

BC480

PDF-Based Print Forms

SAP NetWeaver

Date	<hr/>
Training Center	<hr/>
Instructors	<hr/>
Education Website	<hr/>

Participant Handbook

Course Version: 62 Revision A

Course Duration: 3 Day(s)

Material Number: 50093314



An SAP course - use it to learn, reference it for work

Copyright

Copyright © 2009 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Trademarks

- Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® and SQL Server® are registered trademarks of Microsoft Corporation.
- IBM®, DB2®, OS/2®, DB2/6000®, Parallel Sysplex®, MVS/ESA®, RS/6000®, AIX®, S/390®, AS/400®, OS/390®, and OS/400® are registered trademarks of IBM Corporation.
- ORACLE® is a registered trademark of ORACLE Corporation.
- INFORMIX®-OnLine for SAP and INFORMIX® Dynamic ServerTM are registered trademarks of Informix Software Incorporated.
- UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of the Open Group.
- Citrix®, the Citrix logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® and other Citrix product names referenced herein are trademarks of Citrix Systems, Inc.
- HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.
- JAVA® is a registered trademark of Sun Microsystems, Inc.
- JAVASCRIPT® is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.
- SAP, SAP Logo, R/2, RIVA, R/3, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo and mySAP.com are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other products mentioned are trademarks or registered trademarks of their respective companies.

Disclaimer

THESE MATERIALS ARE PROVIDED BY SAP ON AN "AS IS" BASIS, AND SAP EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR APPLIED, INCLUDING WITHOUT LIMITATION WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THESE MATERIALS AND THE SERVICE, INFORMATION, TEXT, GRAPHICS, LINKS, OR ANY OTHER MATERIALS AND PRODUCTS CONTAINED HEREIN. IN NO EVENT SHALL SAP BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR PUNITIVE DAMAGES OF ANY KIND WHATSOEVER, INCLUDING WITHOUT LIMITATION LOST REVENUES OR LOST PROFITS, WHICH MAY RESULT FROM THE USE OF THESE MATERIALS OR INCLUDED SOFTWARE COMPONENTS.

About This Handbook

This handbook is intended to complement the instructor-led presentation of this course, and serve as a source of reference. It is not suitable for self-study.

Typographic Conventions

American English is the standard used in this handbook. The following typographic conventions are also used.

Type Style	Description
<i>Example text</i>	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths, and options. Also used for cross-references to other documentation both internal (in this documentation) and external (in other locations, such as SAPNet).
Example text	Emphasized words or phrases in body text, titles of graphics, and tables
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example SELECT and INCLUDE.
<i>Example text</i>	Screen output. This includes file and directory names and their paths, messages, names of variables and parameters, and passages of the source text of a program.
Example text	Exact user entry. These are words and characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.

Icons in Body Text

The following icons are used in this handbook.

Icon	Meaning
	For more information, tips, or background
	Note or further explanation of previous point
	Exception or caution
	Procedures
	Indicates that the item is displayed in the instructor's presentation.

Contents

Course Overview	vii
Course Goals	vii
Course Objectives	vii
Unit 1: Overview.....	1
Overview: Architecture	2
Unit 2: Interface	13
The Interface as the Link Between Program and Form.....	14
Unit 3: Context.....	35
Form Context: Using and Extending a Form Interface.....	36
Unit 4: Designer	79
Adobe LiveCycle Designer: Overview	80
Adobe LiveCycle Designer: Structuring a Form	108
Unit 5: Layout.....	127
Static Form Elements.....	129
Dynamic Form Elements.....	143
Tables.....	171
Unit 6: Scripting in the Form	193
Scripting for Form Elements.....	194
Unit 7: Integration into ABAP Programs	217
Integration of Forms into ABAP Programs.....	218
Unit 8: Tips and Tricks.....	233
Download/Upload and Import	234
Accessibility Aspects	240
Unit 9: Appendix	249
Styles	251
ABAP Exception Handling	262
Customizing	267
Basic Administration	271
Tables in Adobe LiveCycle Designer 6.0 or 7.0	276

Migration	282
Index	305

Course Overview

In this course, you will learn about the tool “SAP Interactive Forms by Adobe” and its key capabilities. You will learn how to design forms and how to integrate them into ABAP print scenarios.

Target Audience

This course is intended for the following audiences:

- Project team members, developers and consultants who are responsible for form printing

Course Prerequisites

Required Knowledge

- Programming experience with ABAP

Recommended Knowledge

- Experience with SAPscript or Smart Forms strongly recommended, but not compulsory



Course Goals

This course will prepare you to:

- Create print forms with the tool “SAP Interactive Forms by Adobe”
- Integrate forms into ABAP application programs



Course Objectives

After completing this course, you will be able to:

- Use transaction SFP
- Create and model form contexts
- Create and model form interfaces
- Use Adobe LiveCycle Designer
- Create and design complex forms for printing with the tool “Interactive Forms”
- Integrate forms into ABAP programs

SAP Software Component Information

The information in this course pertains to the following SAP Software Components and releases:

- SAP NetWeaver 7.0

Unit 1

Overview

Unit Overview

This unit gives an overview of the transactions and tools involved at design time. It also provides a brief overview of the technology that is actually used at runtime.



Unit Objectives

After completing this unit, you will be able to:

- Name relevant parts of the architecture of the tool “Interactive Forms”
- Name the steps to create a printing scenario with the tool “Interactive Forms”

Unit Contents

Lesson: Overview: Architecture	2
--------------------------------------	---

Lesson: Overview: Architecture

Lesson Overview

This lesson will introduce you to the components of the “SAP Interactive Forms” tool that are required for a printing scenario, and show you how these components interact with the SAP system.



Lesson Objectives

After completing this lesson, you will be able to:

- Name relevant parts of the architecture of the tool “Interactive Forms”
- Name the steps to create a printing scenario with the tool “Interactive Forms”

Business Example

A travel agency wants to print invoices for their customers' flight bookings using the tool “Interactive Forms”. In the project planning phase, an overview of the architecture and some basic technical background is required.

Parts involved in a printing scenario

How does creating a document work from the user's perspective?

- The user starts the application, for example using a transaction code.
- The user enters the data required on the screen and triggers the business process.
- There are a number of possible printing scenarios: One is previewing the PDF document in Adobe Reader, which is automatically loaded into the SAP GUI. From here it is possible to trigger a printout so that a spool request is created, which can be administered in transaction SP01.

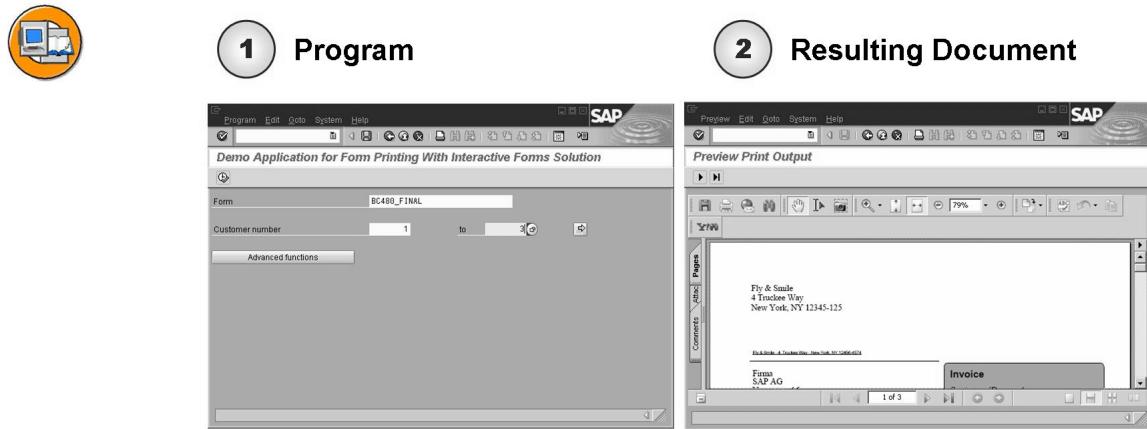


Figure 1: User's Perspective: Printing with PDF-Based Forms

→ **Note:** For an overview of available forms for mySAP ERP 2004, see SAP Note 735050. Please note that the forms delivered with mySAP ERP 2004 are for pilot customers only.

For an overview of limitations for the solution Interactive Forms in SAP NetWeaver 2004s, please see SAP Note 894389.

For an overview of general limitations for mySAP ERP 2005, see SAP note 852235.

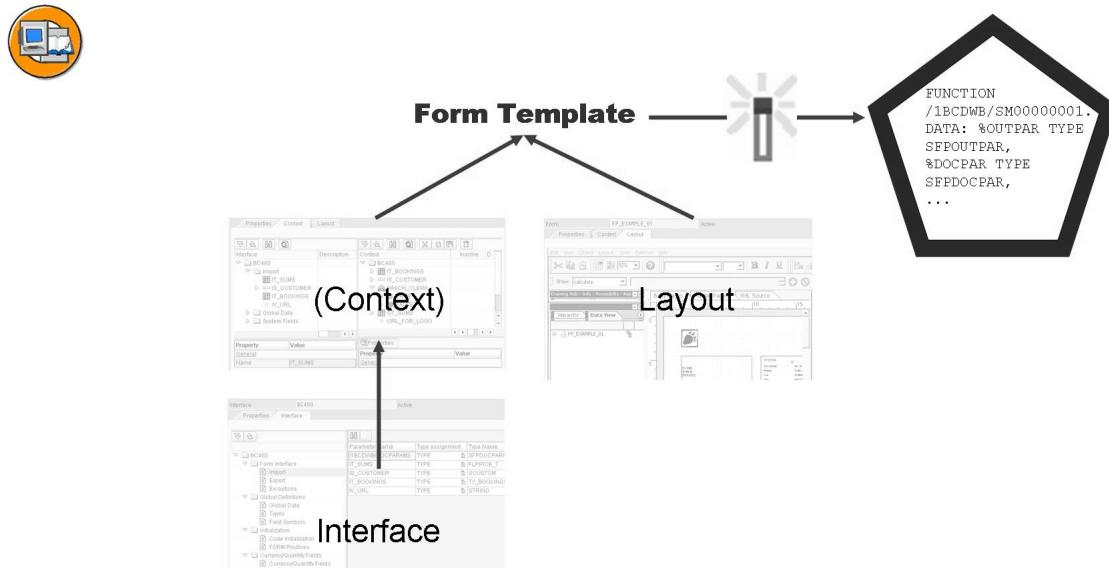


Figure 2: The Tools Involved (Design Time)

At design time, the following needs to be done:

Interface (transaction SE80 or SFP): The interface is used to define which data a program can pass on to a form. It also contains global data and initialization coding that can be used in a form. Interfaces are reusable. They are Repository objects.

Form design (transaction SE80 or SFP): A print form design normally consists of the **context** and the **layout**.

- In the context, you define which parts of the interface you actually want to use in that particular form. You can also add elements like text modules or images.
- The layout typically consists of static elements and dynamic elements (which often have equivalents in the context). The layout is defined in Adobe LiveCycle Designer. It is displayed in a special XML format: XDP = XML Data Package.

Before a form template can be used, it must be activated. Activating a form template involves a context syntax check, saving the form template, and, most importantly, the generation of a function module. The generated function module encapsulates all properties of a form and is called whenever a program triggers form processing.

In some parts of the documentation, you will also find a **form object** instead of a **form template**.

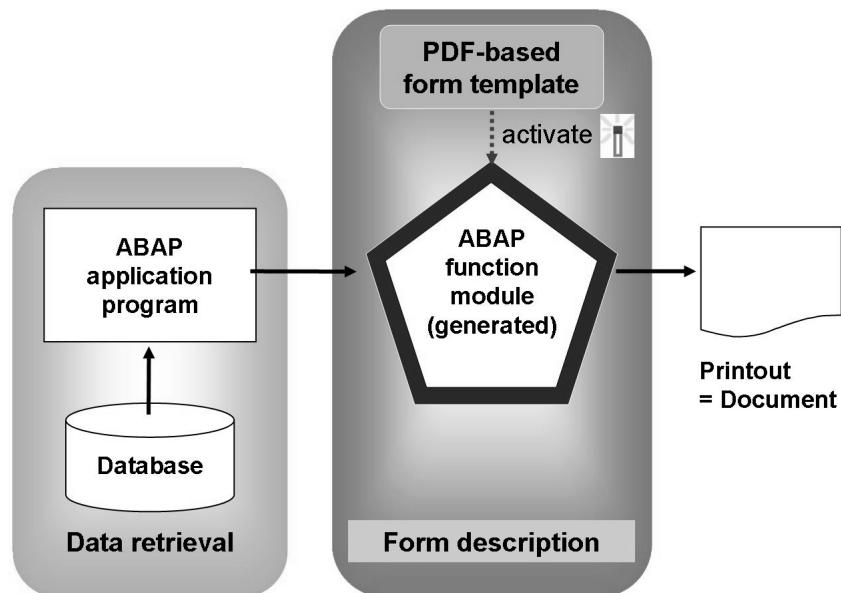


Figure 3: What Happens at Runtime

At runtime, an ABAP program retrieves/calculates all necessary data and passes it on to the generated function module. As interfaces are reusable, a single program could be used for various form function modules, as long as they use the same form interface.

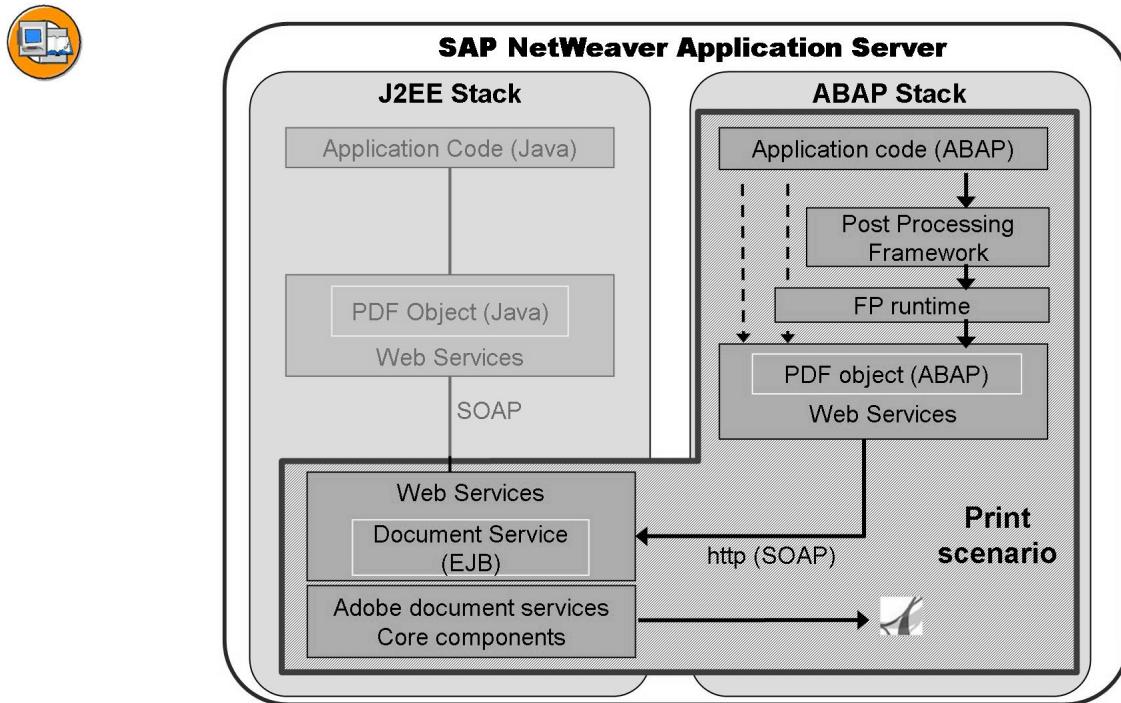


Figure 4: Architecture of a print scenario

A look behind the scenes:

Since SAP Web Application Server (SAP Web AS) 6.20, an SAP system can not only run ABAP programs, but also Java programs. For this purpose, a Java engine must be installed (Java 2 Enterprise Edition, or J2EE).

