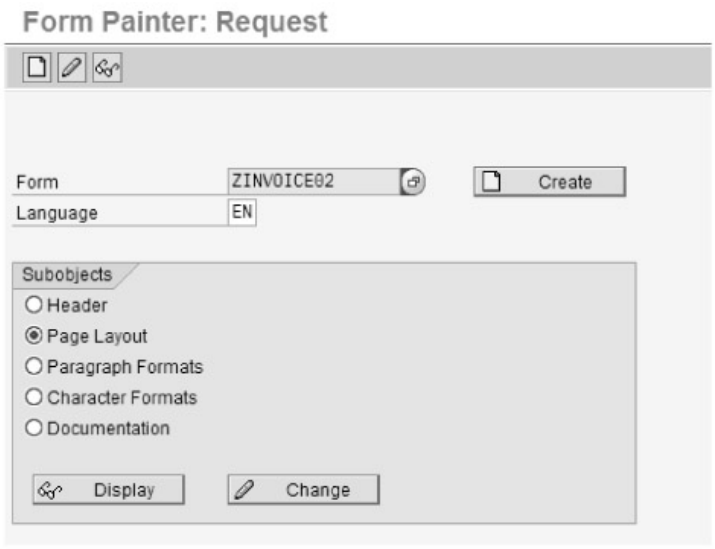


Figure 12.16: Displaying the ZINVOICE02 form in the form painter: request screen

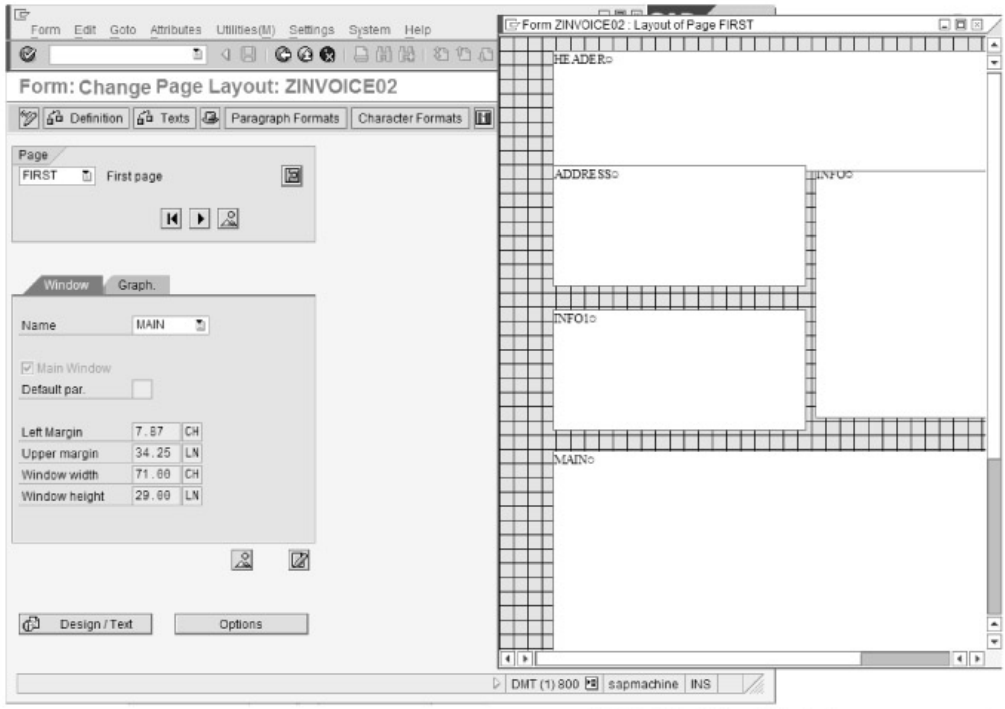


Figure 12.17: The form ZINVOICE02: layout of page FIRST window

In Figure 12.17, the `Form ZINVOICE02: Layout of Page FIRST` window shows the initial layout of the ZINVOICE02 form. The layout of the ZINVOICE02 form contains five windows: `HEADER`, `ADDRESS`, `INFO`, `INFO1`, and `MAIN`. The description of these windows can be accessed in PC Editor. For instance, select the MAIN window in the `Form ZINVOICE02: Layout of Page FIRST` window and click the `Text` icon in the `Form: Change Page Layout: ZINVOICE02` screen, as shown in Figure 12.18:

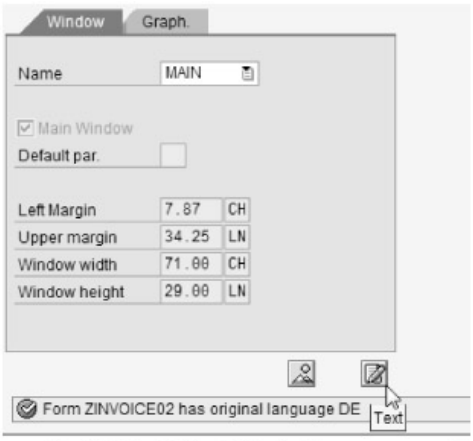


Figure 12.18: Text icon in the form: change page layout: ZINVOICE02 screen

The `Window MAIN` screen opens and displays a description of the main window in PC Editor, as shown in Figure 12.19:

You can also modify the text of the main window in PC Editor.

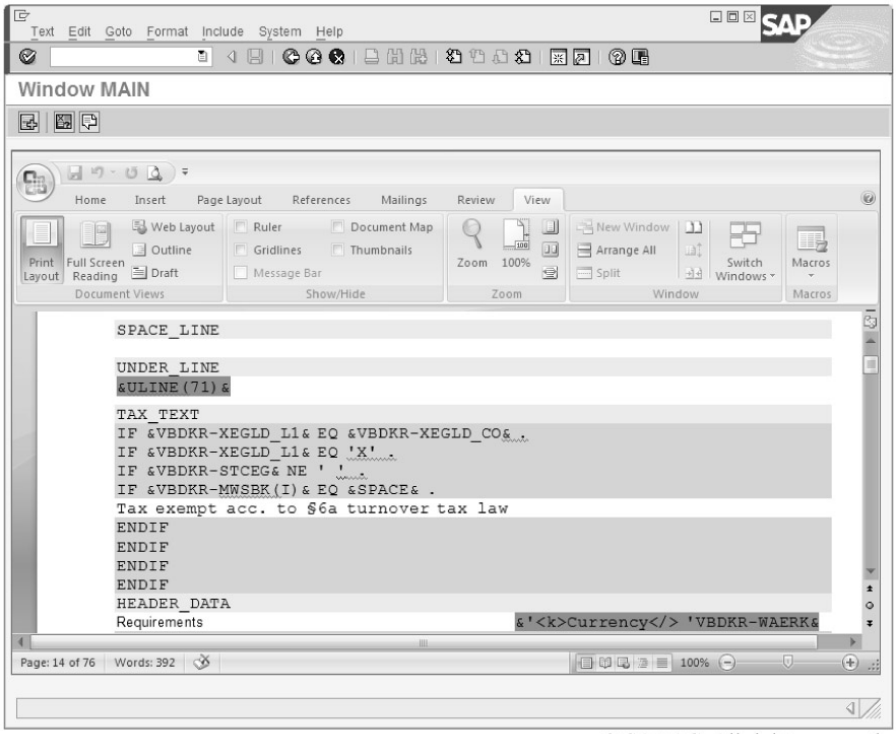


Figure 12.19: Displaying text in the window MAIN screen

Note When you change the text of any window, for example, the `Window MAIN`, remember to click the `Save` icon from the standard toolbar to save the changes.

Next, let's discuss the options available in the `Subobjects` group box in the `Form Painter: Request` screen.

Form Subobjects

The `Subobjects` group box in the `Form Painter: Request` screen contains the following options (see Figure 12.16):

- `Header`
- `Page Layout`

- Paragraph Formats
- Character Formats
- Documentation

Now, let's discuss these options in detail.

The *Header* Option

The Header option or subobject is used to display or accept the information related to an SAPscript form, such as its name and language, and the font used for the text of the SAPscript form. You select the Header radio button in the Form Painter: Request screen (see Figure 12.16) and open the SAPscript form by clicking the Display or Change button. In this case, we have clicked the Change button, so that the header of the ZINVOICE02 form is displayed in the change mode. Figure 12.20 shows administrative and language-related information of the ZINVOICE02 form:

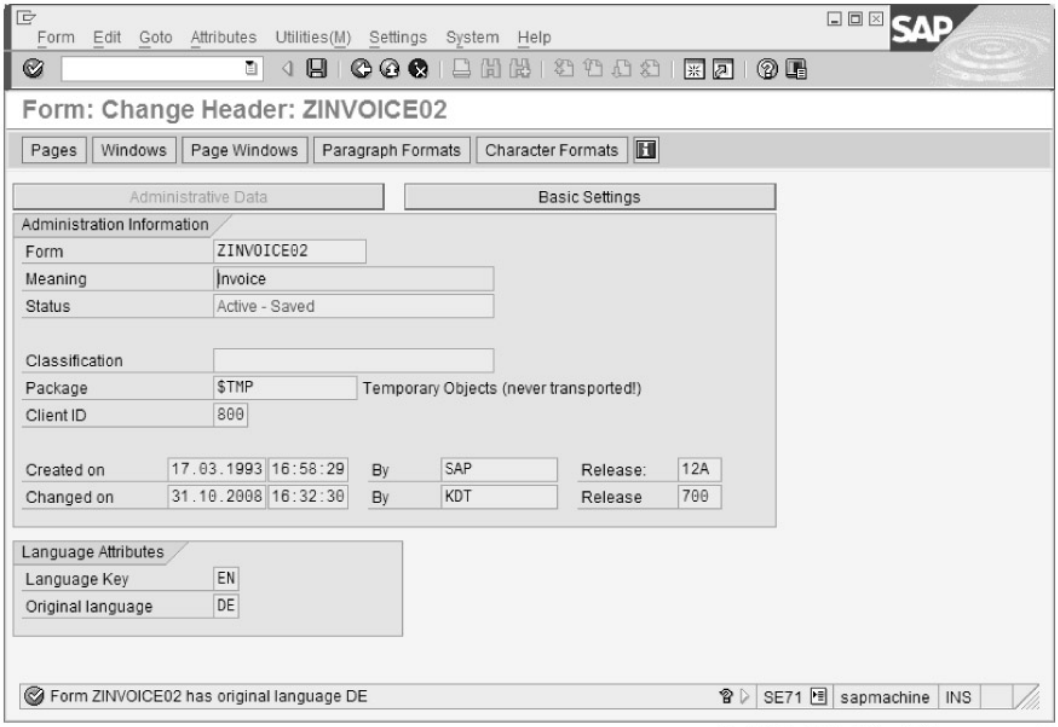


Figure 12.20: Displaying administration information of an invoice form

Figure 12.20 shows the administrative data of the ZINVOICE02 form in change mode. The Form: Change Header: ZINVOICE02 screen has two controls, Administrative Data and Basic Settings, which are used to specify the administrative and formatting information, respectively, of a form. In the Administrative Data control, the Administration Information group box is used to specify form-related information, such as its description, client number, and the dates on which the form was created and last modified.

The Language Attributes group box contains the Language Key field representing the current language of the SAPscript form and the Original language field representing the language in which the form was created.

The Basic Settings control two group boxes, Set up Page and Default Values for Text Formatting, which are used to set the values of an SAPscript form. Figure 12.21 shows the various elements of the Basic Settings control of an invoice form:

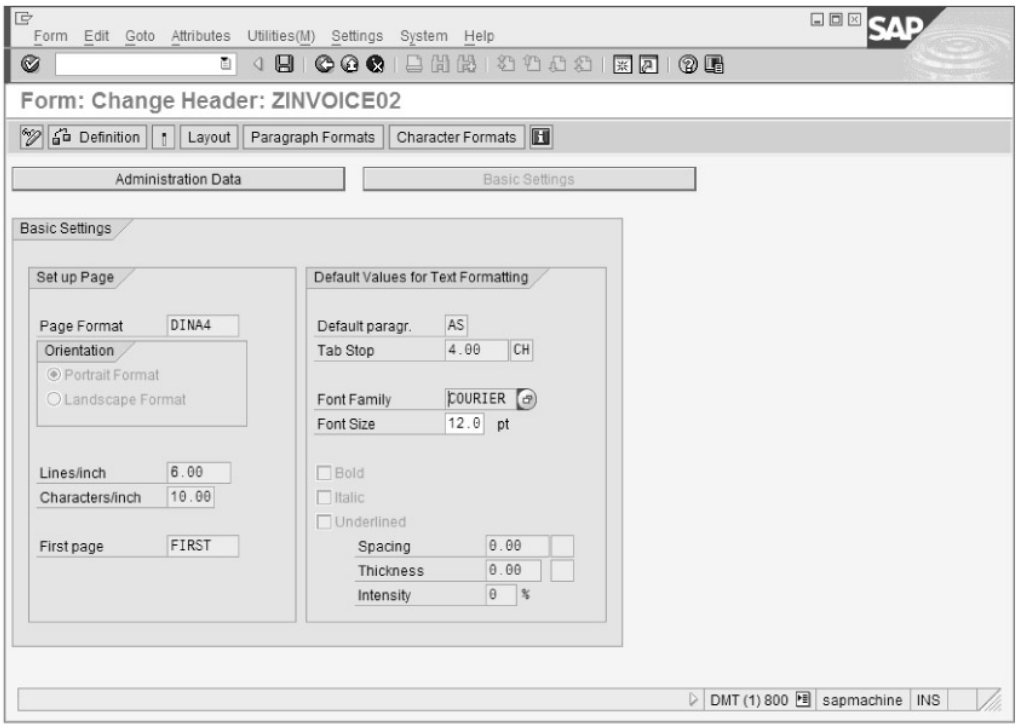


Figure 12.21: The basic settings control of an invoice form

Figure 12.21 shows that the `Basic Settings` control contains two group boxes, `Set up Page` and `Default Values for Text Formatting`. In the `Set up Page` group box, you can select the page format and orientation of an SAPscript form. For example, the page format of the SAPscript form shown in Figure 12.21 is DINA4. You can also specify values for the `Lines/Inch` and `Characters/inch` fields to set the number of lines or characters in the pages of an SAPscript form. In addition, you can specify the font family, font size, and tab position of the form. For example, the font family and font size of the form shown in Figure 12.21 are courier and 12.0 points, respectively.

Note You can navigate easily to other subobjects of an SAPscript form, such as `Layout` and `Paragraph Format`, from the screen for the `Header` subobject by selecting the corresponding button in the Application toolbar. For example, to navigate from the `Header` option to the `Page Layout` option, select the `Layout` button in the Application toolbar.

The *Page Layout* Option

Using the `Page Layout` option, you can define the individual pages and new windows needed for an SAPscript form. When you activate the Form Painter tool and select the `Page Layout` radio button in the `Subobjects` group box in the `Form Painter: Request` screen, the following interfaces appear:

- The `Form: Change Page Layout` screen (also called the `Administrative` screen)
- The `Form Layout of Page FIRST` window (also called the Design window)

Figure 12.22 shows the `Administrative` screen and the Design window, which are used to modify the layout of an SAPscript form:

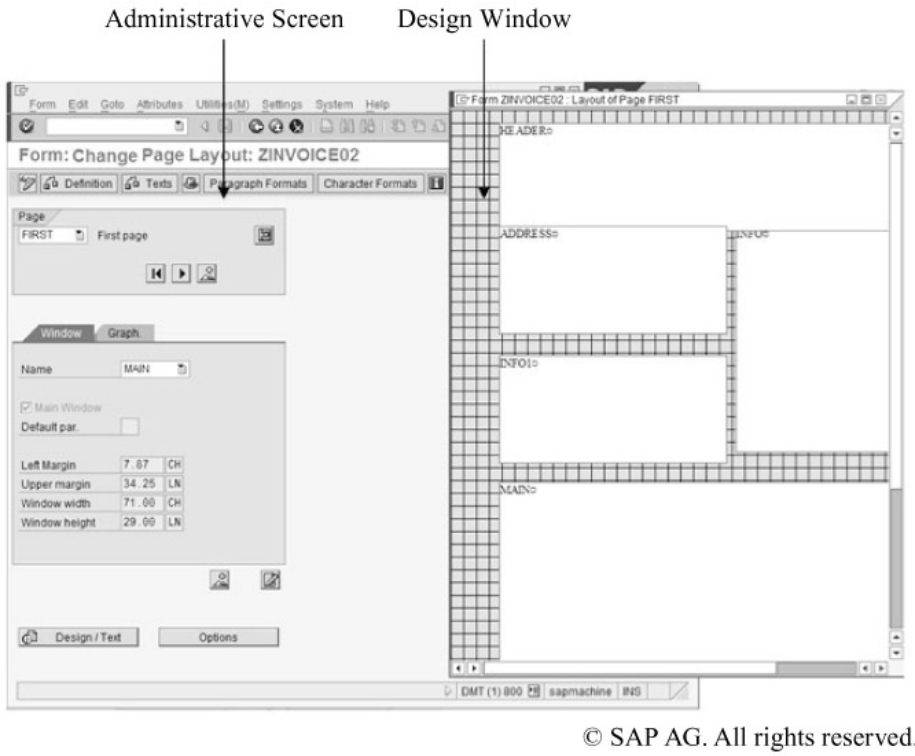


Figure 12.22: The administrative screen and the design window of an SAPscript form

In Figure 12.22, you see the `Administrative` screen, where you can enter page-related information such as page name, its windows, and their size and position on the page. In the Design window, you determine the layout of different windows of a page graphically. The SAP system always synchronizes the `Administrative` screen with the Design window and vice versa; that is, the changes made in any one of these are reflected in the other.