There are basically two scenarios how PDF based forms can be used in an SAP system:

- Integration into classical ABAP programs.

This is typically the case when mass processing of forms is required, e. g. for printing, mailing or faxing them. Data retrieval, user communication, or result processing (like spool processing) is done with the logic of ABAP screens (using SAP GUI). Technically speaking, interactive scenarios are also possible with SAP GUI integration, but you would typically have a browser-based UI for interactive scenarios.

- Integration into browser-based, interactive scenarios.

In interactive scenarios, individual forms are processed and displayed in a web browser. The user can then enter data into the form and trigger the further processing. For example, you can use Java Web Dynpro or (with SAP NetWeaver 2004s or higher) ABAP Web Dynpro.

In both cases, the form rendering is done by Adobe document services, which are part of SAP NetWeaver's J2EE engine.

A more detailed view of the ABAP scenario (= print scenario):

At runtime, an ABAP application program uses the Post Processing Framework (PPF) to determine whether an output is required and, if so, which output. The Post Processing Framework provides SAP applications with a uniform interface for condition-dependent generation of actions (for example, printing delivery notes, faxing order confirmations, or triggering approval procedures). The actions are generated if specific conditions occur for an application document. They are then either processed immediately or at a later time. PPF is the successor to Output Control.

The form processing runtime provides functionalities like opening and closing a spool job for PDF-based forms. It will then call a Web service, which will communicate with Adobe document services via Simple Object Access Protocol, or SOAP (an XML standard), using the http protocol. Adobe document services are responsible for form rendering, including filling in fields, page breaks, or applying layout settings.

Adobe document services must be deployed on the J2EE engine. They consist of the Adobe document services Web service and the following SAP J2EE services: Document Services Data Manager, Document Services Font Manager, PDF Manipulation Module, XML Form Module, and Document Services License Service. These services can be installed together with SAP NetWeaver Application Server.

Adobe document services will return the PDF, which will then be processed further, depending on the settings of the ABAP program. For example, it might be sent to the spool for a printout or mailed.

In some parts of the documentation, Adobe document services might be abbreviated as ADS.

Advantages over Smart Forms/SAPscript



- PDF is a de-facto standard for forms in the Web
- Adobe LiveCycle Designer as an easy to use, flexible tool for designing forms
- Adobe LiveCycle Designer is fully integrated into the SAP IDEs: SAP NetWeaver Development Studio (Java) and ABAP Workbench
- Graphics (BMP, JPEG, GIF, PNG, EXIF) can be inserted into forms directly – no conversion required
- Objects (including texts) can be rotated
- Different page orientations (landscape, portrait) are possible within one form
- Graphical elements can be included in forms
- Forms can be created so that they conform to accessibility standards
- Complex layout elements can be shared between form developers
- Existing PDF or Word documents can be imported
- TrueType Fonts can be used; installation requires no upload
- Barcodes can be printed on all printers of types Postscript, PCL, PDF, or Zebra
- Mailing and faxing is easier
- Forms are regular Repository objects with standard transport and versioning
- Interactive scenarios and integration into browser-based applications are possible (Web Dynpro for Java or ABAP)



Lesson Summary

You should now be able to:

- Name relevant parts of the architecture of the tool “Interactive Forms”
- Name the steps to create a printing scenario with the tool “Interactive Forms”



Unit Summary

You should now be able to:

- Name relevant parts of the architecture of the tool “Interactive Forms”
- Name the steps to create a printing scenario with the tool “Interactive Forms”



Test Your Knowledge

1. An interface can exist without a form.
Determine whether this statement is true or false.
 True
 False

2. A context can exist without a form.
Determine whether this statement is true or false.
 True
 False

3. The form layout can include elements from the context.
Determine whether this statement is true or false.
 True
 False

4. A J2EE engine is required for interactive scenarios only.
Determine whether this statement is true or false.
 True
 False

5. A function module is generated whenever a PDF-based form is printed.
Determine whether this statement is true or false.
 True
 False



Answers

1. An interface can exist without a form.

Answer: True

An interface is an independent repository object. The interface can also exist without a form by means of which it is used.

2. A context can exist without a form.

Answer: False

A context is always part of a form.

3. The form layout can include elements from the context.

Answer: True

Most of the dynamic elements of a form typically come from the context.

4. A J2EE engine is required for interactive scenarios only.

Answer: False

A J2EE engine is also needed for print scenarios, as Adobe document services need to be deployed on the J2EE engine.

5. A function module is generated whenever a PDF-based form is printed.

Answer: False

A function module is generated only when a PDF-based form is activated. This function module is called whenever a PDF-based form is processed.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Unit 2

Interface

Unit Overview

Every PDF-based form needs to have an interface; it is the link between the ABAP program and the form. The program can pass data to the form only if it is defined in the interface. Most of the dynamic data used in the form layout is defined in the interface. This unit walks you through the process of creating an interface that can then be used in any number of forms.



Unit Objectives

After completing this unit, you will be able to:

- List the components of an interface
- Create and design an interface for a form

Unit Contents

Lesson: The Interface as the Link Between Program and Form	14
Exercise 1: The Interface as the Link Between Program and Form ..	23

Lesson: The Interface as the Link Between Program and Form

Lesson Overview

When you create a PDF-based form, you must first create an interface, in which the fields available in the form are defined. This lesson will show you how to create an interface.



Lesson Objectives

After completing this lesson, you will be able to:

- List the components of an interface
- Create and design an interface for a form

Business Example

A travel agency wants to print invoices for their customers' flight bookings, using the tool "Interactive Forms". An application program is available, but not the form with its interface.

Types of an Interface



Interface	BC480	Active
Properties	Interface	
Description	Training interface for BC480 forms	
Interface Type	ABAP Dictionary-Based Interface	

- **ABAP Dictionary based**
- **Smart Forms compatible**
- **XML schema based (mainly Web Dynpro)**

Figure 5: Interface: Properties

To access interface maintenance, use transaction SFP. Alternatively, use transaction SE80 and choose *Other Object*. Choose the *More* tab page and select *Interface*. In the example above, the interface is called BC480.

On the *Properties* tab page you determine the type of the interface. These types differ in the number and types of parameters and other elements they have. The types "ABAP Dictionary-Based" and "Smart Forms-Compatible" are used for print scenarios, whereas the type "XML Schema-Based" (which was introduced in SAP NetWeaver 2004s) is primarily used for Web Dynpro scenarios.

If you create an interface to be used in an ABAP print program, you should choose “ABAP Dictionary-Based”. While it is possible to use the Smart Forms-Compatible interface type (for the sake of keeping an old print program without changes), it is not recommended to do so because:

- You will get incorrect print parameters (for example, for the cover page) and XSF/XDF parameters that cannot be used any more.
- If the spool job contains several PDF documents, you will get only one document back.
- Only the “ABAP Dictionary” interface will allow you to make settings for the Business Communication Service.

Parts of an Interface: Types “ABAP Dictionary” or “Smart Forms”

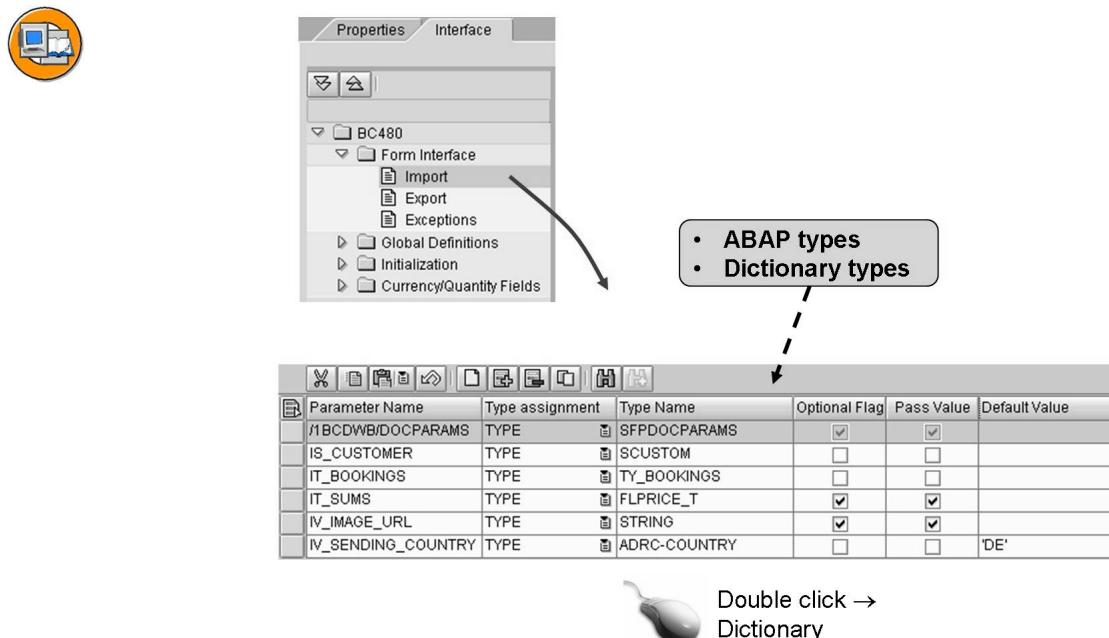


Figure 6: Form interface

When you double-click on one of the nodes in the interface tree on the left-hand side of the screen, an area opens up on the right-hand side that lets you enter details. In this area you can enter information that you would also enter for an ordinary function module in the Function Builder (transaction SE37):

- Parameter Name
- For type assignment, TYPE is the only option (in interfaces from Smart Form migrations, you can also use LIKE).
- Type name: You can enter ABAP types here (c, i, n, among others) and Dictionary types (such as data elements or tables).
- As several forms can use the same interface, you should think about including optional parameters that might not always be needed in all forms. This will help you to keep the number of interfaces small.
- If you set the *Pass Value* flag, a copy of the parameter (not the address) will actually be passed from the program to the form. Such parameters can be changed in the interface coding - the original value remains unchanged. Parameters for which this option is activated slow down system performance, particularly in the case of large parameters (such as internal tables).
- For non-structured parameters, you can also specify a default value, which will be used if the program does not pass on a value for that parameter. All parameters with default values are optional.

The **import parameters** defined in the interface can be passed from the application program to the form at runtime, and vice versa for export parameters.

A form interface of the type “ABAP Dictionary” has only one default import parameter (*/Ibcdwb/docparams* of type *sfpdocparams*). This parameter is used to determine a form's locale (language and country) and whether the form allows interactive functions. The default **export** parameter is */Ibcdwb/formoutput* of type *fpformoutput*. This parameter has to be evaluated in the calling program if the resulting document needs to be handled by the program itself. Export parameters can be added only for those interfaces that are compatible with Smart Forms.

If an interface is compatible with Smart Forms, you can also use **table parameters**. However, you should use them only if you have an internal table that needs to be changed with ABAP coding within the form.

Exceptions that you declare in an interface can be raised in the ABAP coding of a form. They are based on the traditional exception concept (not the class-based concept that was introduced in SAP Web Application Server 6.10).

Example of raising an exception:

```
RAISE <exception>.
```

Alternatively:

```
MESSAGE <message_type><message>(<message_class>) RAISING <exception>
```

If an exception is raised in the interface coding, it will actually be raised in the generated function module. You can evaluate exceptions by querying *sy-subrc* in the ABAP application program.

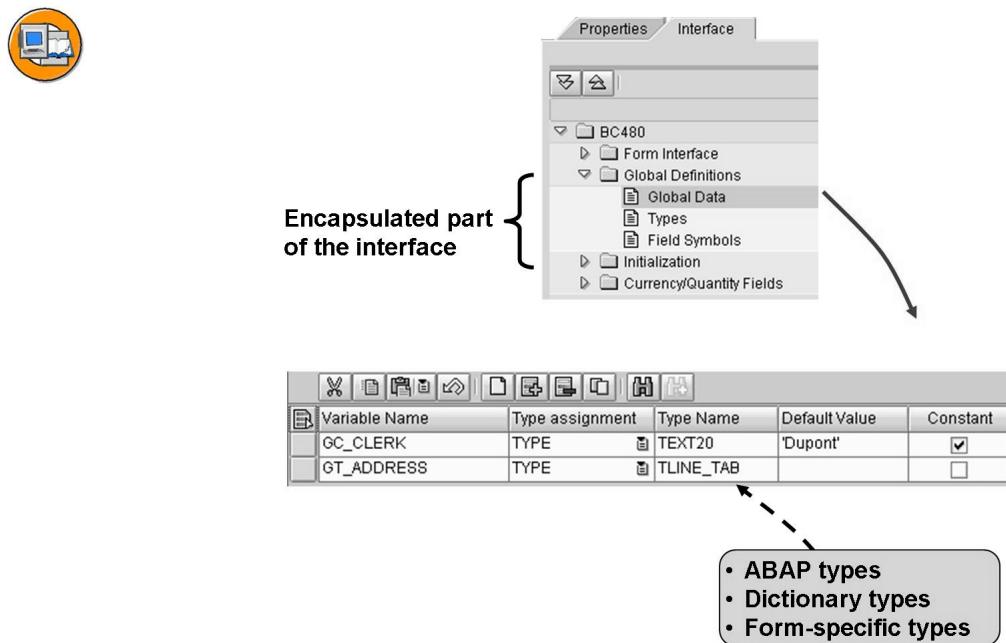


Figure 7: Global Definitions

There are also parts of an interface that are actually invisible from outside and that cannot be accessed from the application program. They include **global definitions**, **initialization coding**, and **currency/quantity fields**.

The global definitions contain the following nodes:

- Global fields: You can set the values of global fields by specifying a default value or by using ABAP initializing coding. In migrated Smart Forms, you can also use them in program line (ABAP) nodes. Global fields can be integrated into the form layout.
- Field symbols: Field symbols can be used as placeholders for variables. They are useful in dynamic programming and for speeding up the processing of internal tables. In the form, they can be used in the same places as global fields, although the coding differs.

In PDF-based forms, values can be assigned to field symbols in the ABAP coding.

For details on how to use field symbols in ABAP coding, see the ABAP documentation, keyword FIELD-SYMBOLS.

- Global types: If your global fields (or any fields declared in the ABAP coding) need types other than ABAP types (i, n, f, c, p, and so on) or Dictionary types, you can create local types in the editor that opens when you choose *Types*. For details on how to create local types, check the ABAP documentation, keyword TYPES.