The `Administrative` screen and the Design window are used to determine the output areas (windows), define pages, and position the windows on the respective pages. In the `Administrative` screen, the different pages and the respective windows can be edited by specifying the values in the following areas:

- **The page area**—Specifies page-related information, such as the sequence and description of the page. Moreover, you can select `Edit > Page > Create` on this area to create individual pages, `Edit > Page > Copy` to create a page by using the copy function, `Edit > Page > Rename` to rename the page, and `Edit > Page > Delete` to delete individual pages.
- **The window area**—Specifies window-related information, such as a list of all the windows on the current page, and the text elements of the windows. You can select `Edit > Windows > Create` to create new windows (of the Variables or Main type) and the labels for the windows. By default, the SAP system creates a variable window (type VAR) and assigns it a default name. You can select `Edit > Windows > Rename` to rename a window, `Edit > Windows > Change type` to change the type of a window (Main or Var), `Edit > Windows > Delete` to delete individual windows on the current page, and `Edit > Windows > Text Elements` to edit the text elements of a window.
    - In addition, you can select `Edit > Clipboard > Cut to Clipboard` to cut individual windows and `Edit > Clipboard > Copy to Clipboard` to copy individual windows to the clipboard. The clip board stores the size and position of a window as well as its text elements. You can also paste a window, including its text elements, size, and position, by selecting `Edit > Clipboard > Paste From Clipboard` to any page of the form. In addition, you can align all the windows by selecting `Edit > Align With Grid`. Remember that the main window, unlike the variable window, must always have the same width on all the pages. If a user changes the width of the main window on a page, this change is reflected on all the pages as well.

The `Text` icon is used to edit the text elements of a selected window in the Line Editor or WYSIWYG PC Editor (see Figure 12.19). You can switch between these two editors by selecting `Go To > Change editor`. For example, the text elements of the `Window MAIN` are displayed in Line Editor, as shown in Figure 12.23:

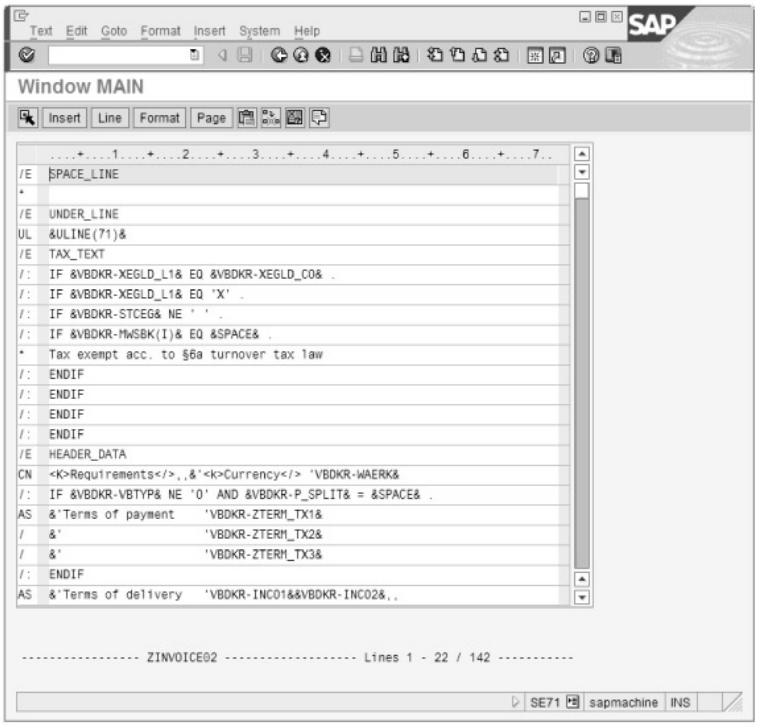


Figure 12.23: Description of the window MAIN in line editor

The text area in Line Editor consists of the ruler and the subsequent lines where you can enter text.

Note When you define the window of a form, you have to select its window type. The following are the three types of windows:

- Constant windows (CONST)—Specifies the windows of the same size in all the pages of an SAPscript form.
- Variable windows (VAR)—Represents the windows of different sizes on different pages of an SAPscript form. If the text exceeds the window size, the text is truncated, but the SAP system does not trigger a page break.
- Main windows—Contains the text body of an SAPscript form, which may span several pages. The Window MAIN controls the page break and allows you to specify the position of the text elements at the upper and lower margins of the window.

The Design window has a grid surface that allows you to align various windows of a form easily. From the `Administrative` screen, you can assign two different modes, design and text, to the Design window, by clicking the `Design/Text` (Design / Text) button. In the Design mode, you can use the mouse to position or change the size of a window. In the Text mode, however, the SAP system displays the text elements of the individual windows. You can view the text elements by using the scrolling feature of the window. However, the text is read-only.

The Paragraph Formats Option

The `Paragraph Formats` option is used to define the information needed to format text in an SAPscript form. Usually, not all formatting features are used in a form because most form paragraphs consist of only a line or word.

The font and tab attributes generally are used to specify a paragraph format in an SAPscript form. If no font is specified, the default font is used for the header of the form. Figure 12.24 shows the font attributes of a paragraph definition:

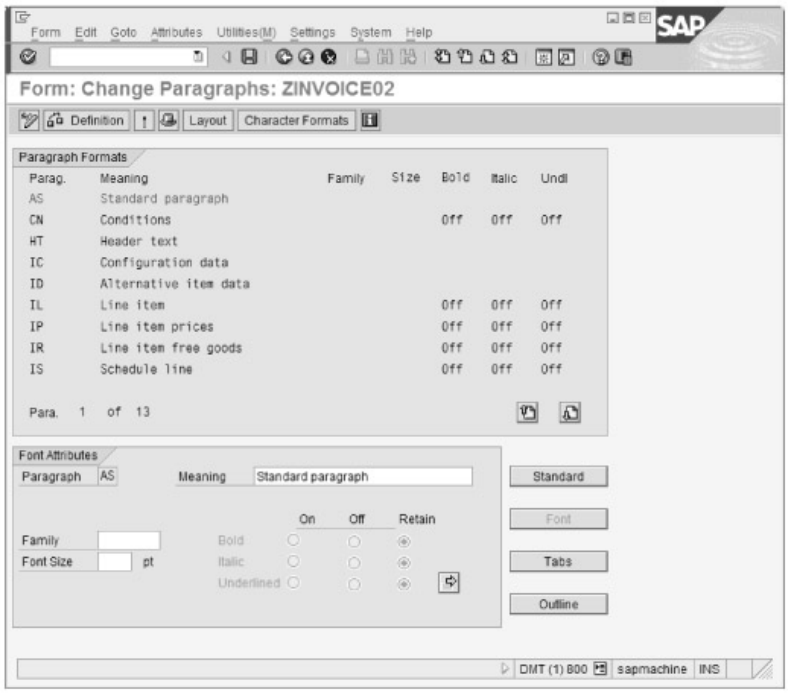


Figure 12.24: Font attributes in the form: change paragraphs screen

Using tab attributes, you can display the text data of a paragraph in the columns format.

Figure 12.25 shows the tab attributes of a paragraph definition:

In Figure 12.25, you can specify the values of the fields in the `Tabs` group box, such as meaning (or description) of a paragraph and position and alignment of tabs.

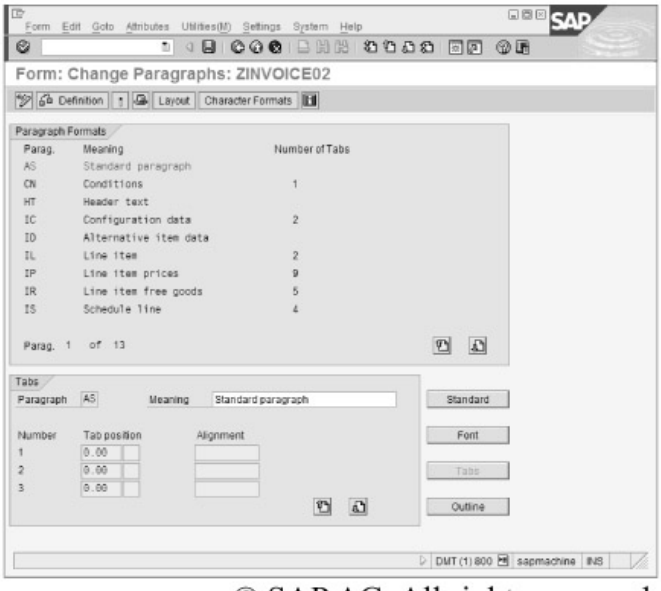


Figure 12.25: Tab attributes in the form: change paragraphs screen

The Character Formats Option

The `Character Formats` option is used to specify the formatting options for the character data of an SAPscript form; for example, italicizing a single word in a paragraph, instead of the entire paragraph. Figure 12.26 shows the `Form: Change Character Strings: ZINVOICE02` screen, which accepts character formatting information of the ZINVOICE02 form in

the change mode:

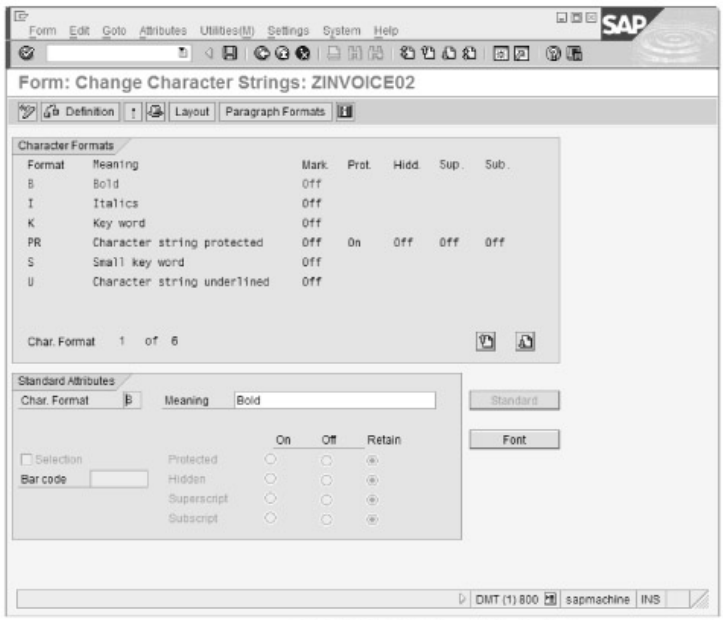


Figure 12.26: Font attributes in the form: change character strings screen

In the `Standard Attributes` group box (see Figure 12.26), you can specify a bar code format as a character format for the text. You can use PC Editor to include a character string by enclosing the name of the character string in angular brackets (< >) before the specific text and inserting a slash within angular brackets (</>) at the end of the specific text. For example, <B>These words are in the bold format </B>.

The font attributes of a character string can be accessed by clicking the `Font` button in the `Form: Change Character Strings` screen. You can specify the font attributes to underline, italicize, or bold a character string by selecting the respective radio button.

In Figure 12.26, the `Retain` radio button is used to retain the underline, italic, or bold setting in a paragraph. If a character string has no specified font, the default paragraph font is used.

Figure 12.26 displays the character string B, which changes the format to bold while retaining the Italics and Underline settings in the paragraph.

The Documentation Option

The `Documentation` option is used to create, display, or edit information about the windows and text elements required in an SAPscript form. For instance, you can create a help document corresponding to an SAPscript form, which can include the description of the program symbols used in the program or the purpose of a specific window.

In an SAP system, the documentation of a form is a form itself. This means that the documentation of a form is also language dependent and is involved in the transportation of the form. When you create a document for a form, the SAP system automatically generates a template for the form. This template contains certain sections of read-only lines; however, you can provide additional information about these sections.

In the documentation of an SAPscript form, the read-only lines represent different objects of the form; that is, general information of the form, its pages, windows, and text elements. Figure 12.27 shows the documentation of the `RVORDER01` SAPscript form in the display mode:

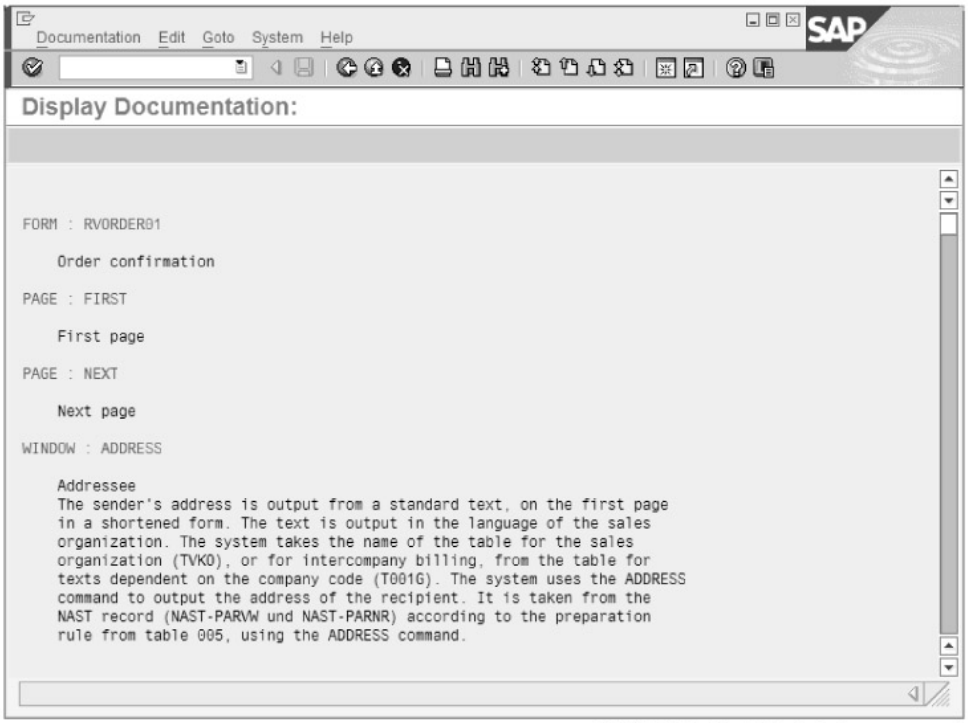


Figure 12.27: The display documentation screen

The `Display Documentation` screen displays useful information about an SAPscript form, such as its description, the windows it uses, and the text elements it contains. There is a keyword for each text element (`FORM`, `PAGE`, `WINDOW`, and `ELEMENT`) that is followed by its corresponding definition. The `FORM` keyword is followed by the name of the form, the `PAGE` keyword is followed by different page descriptions, the `WINDOW` keyword is followed by the description for different kinds of windows, and the `ELEMENT` keyword is followed by the description of text elements.

You can enhance the already-existing documentation of an SAPscript form by adding new objects to it, which become active after saving the form. When you delete the pages, windows, or text elements of an SAPscript form, the SAP system does not delete the existing documentation; it flags it as "`no longer needed *`" beside the corresponding keyword. Moreover, if you create an object with the same name later, the SAP system deletes the flag and preserves the documentation. Note that the objects flagged as "`no longer needed *`" always appear at the end of the object list but in the same sequence of objects (`FORM`, `PAGE`, `WINDOW`, and `ELEMENT`).

You can select Utilities > Documentation > Clean up to delete all sections that are flagged as no longer needed `*`. This also deletes all the read-only lines for which no documentation exists. You can also delete the entire documentation of a form by selecting `Utilities > Documentation > Delete` in the `Form Paint: Request` screen. Moreover, you can print the documentation of a form by selecting `Utilities > Documentation > Print`. However, only documented objects are printed.

The SAPscript Runtime Environment

The SAPscript runtime environment coordinates the processing of SAPscript forms. It performs the following tasks:

- Retrieving the layout and content data from an SAPscript form
- Collecting the necessary business data from the SAP R/3 database
- Generating the final SAPscript form

The resulting business form can be printed, e-mailed, faxed, or displayed on your computer. Figure 12.28 shows the SAPscript runtime environment in the SAP R/3 system:

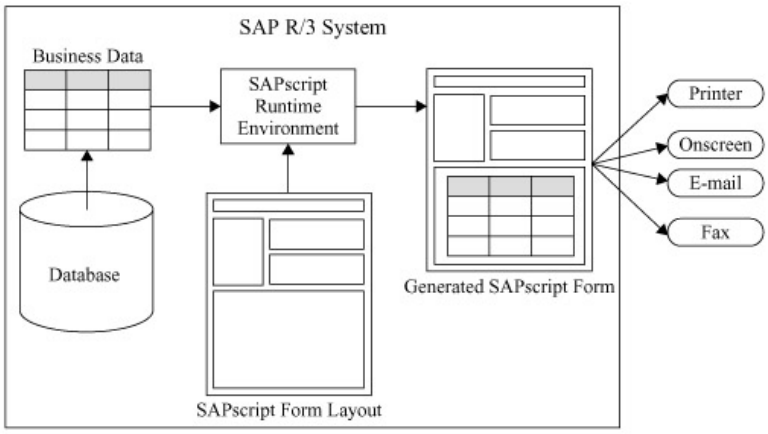


Figure 12.28: SAPscript runtime environment

Print Program

A print program is an application program (such as Report or ModulePool) used to print forms. The print program retrieves the required data from a database table, defines the order in which the elements of the text are printed, selects the form to print, selects an output device and print options, and processes and prints the form.

A print program uses the following function modules to print a form:

- `OPEN_FORM`
- `CLOSE_FORM`
- `WRITE_FORM`
- `START_FORM`
- `END_FORM`
- `CONTROL_FORM`

Note To know more about these function modules, refer to the "SAPscript Function Modules" section of this chapter.

The `OPEN_FORM` and `CLOSE_FORM` function modules are used to open and close a form, respectively; and the `START_FORM` and `END_FORM` function modules combine forms into a single spool request for printing. The `WRITE_FORM` function module determines the order in which the text elements are printed. Finally, the `CONTROL_FORM` function module transfers the control statements (for instance, `NEW-PAGE`) from an ABAP program to a form at runtime. A print program can be divided into six parts:

1. **Part 1—** Contains statements to read data from a table. The syntax to read data from a table is:
   ```
   Tables: abc.
   SELECT * FROM abc.
   ```
2. **Part 2—** Contains the `OPEN_FORM` function module to open a form.
3. **Part 3—** Contains the `START_FORM` function module to start a spool request for printing.
4. **Part 4 —** Contains the `WRITE_FORM` function module to write the text elements of the windows of the form.
5. **Part 5—** Contains the `END_FORM` function module to end the spool request started by the START_FORM function module.
6. **Part 6—** Contains the `CLOSE_FORM` function module to close the form after displaying or printing the required data.

For example, RSTXEXP1 is a standard print program in the mySAP ERP system and corresponds to the S_EXAMPLE_1 form. Listing 12.1 shows the code of the RSTXEXP1 print program:

Listing 12.1: Code of the RSTXEXP1 print program

```
report rstxexp1.
tables: scustom, sbook, spfli.
select-options: s_id for scustom-id default 1 to 1,
              s_fli for sbook-carrid default 'LH' to 'LH'.
data customers like scustom occurs 100
              with header line.
data bookings like sbook occurs 1000
              with header line.
data connections like spfli occurs 1000
              with header line.
data: begin of sums occurs 10,
              forcuram like sbook-forcuram,
              forcurkey like sbook-forcurkey,
      end of sums.

* Get data
select * from scustom into table customers
              where id in s_id
              order by primary key.
select * from sbook into table bookings
              where customid in s_id and carrid in s_fli
              order by primary key.
select * from spfli into table connections
              for all entries in bookings
              where carrid = bookings-carrid
              and connid = bookings-connid
              order by primary key.
* Open print job

call function 'OPEN_FORM'
      exporting
                    device      = 'PRINTER'
                    form        = 'S_EXAMPLE_1'
                    dialog      = 'X'
      exceptions
                    canceled    = 1
                    device      = 2
                    form        = 3
                    options     = 4
                    unclosed    = 5
                    others      = 6.
if sy-subrc <> 0.
      write 'Error in open_form'(001).
      exit.
endif.

* Print form for all customers
loop at customers.
* Set customer address
      scustom = customers.
* Open form of respective customer
      call function 'START_FORM'
            exceptions
                     others     = 1.
if sy-subrc <> 0.
      write 'Error in start_form'(002).
      exit.
endif.

* Display column headings of main window
      call function 'WRITE_FORM'
            exporting
                   element      = 'HEADING'
                   function     = 'SET'
                   type         = 'TOP'
                   window       = 'MAIN'
            exceptions
```

```
                    others       = 1.

  if sy-subrc <> 0.
        write 'Error in write_form printing top element of
        main'(003).
        exit.
  endif.
  * Customer bookings
        clear sums. refresh sums.
        loop at bookings
                where customid = customers-id.
        sbook = bookings.
  * Get departure time
        read table connections with key carrid = bookings
                                                -carrid
                                        connid = bookings
                                                -connid.
        if sy-subrc = 0.
                spfli = connections.
        else.
                clear spfli.
        endif.
  * Print position
        call function 'WRITE_FORM'
                exporting
                       element        = 'BOOKING'
                       function       = 'SET'
                       type           = 'BODY'
                       window         = 'MAIN'
                exceptions
                       others         = 1.
        if sy-subrc <> 0.
                write 'Error in write_form printing body of
                main'(004).
                exit.
        endif.
  * Add current position to corresponding entry in table
  sums
        move-corresponding sbook to sums.
        collect sums.
  endloop.              " at bookings
  * Print sum

        loop at sums.
        move-corresponding sums to sbook.
        call function 'WRITE_FORM'
                exporting
                       element       = 'SUM'
                       function      = 'SET'
                       type          = 'BODY'
                       window        = 'MAIN'
                exceptions
                       others        = 1.
        if sy-subrc <> 0.
        write 'Error in write_form printing sum of
        invoice'(005).
        exit.
  endif.
        endloop.       " at sums

  * Close customer form
        call function 'END_FORM'
                exceptions
                       others         = 1.
  if sy-subrc <> 0.
                write 'Error in end_form'(006).
                exit.
        endif.
  endloop.      " at customers
```

```
* close print job
call function 'CLOSE_FORM'
              exceptions
                     others         = 1.
if sy-subrc <> 0.
      write 'Error in close_form'(007).
      exit.
endif.
```

The code given in Listing 12.1 creates an invoice form that contains company-related information, such as the flight bookings of a customer, customer address, and flight booking dates.

In Listing 12.1, the data of the form is read from a database table and populated in internal tables (for example, BOOKINGS). The `OPEN_FORM` function module is called to open the S_EXAMPLE_1 form for printing. The `WRITE_FORM` function module is used to write the `HEADING` text element in the `MAIN` window of the form. The `HEADING` text element displays the column or field names of the invoice in the `MAIN` window of the form. The `BOOKING` text element in the `MAIN` window provides information related to the bookings of the customer. The information displayed by the BOOKING text element is retrieved by using the BOOKINGS internal table in a loop. The address of the customer and company-related information are displayed by using default text elements. The `CLOSE_FORM` function module finally closes the form.

Note The various function modules are discussed in detail later in this chapter.

Now, perform the following steps to view a print preview of the S_EXAMPLE_1 form through the RSTXEXP1 print program:

1. Open ABAP Editor by either navigating through the SAP menu or executing the SE38 transaction code and entering the name of the print program, RSTXEXP1, in the specified `Program` text field, as shown in Figure 12.29:
2. Select the `Source code` radio button in the Subobjects list and then click the `Display` button (see Figure 12.29). ABAP Editor displays the source code of the RSTXEXP1 print program, as shown in Figure 12.30:
3. Click the `Check` icon to check the source code of the program. Next, click the `Activate` icon to activate the program and the `Direct Processing` icon to process the RSTXEXP1 program (see Figure 12.30).

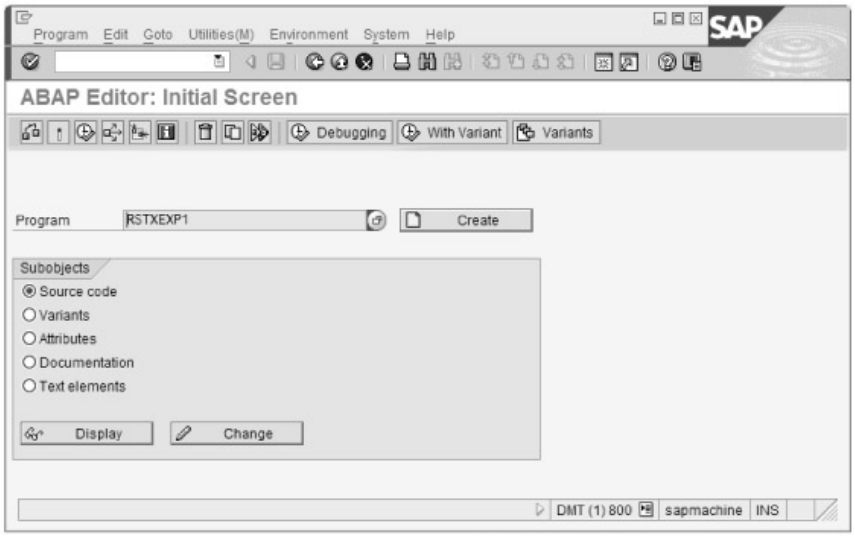


Figure 12.29: Entering the program name in ABAP editor

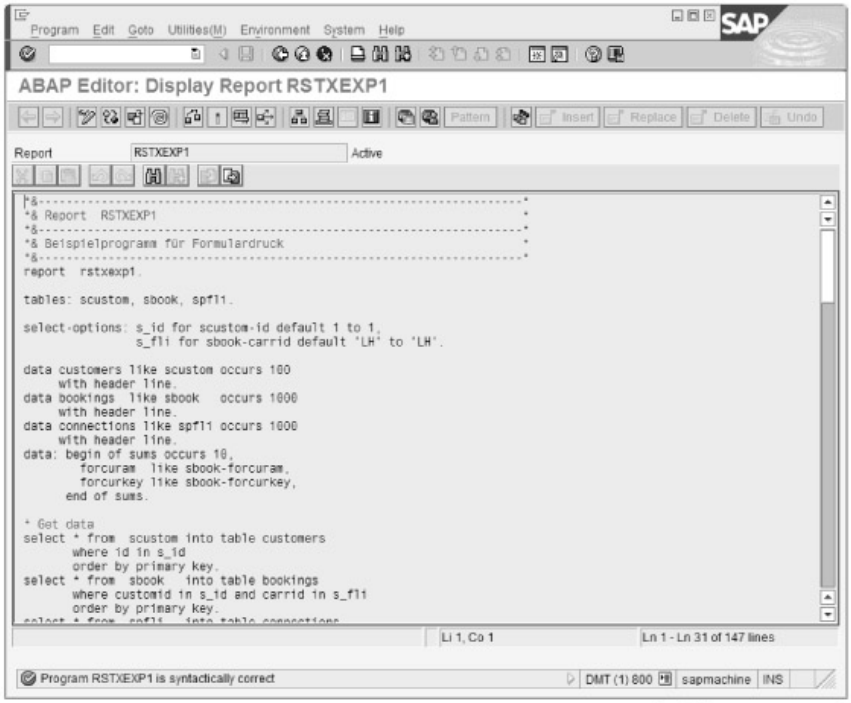


Figure 12.30: Source code of the RSTXEXP1 print program

The `SAPscript: Sample Program for Form Printing` screen appears, as shown in Figure 12.31:

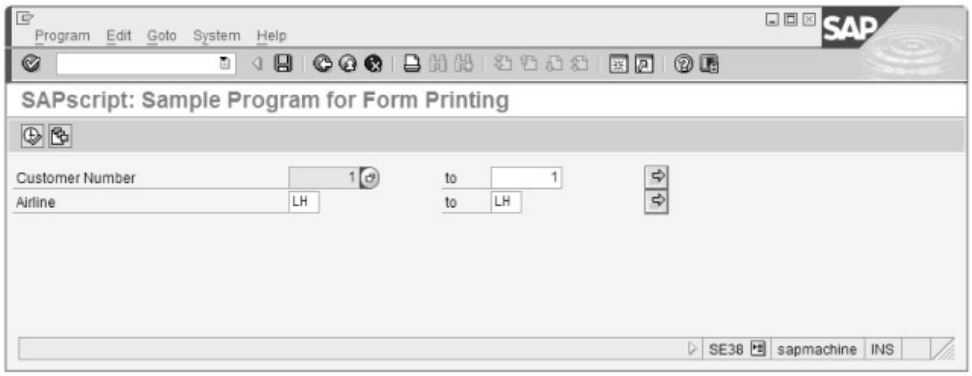


Figure 12.31: The SAPscript: sample program for form printing screen

4. Click the `Execute` (F8) icon or press the F8 key, as shown in Figure 12.31. The `Print` dialog box appears.
5. Enter the value lp01 in the `Output Device` text field of the `Print` dialog box, as shown in Figure 12.32:
6. Click the `Print Preview` button of the `Print` dialog box (see Figure 12.32) to display the print preview of the S_EXAMPLE_1 form.

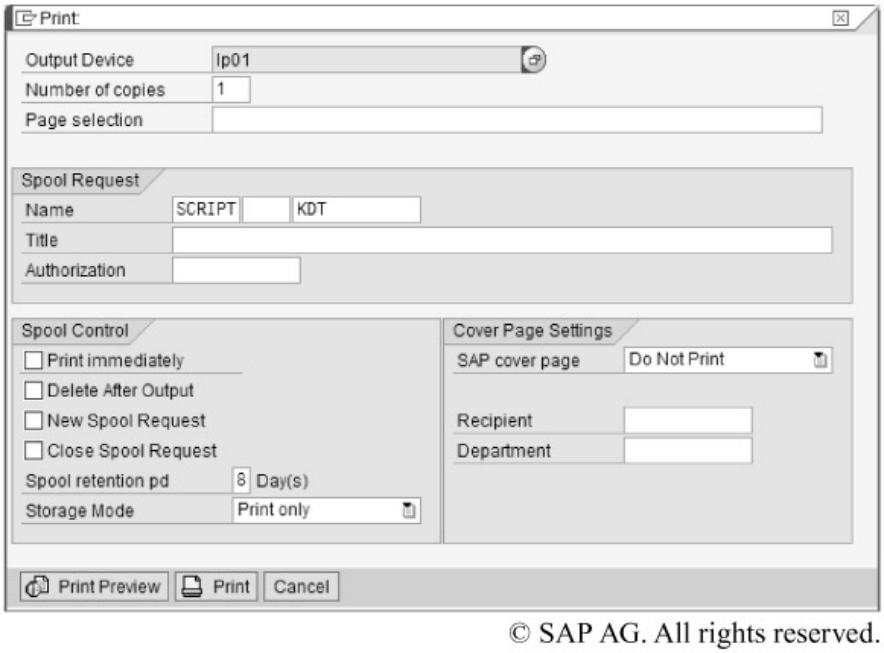


Figure 12.32: The print dialog box

Figure 12.33 shows the print preview of the S_EXAMPLE_1 SAPscript form:

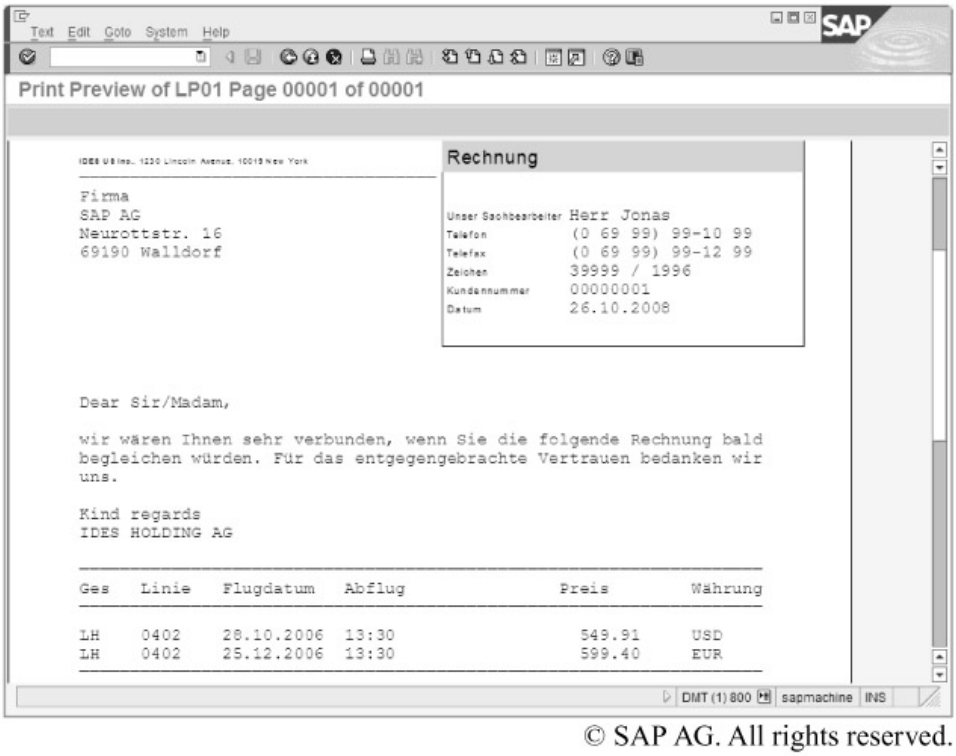


Figure 12.33: Print preview of the S_EXAMPLE_1 form

SAPscript Function Modules

Table 12.3 describes some commonly used function modules to process SAPscript forms:

Table 12.3: Commonly used function modules for SAPscript forms

Function Module	Description
OPEN_FORM	Opens a form to display or print it.