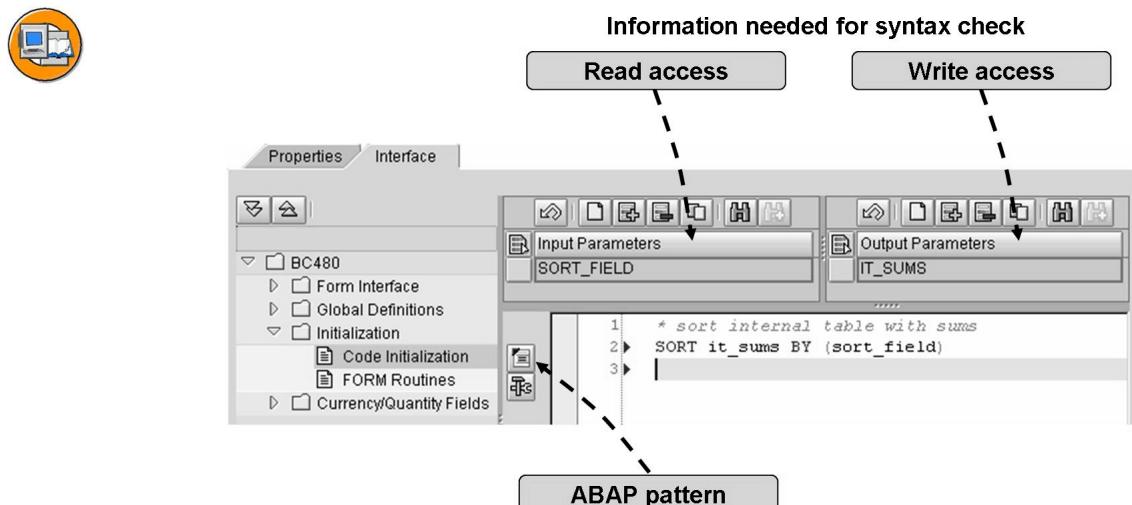


Figure 8: Initialization

During **initialization**, data passed on from the program can be changed before it is sent to the form. In forms that have not been migrated from Smart Forms, initialization is the only time when ABAP coding can be executed.

Even if the initialization coding makes use of form interface fields or global fields, you still have to make them known to the initialization coding. Enter those fields that were read under *Input Parameters*, and those that were read under *Output*

Parameters. You can also work with local variables that you create with the DATA statement. Note, however, that these local variables are not known outside the initialization coding. They cannot be used in the form layout.

The distinction between input parameters and output parameters is made only for structuring purposes. It has no effect on the option to change parameters, as both input and output parameters are passed to the initialization coding by reference. As a consequence, changes to input parameters are permanent.

System fields from the ABAP system structure *SYST* (sy-...) do not need to be declared, but can be used directly in the coding. System fields should not be changed.

In the initialization coding, you can call form routines (subroutines) that you have created in the interface: PERFORM Form routines make sense for coding that must be executed several times. You define form routines by using the ordinary ABAP syntax FORM xyz... ENDFORM.

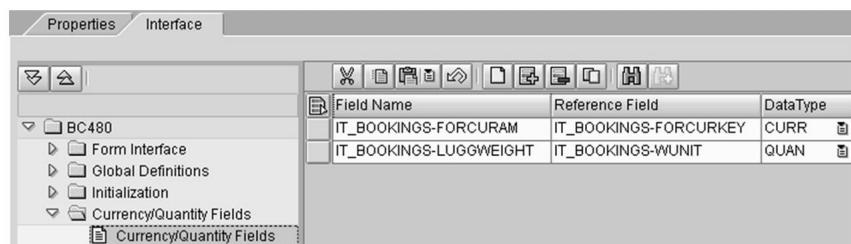
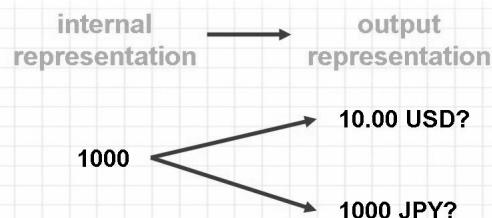


Figure 9: Currency and Quantity Fields

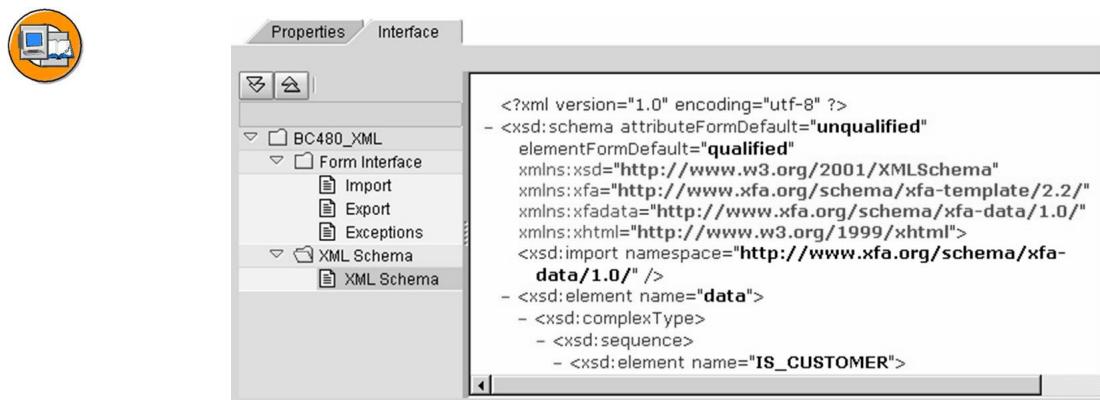
ABAP distinguishes between internal and external representations of currencies and quantities. Whether or not decimals are included in the output format depends on the reference field that contains the amount. Example: The internal numeric string 1000 can be output as 10.00 USD or 1000 JPY. For structured types and table types, these references are typically described in the ABAP Dictionary and evaluated automatically on the screen. (Information on decimal places for currencies and quantities can be found in tables TCURX and T006. These tables are automatically evaluated whenever a Dictionary reference field is used.)

For SAP NetWeaver 2004, this information – even if it is available in the Dictionary – cannot be evaluated for any fields from the form interface. You must explicitly specify a reference field for all currency and quantity fields that you want to print in your form. Only then will you have the correct decimal places.

Use the F4 help to enter the relevant currency and quantity fields. Then select the appropriate reference fields. These fields are often, but not always, in the same structure. Finally, specify for every field whether it is a currency field or a quantity field.

As of SAP NetWeaver 2004s, the currency and quantity information is evaluated automatically. You can define reference fields for your interface fields if you want to override the Dictionary settings.

XML Schema-Based Interface



- Only one import parameter for form data
- Data must conform to XML schema

Parameter Name	Type assignment	Type Name	Optional	Pass Value
/1BCDWB/DOCPARAMS	TYPE	SFPDOCPARAMS	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
/1BCDWB/DOCXML	TYPE	XSTRING	<input type="checkbox"/>	<input type="checkbox"/>

Figure 10: XML Schema-Based Interface

If you want to create an XML schema-based interface, you must have a suitable XSD file to upload. Go to the *Properties* tab to upload your schema. (Typically, an XML-based interface is not created manually, but is generated by a Web Dynpro application.)

Apart from the import parameter /1BCDWB/DOCPARAMS, an XML interface has only one more import parameter, /1BCDWB/DOCXML. All the business data that is transferred to the form at runtime must be passed over with this XSTRING

parameter. It is not possible to add more parameters to an XML form interface. As you cannot have ABAP initialization coding either, you cannot create global data, types, field symbols, or form routines.

Exercise 1: The Interface as the Link Between Program and Form

Exercise Objectives

After completing this exercise, you will be able to:

- Create and design an interface for a form

Business Example

A travel agency wants to print invoices for their customers' flight bookings, using the tool "Interactive Forms". Creating the interface is the first step.

Task 1:

Create an interface.

Please stick closely to the exercise description, because at the end of the exercise, a program will check your interface for errors.

1. Create an interface: ZBC480_##. Save it in package ZBC480_## (## stands for your two-digit group number). The model solution for this exercise is interface BC480.

Task 2:

Specify the type of the interface.

1. Make sure your interface is based on the ABAP Dictionary.

Task 3:

Create the following import parameters:

1. A mandatory structure *IS_CUSTOMER* of type *scustom*.
2. A mandatory internal table *IT_BOOKINGS* of type *ty_bookings*.
3. An optional parameter *IV_IMAGE_URL* of type *STRING* as a value parameter. (This parameter will later contain a hyperlink for an image (for example, a company logo).)
4. Create a parameter *IV_SENDING_COUNTRY* of type *adrc-country* with the default value 'US'. (This parameter will later determine from which country the invoice is to be sent, so that the customer address can be formatted accordingly.)

Continued on next page

Task 4:

Create the following global data:

1. Constant *GC_CLERK* of type *text20*. Assign the value *Dupont*.
2. A variable *GT_ADDRESS* of type *tline_tab*.

Task 5:

Create the initialization coding as follows:

1. Specify the URL for the company logo. If *IV_IMAGE_URL* has no value, assign the address http://wts.wdf.sap.corp:1080/portal/Image/sap_logo.gif.
2. The global field *GT_ADDRESS* should contain the formatted customer's address. Call function module *CUSTOMER_ADDRESS_TO_ITF* and set the parameters appropriately.

Task 6:

Check your interface for correctness.

1. Activate your interface.
2. Run report *SAPBC480_CHECK*. Green, yellow, and red indicators will show you which of your fields are correct, not completely correct, or incorrect.

 **Note:** The coding is not checked.

Solution 1: The Interface as the Link Between Program and Form

Task 1:

Create an interface.

Please stick closely to the exercise description, because at the end of the exercise, a program will check your interface for errors.

1. Create an interface: ZBC480_##. Save it in package ZBC480_## (## stands for your two-digit group number). The model solution for this exercise is interface BC480.
 - a) Start transaction SFP.
 - b) Select *Interface*. Enter the name **ZBC480_##** and choose *Create*.
 - c) In the following window, enter a description and choose *Save (Enter)*.
 - d) In the following window, enter **ZBC480_##** as the package and choose *Save (Enter)*.
 - e) In the following window, press *F4* to select the workbench request the instructor has created for you.
 - f) Continue by pressing *Enter*.

Task 2:

Specify the type of the interface.

1. Make sure your interface is based on the ABAP Dictionary.
 - a) This is automatically the case when you create an interface. Go to the *Properties* tab to check the setting.

Task 3:

Create the following import parameters:

1. A mandatory structure **IS_CUSTOMER** of type *scustom*.
 - a) Choose the *Interface* tab. Double-click on *Import* on the left side. On the right side, choose the icon with the white sheet to append a new row.
 - b) In the column *Parameter Name*, enter **IS_CUSTOMER**.
 - c) In the column *Type assignment*, select **TYPE**. In the column *Type Name*, enter **SCUSTOM**. Leave the remaining columns empty.

Continued on next page

2. A mandatory internal table *IT_BOOKINGS* of type *ty_bookings*.
 - a) See previous task.
3. An optional parameter *IV_IMAGE_URL* of type *STRING* as a value parameter. (This parameter will later contain a hyperlink for an image (for example, a company logo).)
 - a) See previous task.
4. Create a parameter *IV_SENDING_COUNTRY* of type *adrc-country* with the default value 'US'. (This parameter will later determine from which country the invoice is to be sent, so that the customer address can be formatted accordingly.)
 - a) Same procedure as above, but now you deselect the *Pass Value* checkbox. Also, enter 'US' (with single quotes!) in the *Default Value* column.

Task 4:

Create the following global data:

1. Constant *GC_CLERK* of type *text20*. Assign the value *Dupont*.
 - a) Go to the *Interface* tab. Double-click on *Global Data* on the left side. On the right side, choose the icon with the white sheet to append a new row.
 - b) In the *Variable Name* column, enter *GC_CLERK*. In the column *Type assignment*, select *TYPE*. In the column *Type Name*, enter *text20*. Enter '*Dupont*' (with single quotes!) in the column *Default Value*. Select the *Constant* option.
2. A variable *GT_ADDRESS* of type *tline_tab*.
 - a) Same procedure as above, but here you deselect the *Constant* checkbox option and leave the *Default Value* column empty.

Continued on next page

Task 5:

Create the initialization coding as follows:

1. Specify the URL for the company logo. If **IV_IMAGE_URL** has no value, assign the address http://wts.wdf.sap.corp:1080/portal/Image/sap_logo.gif.

- a) Go to the *Output Parameters* and click the icon with the white sheet to append a new row. Enter **IV_IMAGE_URL**.
- b) In the ABAP editor, enter the following coding:

```
IF iv_image_url IS INITIAL.  
  iv_image_url =  
    'http://wts.wdf.sap.corp:1080/portal/Image/sap_logo.gif'  
ENDIF.
```

2. The global field **GT_ADDRESS** should contain the formatted customer's address. Call function module **CUSTOMER_ADDRESS_TO_ITF** and set the parameters appropriately.

- a) Go to the *Input Parameters* and click twice on the icon with the white sheet to append two new rows. Enter **IS_CUSTOMER** and **IV_SENDING_COUNTRY**.
- b) Go to the *Output Parameters* and click on the icon with the white sheet to append a new row. Enter **GT_ADDRESS**.
- c) In the ABAP editor, enter the following coding:

```
CALL FUNCTION 'CUSTOMER_ADDRESS_TO_ITF'  
  EXPORTING  
    is_customer      = is_customer  
    iv_sendig_country = iv_sendig_country  
  IMPORTING  
    et_address       = gt_address.
```

Task 6:

Check your interface for correctness.

1. Activate your interface.
2. Run report **SAPBC480_CHECK**. Green, yellow, and red indicators will show you which of your fields are correct, not completely correct, or incorrect.

 **Note:** The coding is not checked.

- a) Call transaction **SA38** and enter **SAPBC480_CHECK**. Press F8 to start the report.



Lesson Summary

You should now be able to:

- List the components of an interface
- Create and design an interface for a form



Unit Summary

You should now be able to:

- List the components of an interface
- Create and design an interface for a form



Test Your Knowledge

1. Every form interface is compatible with Smart Forms.
Determine whether this statement is true or false.
 True
 False

2. The values of import parameters can always be changed in the initialization coding of a form interface.
Determine whether this statement is true or false.
 True
 False

3. Interfaces can be maintained with transaction SE80.
Determine whether this statement is true or false.
 True
 False

4. You can define your own local types for the important parameters of a form interface.
Determine whether this statement is true or false.
 True
 False

5. You can define your own local types for the global data of a form interface.
Determine whether this statement is true or false.
 True
 False

6. Import parameters of a form interface are always optional.
Determine whether this statement is true or false.
 True
 False

7. Initialization coding is executed by Adobe document services.
Determine whether this statement is true or false.
 True
 False



Answers

1. Every form interface is compatible with Smart Forms.

Answer: False

You can decide whether a form interface should be compatible with Smart Forms. You do this on the *Properties* tab.

2. The values of import parameters can always be changed in the initialization coding of a form interface.

Answer: False

Only those parameters for which the *Pass Value* option is activated can be changed.

3. Interfaces can be maintained with transaction SE80.

Answer: True

You can either use the standalone Form Builder (transaction SFP) or the version that is integrated into the ABAP Workbench (transaction SE80).