CLOSE_FORM	Closes a form.
START_FORM	Starts a new form for spooling a print request.
WRITE_FORM	Writes a text element in a form.
WRITE_FORM_LINES	Writes lines of text in a form.
END_FORM	Ends or terminates the spool request of printing the current form.
CONTROL_FORM	Sends a control statement to a form.
READ_FORM_ELEMENTS	Retrieves the elements of a form.
READ_FORM_LINES	Reads the lines of the elements of a form and stores them in an internal table.

Now, let's describe each function module in detail.

The *OPEN_FORM* Function Module

The OPEN_FORM function module opens an SAPscript form to display or print its data. It is, therefore, used before other function modules. Using the name of a form is optional with the OPEN_FORM function module. However, if the name of the form is omitted, the START_FORM function module must be used to open the form. A form must be closed by the CLOSE_FORM function module; otherwise, the SAP system does not print or display the form.

Note that a pair of OPEN_FORM... CLOSE_FORM function modules can be used any number of times in a program. This allows you to display or print a form for several different spool requests from one program. The following is the syntax used to call the OPEN_FORM function module:

```
CALL FUNCTION 'OPEN_FORM'
EXPORTING FORM = SPACE
LANGUAGE = SY-LANGU
DEVICE = 'PRINTER'
DIALOG = 'X'
OPTIONS = SPACE
APPLICATION = 'TX'
ARCHIVE_INDEX = SPACE
ARCHIVE_PARAMS = SPACE
IMPORTING LANGUAGE =
RESULT =
NEW_ARCHIVE_PARAMS =
EXCEPTIONS CANCELED =
DEVICE =
FORM =
OPTIONS =
UNCLOSED =
```

The OPEN_FORM function module is called by using the export parameters listed in Table 12.4:

Table 12.4: Export parameters in the OPEN_FORM function module

Parameter	Description
FORM	Specifies the name of a form. The default value of this parameter is SPACE. If this parameter is left blank, the START_FORM function module must be called with a valid form name before starting a spool request for printing.
LANGUAGE	Specifies the desired language, as forms are language-dependent. The default value of this parameter is SY-LANGU.
DEVICE	Specifies the desired device type for the output of a form. The possible values for this parameter are PRINTER for the print output, TELEX for the telex output, TELEFAX for the telefax output, ABAP for the output as an ABAP list, and SCREEN for the screen output. The default value of this parameter is PRINTER.
DIALOG	Determines whether or not to display a dialog box before printing the form. The default value of this parameter is X, which displays the print parameter screen. The other possible value for this parameter is ' ', which does not display the print parameter screen.
OPTIONS	Sets several options to specify the format to print a form. The default value of this paramter is SPACE.
APPLICATION	Specifies an interface name provided by SAPscript. The default value of this parameter is TX.
ARCHIVE_INDEX	Specifies index information for the print output that needs to be archived. The default value of this parameter is SPACE.

ARCHIVE_PARAMS	Specifies a value that is interpreted by the SAP system to archive the output. The default value of this parameter is SPACE.

The OPEN_FORM function module uses the import parameters listed in Table 12.5:

Table 12.5: Import parameters in the OPEN_FORM function module

Parameter	Description
LANGUAGE	Specifies the language of a form.
RESULT	Contains the results of the process to specify a format to print a form.
NEW_ARCHIVE_PARAMS	Contains the results of the archive process.

Calling the OPEN_FORM function module can raise exceptions when the parameters specified in Table 12.6 contain invalid values:

Table 12.6: Exception parameters in the OPEN_FORM function module

Parameter	Description
CANCELED	Handles the exception raised when no form is opened for output.
DEVICE	Handles the exception raised when an invalid device type is found.
FORM	Handles the exception raised when a form of the specified name is not found in an SAP system. The possible reasons for this may be that either the form does not exist or there is no active version of the form.
OPTIONS	Handles the exception raised when invalid values are specified for the formatting options while printing a form.
UNCLOSED	Handles the exception raised when an SAP system tries to open another form without closing the earlier form.

Note In an SAP system, the searching criteria to find a form can be explained in the following way: SAPscript first searches for the form in the current client and in the specified language. If the form is not found, SAPscript tries the original language of the form. If the form is still not found, it searches for the form in client 000, first in the specified language and then in the original language of the form.

The *CLOSE_FORM* Function Module

The CLOSE_FORM function module closes an open form. You can use this function module to close the form; otherwise, no output appears on the printer or screen.

The syntax to call the CLOSE_FORM function module is:

```
CALL FUNCTION 'CLOSE_FORM'
IMPORTING RESULT =
TABLES OTFDATA = ?...
EXCEPTIONS UNOPENED =
```

In this syntax, RESULT is the import parameter that receives the results of formatting a form for printing. By comparing the fields of the OPTIONS parameter (of the OPEN_FORM function module) with the corresponding fields of the RESULT parameter, you can find out whether any changes have been made to the settings on the print control screen. OTFDATA is the table parameter, which is optional if the value of the OPTIONS parameter is X for the TDGETOTF field. This is because in this case, the SAP system returns the formatted output in the OTF format, which means that the SAP system does not produce any output to printer, screen, faxmachine, or telex. The SAP system throws the UNOPENED runtime exception if it is not able to execute the current form function because the output of the form has not yet been initialized by the OPEN_FORM function module.

Note that you can open or close one or more forms by using the OPEN_FORM and CLOSE_FORM function modules, respectively. In addition, you can combine multiple forms into one print output, but all these forms must have the same page format.

The *START_FORM* Function Module

The START_FORM function module is used to start a spool request to print or display a form. On the contrary, the END_FORM function module is used to stop or end the spool request for printing or displaying the form. You can use the pair of START_FORM and END_FORM function modules more than once inside the pair of the OPEN_FORM and CLOSE_FORM function modules.

You can also use the START_FORM function module to switch from one form to another. For this, you must end the spool request to print the first form by using the END_FORM function module and then use the START_FORM function module to start the the spool request to print the second form.

If you do not specify the name of a form when calling the START_FORM function module, the SAP system restarts the process of formatting the last opened form. If no form is activated after the OPEN_FORM function module is called, the SAP system raises the UNUSED exception.

The syntax to call the START_FORM function module is:

```
CALL FUNCTION 'START_FORM'
EXPORTING FORM = SPACE
LANGUAGE = SPACE
STARTPAGE = SPACE
PROGRAM = SPACE
ARCHIVE_INDEX = SPACE
IMPORTING LANGUAGE =

EXCEPTIONS FORM =
FORMAT =
UNENDED =
UNOPENED =
UNUSED =
```

The START_FORM function module is called by using the export parameters listed in Table 12.7:

Table 12.7: Export parameters in the START_FORM function module

Parameter	Description
FORM	Specifies the name of the form a user wants to print. If no form is specified, the SAP system restarts the printing operation of the last active form. The default value of this parameter is SPACE
LANGUAGE	Specifies the language of a form. If the form does not exist in the specified language, the SAP system calls the form in its original language. If the form is still not found, the SAP system uses the language of the last activated form. The default value of this parameter is SY-LANGU.
STARTPAGE	Specifies the starting page of a form. If this parameter is not defined, the SAP system uses the start page defined in the page layout of the form. The default value of this parameter is SPACE.
PROGRAM	Specifies a program name that is replaced by the original program in the final output. The default value of this parameter is SPACE.
ARCHIVE_INDEX	Specifies the index information of the print output to be archived. The default value of this parameter is SPACE.

The LANGUAGE import parameter is used in the START_FORM function module to specify the language variant of the form being used by the SAP system.

Calling the START_FORM function module can raise exceptions in case the parameters specified in Table 12.8 contain invalid values:

Table 12.8: Exception parameters in the START_FORM function module

Parameter	Description
FORM	Handles the exception raised when the form of the specified name is not found in an SAP system. The possible reasons may be that either the form does not exist or there is no active version of the form.
FORMAT	Handles the exception raised when the page format of one form differs from the other forms; while both the forms have the same print format.
UNENDED	Handles the exception raised when the last form is still open. The user must end the form by using the END_FORM function module or close the form by using the CLOSE_FORM function module.
UNOPENED	Handles the exception raised when the specified form is not opened by using the OPEN_FORM function module.
UNUSED	Handles the exception raised when either the FORM or LANGUAGE parameter is left blank, or when a form is opened by using the default name and language.

The *WRITE_FORM* Function Module

The WRITE_FORM function module is used to write the output of an SAPscript form in the specified element and to specify a window of the output. The following syntax is used to call the WRITE_FORM function module:

```
CALL FUNCTION 'WRITE_FORM'
EXPORTING ELEMENT = SPACE
WINDOW = 'MAIN'
FUNCTION = 'SET'
TYPE = 'BODY'
IMPORTING PENDING_LINES =

EXCEPTIONS ELEMENT =
FUNCTION =
TYPE =
UNOPENED =
UNSTARTED =
WINDOW =
```

The WRITE_FORM function module is called by using the export parameters listed in Table 12.9:

Table 12.9: Export parameters in the WRITE_FORM function module

Parameter	Description
ELEMENT	Specifies the name of the text element that needs to be printed in a form. The SAP system uses the default text element if you do not specify any text element. The default value of this parameter is SPACE.
WINDOW	Specifies the name of the window in which the user wants to provide the output of a text element of an SAPscript form. The default value of this parameter is MAIN.
FUNCTION	Specifies the output of a text element in the respective windows contained in a page of a form. The default value of this parameter is SET.
TYPE	Specifies the area of the MAIN window in which the user wants to provide the output of a text element. The possible values for this parameter are TOP for the header area and BOTTOM for the footer area. The default value of this parameter is BODY for the main area of the window.

The PENDING_LINES import parameter of the WRITE_FORM function module contains the value X when the text lines of a form are pending for print output, which is handled by the print program.

Calling the WRITE_FORM function module can raise exceptions when the parameters specified in Table 12.10 contain invalid values:

Table 12.10: Exception parameters in the WRITE_FORM function module

Parameter	Description
ELEMENT	Handles the exception raised when an SAP system does not find the name of a form element. The following are possible reasons for this: ▪ The element does not exist for a specified window in the defined form. ▪ The element is specified for a window, but not defined in the form. ▪ The defined form, having the text element in the specified window, is not active.
FUNCTION	Handles the exception raised when the function specified in this parameter is unknown. The possible values for the FUNCTION export parameter are SET, APPEND, and DELETE.
TYPE	Handles the exception raised when the TYPE parameter specifies an invalid type of window. The possible values for this parameter are BODY for all windows and TOP or BOTTOM for the MAIN window.
UNOPENED	Handles the exception raised when the WRITE_FORM function module executes a form that has not been opened.
UNSTARTED	Handles the exception raised when no form is open. The following are possible reasons for this exception: ▪ The OPEN_FORM function module starts processing a form without specifying the form name in the function module. ▪ The END_FORM function module ends the processing of a form, which had been started by using the START_FORM function module. ▪ The current form has no subsequent page after populating the last page. In this case, the SAP system automatically terminates the printing of the form after this page and raises the UNSTARTED exception.

	▪ The current form does not contain any page having the MAIN window, but a text element is specified to be displayed in the MAIN window.
WINDOW	Handles the exception raised when the form window specified in the WINDOW export parameter does not exist in the current form.

The *WRITE_FORM_LINES* Function Module

The WRITE_FORM_LINES function module is used to insert records or lines stored in a table to a specified window of a form. The following syntax is used to call the WRITE_FORM_LINES function module:

```
CALL FUNCTION 'WRITE_FORM_LINES'
EXPORTING HEADER = ?...
WINDOW = 'MAIN'
FUNCTION = 'SET'
TYPE = 'BODY'
IMPORTING PENDING_LINES =
FROMPAGE =
TABLES LINES = ?...
EXCEPTIONS FUNCTION =
TYPE =
UNOPENED =
UNSTARTED =
WINDOW =
```

The WRITE_FORM_LINES function module is called by using the export parameters listed in Table 12.11:

Table 12.11: Export parameters in the WRITE_FORM_LINES function module

Parameter	Description
HEADER	Specifies the header field of a form to be printed.
WINDOW	Specifies the name of the window where the user wants to provide the output of the form element, specified in the ELEMENT parameter. The default value of this parameter is MAIN.
FUNCTION	Specifies the output of the text element in the respective windows of the form. The default value of this parameter is SET.
TYPE	Specifies the area of the MAIN window where you want to display the output of a text element. The possible values of this parameter are TOP for the header area and BOTTOM for the footer area. The default value of this parameter is BODY for the main area of the window.

The WRITE_FORM_LINES function module is called by using the import parameters listed in Table 12.12:

Table 12.12: Import parameters in the WRITE_FORM_LINES function module

Parameter	Description
PENDING_LINES	Specifies the value X when the text lines of a form that is handled by the print program are pending.
FROMPAGE	Specifies the starting page of a form where printing has to begin.

The WRITE_FORM_LINES function module is called by using the LINES table parameter if the name of the table containing the text lines to be printed is specified.

Calling the WRITE_FORM_LINES function module can raise exceptions when the parameters specified in Table 12.13 contain invalid values:

Table 12.13: Exception parameters in the WRITE_FORM_LINES function module

Parameter	Description
FUNCTION	Handles the exception raised when the function specified in the FUNCTION export parameter is unknown. The possible values of this parameter are SET, APPEND, and DELETE.
TYPE	Handles the exception raised when the type of window area specified in the TYPE export parameter is invalid. The possible values of this parameter are BODY for all windows and TOP or BOTTOM for the MAIN window.
UNOPENED	Handles the exception raised when the WRITE_FORM_LINES function module cannot be executed because the form has not yet been opened by the OPEN_FORM function module.
UNSTARTED	Handles the exception raised when no open form is found. The possible reasons for this exception are as follows:

	▪ The OPEN_FORM function module starts a form without specifying the name of the form in the START_FORM function module. ▪ The END_FORM function module ends the form whose processing had not been started by using the START_FORM function module. ▪ The current form has no subsequent page after populating the last page. In this case, the SAP system automatically terminates the printing of the form after this page and raises the UNSTARTED exception. The current form does not contain any page having the MAIN window, but a text element is specified to be displayed in the MAIN window.
WINDOW	Handles the exception raised when the window of the form specified in the WINDOW parameter does not exist in the current form. This may be because a wrong window name is specified or the form containing this window is not active.

The *END_FORM* Function Module

The END_FORM function module ends or terminates the currently opened form and performs the required processing to terminate it. The task of the END_FORM function module is different from that of the CLOSE_FORM function module; the CLOSE_FORM function module is used to close an SAPscript form that you want to print. The following syntax is used to call the END_FORM function module:

```
CALL FUNCTION 'END_FORM'
IMPORTING RESULT =
EXCEPTIONS UNOPENED =
```

In this syntax, the RESULT parameter contains the results of the formatting of a form before the form is printed. By comparing the corresponding fields of the OPTIONS parameter with those of the RESULT parameter, you can determine if changes have been made in the settings of the print control screen. The SAP system throws the UNOPENED runtime exception when it is not able to execute the END_FORM function module, because the form has not yet been opened by the OPEN_FORM function module.

The *CONTROL_FORM* Function Module

The CONTROL_FORM function module is used to pass SAPscript control statements to a form. The following syntax is used to call the CONTROL_FORM function module:

```
CALL FUNCTION 'CONTROL_FORM'
EXPORTING COMMAND = ?...
EXCEPTIONS UNOPENED =
UNSTARTED =
```

In this syntax, the COMMAND export parameter is used to enter the SAPscript statement that you want to execute in the Interchange Text Format (ITF), but without the '/:' statement paragraph attribute.

The SAP system throws the UNOPENED runtime exception when it is not able to execute the CONTROL_FORM function module because the form has not yet been opened by the OPEN_FORM function module.

The UNSTARTED exception is thrown when an SAP system cannot find a form to start. The possible reasons for this exception are as follows:

- The OPEN_FORM function module starts processing a form without specifying the name of the form in the START_FORM function module.
- The END_FORM function module ends the processing of a form whose processing was not started by using the START_FORM function module.
- The current form has no subsequent page after filling the last page. In this case, the system automatically terminates the printing of the form after this page and raises the UNSTARTED exception.
- The current form does not contain any page having the MAIN window, but a text element is specified to be displayed in the MAIN window.

The *READ_FORM_ELEMENTS* Function Module

The READ_FORM_ELEMENTS function module populates a table with all the text elements of a form. The following syntax is used to call the READ_FORM_ ELEMENTS function module:

```
CALL FUNCTION 'READ_FORM_ELEMENTS'
EXPORTING FORM = SPACE
LANGUAGE = SPACE
TABLES ELEMENTS = ?...
EXCEPTIONS FORM =
UNOPENED =
```

The READ_FORM_ELEMENTS function module is called by using the export parameters listed in Table 12.14:

Table 12.14: Export parameters in the READ_FORM_ELEMENTS function module

Parameter	Description
FORM	Specifies the name of the form whose list of elements need to be created. If you leave this parameter blank, the SAP system uses the current active form. The default value of this parameter is SPACE.
LANGUAGE	Specifies the desired language of a form. The default value of this parameter is SPACE.