4. You can define your own local types for the important parameters of a form interface.

Answer: False

As the interface contains those parts of a form that can be accessed from an ABAP program, you can use only Dictionary types or ABAP types.

5. You can define your own local types for the global data of a form interface.

Answer: True

You define your own local types with the ABAP command TYPES.

6. Import parameters of a form interface are always optional.

Answer: False

The import parameters of a form interface are optional only if they are explicitly selected as optional or if they have a default value.

7. Initialization coding is executed by Adobe document services.

Answer: False

Initialization coding is executed by the ABAP processor before the XML data stream is passed on to Adobe document services.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Unit 3

Context

Unit Overview

The context is essential, as it provides the source for data for a form. Apart from static elements, only those texts, fields, images, and so on can be included in the layout of a form that have been integrated into the form's context. However, the context should not be overloaded, as this will have a negative impact on printing performance.

This unit will teach you all the details of a form context, including how to create and maintain it.



Unit Objectives

After completing this unit, you will be able to:

- Effectively use transaction SFP to create a form context
- Create a form context using an existing interface
- Integrate folders, alternatives, graphics, and addresses into a form context
- Create and integrate long texts into a form context

Unit Contents

Lesson: Form Context: Using and Extending a Form Interface	36
Exercise 2: Form Context: Using and Extending a Form Interface ...	61

Lesson: Form Context: Using and Extending a Form Interface

Lesson Overview

Once you have created an interface, you can use it in a form. More precisely, the form's context makes use of the interface. It can be seen as a subset of the interface enriched with some form specific information. The form context contains the concrete data for the form. (If you use an XML-based interface - typically within a Web Dynpro form scenario - the form does not contain a context. In that case, all data from the interface is available in the form directly.)

This lesson shows you how to create a form context, use an existing form interface, choose the elements that are needed in the form, and add additional elements like texts or graphics.



Lesson Objectives

After completing this lesson, you will be able to:

- Effectively use transaction SFP to create a form context
- Create a form context using an existing interface
- Integrate folders, alternatives, graphics, and addresses into a form context
- Create and integrate long texts into a form context

Business Example

A travel agency wants to print invoices for their customers' flight bookings, using the tool "SAP Interactive Forms by Adobe". A program and interface are available, but no form yet.

Form Properties

To access the Form Builder, use transaction SFP. Alternatively, use transaction SE80 and choose *Other object*. Then choose the *More* tab page and select *Form*.

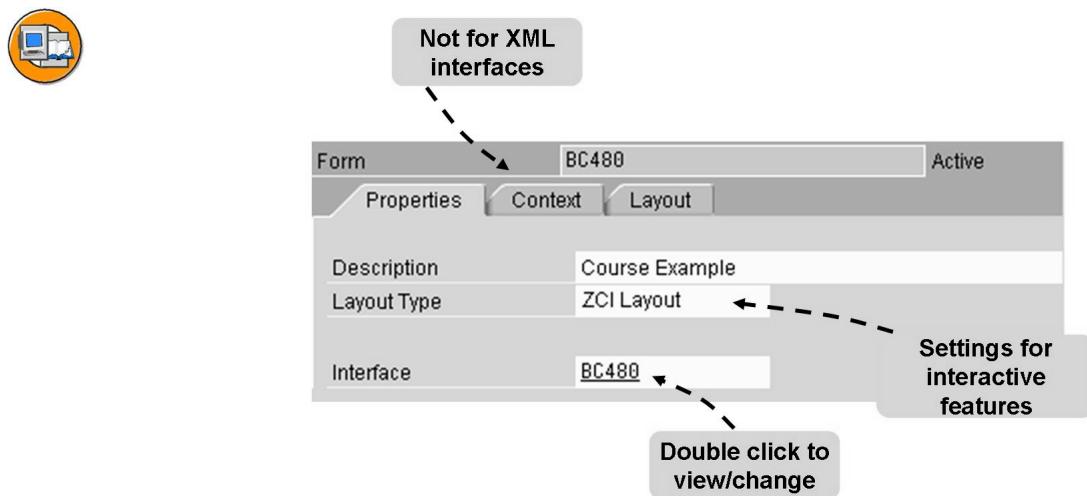


Figure 11: Form Properties

The following details are entered on the *Properties* tab:

- The description of the form
- The layout type. This is relevant only if you create forms with scripting for interactive scenarios. You can choose between the Active Component Framework (ACF), which requires additional software installation (see SAP Note 766191) on the user's PC, and the newer version, Zero Client Installation (ZCI), which requires no extra installation, only a current version of Adobe Reader. See SAP Note 955795 for details.
- The name of the interface to be used: Double-click on the name to display or change the interface. Be careful when you change a form interface that is used in a form; you will not get a warning if you do so, even though you might corrupt the form.

The name of the package will automatically be added to the form properties when you first save the form. If you want to change the package assignment later, choose *Goto → Object directory entry*.

→ **Note:** As a form with an XML-based interface does not contain a context, all of the following information relates to a form with an ABAP Dictionary-compatible or Smart Forms-compatible interface.

General Handling of the Context

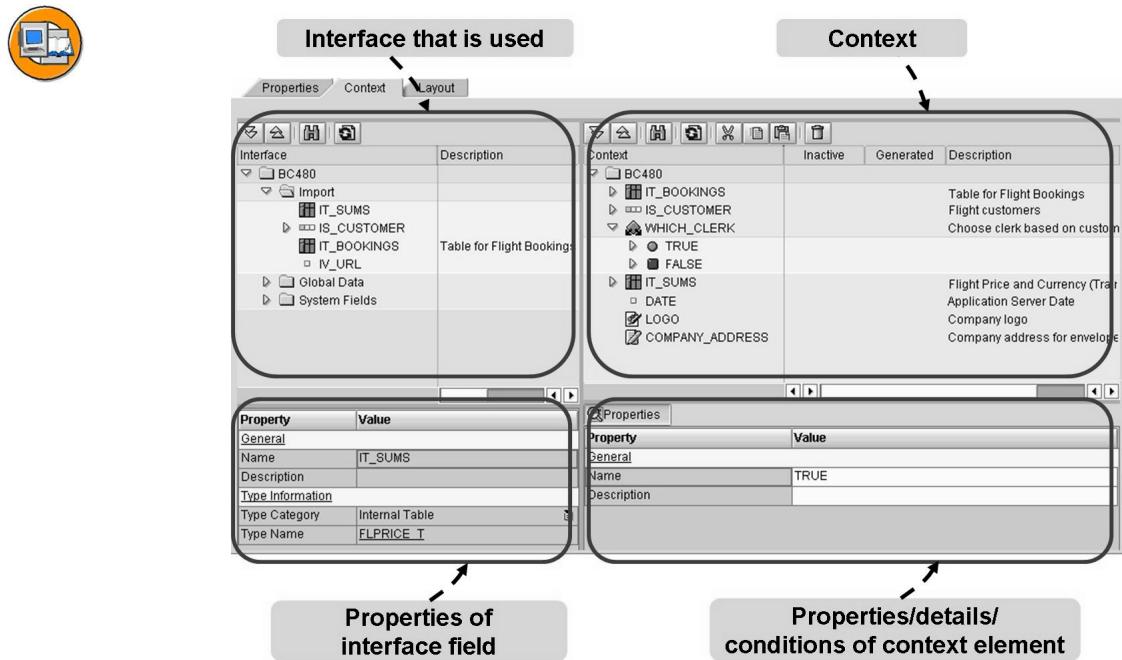


Figure 12: Context: Overview

The screen on the *Context* tab of transaction SFP is split into four areas:

- **Interface**

All fields that are displayed in the *Interface* area (all interface fields plus the system fields *sfpsy-date*, *sfpsy-time*, *sfpsy-username*, *sfpsy-subrc*) can be selected for the context by dragging and dropping them there. If your interface is compatible with Smart Forms, you will have the Smart Forms system fields (*SFSY...*) instead.

- **Context**

The context area contains those elements that will be available in the form layout. These fields are a subset of the interface, and include additional elements like graphics or texts. You can create new elements by right-clicking on the context.

- **Properties of an interface field**

Double-clicking on a field from the interface will display its details in the bottom left area.

- **Properties/details/conditions of a context element**

Double-clicking on an element from the context will display its details in the bottom right area. For example, you would set the source for an image or the name of an external text here.

From NetWeaver 2004s on, for fields with Dictionary reference, their texts can automatically be integrated into the form layout to be used as captions. As the Dictionary (or more precisely: the data element) offers four different kinds of texts, you can choose which one to use.

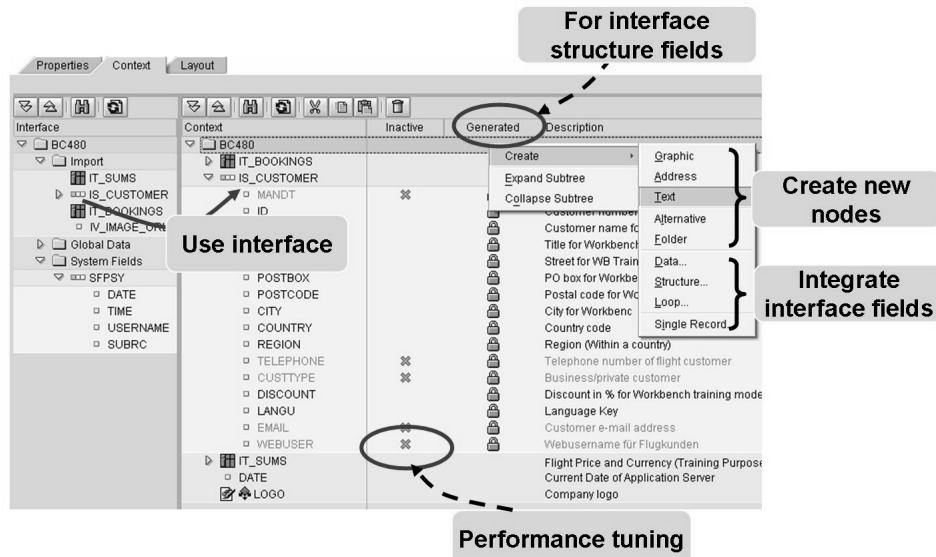


Figure 13: Using the Interface

You can drag and drop elements from the interface tree into the context tree. You can achieve the same effect by right-clicking in the context tree and selecting *Create → Data..., Structure... or Loop....*

If you are using an interface compatible with Smart Forms, you can also integrate ABAP system fields (*sy-...*). To do so, right-click in the context tree and select *Create → Data... or Structure....*

Original fields from a structure or an internal table will be marked as *generated*.

It is possible to drag and drop a field into an internal table, which, for example, allows you to have individual long texts for every dataset.

It is possible to name objects in the context in a different way from those in the interface. Double-click on the element and rename it in the *Properties* section in the bottom right area.

If you include fields from the interface with ABAP Dictionary reference, their Dictionary descriptions will automatically be referenced. Consequently, you cannot change the texts for these fields and you do not have to translate them separately.

You can create new nodes, for example texts and graphics by using the context menu.

In the figure above, the graphic node LOGO was created in this form.

The order in which you integrate fields in the context is irrelevant unless you nest one element in another.

Fields that are not required can be set to inactive (using the right mouse button). They will have a cross in the *Inactive* column. Deactivating a field in the context means that it will not be part of the XML data stream (XFD). Thus, performance can be improved. However, note that deactivating context elements does not turn mandatory fields into optional fields. This means that these fields must always be passed to the form interface by the print program. Setting fields that do not come from the interface (that you define in the context) to inactive makes sense for testing scenarios.

Tables

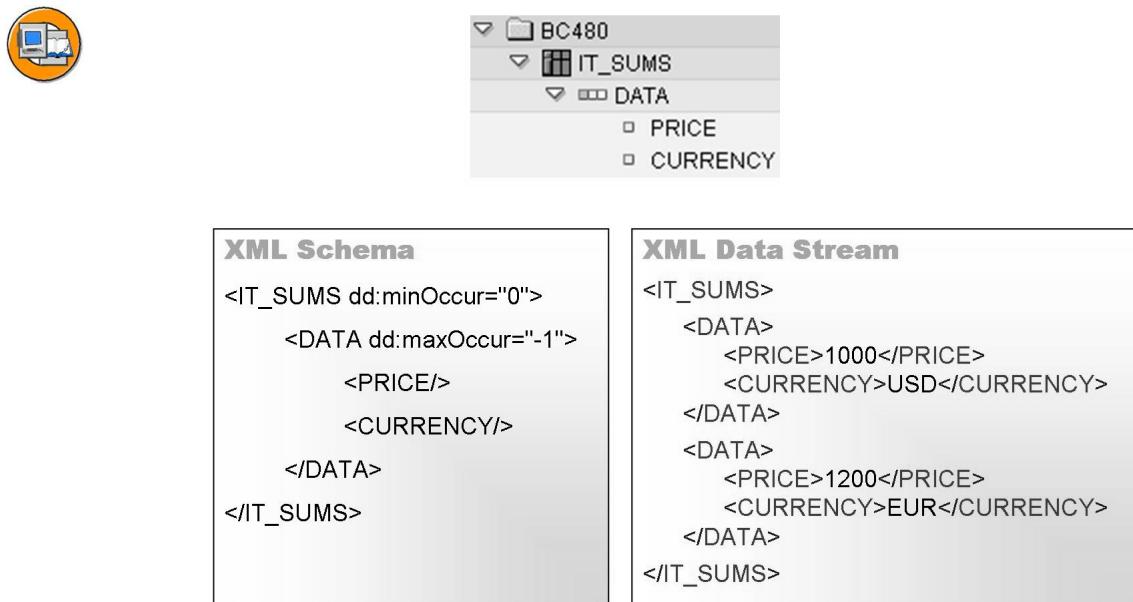


Figure 14: Addendum: Internal Tables Represented in XML

Internal tables are always displayed in nested form in the context; this corresponds to the display in the XML schema (XML schema definition, XSD). The XSD describes how data must be structured in order to be valid. As Adobe LiveCycle Designer requires this XSD, the SAP transaction SFP automatically generates it from the form context. It cannot be changed manually.

In the XSD representation of an internal table, the structure DATA reflects the line type. Hence, if you want to include other fields in the internal table, you must create them in the structure DATA.

You can integrate an internal table into the context by dragging it from the left-hand side to the right-hand-side. Alternatively, you can choose *Create* → *Loop...* in the context menu of the context.

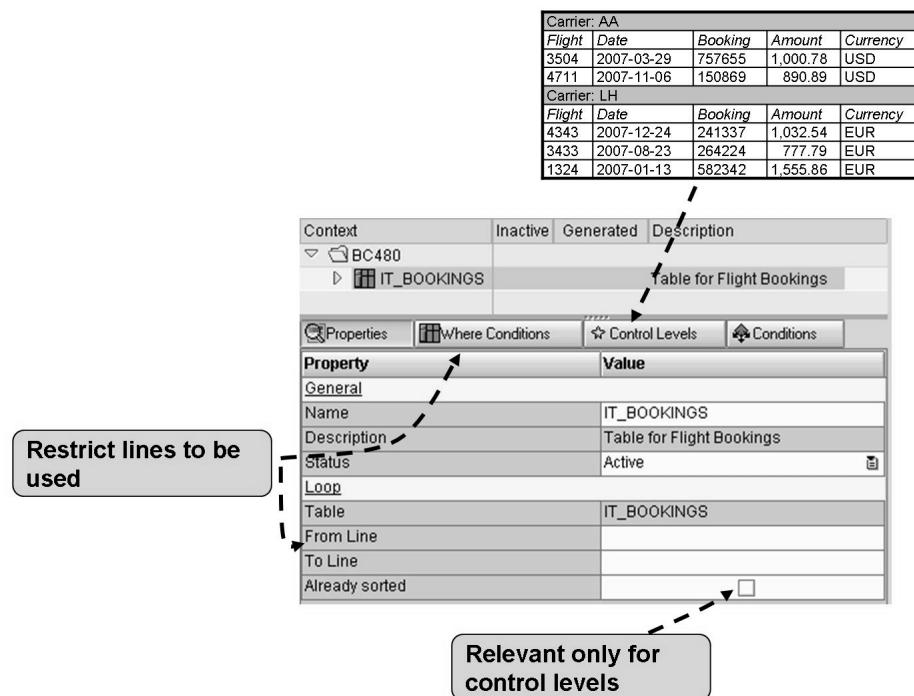


Figure 15: Internal Tables (Loops) in the Context

On the *Properties* tab, you can specify which lines of the internal table will be processed in ABAP and passed to the XML data stream. The corresponding ABAP command is:

```
LOOP AT it INTO wa FROM <r1> TO <r2>.
```

You can also give a WHERE clause, stating that a row will be part of the XML data stream only if a field has a certain value. The corresponding ABAP command is:

```
LOOP AT it INTO wa WHERE <field> = <value>.
```

A WHERE clause is in particular helpful if you want to nest one table T2 into a table T1. Let's assume the ABAP program passes two internal tables to the form: one with all customers (T1) and one with all bookings (T2). Within the document, you would like to print the bookings just for the customer for whom the form is printed. In that case, you would nest T2 into T1 and add a suitable WHERE clause for T2, like CUST_NUMBER = T1-CUST_NUMBER. The result would be a single XML table, which can easily be integrated into the layout later on.

On the *Control Level* tab, you can create control levels to group tables by data fields. In this way, you can, for example, define a nested output for a flat internal table.

Group levels make sense only if the data is sorted according to these levels. You can sort the table yourself in the ABAP coding of the print program or in the initialization coding of the form. You do this by selecting the *Already sorted* checkbox on the *Properties* tab. You also have the option of sorting the

table automatically before rendering the form. Mind, however, that the latter option is possible only if the table is either defined as a global interface field or as a call-by-value import parameter. (A call-by-reference import parameter is read-only.) Please note that control levels were introduced in SAP NetWeaver 2004s.)

If you want to have only one specific row of an internal table, you should not use the entire internal table with a WHERE clause or an index. Instead, you right-click on the context, choose *Create → Single Record...*, specify the internal table to be used and set either the row number or the key for determining the row. This way, the handling in Designer will be easier, as the result will be treated as a structure rather than an internal table. (You need at least SAP NetWeaver 2004s for single records.) The equivalent ABAP command is `READ TABLE`. You can query the result of this single record access using the field `SFPSY-SUBRC`, which is a copy of `SY-SUBRC`.

Conditions and Alternatives

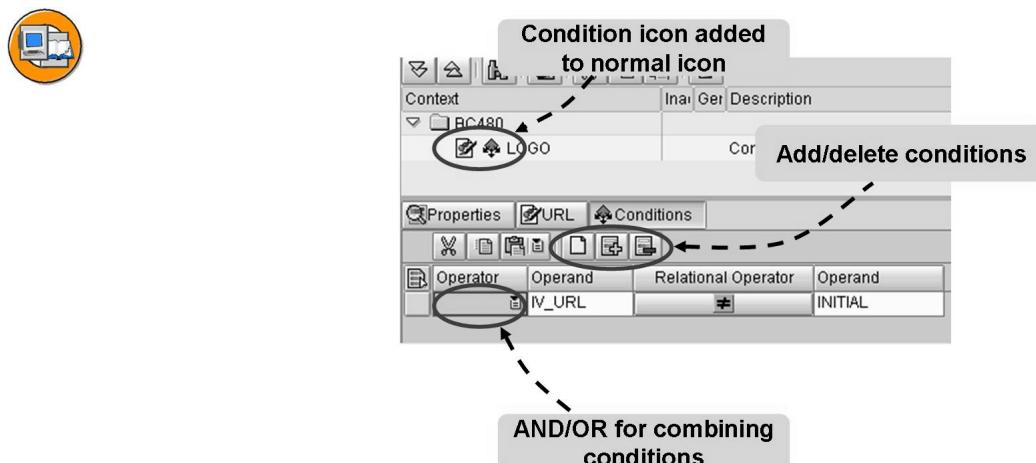


Figure 16: Conditions for Context Elements

You can enter conditions for most node types of the context. If a condition for a context element fails to be correct at runtime, the element field will not be part of the data stream. Conditions are also possible for more complex elements that have “children” in the context tree (structures, internal tables, or folders). If such a condition fails at runtime, none of the subordinate elements will be part of the data stream.

Conditions help to reduce data traffic. They also help in hiding fields from the layout.

All of the fields from the interface (even those that you do not drag and drop to the context) are known to all context fields and can be used in conditions. You cannot refer to other context fields or to ABAP system fields.

You can see at a glance which context nodes have conditions, as they will have an additional condition symbol next to the node symbol itself.

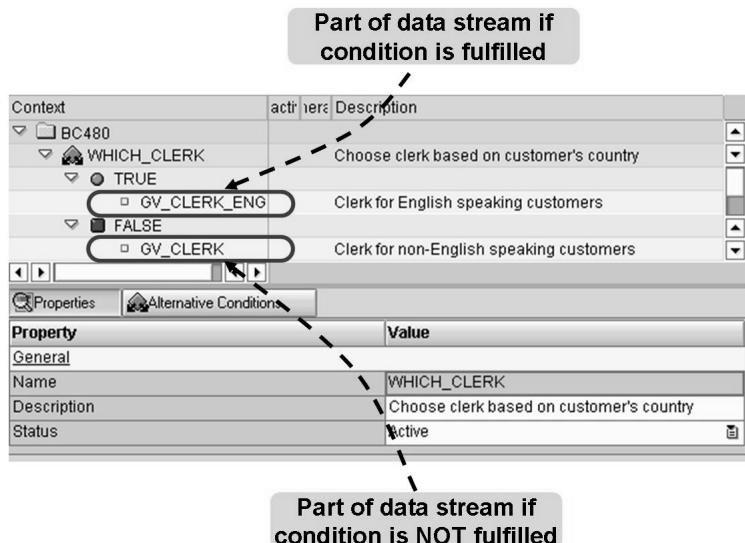


Figure 17: Including Alternatives

An alternative is a node with two subnodes (*TRUE* and *FALSE*), each of which can contain further subnodes of any type. The condition(s) you enter when you choose the *Alternative Conditions* button determine which of the two subnodes will be processed at runtime. If the condition is met, the *TRUE* node is processed, including all of its subnodes. If the condition is not fulfilled, the *FALSE* node is processed together with its subnodes. This query is similar to that of the ABAP commands *IF* and *ELSE*. You can enter the same kinds of conditions as for other nodes.



Caution: The meaning of the condition for an alternative differs from the meaning of conditions of other nodes. The condition for an alternative determines whether the *TRUE* and *FALSE* nodes will be part of the resulting data stream. The condition of an alternative does not determine whether the alternative itself is evaluated or not. (The logic of the alternative is always processed.)

Folders

The more detailed a form becomes, the more difficult it is to get a clear overview of the hierarchy of nodes. To make the hierarchy easier to understand, you can organize nodes in folder nodes. Folders can be collapsed and expanded so that you will see only those nodes in the context that you actually need to see.

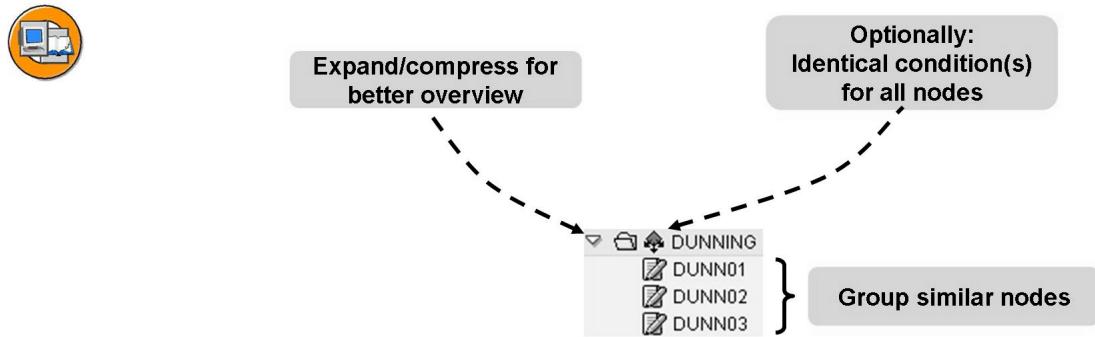


Figure 18: Including Folders

Examples of the usage of folders:

- A dunning form has different dunning texts, of which only one is to be passed on to the data stream at runtime, depending on the reminder days exceeded. You can create these texts (with individual conditions) and group them into a folder.
- Several nodes share the same condition(s). Instead of setting the condition(s) individually for each node, you can create a folder and then assign the condition(s) to this folder only. They will be passed on to its nodes.

Folders make working with Adobe LiveCycle Designer easier, too, as they help you to organize your layout.

Graphics

In the context, you can also define graphics. To do so, right-click on the top node of the context and choose *Create → Graphic*.

Type Graphic Content	
Properties	Conditions
Property Value	
General	
Name	LOGO1
Description	Company logo
Status	Active
Graphic	
Graphic Type	Graphic Content
Graphic Content	
Field	G1
MIME Type	'image/bmp'

Type Graphic Reference		
Properties	URL	Conditions
Property Value		
General		
Name	LOGO2	
Description	Company logo	
Status	Active	
Graphic		
Graphic Type	Graphic Reference	

An arrow points from the 'Graphic Content' table to the 'Graphic Reference' table, indicating the relationship between the two types of graphics.

Figure 19: Including Graphics

In the attributes section of the lower right part of the screen, specify details of the graphic. In particular, specify the graphic type:

- *Graphic Content*: When you select this type, you must specify the name of a field from the context from which the graphic data is to be passed on at runtime. Permissible types are STRING (graphical data is Base64-coded) or XSTRING (for binary-coded graphical data). You must also enter a valid MIME type, such as such as 'image/bmp' or 'image/jpeg'.
- *Graphic Reference*: This type allows you to specify the graphic address dynamically. You must enter the details by choosing the URL button. You can split the URL into dynamic parts (which are represented by individual STRING variables from the interface) and static parts (literals), which must be enclosed in single quotes. The different parts of the graphic URL can be separated by a delimiter, which you can set. In the above example, possible locations of the image at runtime would be <http://www.sap.com/IMAGES/logo1.bmp> or <http://www.sap.com/IMAGES/logo2.gif>.

You can integrate images in the following common formats: Windows Bitmap (*.bmp), JPEG (*.jpg), TIFF (*.TIF), PNG (*.png), and GIF (*.gif – animated GIF images are not supported). From Adobe LiveCycle Designer 7.0 on, you can also add EXIF graphics.

Including Texts

In a form context, four types of long texts can be included:

- Addresses from the Business Address Services
- Text modules (Smart Forms texts)
- Include texts (SAPscript texts)
- Dynamic texts

All these text types are automatically converted by transaction SFP into a special format that can be evaluated by Adobe document services: XHTML.

 **Note:** XHTML is an XML standard that extends HTML 4. (HTML = Hypertext Markup Language – the most commonly used language for publishing pages in the World Wide Web.) XHTML can be viewed in most Web (HTML) browsers.

Addresses



Example 1:



Barbara McCloskey
74 Court Oak Road
Harborne
Birmingham B17 9TN
GREAT BRITAIN



Peter Dennebaum
Alfred-Mumbächer-Str. 28a
55128 Mainz

Example 2:

Frau
Karin Kottenhoff
Geschäftsleitung
Röderwiese 10
58093 Hagen

Karin Kottenhoff
Röderwiese 10
58093 Hagen

Figure 20: Formatted, Country-Specific Addresses

Instead of using their own tables for address information, many applications now access Business Address Services (BAS). In the Business Address Services, addresses are identified by means of numbers.

Before SAP Web Application Server 6.10, BAS was known as Central Address Management.

The addresses that you integrate into your form's layout will be formatted in accordance with country-specific conventions (based on ISO 11180 and the guidelines of the Universal Postal Union). For detailed information, see the documentation on the function module ADDRESS_INTO_PRINTFORM.



BC480
ADDRESS

Address type	1 Organization, company 2 Private address 3 Contact person (in company) <input checked="" type="checkbox"/> Determine dynamically
Address number	Required for all address types
Person number	Required for address types 2 and 3
Sending country	Required to distinguish between domestic and international addresses

Figure 21: Including Addresses in the Context

In the context, you can create nodes that make use of Business Address Services. You do not need to know the technical details of Business Address Services, or worry about the correct formatting of the addresses.

Address nodes will be rendered as regular texts in the form.

Some of the relevant fields in Business Address Services include:

Address type:

- *Company addresses:* Typical examples are delivery addresses or company codes. These addresses are uniquely identified by their address numbers. In the context, you must specify the value of the address number. It must be a ten-digit character field. Typically, this is an interface parameter.
- *Private addresses:* Addresses of this type are assigned to exactly one natural person, along with other associated attributes, such as the form of address. Because a person can have more than one address, enter both the address number and the person number for identification. Both must be ten-digit character fields, and both are usually interface parameters.
- *Contact person in company:* These are personal addresses in companies, which means they have additional attributes such as department or room number. You identify such an address by means of the address number and the person number. Both must be ten-digit character fields, and both are usually interface parameters.
- *Define Dynamically:* If you want the address type to be defined at runtime, enter the name of an appropriate interface parameter. At runtime, this parameter must be filled with 1, 2, or 3.

You must specify the *sending country*. If you enter an interface parameter here, the country can be determined dynamically at runtime. If, at runtime, the address is found to be domestic, the addressee's country will not show.

If the system does not find an address with the numbers specified in Business Address Services at runtime, the generated function module of the form terminates with an error message.

For addresses that have both a post office box number and a street address, use the drop-down list to determine which one to use. You can also choose *Define dynamically*, in which case you must also enter the name of a field that contains X (street) or a space (post office box) at runtime.

In the *Priority of Lines* field, you can determine which part of the address is first suppressed if there is not enough space for the address node. For example, A stands for form of address, P for an obligatory empty line, and D for department. For more information, see the documentation for the function module ADDRESS_INTO_PRINTFORM.