ELEMENTS, a table parameter of the READ_FORM_ELEMENTS function module, is used to specify the name of a table that stores the name of all windows and their respective text elements. Note that the SAP system assigns a specific number to each text element, which is the number of text lines contained by the text element. Calling the READ_FORM_ELEMENTS function module can raise exceptions when the parameters listed in Table 12.15 contain invalid values:

Table 12.15: Exception parameters in the READ_FORM_ELEMENTS function module

Parameter	Description
FORM	Handles the exception raised when a form of the specified name is not found in an SAP system. The reason for this may be that either the form does not exist or there is no active version of the form.
UNOPENED	Handles the exception raised when the specified form cannot be opened by using the OPEN_FORM function module.

The *READ_FORM_LINES* Function Module

The READ_FORM_LINES function module is used to transfer the lines of a text element of a form to an internal table. If the name of a form is not specified, the SAP system transfers the text lines of a currently opened form. However, if the name of a form is specified, the SAP system transfers the lines of the text elements of the active form to an internal table. The following syntax is used to call the READ_FORM_LINES function module:

```
CALL FUNCTION 'READ_FORM_LINES'
EXPORTING FORM = SPACE
LANGUAGE = SPACE
WINDOW = 'MAIN'
ELEMENT = SPACE
TABLES LINES = ?...
EXCEPTIONS ELEMENT =
FORM =
UNOPENED =
```

The READ_FORM_LINES function module is called by using the export parameters listed in Table 12.16:

Table 12.16: Export parameters in the READ_FORM_LINES function module

Parameter	Description
FORM	Specifies the name of the form whose element list is to be created. If this field is left blank, the SAP system uses the currently active form. The default value of this parameter is SPACE.
LANGUAGE	Specifies the desired language of the form. The default value of this parameter is SPACE.
WINDOW	Specifies the name of the window where the text element of a form has to be read. The default value of this parameter is MAIN.
ELEMENT	Specifies the name of the element of a form whose text lines need to be read. The default value of this parameter is SPACE.

LINES, a table parameter of the READ_FORM_LINES function module, is used to specify the name of a table that stores the text lines of the specified element of a form. Calling the READ_FORM_LINES function module can raise exceptions when the parameters specified in Table 12.17 contain invalid values:

Table 12.17: Exception parameters in the READ_FORM_LINES function module

Parameter	Description
ELEMENT	Handles the exception raised when an SAP system does not find the name of a element of a form. The possible reasons for this are as follows: ▪ The element does not exist for a specified window in the defined form. ▪ The element is specified for a specific window but not defined in the form. ▪ The defined form, having the text element in the specified window, is not active.
FORM	Handles the exception raised when a form of the specified name is not found in the SAP system. The possible reasons may be that either the form does not exist or there is no active version of the form.
UNOPENED	Handles the exception raised when the `READ_FORM_LINES` function module is not executed, as the form has not yet been opened by the `OPEN_FORM` function module.

Controlling the SAPscript Forms

To retrieve the output of an SAPscript form through a print program, you must always open a form using the `OPEN_FORM` function module and close it using the `CLOSE_FORM` function module. The `OPEN_FORM` function module initializes the SAPscript composer and opens the specified form for print output. The SAP system continuously collects data for the output of the form until the form is closed by using the `CLOSE_FORM` function module. If the `CLOSE_FORM` function module is not specified in the print program of the form, nothing is printed. The `WRITE_FORM`, `WRITE_FORM_LINES`, and `CONTROL_FORM` function modules are used to write data in a form. You can use these function modules any number of times and in any order after opening and before closing a form.

Note The SAPscript composer is a program that converts the format of a form's text into a printable format.

The following are some additional functions that can be performed with an SAPscript form or with the controls of the form:

- **Processing multiple print requests—** Opens and closes several forms to print. However, you cannot print multiple forms simultaneously.
- **Starting a form again—** Starts a form for printing repeatedly by using the `START_FORM` and `END_FORM` function modules.
- **Switching forms—** Switches from one form to another in a print request, but the forms must have the same page format.
- **Finding forms—** Searches for another version of the form, if the specified version is not found.

Let's explore these tasks in detail.

Multiple Print Requests

You can open and close various forms within one transaction by using the `OPEN_FORM` and `CLOSE_FORM` function modules. However, these forms cannot be opened simultaneously.

Starting a Form Again

A print program allows you to print multiple copies of one letter or form for different customers. To print several copies of a form for different customers, you must start the form every time it needs to be printed.

To start a form, you must end the form (if it is already started) and then start it again. Within one print request, you need to call the `END_FORM` function module to execute the final processing for the form. Next, start the form again by using the `START_FORM` function module, as shown in the following syntax:

```
CALL FUNCTION 'OPEN_FORM'
:
CALL FUNCTION 'START_FORM'
:
CALL FUNCTION 'END_FORM'
:
CALL FUNCTION 'START_FORM'
:
CALL FUNCTION 'END_FORM'
```

```
:
CALL FUNCTION 'CLOSE_FORM'
```

Note When you use the START_FORM and END_FORM function modules, you must not use the OPEN_FORM function module to open the form.

Switching Forms

A user can switch forms within one print request. This may be necessary if you need to customize the format of the form's output depending on customer preferences. For example, you may need to switch forms to generate the output in a different layout, depending on the geographical location of a customer. In this case, you need to switch from one form to another by using the START_FORM function module.

Note When switching forms, ensure that you use only those forms that have the same page format, such as DINA4 or LETTER page format. However, you can mix forms with different page orientations easily; that is, LANDSCAPE or PORTRAIT.

Finding Forms

The SAPscript form processor generates business forms based on SAPscript forms that act as templates for business forms. The SAPscript form processor searches for a predefined SAPscript form in the available clients. If the SAPscript form processor does not find the specified SAPscript form, it automatically searches for another version of the same form. Figure 12.34 shows how the SAPscript form processor searches for an SAPscript form in an SAP system:

In Figure 12.34, note that the SAPscript form processor executes a cross-client search only if the current client does not contain the form. If the SAPscript form processor finds the form in the current client, it generates and stores it in the client that has started the search. If the request for the form is started again, the SAPscript form processor finds it in the current client.

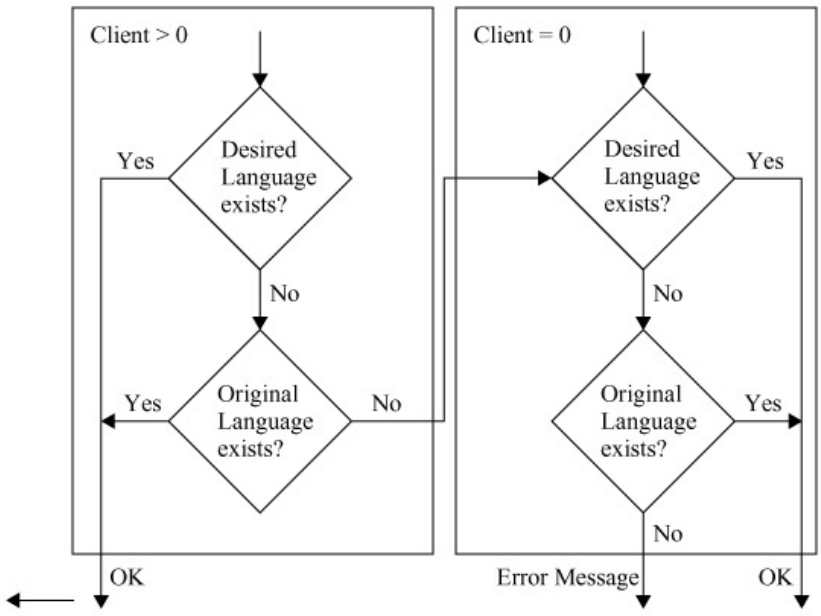


Figure 12.34: Searching forms in different versions

SAPscript Control Commands

SAP control commands allow you to format the final output. These commands are passed to the SAPscript composer to process a form. Some common tasks performed by the SAPscript composer are line and page formatting, replacing symbols with their current values, and formatting the text of a form according to the specified paragraph and character formats. Table 12.18 shows a list of SAPscript control commands:

Table 12.18: List of SAPscript control commands

Control Command	Description

ADDRESS	Formats the text specified for the `Address` window in a form.
BOTTOM … ENDBOTTOM	Defines the footer text in a window.
BOX, POSITION, SIZE	Specifies a box or line in a form that needs to be formatted.
CASE, ENDCASE	Specifies multiple conditions based on text printed.
DEFINE	Assigns values to text symbols in a form.
HEX, ENDHEX	Specifies hexadecimal values in a form.
IF, ENDIF	Specifies conditional text in a form.
INCLUDE	Includes other text in a form.
NEW-PAGE	Inserts a page in a form explicitly.
NEW-WINDOW	Inserts a new main window in a form.
PRINT-CONTROL	Inserts a print control character in a form.
PROTECT, ENDPROTECT	Prevents a page break in a form.
RESET	Initializes the outline paragraph in a form.
SET COUNTRY	Sets country-specific formatting in a form.
SET DATE MASK	Specifies the format of the date fields in a form.
SET SIGN	Inserts the + or - sign in a form.
SET TIME MASK	Specifies the format of the time fields in a form.
STYLE	Changes the style of the text in a form.
SUMMING	Inserts the sum of the variables in a form.
TOP	Sets the header text in the main window.

SAPscript Symbols

SAPscript symbols are used to include a program, system data, or predefined text in a form. The name of the symbol is specified by using the ampersand (&) symbol. For example, the following syntax is used for the DATE symbol: `&DATE&.`

You can insert a symbol anywhere in the text of any window or page of a form. When the form is printed or displayed, SAPscript substitutes the current value of the symbol with the name of the symbol in the form.

The following four categories of SAPscript symbols are used in the SAP system:

- **System symbols**—Receive their values from SAPscript. The names and text of the system symbols are fixed. Table 12.19 describes a list of system variables:

Table 12.19: List of system symbols used in a form

System Symbol	Description
DATE	Displays the current date.
DAY	Displays the current day of the week.
DEVICE	Specifies an output device, such as PRINTER, SCREEN, or FAX.
HOURS	Displays the hours in the context of the current time.
MINUTES	Display the minutes in the context of the current time.
MONTH	Displays the current month.
NAME_OF_MONTH	Displays the name of the month in the context of the current date.
NEXTPAGE	Displays the next page number of a form.
PAGE	Displays the current page number of a form.
SECONDS	Displays the seconds in context with the current time.
SPACE	Inserts a blank space in a form.
TIME	Displays the current time of day.

ULINE	Underlines text in a form.
VLINE	Displays a vertical line in a form.
YEAR	Displays the current year.

You can access system symbols by selecting `Include > Symbols > System from PC Editor for a window`. They are of the following types:

- **Program symbols—** Receive the data either from the fields of a database table or from the globally stored data in an SAP system. Note that when you print an SAPscript form, the data is retrieved directly from the fields of the database table, instead of the symbols. For instance, `&MARA-MAKTL&`, is a program symbol, in which `MARA` is the name of a standard database table and `MAKTL` is one of the fields of this table. To access a program symbol, select `Include > Symbols > Program from PC Editor for a window`.
- **Standard symbols—** Receive the data from the TTDTG table. TTDTG is a predefined table in the SAP system to store SAPscript standard symbols for word processing. You can access the TTDTG table using the `SM30` or `SE11` transaction code. Standard symbols are static text, which are assigned by a user. The `&WR&` expression can represent an example of a standard symbol to denote the With Regards text. Standard symbols are language-dependent, which means that a symbol is first translated and then inserted in the text in the defined language of a form. Standard symbols are accessed by selecting `Include > Symbols > Standard from PC Editor for a window`.
- **Text symbols—** Store user-defined text for a form. These symbols are defined locally in a text module. Text symbols can be used only in the form for which they are defined. The `DEFINE` command is used to define a text symbol by using the following syntax:

```
/: DEFINE &symbolname&='Symbol text'
```

The following examples show the use of text symbols:

```
/: DEFINE &MYSYMBOL& = '100'.
```

and

```
/: DEFINE &OBJECT&='my country'.
```

After the text is assigned to a text symbol, the text symbol can be included in the text by typing its name or by selecting `Include > Symbols > Text from PC Editor for a window`.

Note When you use any symbol in your SAPscript form, the SAP system automatically recognizes the type of the symbol. The SAP system first checks whether the symbol is a system symbol. If the symbol is not a system symbol, the SAP system checks whether the symbol is defined in the calling program. If it is so, the symbol is specified as a program symbol. Otherwise, the SAP system checks the symbol from the entries stored in the TTDTG table, and in this case the symbol is specified as a standard symbol. Finally, if a symbol neither belongs to the category of system symbols nor to the categories of program and standard symbols, it is considered a text symbol.

The SAP Smart Forms Tool

The SAP Smart Forms tool was introduced with SAP R/3 4.6 to print and send documents through e-mail, the Internet, or facsimile. This tool helps develop forms, PDF files, e-mails, and documents for the Internet. The SAP Smart Forms tool provides an interface to build and maintain the layout and logic of a form. In addition to the SAP Smart Forms tool, SAP delivers a selection of forms for business processes, such as those used in Customer Relationship Management (CRM), Sales and Distribution (SD), Financial Accounting (FI), and Human Resources (HR).

In the upcoming sections, we describe the following:

- Overview of the SAP Smart Forms tool
- Smart Form components
- Explaining the Smart Forms process
- Advantages of Smart Forms
- Important objects for form development

- Creating and maintaining Smart Forms
- Style builder

Overview of the SAP Smart Forms Tool

In the earlier versions of SAP, SAPscript was the only option available to create printable documents or forms. However, implementing and maintaining SAPscript forms can be difficult because these forms are not fully integrated with the other applications in an SAP system since they make use of batch printing and spool control. Therefore, to simplify and facilitate the printing of these forms, SAP introduced a new tool called SAP Smart Forms. Introduced with SAP R/3 4.6, the SAP Smart Forms tool is fully integrated with the core applications of an SAP system, such as HR and mySAP ERP. The SAP Smart Forms tool also supports Web applications, which is a key area of expansion for most companies.

The SAP Smart Forms tool allows you to modify forms by using simple graphical tools instead of using any programming tool. This means that a user with no programming knowledge can configure these forms with data for a business process easily.

In a Smart Form, data is retrieved from static and dynamic tables, the table heading and subtotal are specified by the triggered events, and the data is then sorted before the final output. A Smart Form allows you to incorporate graphics, which can be displayed either as part of the form or as the background. You can also suppress a background graphic, if required, while taking a printout of a form.

Some examples of Smart Forms available in the mySAP ERP system are as follows:

- **SF_EXAMPLE_01—** Represents an invoice with a table output for flight booking for a customer.
- **SF_EXAMPLE_02—** Specifies an invoice similar to SF_EXAMPLE_01, but with subtotals.
- **SF_EXAMPLE_03—** Represents an invoice similar to SF_EXAMPLE_02, but one in which several customers can be selected in an application program. A form is created for each customer and then all the forms are included in a single request for the output.

SAP Smart Form Components

The basic structure of an SAP Smart Form comprises four basic components: Smart Form Builder, the Smart Form print form template, the Smart Form function module, and the Smart Form print program.

Now, let's discuss these components in detail.

Smart Form Builder

Smart Form Builder is the main interface used to build a Smart Form. It is available on the initial screen of SAP Smart Forms. You can type the `SMARTFORMS` transaction code in the `Command` field to open the initial screen of SAP Smart Forms, as shown in Figure 12.35:

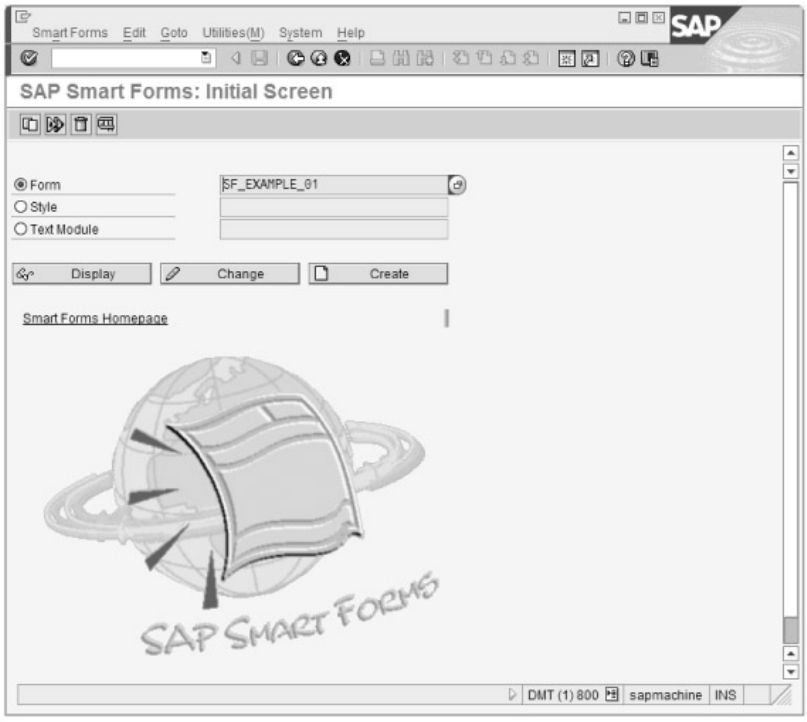


Figure 12.35: The initial screen of SAP smart forms

Note An alternative way to open the initial screen of SAP Smart Forms is by selecting SAP `menu > Tools > Forms > Smart Forms` in the SAP `Easy Access` screen.

Next, you need to enter the name of the form that you want to display, copy, or create in the initial screen of SAP Smart Forms. In this case, we click the `Display` button after entering a predefined Smart Form name in the `Form` field (see Figure 12.35). The SF_EXAMPLE_01 form appears in the `Form Builder` screen, as shown in Figure 12.36:

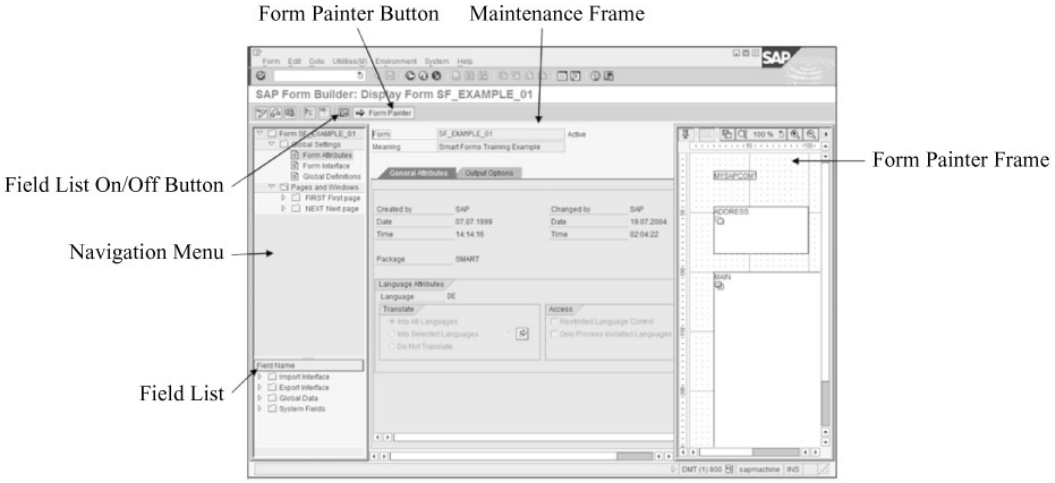


Figure 12.36: The form builder screen

Figure 12.36 shows three frames in the `Form Builder` screen. The names and positions of these frames are as follows:

- **Navigation menu—** Displays the form as a hierarchy in which each element of the form is represented by a node. The `Navigation` menu appears on the left of the screen and provides you direct access to the elements of the opened form.
- **Maintenance frame—** Maintains the attributes of the currently selected node in the navigation tree. This frame

appears in the middle of the screen.

- **Field List—** Displays the data currently defined in the form. If Field List is not displayed on the `Form Builder` screen, click the `Field List On/Off`  icon (see Figure 12.36).
- **Form Painter frame—** Acts as a graphical tool to design the layout of a form by using the output areas, such as the size and location of the windows. This frame appears at the right side of the screen. If the Form Painter frame is not visible on the screen, click the `Form Painter` button (see Figure 12.36).

Each frame can be resized by clicking and dragging its borders. When you select a node in the `Navigation` menu, the view in the other frames also changes. For example, when you double-click a node in the `Navigation` menu, the corresponding information related to the node is displayed in the maintenance frame. In addition, the selection of the output area in the Form Painter frame can be changed by using the drag-and-drop feature, according to the node selected in the `Navigation` menu.

In the `Navigation` menu, the form is represented as branches. The top hierarchy level of the `Navigation` menu contains the following nodes and subnodes:

- **Global Settings—** Contains the `Form Attributes`, `Form Interface`, and `Global Definitions` subnodes.
- **Pages and Windows—** Contains the `FIRST First page` and `NEXT Next page` subnodes.

You can add more subnodes under the `Pages and Windows` node, while creating a form. However, no subnode can be added to the `Global Settings` node.

The Smart Form Print Form Template

The Smart Form print form template provides a preconfigured design and layout to create a Smart Form in a printable format. It can be created by using Smart Form Builder. A Smart Form print form template contains the layout of the form, the fields required in the form, the conditions that specify how to populate the form, and some special programming instructions for printing the form. The form created by a Smart Form print form template generates the final output after using the Smart Form function modules.

The Smart Form Function Module

The Smart Form function module is a sequence of statements that are generated automatically when the Smart Form print form is activated. If you make any change in the print form of Smart Form Builder, you need to save the change and activate the form again in Smart Form Builder to get the desired output.

The Smart Form Print Program

The Smart Form print program controls the printing of a Smart Form. Generally, one Smart Form print program is associated with one type of form. For example, the purchase order Smart Form has a corresponding purchase order print program. Note that customer-specific customization performed in the Smart Form print form is more efficient than that performed in the Smart Form print program.

Note The print program of an SAPscript form is different from the print program of a Smart Form and cannot be used with a Smart Form.

Explaining the Smart Form Process

As discussed earlier, the structure of an SAP Smart Form consists of Smart Form Builder, the Smart Form print form template, the Smart Form function module, and the Smart Form print program. These components work together in the program or workflow of a Smart Form. The program flow of a Smart Form process can be explained in the following way:

- A user creates a Smart Form for an application and includes application data into the form by using a form template in the Smart Form Builder.
- After designing the form, it needs to be activated before it can be tested or made accessible to print programs.

Activating the form generates a function module that handles the processing of the form. This function module interacts with an application and creates the output according to the specified output device.

Figure 12.37 shows the program flow of a Smart Form process:

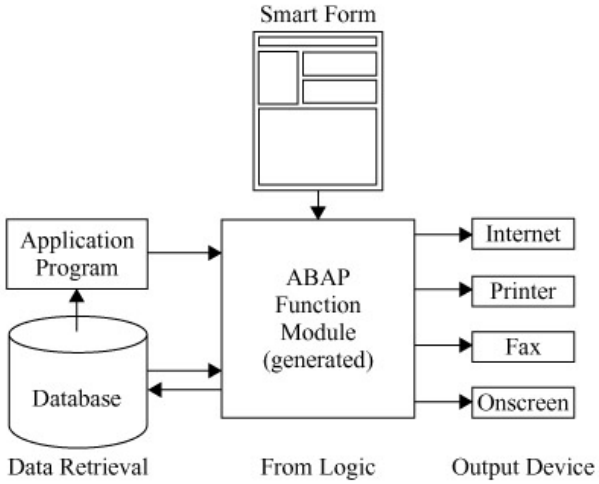


Figure 12.37: The program flow of a smart form process

In Figure 12.37, you can see that to take a printout of a Smart Form, a user requires an application program for data retrieval and a Smart Form that consists of the entire logic of the form. The application program passes the data through a function module interface to a Smart Form. When activating a Smart Form, the SAP system automatically generates a function module, which is then processed by the SAP system at runtime. The function module generated by the SAP system encapsulates all the attributes of the Smart Form. As soon as the application program calls the function module, the Smart Form uses the module interface (which corresponds to the interface of the form) to transfer previously selected table data and print the form as described in the form description. This process is also known as calling a Smart Form for printing.

Note The description of a Smart Form includes the following:

- The layout of the form
- Individual elements, such as text, graphics, addresses, and tables
- The logic of the form, for example, to read application data from internal tables and to control the process flow
- A form interface to transfer application data into the form definition

Advantages of Smart Forms

A Smart Form has the following advantages:

- Enables you to create interactive Web forms with input fields, buttons, and radio buttons
- Enables you to create the layout and logic of a form by using graphic tools, an operation for which programming knowledge is not necessary
- Displays table structures
- Displays a colored output of the text
- Enables you to use the user-friendly and integrated Form Painter to design forms graphically
- Enables you to draw Smart Form tables by using a special control; that is, Graphical Table Painter
- Enables you to display the background graphics using templates
- Enables you to use the format of fonts and paragraphs defined in a Smart style
- Enables you to display the output in the HTML format

Important Objects for Form Development

To create a business form, the SAP Smart Forms tool uses some important objects, such as a Smart style, text module, and Smart Form template. A Smart style is a collection of paragraph and character formats. Each Smart Form uses at least one Smart style. A text module is an encapsulated text that allows a text element to be used in one or more Smart Forms. A Smart Form template is a collection of form processing logic, content, and layout, and also refers to Smart styles and text modules.

Besides the Smart style and text module objects, a Smart Form can also include a graphic object, such as company's logo or preprinted forms, scanned as background images.

Another important object is the Table Painter, which is used to create dynamic expandable tables in Smart Forms. A Smart Form template has a fixed number of rows, while a Smart Form table can have a dynamic number of rows. In the Maintenance frame of SAP Form Builder, a new type of tab, the `Table` tab, appears. In the `Table` tab, you can define table characteristics and turn on the `Table Painter` icon to specify the rows and columns of dynamically expandable tables. However, the `Table Painter` icon does not show the actual output, because the depth of a table row output depends dynamically on the number of records received from an application.

Creating and Maintaining Smart Forms

In this section, you learn how to create a form by using the SAP Smart Forms tool. You also learn how to add a node in the Smart Form and test the form.

We begin with creating a copy of the SF_EXAMPLE_01 form by using the SAP Smart Forms tool. The SF_EXAMPLE_01 form is a standard Smart Form available in an SAP system. Perform the following steps to create a form:

1. In the SAP menu path, select Tools > Forms > Smart Forms or use the `SMARTFORMS` transaction code in the `Command` field. The initial screen of SAP Smart Forms appears. In this screen, enter the form name, SF_EXAMPLE_01, in the `Form` field (see Figure 12.35).
2. Select `Smart Forms > Copy` or click the `Copy` icon to open the `Copy Form or Text` dialog box, as shown in Figure 12.38:
3. In the `Target Object` field, enter a name for the new form. The name must begin with the Y or Z letter. In our case, the name of the form is ZSF_EXAMPLE_01 (see Figure 12.38).
4. Click the `Continue` icon or press the ENTER key in the `Copy Form or Text` dialog box so that the ZSF_EXAMPLE01 form is created as a copy of the predefined form SF_EXAMPLE_01. The `Create Object Directory Entry` dialog box appears.
5. In the `Create Object Directory Entry` dialog box, enter the package name, ZKOG_PCKG, in the `Package` field and then click the `Save` icon. The name of the form is displayed in the `Form` field on the initial screen of SAP Smart Forms, as shown in Figure 12.39:
6. Click the `Create` button on the initial screen of SAP Smart Forms. The ZSF_EXAMPLE_01 form appears in Form Builder, as shown in Figure 12.40:

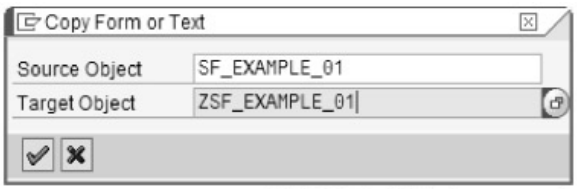


Figure 12.38: The copy form or text dialog box

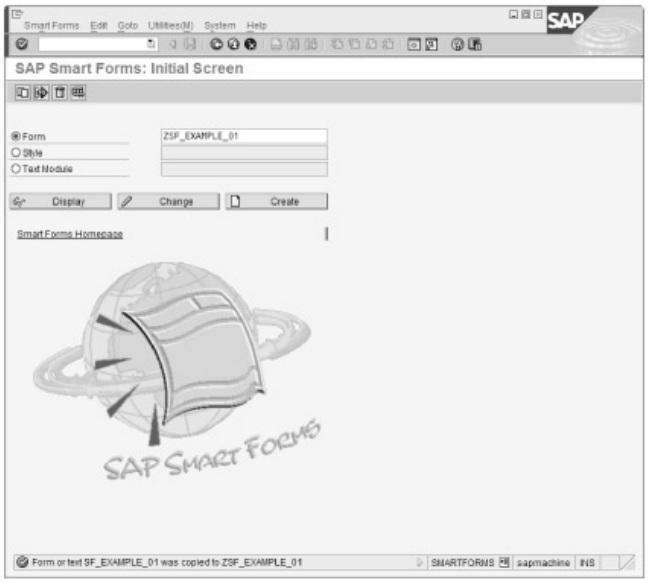


Figure 12.39: Displaying the copied form in the initial screen of SAP smart forms

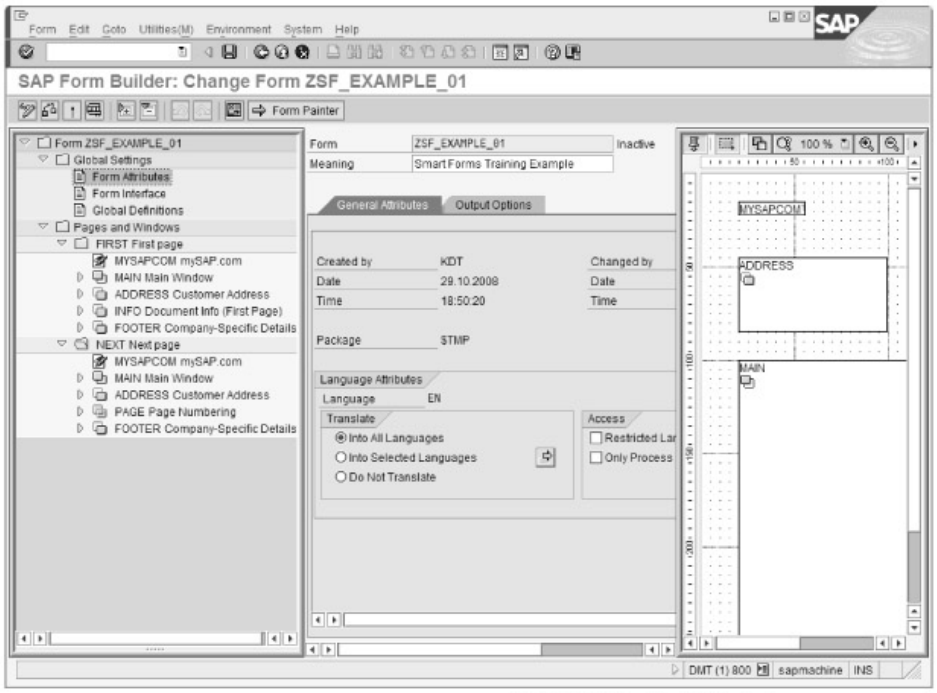


Figure 12.40: The ZSF_EXAMPLE_01 form in form builder

In Figure 12.40, the first draft page is created with a `MAIN` window. All the components of the new form are based on the SF_EXAMPLE_01 predefined form. You can click a node in the `Navigation` menu to view its content.

Nodes are created or moved by using the `Navigation` menu. The order of the nodes determines the order in which the output of the form is processed. In addition, the order of the output is determined by the order in which the nodes are arranged in the `Navigation` menu.

A node can be moved by clicking and dragging it to the desired location in the navigation tree. The attributes for each node, such as the text or the name of an image, are defined in the Form Builder's maintenance frame.

Now, performing the following steps to create a text node in the form:

1. Open a form in the change mode of the `SAP Form Builder` screen (see Figure 12.40). Next, right-click the `Main`

`Window` option in the `First Page` node and select `Create > Text` from the context menu, as shown in Figure 12.41:

2. Modify the text in the `Text` field to MYTEXT and the text in the `Meaning` field to Text Node Demo, as shown in Figure 12.42:
3. Enter the text "Hello World" in the text-editing box in the center frame of Form Builder (see Figure 12.42).
4. Click the `Save` icon to save the node.
5. Next, activate and test the node by clicking the `Activate` and `Test` icons, respectively. The initial screen of Function Builder appears, as shown in Figure 12.43:
6. Activate and test the function module by clicking the `Activate` and `Test/ Execute` icons. The parameters of the function module are displayed in the initial screen of Function Builder, as shown in Figure 12.44:
7. Execute the function module by clicking the `Execute` icon (see Figure 12.44). The `Print` dialog box appears, as shown in Figure 12.45:
8. Specify the output device as lp01 and click the `Print preview` button, as shown in Figure 12.45.