Text Modules and SAPscript Texts

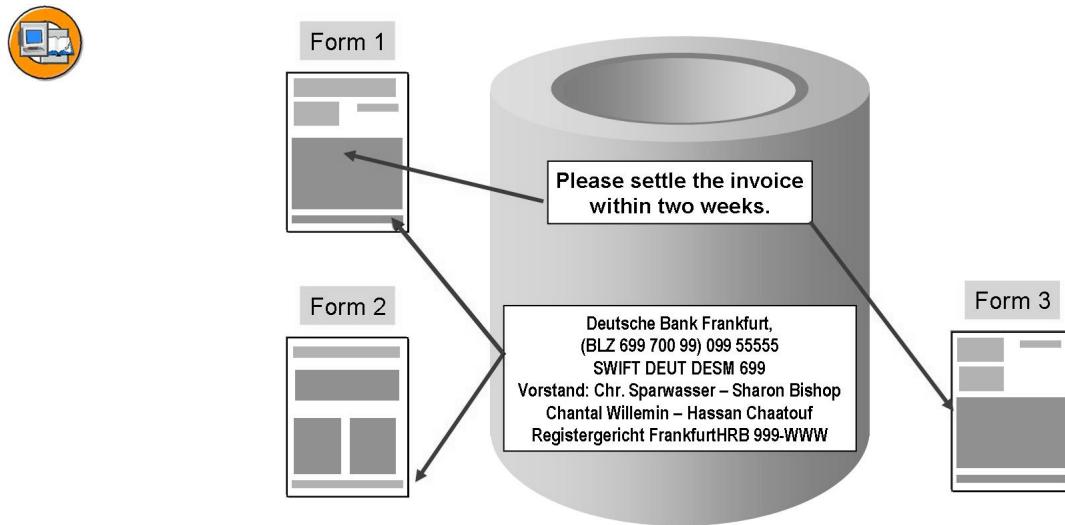


Figure 22: Long Texts from the Database

In some cases, it makes sense to store texts not in the form itself, but centrally in the database. The form then contains only a reference indicating which text should be used at application program runtime. This method has the following advantages:

- You need to create the texts only once and can then reuse them as required.
- You make changes centrally only once without having to modify the actual forms. (The reference in the form remains unchanged.) Typical examples include headers (company address), footers (company information like board members and so on), and whole pages containing introductions or terms of trade.

You can use text modules of Smart Forms and also include texts (that is, SAPscript texts). Run transaction SMARTFORMS (for text modules) or SO10 (for include texts).

With regards to PDF-based forms, the main difference between the two text types is that text modules are client-independent and have an automatic link to the Transport Organizer, whereas SAPscript texts are client-dependent and must be manually included in a transport request.

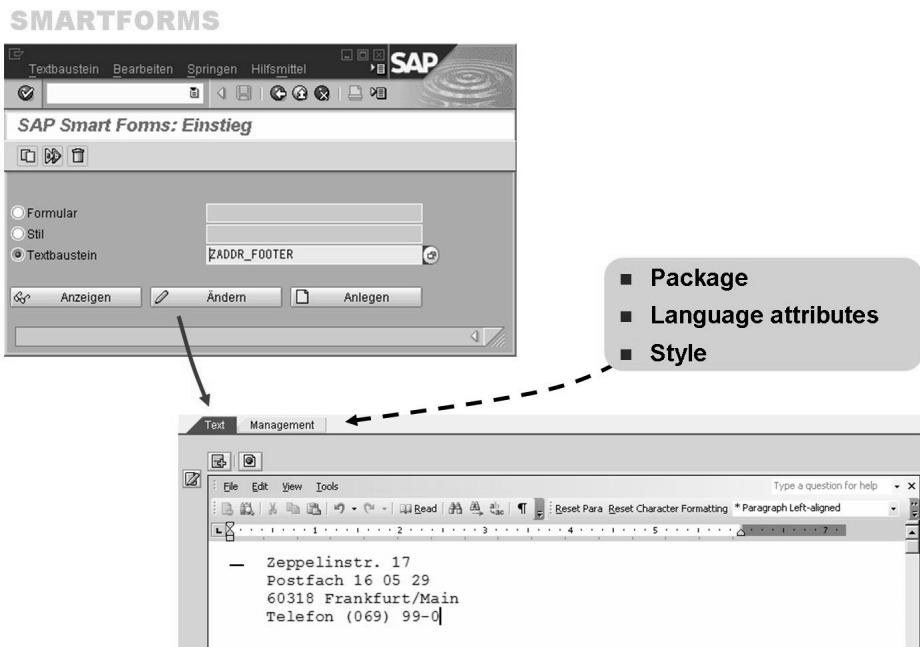


Figure 23: Creating Text Modules

The maintenance transaction for text modules is SMARTFORMS. Select *Text Module* on the initial screen and then choose *Display*, *Change* or *Create*, depending on what you want to do. On this screen, you can also copy, rename, or delete existing text modules. To do this, choose the appropriate button from the toolbar or use the *Text Module* menu.

Since text modules are integrated with the SAP transport system, you must assign them to a package. You do this when you first save your text module.

You must assign a style to each text module on the *Management* tab. The default style is *System*.

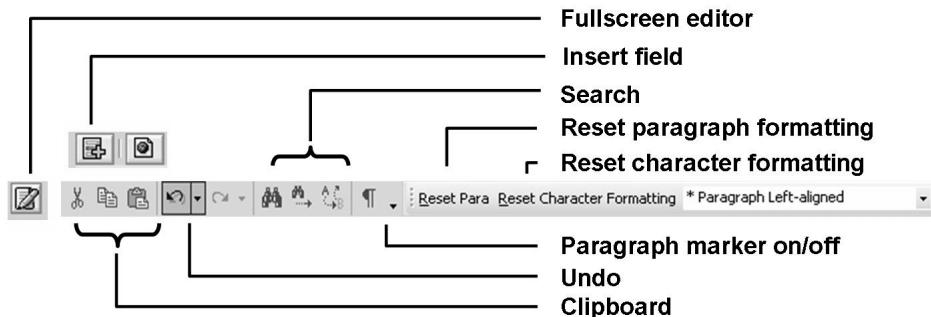


Figure 24: Text Modules: Editor

You can enter text in the same way as you would in any common word-processing system. Alternately, you can switch to full-screen mode by choosing the appropriate button.

You can use the clipboard by selecting text blocks with the mouse and then choosing the *Cut*, *Copy*, or *Paste* buttons. This way, you can copy text sections between different text modules.

Lines in text modules are broken automatically at runtime depending on the width of the form element where they are included. The editor assumes a certain standard width and breaks your text as you type it in, but this does not influence line breaks in the output.

If you want to determine line breaks manually, you have two options:

- You can use the Enter key to create a new paragraph, which may then have a different format than the preceding one.
- Shift+Enter allows you to create a line break within a paragraph.

The *Paragraph mark on/off* button allows you to determine whether you want to display nonprinting characters (blanks, tabulators, paragraph marks, and line breaks).

Though technically it is possible to include hyperlinks in a text module, they will not show in the resulting PDF that includes the text module. (This is true as of SAP Web Application Server 6.40.)

The full-screen editor also allows you to switch to the old line editor used by SAPscript, which technically enables you to type in SAPscript commands. However, all SAPscript commands will be ignored at runtime, so it is best not to use this option.

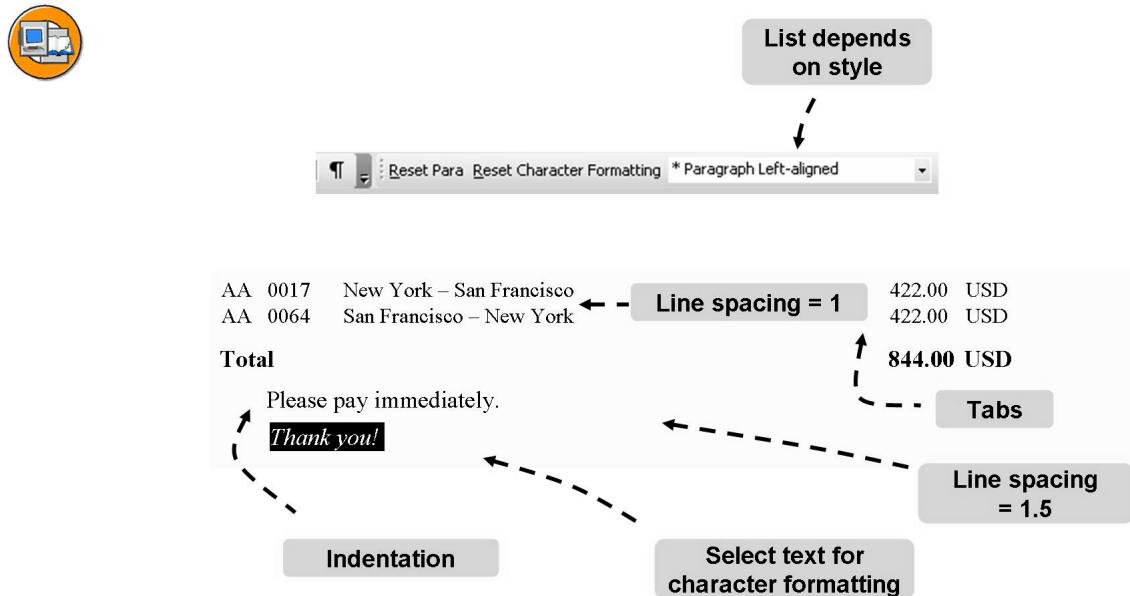


Figure 25: Text Modules: Formats

You can format selected text sections. These text sections are then displayed in the editor as they will appear when printed (WYSIWYG = **What You See Is What You Get**).

For each paragraph, you can choose a paragraph format from the selection list in the editor. A paragraph format is a collection of format settings, such as tabs, type of justification, and so on. The paragraph formats available in the list depend on the style you have chosen.

All formats have a one- or two-digit name, which is defined in the style. An asterisk (*) indicates the default format.

The system automatically displays the set format at the current cursor position in the paragraph and character format list. You therefore only need the *Display formats* function if you want to obtain detailed information on the format, or if text has been formatted using several character formats.

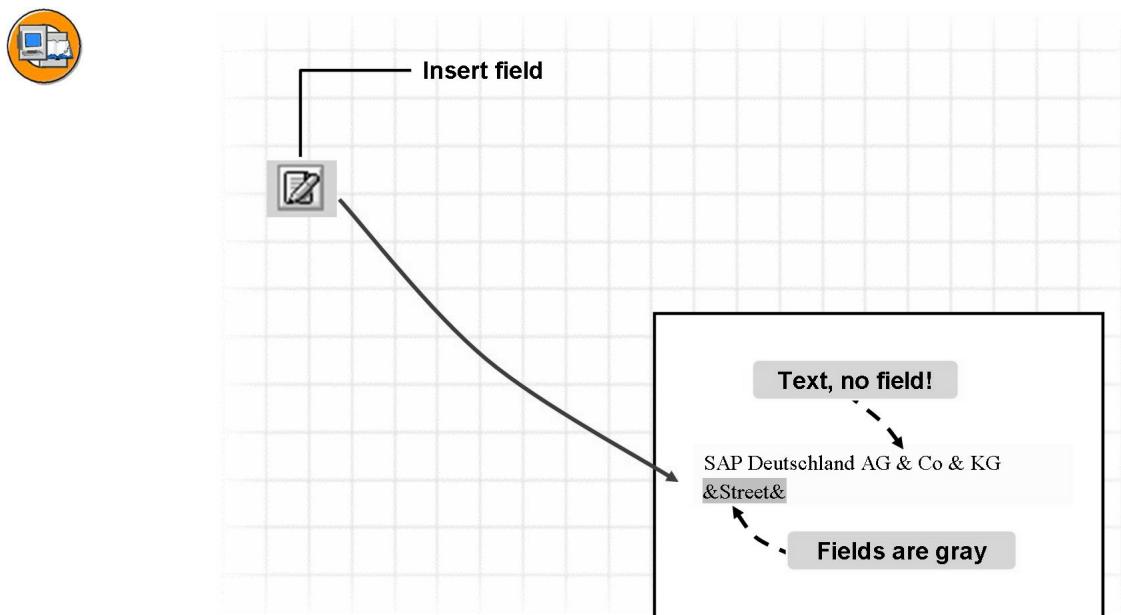


Figure 26: Text Modules: Inserting Fields

Frequently, text modules contain not only static text, but also variable data, referred to as fields. In the editor, fields cannot be inserted or manipulated directly. Instead, use the relevant buttons to insert, edit, or delete a field.

All fields start and end with an ampersand, such as `&SFPSY-DATE&`. You can enter the following fields:

- Fields that are used in a form interface (typically containing business data)
- *sfpsy* fields
- Fields from Dictionary structure *syst*, such as *sy-mandt* (client) or *sy-sysid* (system ID): Note that not all fields seem to make sense. In particular, fields used in list processing, like *sy-pagno*, are not filled by the system and should not be used.

Field names are case-insensitive.

Formatting options as they can be used in SAPscript or Smart Forms (like offsets or the “no zeroes” option) will be ignored in texts when included in PDF-based forms.

Be sure to include only those fields into your text that are actually known at form processing time, that is, those that are defined in the form interface that is actually used. If, at runtime, a field turns out not to be defined, the program execution will terminate. *SFPSY* fields can always be used in text modules, even if they have not been included in the context. The syntax check of the context only includes a check on the text modules used as of SAP NetWeaver 2004s.



Context	Inactive	Generated	Description																								
BC480			Text Node TEXT																								
TEXT																											
<input type="button" value="Properties"/> <input type="button" value="Conditions"/>																											
Property	Value																										
<table border="1"> <tr> <td colspan="2"><u>General</u></td> </tr> <tr> <td>Name</td> <td>TEXT</td> </tr> <tr> <td>Description</td> <td>Text Node TEXT</td> </tr> <tr> <td>Status</td> <td>Active</td> </tr> <tr> <td colspan="2"><u>Text</u></td> </tr> <tr> <td>Text Type</td> <td>Text Module</td> </tr> <tr> <td>Text Module</td> <td></td> </tr> <tr> <td>Text Name</td> <td>'ADDR_FOOTER'</td> </tr> <tr> <td>Text Language</td> <td>LAN</td> </tr> <tr> <td>No error if text not available</td> <td><input type="checkbox"/></td> </tr> <tr> <td>Copy Style From Text Module</td> <td><input checked="" type="checkbox"/></td> </tr> <tr> <td>Style</td> <td></td> </tr> </table>				<u>General</u>		Name	TEXT	Description	Text Node TEXT	Status	Active	<u>Text</u>		Text Type	Text Module	Text Module		Text Name	'ADDR_FOOTER'	Text Language	LAN	No error if text not available	<input type="checkbox"/>	Copy Style From Text Module	<input checked="" type="checkbox"/>	Style	
<u>General</u>																											
Name	TEXT																										
Description	Text Node TEXT																										
Status	Active																										
<u>Text</u>																											
Text Type	Text Module																										
Text Module																											
Text Name	'ADDR_FOOTER'																										
Text Language	LAN																										
No error if text not available	<input type="checkbox"/>																										
Copy Style From Text Module	<input checked="" type="checkbox"/>																										
Style																											

Determined at design time
Determined at runtime

Figure 27: Including Text Modules

If you want to integrate a text module into the context, right-click somewhere in the context tree and choose *Create → Text*.

To determine which text module should actually be taken, specify the name and the language. Both of these can be determined with fixed values or dynamically. In the first case, enter the name and language in single quotes. In the second case, enter the names of interface fields.