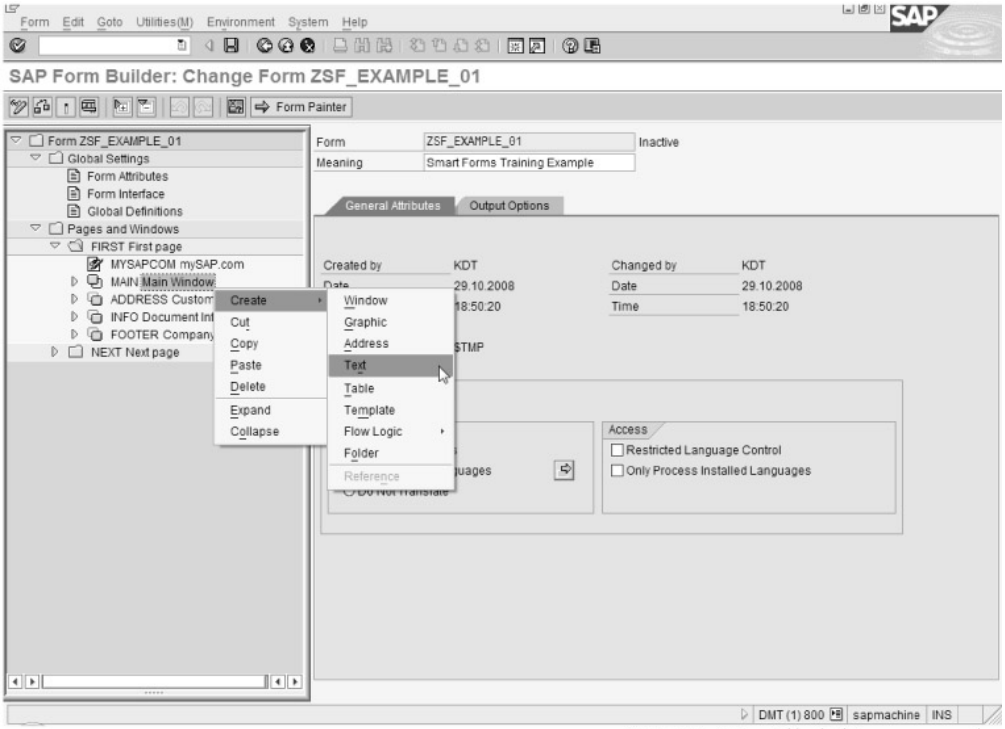


Figure 12.41: Selecting a new node in the main window

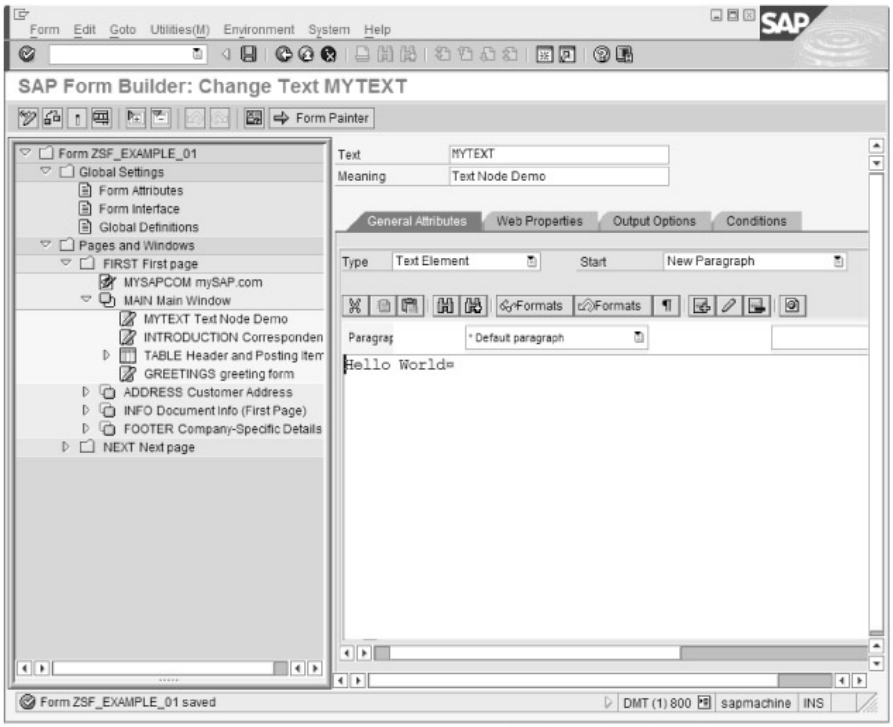


Figure 12.42: Specifying the attributes of the new text node

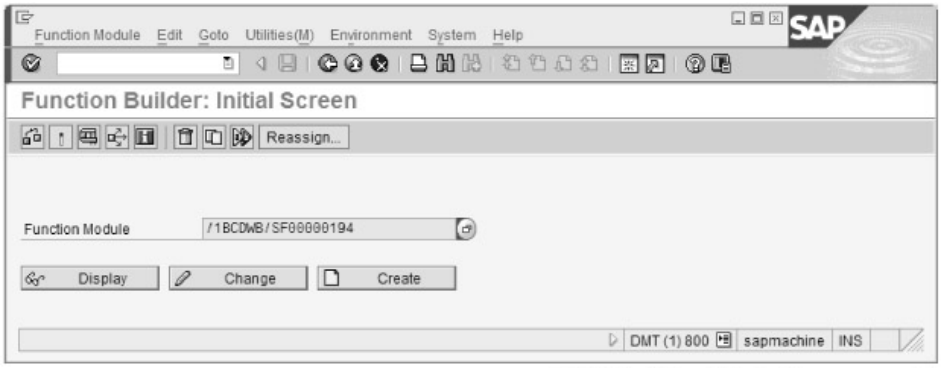


Figure 12.43: The initial screen of function builder

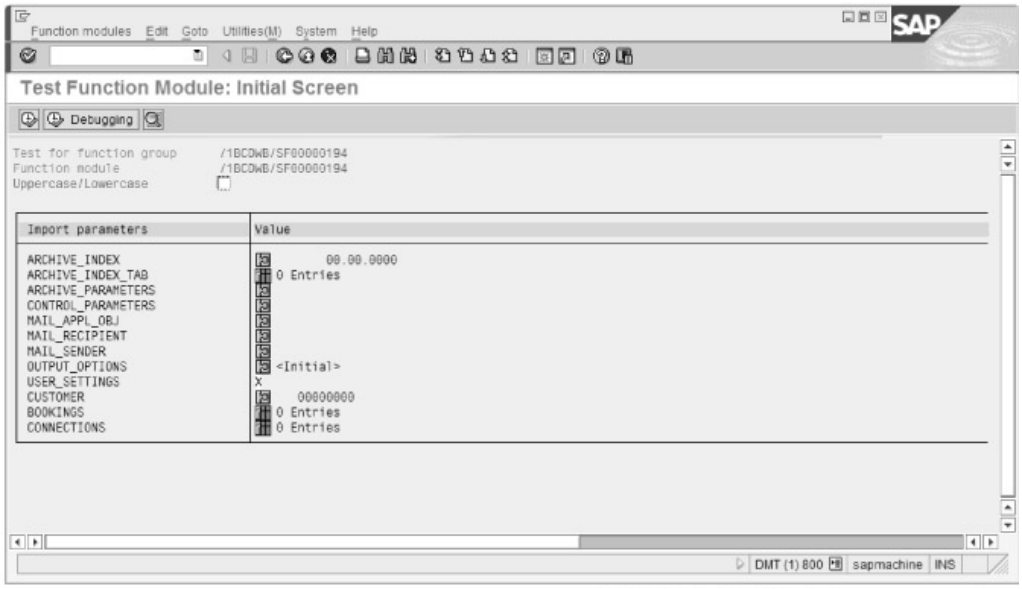


Figure 12.44: Displaying the parameters in the initial screen of test function module

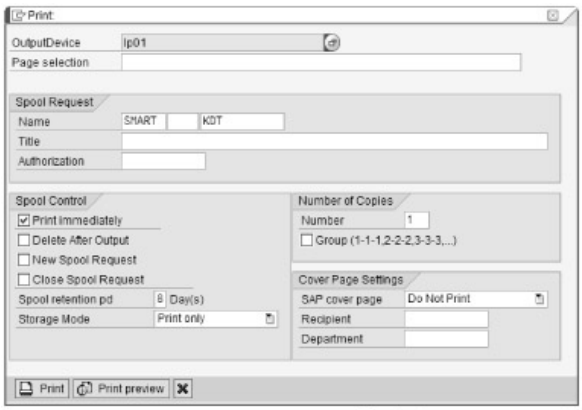


Figure 12.45: The print dialog box

The print preview of the ZSF_EXAMPLE_01 form appears, with the text Hello World, as shown in Figure 12.46:

Figure 12.46 shows the print preview of the ZSF_EXAMPLE_01 form.

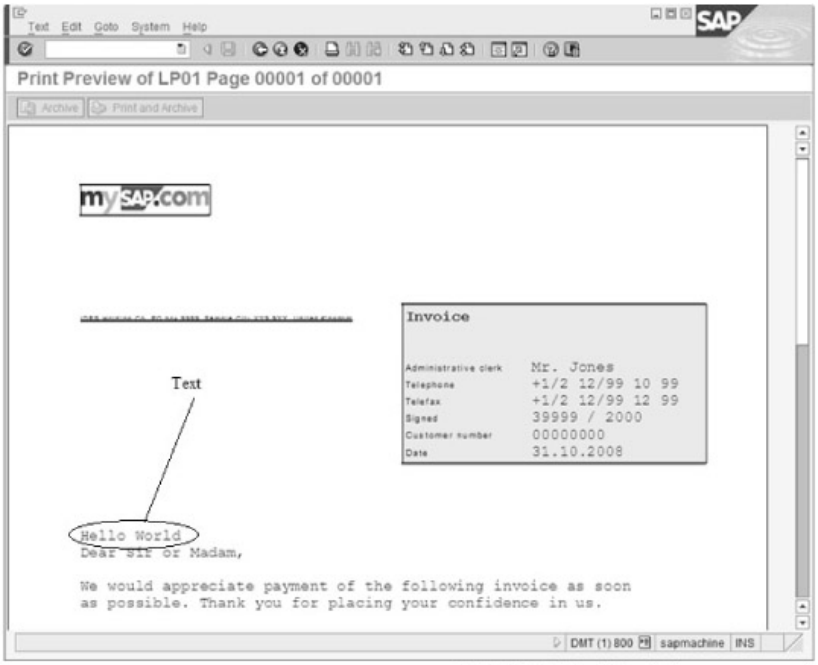


Figure 12.46: Print preview of the form

Style Builder

Style Builder is a tool used to define a Smart style for a Smart Form. It is available on the initial screen of Smart Styles, which is opened by using the SMARTSTYLES transaction code. Figure 12.47 displays the initial screen of Smart styles:

Figure 12.47: The initial screen of smart styles

In Figure 12.47, note that the initial screen of Smart Styles contains the `Style` field to specify the Smart style name that you want to create, change, or display. Now, let's enter a name for a Smart style (for example, ZSF_STYLE_1), and click the `Create` button, as shown in Figure 12.47. The ZSF_STYLE_1 Smart style appears in Style Builder (in change mode), as shown in Figure 12.48:

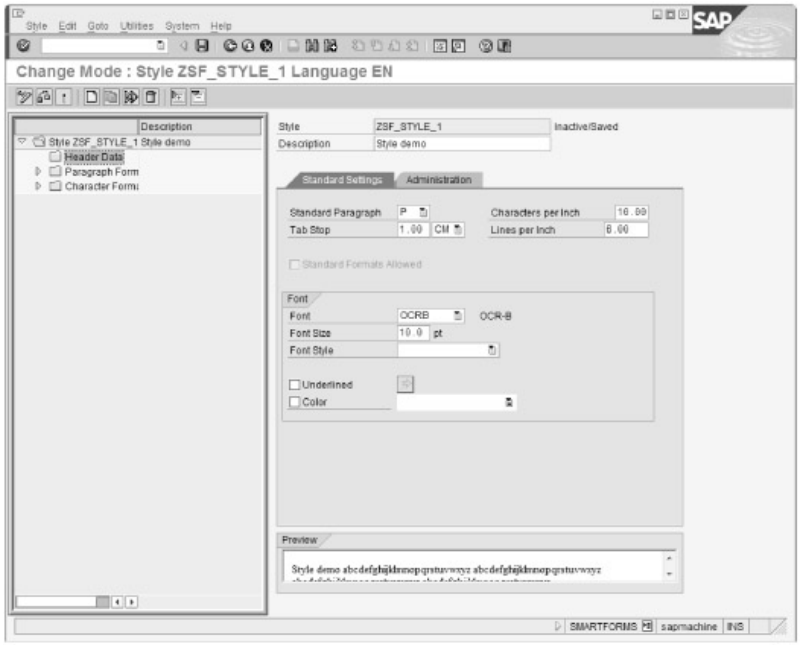


Figure 12.48: The ZSF_STYLE_1 smart style in style builder

Note The interface of Style Builder, as shown in Figure 12.48, can also be accessed by entering the name of the Smart style in the initial screen of SAP Smart Forms and clicking the `Create` button, as shown in Figure 12.49:

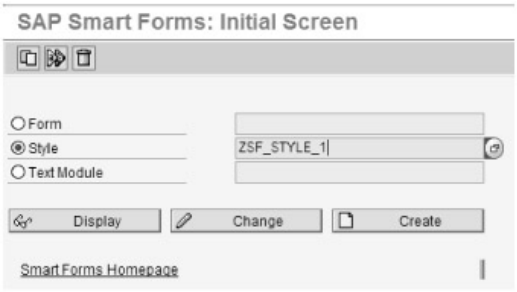


Figure 12.49: Entering a smart style name in the initial screen of SAP smart forms

Figure 12.48 shows a style tree to the left of the `Style Builder` screen, which consists of predetermined nodes (`Header Data`, `Paragraph Formats`, and `Character Form`). You can navigate between the nodes or create new nodes. On the right is a maintenance screen, where you specify various settings in a Smart style. Note that when you specify any settings in the maintenance frame, the `Preview` group box displays the preview of the specified font settings.

You must activate a Smart style in a Smart Form before using it. A Smart style is activated by clicking the `Activate` icon. During activation, the SAP system checks the Smart style for errors and, if necessary, displays an error list.

Comparing Sapscript and Smart Forms

SAPscript forms and Smart Forms have various similarities, such as their use to generate business forms; there are differences between the two as well. One major difference is that SAPscript forms are client-dependent, while Smart Forms are client-independent.

SAPscript is a word processing tool that displays the data on an SAPscript form by using various text elements and a print program that stores the logic of these text elements. SAPscript forms are, therefore, designed to be driven by their specific print programs and are often called client-dependent. Figure 12.50 shows the process of creating a form by using the SAPscript tool:

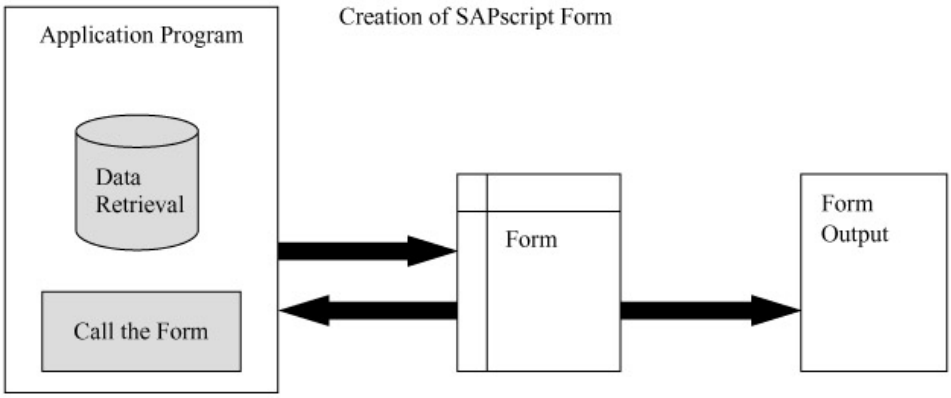


Figure 12.50: The process of creating a form by using SAPscript

A print program determines the output document, the values of the areas, and the frequency of the output, during the processing of an SAPscript form. The print program also accesses the SAP database to retrieve data that is populated in the corresponding fields of the form, which means that the SAPscript tool directs the print program to print the output.

Moreover, an SAPscript form uses a layout set that describes the layout of the individual print pages and uses text elements to supply definable output blocks, which a print program can call.

Smart Forms, on the other hand, are executed through a function module. When a print program calls a Smart Form, the form itself takes over the task of generating the output, without any intervention from the print program. Figure 12.51 shows the process of creating a form by using the SAP Smart Forms tool:

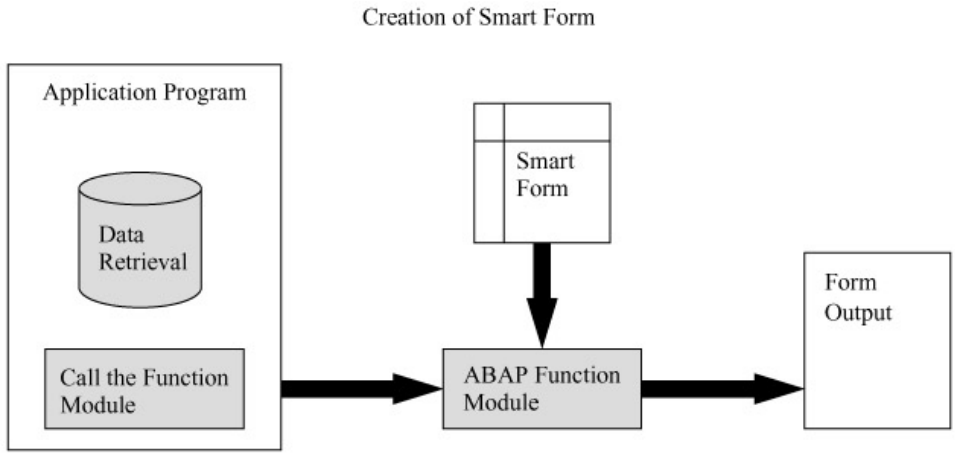


Figure 12.51: The process of creating a form by using SAP smart forms

The following tasks are performed in a Smart Form:

- Designing the form
- Activating the form and automatically generating the function modules
- Using an application program to retrieve data and call the Smart Form

In Figure 12.51, you can see that ABAP function modules use the layout of a Smart Form and an application program, which retrieves the data from different sources, to generate the output of the form.

Other differences that exist between the two types of forms are as follows:

- Smart Forms can have multiple page formats
- Smart Forms do not necessarily have a MAIN window
- Smart Forms cannot have labels

- Smart Forms can have routines
- Smart Forms generate function modules on activation

Migrating Sapscript Forms to Smart Forms

At times, you might need to create a business form that is not provided in the built-in templates provided by Smart Form. In such situations, you can migrate an SAPscript form to a Smart Form. SAP provides a migration tool to migrate the layout and text of an SAPscript form to a Smart Form. However, remember that when you migrate an SAPscript form to a Smart Form, the SAPscript form's logic of the print program is not migrated. You can migrate an SAPscript form to a Smart Form in two ways, by individual migration and by mass migration (multiple SAPscript forms migrated simultaneously).

The SAP system performs the following actions during the migration of an SAPscript form to a Smart Form:

- Converts layout information, such as information about the pages and windows, and their attributes
- Copies language attributes and output options
- Copies the text from an SAPscript form to the Smart Form
- Displays program symbols in the text
- Converts SAPscript commands (such as `NEW-PAGE` or `IF... ENDIF`) to comment lines and displays them in the text

In the upcoming sections, we describe the following:

- Individual migration
- Mass migration
- Converting a style (that is, the conversion of an SAPscript style into a Smart style)

Individual Migration

When converting an SAPscript form to a Smart Form, you have to provide the names of the SAPscript form and Smart Form involved in the migration.

Now, perform the following steps to migrate an SAPscript form to a Smart Form:

1. Open the initial screen of SAP Smart Forms either by selecting `Tools > Forms > Smart Forms` or by using the `SMARTFORMS` transaction code in the `Command` field and entering the name ZMySmartForm in the `Form` field.
2. Select `Utilities(M) > Migration > Import SAPscript form`, as shown in Figure 12.52:

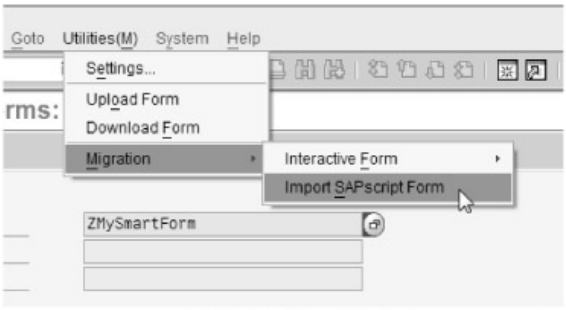


Figure 12.52: Selecting the import SAPscript form option

The `Import SAPscript Form` dialog box appears.

3. Enter the name and the language of the source SAPscript form; for example, enter S_EXAMPLE_1, as shown in Figure 12.53:
4. Click the `Continue` icon or press the ENTER key. Form Builder opens the ZMySmartForm form in the change

mode, as shown in Figure 12.54:

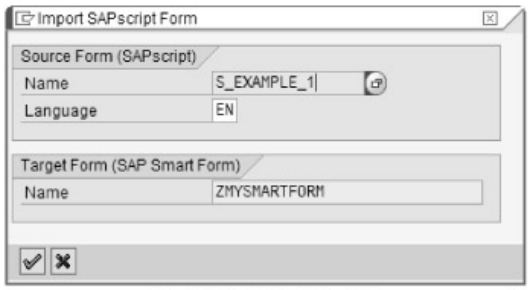


Figure 12.53: The import SAPscript form dialog box

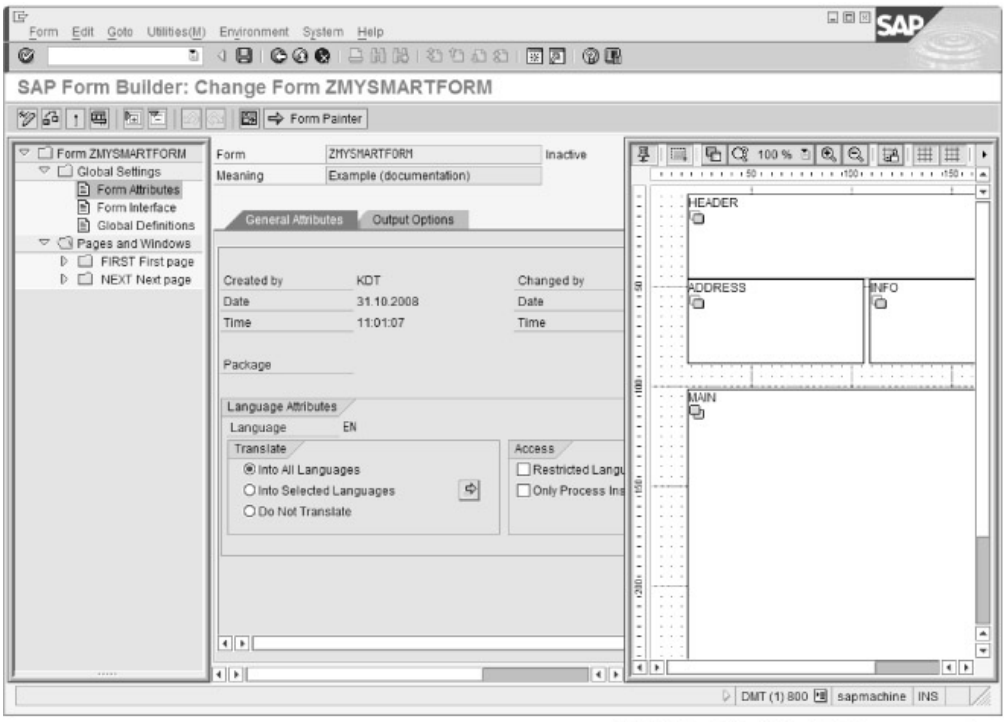


Figure 12.54: Opening the ZMySmartForm form in the change mode

Note If the SAPscript form does not exist in the selected language, a dialog box appears, where you can select one of the existing languages.

Now, you can change the design of a Smart Form, save the changes, and activate the form.

Mass Migration

To convert multiple SAPscript forms into Smart Forms, you have to provide a range of SAPscript forms.

Perform the following steps to migrate multiple SAPscript forms to Smart Forms:

1. Open the initial screen of ABAP Editor by using the SE38 transaction code. In the initial screen of ABAP Editor, enter the program name SF_MIGRATE and click the `Display` button.
2. ABAP Editor opens the SF_MIGRATE report in the display mode, as shown in Figure 12.55:
3. Activate and execute the SF_MIGRATE report by clicking the `Activate` and `Direct Processing` icons, respectively.

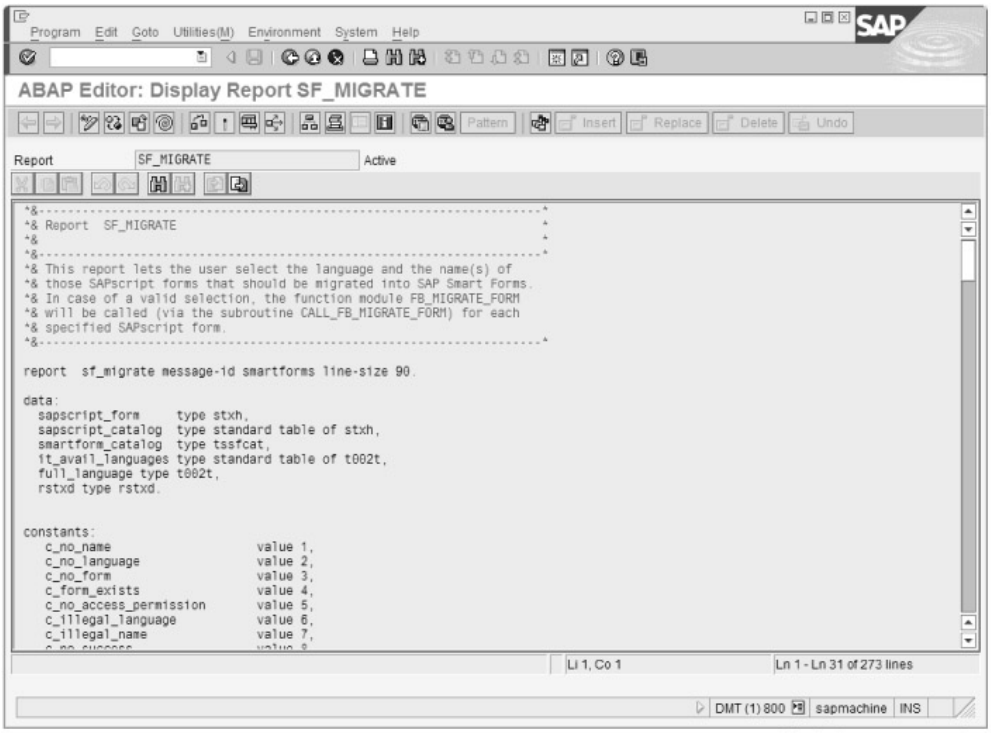


Figure 12.55: The SF_MIGRATE report in display mode

The `Report for migrating SAPscript forms` screen appears, as shown in Figure 12.56:

4. Enter the names and languages of the range of SAPscript forms that you want to migrate; for example, enter the range of names from ZMYSAPSCRIPTFORM to ZSSFORM1 and the language as EN (see Figure 12.56).

5. Specify the client information by selecting either the `from the current client` or `from all clients` radio button. In our case, we have selected the `from all clients` radio button, as shown in Figure 12.56.

6. Click the `Execute` icon to complete the process. The SAP system creates Smart Forms corresponding to the names of SAPscript forms, with the extension _SF. Figure 12.57 shows a list of forms that were migrated and also forms that could not be migrated:

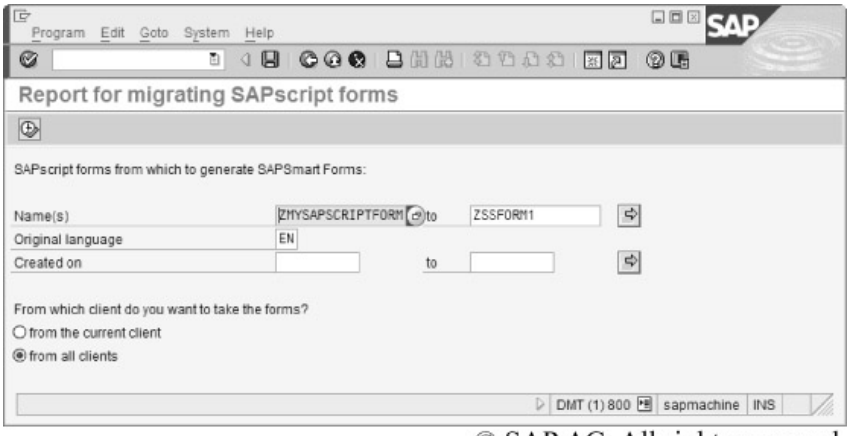


Figure 12.56: The report for migrating SAPscript forms screen

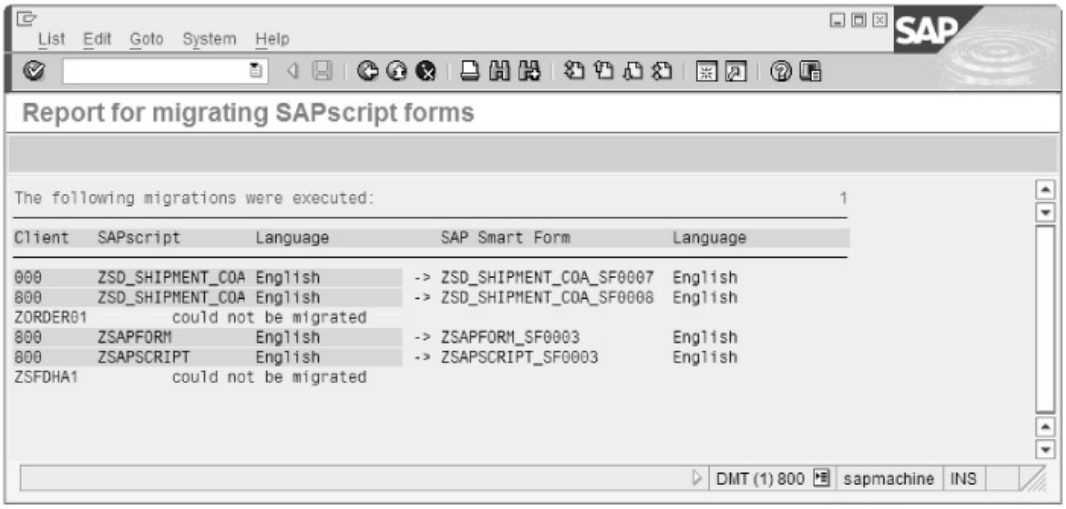


Figure 12.57: List of migrated/not migrated forms

Note Use the SMARTFORMS transaction code to modify a Smart Form and activate the Smart Form.

Now, let's learn how to convert an SAPscript style into a Smart style.

Converting a Style

You can also convert an SAPscript style into a Smart style and apply the converted Smart style to one or more Smart Forms. Now, let's learn how to convert an SAPscript style (in this case, the name of the SAPscript style is YS_STYLE1) to a Smart style. Perform the following steps to do so:

1. Open the initial screen of Smart Styles by using the SMARTSTYLES transaction code and then enter the name of the Smart style that you want to create, for example, enter ZSF_STYLE_1 (see Figure 12.47).
2. Select `Utilities(M) > Convert SAPscript Style`, as shown in Figure 12.58:

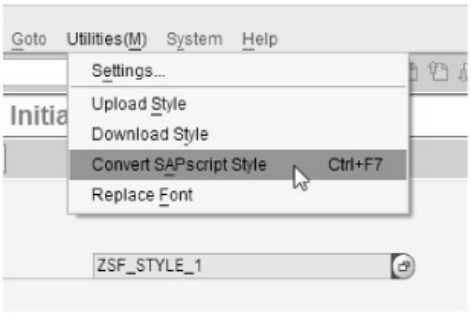


Figure 12.58: Selecting the convert SAPscript style option

The `Convert SAPscript Style` dialog box appears.

3. In the `Convert SAPscript Style` dialog box, enter the name of the SAPscript style in the `From (SAPscript Style)` field. For example, enter YS_STYLE1, as shown in Figure 12.59:
4. Click the `Continue` icon or press the ENTER key in the `Convert SAPscript Style` dialog box to proceed. The `Create Object Directory Entry` dialog box appears.
5. In the `Create Object Directory Entry` dialog box, enter the name of the package as ZKOG_PCKG in the `Package` field and then click the `Save` icon.

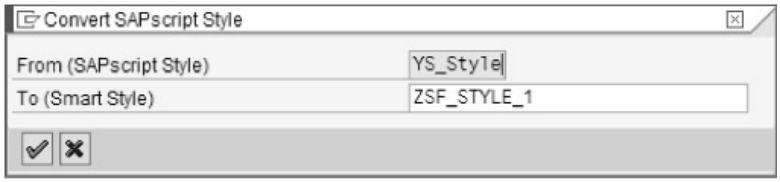


Figure 12.59: The convert SAPscript style dialog box

A list of the converted styles are displayed in the initial screen of Smart styles, as shown in Figure 12.60:

You can click the `Back` icon from the standard toolbar to navigate to the `Smart Styles: Initial Screen` screen and then click the `Change` button to make additional changes in the converted Smart style. Note that you must activate the Smart style by using the `Activate` icon before using it to create a Smart Form.

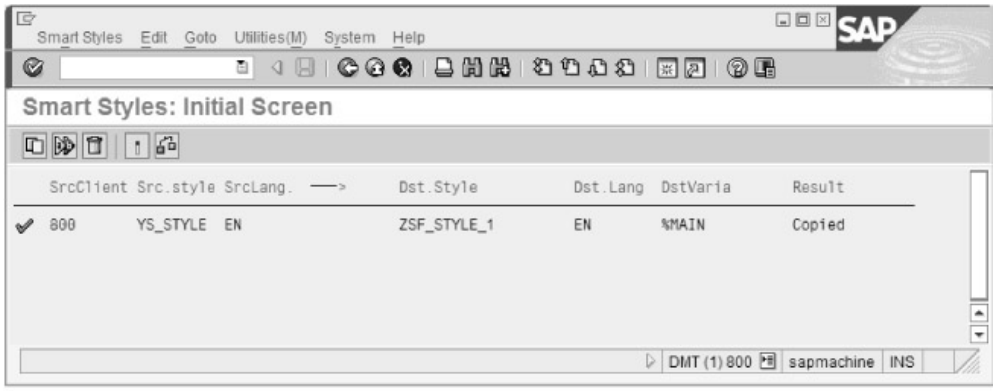


Figure 12.60: Displaying the converted styles in the initial screen of smart styles

Summary

In this chapter, you have learned about the SAPscript and SAP Smart Forms tools in detail. This chapter has also discussed the structure and components of an SAPscript form, the print programs used to print the forms, and the various function modules, control commands, and symbols used in SAPscript. In addition, you have learned about the SAP Smart Forms tool and its advantages. You have also explored the differences between an SAPscript form and a Smart Form. Finally, you have learned how to migrate an SAPscript form to a Smart Form.

In the next chapter, you learn how to create a report based on database tables and explore the concepts of interactive and Application List Viewer (ALV) reports.