If you determine name and language of the text module dynamically, you cannot check from within form maintenance whether the fields are filled with appropriate values at application program runtime. If you want to avoid program termination in case the text does not exist, select *No error if text not available*.

If you do not enter a language, the form's language will be used.

The text module's style determines the available paragraph and character formats. The following options are available:

- *Copy Style From Text Module* means that you want to use the style of the text module.
- You can also specify a different style, which will then be applied to the text module. As the style cannot be set dynamically, you do not enclose the style in quotes.
- If you choose neither of these options, no style will be applied (not even Smart Form's standard style *System*), which will result in plain text.

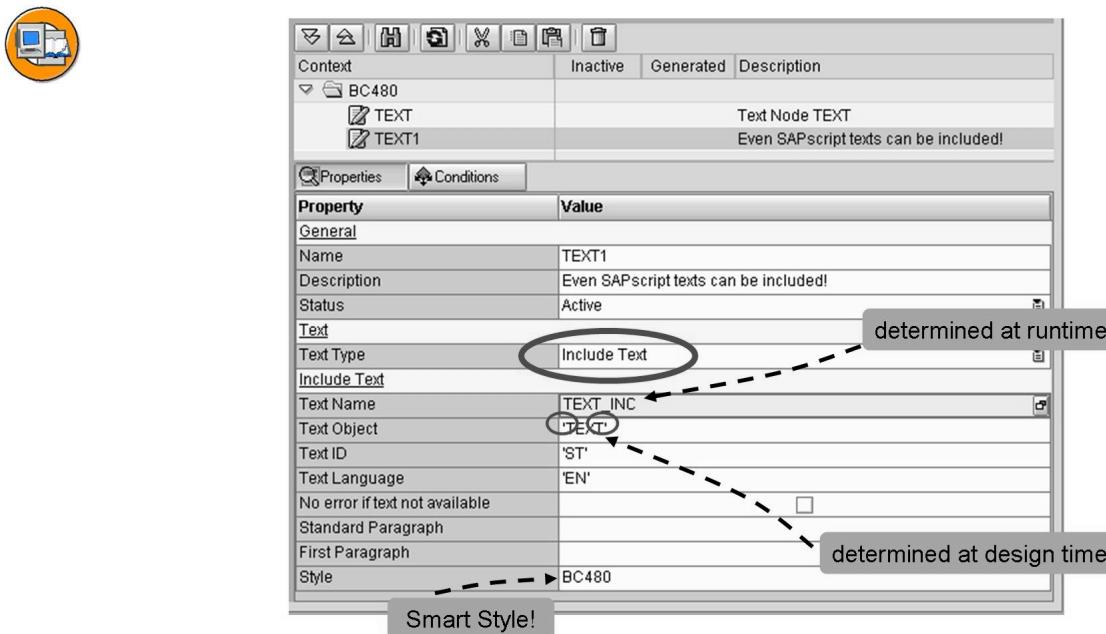


Figure 28: Including SAPscript Texts

If you want to integrate SAPscript text (also known as include text) into the context, right-click somewhere in the context tree and choose *Create → Text*. Then set the *Text Type* to *Include Text*.

You need to specify the text name, the text object, the text ID, and the text language. All of these can be entered as literals (enclosed in single quotes) or as fields (which must be known from the interface). If you do not enter a language, the form's language will be taken.

If you select *No error if text not available*, the generated function module will not terminate if the text specified is not found at runtime.

Note that all SAPscript commands in SAPscript texts will be ignored. In particular, this means that nesting SAPscript texts is not possible.

As to the text formatting, SAPscript styles assigned to the text will be ignored.

All paragraph formats and character formats of an include text are interpreted as they are defined in the style (Smart Style) that you can enter in the *Style* field. For example, the paragraph format B in a SAPscript text might mean bold, but in the Smart Style it could mean small. Each paragraph with format B in the text would therefore be printed in small text. If a format is used that is not defined in the style that you enter in the context, it will be ignored.

You can override the paragraph formatting of include texts:

In the *Standard Paragraph* field, you can select a paragraph format of the Smart Style that you enter in the *Style* field. This format is then used for all paragraphs of the SAPscript text that are formatted using the standard paragraph (*). If you are familiar with SAPscript commands: an entry in this field corresponds to the PARAGRAPH addition of the INCLUDE command.

You use the *First Paragraph* attribute to set a paragraph format for the first paragraph of the include text - independently of how the paragraph is actually formatted in the include text (corresponds to the NEW PARAGRAPH addition of the SAPscript command INCLUDE). If the *Standard Paragraph* field remains empty, all standard paragraphs in the include text also adopt this paragraph format.

Now that you know how to include Smart Forms text modules or SAPscript texts, let's summarize the main differences. If you have a choice, you should normally opt for Smart Forms texts.



Smart Forms text module	SAPscript text
Client independent	Client dependent
Assigned to package → standard transport	w/o package → text must be assigned to transport request manually (report RSTXTRAN)
Default paragraph and first paragraph can <u>not</u> be reformatted	Default paragraph and first paragraph can be reformatted
	Old SAPscript commands might tempt developers to use them (in vain)
Styles assigned to text can be used or overridden	Styles assigned to or within text will be ignored

Figure 29: Text Modules vs. SAPscript Texts

Dynamic Texts

Along with text modules and SAPscript texts, you can integrate a third variant of long texts into the context: dynamic texts. In this case, you must specify an internal table with line type *tline* that will contain the text in ITF format at runtime. *tline* consists of two columns: column *tdformat* contains the name of the paragraph format, and column *tdline* contains the content of the line. (In other words, a dynamic text looks exactly like a text module or a SAPscript text in the line editor.)

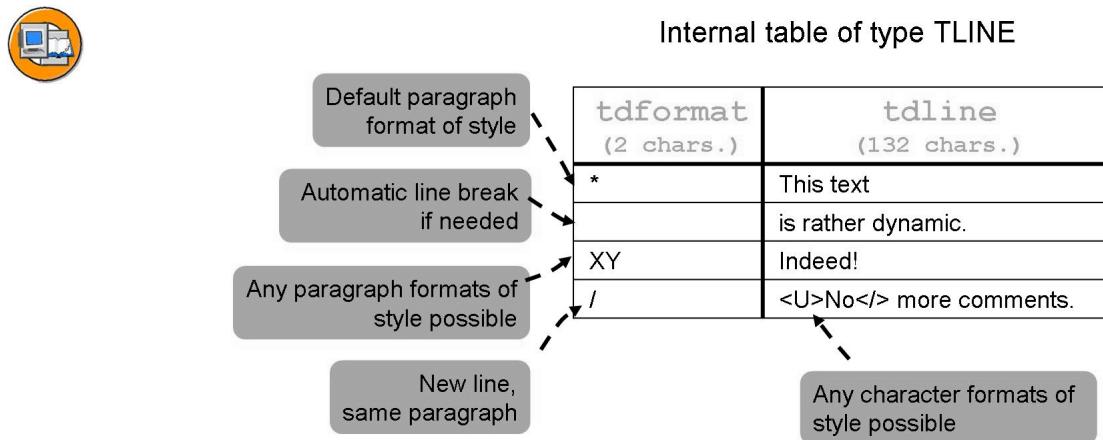


Figure 30: Dynamic Texts

If you enter an asterisk (*) as the paragraph, the default paragraph will be taken from the style (Smart Style) that you specify in the context.

If you want to have continuous text, make sure to enter a paragraph format only for the first line. You can then add as many lines to the internal table as you like, using only the *tdline* column. *tdline* can contain up to 132 characters. Independent of how many characters you enter, the resulting text in the form will have one paragraph.

If you want to apply character formats to some parts of the text, look up the name of a suitable format in the style to be used (it will always have a one- or two-digit name). Put this name in angle brackets, then write the text that needs to be formatted, and close the formatting with </>. For example, if a style contains the character format AB, you can format parts of a text as follows: irrelevant <AB>formatted</> irrelevant .



Context		Inactive	Generated	Description
BC480				
CUSTOMER_ADDRESS		Customer address		
Properties		Conditions		
Property		Value		
General				
Name	CUSTOMER_ADDRESS			
Description	Customer address			
Status	Active			
Text				
Text Type	Dynamic Text			
Dynamic Text				
Field	GT_ADDRESS			
Text Language				
Style	BC480			

Smart Style

Internal table of type
TLINE

Figure 31: Including Dynamic Texts

If you create a dynamic text in a form context, make sure to give the name of the internal table (which has line type *tline*) in the *Field* field. Do **not** include the internal table as an independent context node. It should be referred to only in the dynamic text.

The style that you specify for a dynamic text must be a Smart style – that is, a style that can also be used for text modules.

Addresses Without Business Address Services

If your application makes no use of the Business Address Services but you still want to have country-specific addresses, you must use ABAP coding to achieve this. It is possible to do this in the application program or in the initialization coding of the interface. The example here describes the latter option.

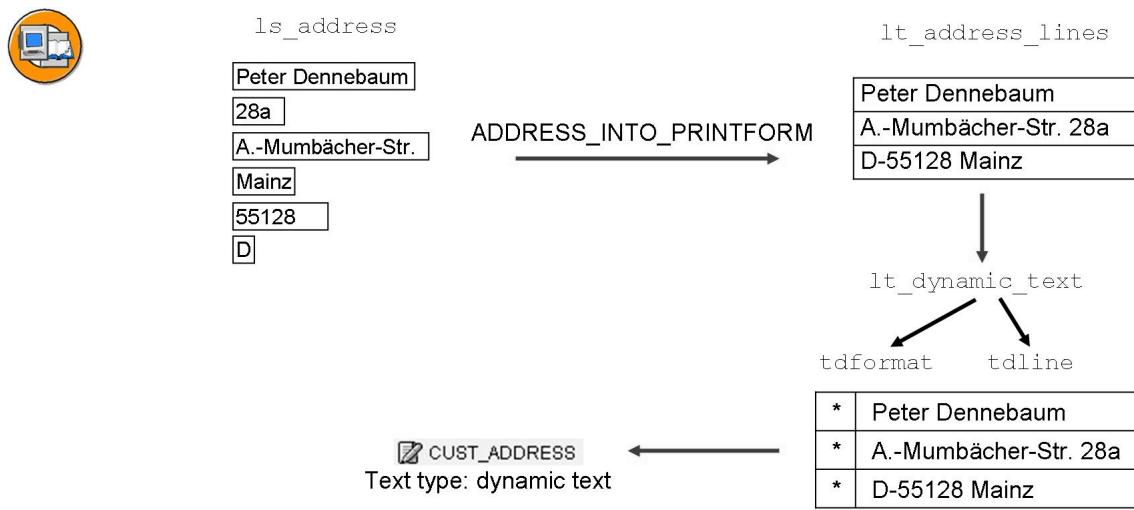


Figure 32: Addresses Without Business Address Services

You need to call the function module **ADDRESS_INTO_PRINTFORM**. Among its parameters, you will find *address_1*. This structure contains all relevant address fields, such as name, street, city, or country. These fields must be filled with the address data from your application. In other words, you must take care to create a correct mapping of the address fields from your application and the individual address fields of structure *address_1*.

You should determine the value of parameter *number_of_lines*. It equals the maximum number of lines that will be created from the address data. Function module **ADDRESS_INTO_PRINTFORM** returns an internal table, *address_printform_table*, which has only one column. Every line contains one line of the address that has been assembled according to the addressee's country. These lines need to be converted into a two-column internal table of type *tline* that can be used as the source of a dynamic text in a PDF form. Its two columns are:

- **TDFORMAT**, which contains the paragraph format of the line (for example, an asterisk for the default format)
- **TDLINE**, which contains the text itself



Addresses Without Business Address Services - Coding I

```

TYPE-POOLS: szadr.
DATA:
  ls_address      TYPE adrs1,
  lt_address_lines TYPE szadr_printform_table,
  ls_address_line  LIKE LINE OF lt_address_lines.

*map address fields from work area to fields from function module
ls_address-title_text = is_customer-form.

```

```

ls_address-name1      = is_customer-name.
ls_address-street     = is_customer-street.
ls_address-post_code1 = is_customer-postcode.
ls_address-city1      = is_customer-city.
ls_address-country    = is_customer-country.

CALL FUNCTION 'ADDRESS_INTO_PRINTFORM'
EXPORTING
  address_1            = ls_address
  address_type          = '1'      "normal/company
  sender_country        = iv_sending_country
  number_of_lines       = 6
IMPORTING
  address_printform_table = lt_address_lines.

```



Addresses Without Business Address Services - Coding II

```

DATA: ls_dynamic_text TYPE tline.
* LT_DYNAMIC_TEXT would be defined as a global field
* of the interface as follows:
* lt_dynamic_text TYPE TABLE OF tline

LOOP AT lt_address_lines
  INTO ls_address_line.
  ls_dynamic_text-tdformat = '*'.
  ls_dynamic_text-tdline   = ls_address_line-address_line.
  APPEND ls_dynamic_text TO lt_dynamic_text.
ENDLOOP.

```

Exercise 2: Form Context: Using and Extending a Form Interface

Exercise Objectives

After completing this exercise, you will be able to:

- Create a form context, using an existing interface
- Integrate alternatives and graphics into a form context
- Create and integrate long texts into a form context

Business Example

A travel agency wants to print invoices for their customers' flight bookings using the tool "Interactive Forms". The interface has been designed already, but there is no form yet. Designing the form context is the next step.

Task 1:

Create a form with a context.

Please stick closely to the exercise description, because at the end of the exercise, a program will add a layout to your form context and show the resulting PDF. (You will use this context for later exercises.)

1. Create a PDF-based form, ZBC480_##, which uses your interface, ZBC480_##, from the last exercise. ## stands for your two-digit group number (monitor number).

If you have not finished the previous exercise, you can use the solution interface BC480 instead.

Save the form in package ZBC480_##.

The reference solution for this exercise is form BC480_CONTEXT_ONLY.

Task 2:

Integrate interface fields into the context.

1. From the interface, drag structure *IS_CUSTOMER* into your context.
Deactivate the fields *MANDT*, *TELEPHONE*, *CUSTTYPE*, *LANGU*, *EMAIL*, and *WEBUSER*.
2. Integrate the internal table *IT_BOOKINGS* into your context.
Deactivate all fields except for *CARRID*, *CONNID*, *FDATE*, *BOOKID*, *FORCURAM*, and *FORCURKEY*.

Continued on next page

3. Integrate the constant *GC_CLERK* into your context.
4. Integrate the system date into your context.

Task 3:

Integrate graphics into the context.

1. Create a graphic node, *LOGO*, of type *graphic reference*.
2. Set the URL to the *IV_IMAGE_URL* field.
3. Set a condition for *LOGO* so that it will be processed only if its URL is not <http://www.sap.com/nonsense.bmp>.

Task 4:

Integrate an alternative into the context.

1. Create an alternative *OFFER*.
2. Set the alternative conditions so that the *TRUE* node will be processed if the customer is a business customer (*IS_CUSTOMER-CUSTTYPE* = 'B').
For all other customers, the *FALSE* node will automatically be processed.
(You will have to fill the *TRUE* and *FALSE* nodes soon.)
3. Save your form before you continue with the next task.

Task 5:

Create a text module.

1. Create a text module *ZBC480_BUSINESS##*. Save it in package *ZBC480_##*.
stands for your two-digit group number (monitor number).
2. Choose the style BC480 for the text module.
3. Suggest some destinations where your business customers could spend their holidays.
4. Choose the default paragraph format (*) for all lines of your text.
5. Save the text module. Go back to your form and integrate this text module into the *TRUE* node of the *OFFER* alternative.
6. Rename the text node of the context *TEXT_BUSINESS*.
7. The text node of the context should use the style assigned to the text module.

Continued on next page

Task 6:

Create a second text module.

1. Create a second text module, ZBC480_OTHER##. Save it in package ZBC480_##.
stands for your two-digit group number (monitor number).
2. Choose the style SAPADRS for the text module.
3. Type in some advertisement lines for those customers that are non business customers.
4. Choose the default paragraph format (*) for all lines of your text.
5. Save the text module. Go back to your form and integrate this text module into the *FALSE* node of the *OFFER* alternative.
6. Rename the text node of the context *TEXT*.
7. The text node of the context should use the style assigned to the text module.

Task 7:

Integrate a SAPscript text into the form context.

1. In your form context, create a text node *ENVELOPE_ADDRESS*. (It will contain the travel agency's address in small print that is visible in the window of the envelope.) Choose type *Include text* (that is, SAPscript text) and refer to include text *FLY_AND_SMILE_SMALL*, object *TEXT*, ID *ADRS*.

Assign style SAPADRS.

Task 8:

Integrate a dynamic text into the form context.

1. In your form context, create a text node *CUSTOMER_ADDRESS*. Choose type *Dynamic text* and refer to the interface variable *GT_ADDRESS*. (Note: *GT_ADDRESS* is filled with the formatted customer address in the initialization coding of the interface.)

Assign style BC480.

Task 9:

Test the form context.

1. Check the syntax of your form. If errors are found, correct them. Ignore the error message that your form has no layout yet. You will design the layout soon.

Continued on next page

2. Save your form and leave the maintenance mode.
3. Execute report SAPBC480_ADD_LAYOUT. It will try to add a test layout to your form, activate the form, and fill it with test data.

When you run the report, you should get two PDF documents: one for a German business customer, and one for a British private customer. Check that you see different texts.

On the selection screen, you can enter the country that will be taken as the sending country. Run the report once with 'DE' and a second time with 'GB'. The customer's country in the address field should appear in one case (for international mail), but not in the other.

The layout elements will have a green color if the corresponding context elements are OK, red if not. In case you get errors, go back to the context and try to fix the error. In particular, make sure you name your context elements exactly as described in the tasks.

Task 10:

Optional: Create a SAPscript text.

1. With transaction SO10, create a SAPscript text: ZBC480_ENVELOPE##, ID ADRS. (The invisible key field OBJECT will automatically be filled with the value 'TEXT'.)
2. Use style SAPADRS.
3. Type in the address line for your travel agency in paragraph format SD.
4. In the context node *ENVELOPE_ADDRESS*, use your newly created SAPscript text instead of FLY_AND_SMILE_SMALL.
5. Test your form again as described before, using report SAPBC480_ADD_LAYOUT.

Solution 2: Form Context: Using and Extending a Form Interface

Task 1:

Create a form with a context.

Please stick closely to the exercise description, because at the end of the exercise, a program will add a layout to your form context and show the resulting PDF. (You will use this context for later exercises.)

1. Create a PDF-based form, ZBC480_##, which uses your interface, ZBC480_##, from the last exercise. ## stands for your two-digit group number (monitor number).

If you have not finished the previous exercise, you can use the solution interface BC480 instead.

Save the form in package ZBC480_##.

The reference solution for this exercise is form BC480_CONTEXT_ONLY.

- a) Start the Form Builder (transaction SFP). Select *Form*.
- b) Enter the name **ZBC480_##** and choose *Create*.
- c) In the following window, enter a description and the name of the interface you want to use (**ZBC480_##**). Press Enter.
- d) In the following window, enter **ZBC480_##** as the package and press Enter.
- e) In the next window, press F4 to select the workbench request the instructor has created for you. Continue by pressing Enter.

Task 2:

Integrate interface fields into the context.

1. From the interface, drag structure *IS_CUSTOMER* into your context.
Deactivate the fields *MANDT*, *TELEPHONE*, *CUSTTYPE*, *LANGU*, *EMAIL*, and *WEBUSER*.
 - a) Switch to the *Context* tab page. On the left side of the Form Builder, open the *Import* folder. Select *IS_CUSTOMER* and drag it over to the right side. Drop it on folder *ZBC480_##*.
 - b) Keep the CTRL key pressed down. On the right side of the Form Builder, select the fields *MANDT*, *TELEPHONE*, *CUSTTYPE*, *LANGU*, *EMAIL*, and *WEBUSER*. Right-click on any of the fields and select *Deactivate*.

Continued on next page

2. Integrate the internal table *IT_BOOKINGS* into your context.
Deactivate all fields except for *CARRID*, *CONNID*, *FLDATE*, *BOOKID*, *FORCURAM*, and *FORCURKEY*.
 - a) See previous step. Make sure not to drop *IT_BOOKINGS* on *IS_CUSTOMER*. Drop it on folder *ZBC480_##*.
3. Integrate the constant *GC_CLERK* into your context.
 - a) Switch to the *Context* tab page. On the left side of the Form Builder, open the *Global Data* folder.
 - b) Select *GC_CLERK* and drag it over to the right side. Drop it on folder *ZBC480_##*.
4. Integrate the system date into your context.
 - a) Switch to the *Context* tab page. On the left side of the Form Builder, open the folder *System Fields*. Then open the structure *SFPSY*.
 - b) Select *DATE* and drag it over to the right side. Drop it on folder *ZBC480_##*.

Task 3:

Integrate graphics into the context.

1. Create a graphic node, *LOGO*, of type *graphic reference*.
 - a) Switch to the *Context* tab page. On the right side of the Form Builder, choose folder *ZBC480_##*. Choose *Create → Graphic*. This will automatically create a graphic of type *graphic reference*.
 - b) Rename the graphic node. In the bottom right area of the Form Builder, enter **LOGO** in the *Name* field.
2. Set the URL to the *IV_IMAGE_URL* field.
 - a) In the bottom right area of the Form Builder, choose the *URL* button.
 - b) In the *Graphic URL* column, enter **IV_IMAGE_URL**. Leave the *Delimiter* column empty.

Continued on next page

3. Set a condition for *LOGO* so that it will be processed only if its URL is not <http://www.sap.com/nonsense.bmp>.
 - a) In the bottom right area of the Form Builder, choose the *Conditions* button. Leave the *Operator* column empty (it is required only if you have several conditions).
 - b) In the *Operand* column, enter **IV_IMAGE_URL**.
 - c) Set the *Relational Operator* field to **≠**. Set the *Operand* field to '<http://www.sap.com/nonsense.bmp>' (with single quotes).

Task 4:

Integrate an alternative into the context.

1. Create an alternative *OFFER*.
 - a) Switch to the *Context* tab page. On the right side of the Form Builder, choose folder **ZBC480_##**. Choose *Create → Alternative*. This will create an alternative node with two subnodes: *TRUE* and *FALSE*.
 - b) Rename the alternative. In the bottom right area of the Form Builder, enter **OFFER** in the *Name* field.
2. Set the alternative conditions so that the *TRUE* node will be processed if the customer is a business customer (**IS_CUSTOMER-CUSTTYPE = 'B'**).
For all other customers, the *FALSE* node will automatically be processed.
(You will have to fill the *TRUE* and *FALSE* nodes soon.)
 - a) In the bottom right area of the Form Builder, choose the *Alternative Conditions* button. Add one row.
 - b) In the row, leave the *Operator* column empty. Enter **IS_CUSTOMER-CUSTTYPE** in the *Operand* column, **=** in the *Relational Operator* column, and '**'B'**' (with single quotes) in the *Operand* column.
3. Save your form before you continue with the next task.
 - a) Press **CTRL+S**.

Task 5:

Create a text module.

1. Create a text module **ZBC480_BUSINESS##**. Save it in package **ZBC480_##**.

Continued on next page

- ## stands for your two-digit group number (monitor number).
- a) Start transaction SMARTFORMS. Select *Text Module*.
 - b) Enter **ZBC480_BUSINESS##**. Choose the *Create* button. Press CTRL+S to save the text module.
 - c) In the following window, enter **ZBC480_##** as the package and press Enter.
 - d) In the next window, press F4 to select the workbench request the instructor has created for you. Continue by pressing Enter.
2. Choose the style BC480 for the text module.
 - a) Go to the *Management* tab. Enter **BC480** as the style name. Type in some advertisement lines.
 3. Suggest some destinations where your business customers could spend their holidays.
 - a) Go to the *Text* tab to type in text.
 4. Choose the default paragraph format (*) for all lines of your text.
 - a) In the text editor, mark all lines by pressing CTRL+A. Choose * from the *Paragraph Formats* drop-down list.
 5. Save the text module. Go back to your form and integrate this text module into the *TRUE* node of the *OFFER* alternative.
 - a) Press CTRL+S.
 - b) On the right side of the Form Builder, right-click the folder *TRUE* and then choose *Create → Text*.
 - c) Locate the *Text Name* field in the bottom right area of the Form Builder and enter '**ZBC480_BUSINESS##**' (in single quotes).
 6. Rename the text node of the context *TEXT_BUSINESS*.
 - a) Locate the *Name* field in the bottom right area of the Form Builder and enter '**TEXT_BUSINESS**'.
 7. The text node of the context should use the style assigned to the text module.
 - a) Locate the *Copy Style from Text Module* field and select its box.

Task 6:

Create a second text module.

1. Create a second text module, **ZBC480_OTHER##**. Save it in package **ZBC480_##**.

Continued on next page

stands for your two-digit group number (monitor number).

- a) See solution of first text module.
2. Choose the style SAPADRS for the text module.
 - a) See solution of first text module.
3. Type in some advertisement lines for those customers that are non business customers.
 - a) See solution of first text module.
4. Choose the default paragraph format (*) for all lines of your text.
 - a) See solution of first text module.
5. Save the text module. Go back to your form and integrate this text module into the *FALSE* node of the *OFFER* alternative.
 - a) Press CTRL+S. On the right side of the Form Builder, right-click the folder *FALSE* and then choose *Create → Text*.
 - b) Locate the *Text name* field in the bottom right area of the Form Builder and enter '**'ZBC480_OTHER##'** (in single quotes).
6. Rename the text node of the context *TEXT*.
 - a) See solution of first text module.
7. The text node of the context should use the style assigned to the text module.
 - a) See solution of first text module.

Task 7:

Integrate a SAPscript text into the form context.

1. In your form context, create a text node *ENVELOPE_ADDRESS*. (It will contain the travel agency's address in small print that is visible in the window of the envelope.) Choose type *Include text* (that is, SAPscript text) and refer to include text *FLY_AND_SMILE_SMALL*, object *TEXT*, ID *ADRS*.

Continued on next page

Assign style SAPADRS.

- a) On the right side of the Form Builder, right-click the folder *ZBC480_##* and then choose *Create → Text*.
- b) Locate the *Text type* field in the bottom right area of the Form Builder and select *Include text*. Enter '**ENVELOPE_ADDRESS**' (with single quotes).
- c) In the *Text name* field, enter '**FLY_AND_SMILE_SMALL**'. In the *Text object* field, enter '**TEXT**'. In the *Text ID* field, enter '**ADRS**'.
- d) In the *Style* field, enter **SAPADRS**

Task 8:

Integrate a dynamic text into the form context.

1. In your form context, create a text node *CUSTOMER_ADDRESS*. Choose type *Dynamic text* and refer to the interface variable *GT_ADDRESS*. (Note: *GT_ADDRESS* is filled with the formatted customer address in the initialization coding of the interface.)

Assign style BC480.

- a) On the right side of the Form Builder, right-click the folder *ZBC480_##* and then choose *Create → Text*.
- b) Locate the *Text type* field in the bottom right area of the Form Builder and select *Dynamic text*. Enter '**CUSTOMER_ADDRESS**' (with single quotes). In the *Field* field, enter **GT_ADDRESS**.
- c) In the *Style* field, enter **BC480**.

Task 9:

Test the form context.

1. Check the syntax of your form. If errors are found, correct them.
Ignore the error message that your form has no layout yet. You will design the layout soon.
 - a) Press CTRL+F2.
2. Save your form and leave the maintenance mode.
 - a) Press CTRL+S to save the form, then CTRL+F1 to go to display mode.
3. Execute report *SAPBC480_ADD_LAYOUT*. It will try to add a test layout to your form, activate the form, and fill it with test data.

Continued on next page

When you run the report, you should get two PDF documents: one for a German business customer, and one for a British private customer. Check that you see different texts.

On the selection screen, you can enter the country that will be taken as the sending country. Run the report once with 'DE' and a second time with 'GB'. The customer's country in the address field should appear in one case (for international mail), but not in the other.

The layout elements will have a green color if the corresponding context elements are OK, red if not. In case you get errors, go back to the context and try to fix the error. In particular, make sure you name your context elements exactly as described in the tasks.

- a) Call transaction SA38. Enter **SAPBC480_ADD_LAYOUT** and press F8.

Task 10:

Optional: Create a SAPscript text.

1. With transaction SO10, create a SAPscript text: ZBC480_ENVELOPE##, ID ADRS. (The invisible key field OBJECT will automatically be filled with the value 'TEXT').
 - a) Start transaction SO10 and type in the text name, and its ID.
2. Use style SAPADRS.
 - a) Choose *Format → Change Style*.
3. Type in the address line for your travel agency in paragraph format SD.
 - a) Type in the text. Choose * from the *Paragraph Formats* drop-down list.
4. In the context node *ENVELOPE_ADDRESS*, use your newly created SAPscript text instead of *FLY_AND_SMILE_SMALL*.
 - a) On the right side of the Form Builder, select the text node *ENVELOPE_ADDRESS* by double-clicking it.
 - b) Locate the *Text* field in the bottom right area of the Form Builder and enter '**ZBC480_ENVELOPE##**'.
5. Test your form again as described before, using report SAPBC480_ADD_LAYOUT.
 - a) See first test.



Lesson Summary

You should now be able to:

- Effectively use transaction SFP to create a form context
- Create a form context using an existing interface
- Integrate folders, alternatives, graphics, and addresses into a form context
- Create and integrate long texts into a form context



Unit Summary

You should now be able to:

- Effectively use transaction SFP to create a form context
- Create a form context using an existing interface
- Integrate folders, alternatives, graphics, and addresses into a form context
- Create and integrate long texts into a form context



Test Your Knowledge

1. A context contains all fields of an interface.
Determine whether this statement is true or false.
 True
 False

2. A context must use exactly one interface.
Determine whether this statement is true or false.
 True
 False

3. Deactivating fields makes sense for testing only. Active forms should have active context fields only.
Determine whether this statement is true or false.
 True
 False

4. Addresses from Business Address Services are automatically formatted when included in a PDF-based form.
Determine whether this statement is true or false.
 True
 False

5. URLs of images are static.
Determine whether this statement is true or false.
 True
 False

6. Text modules can be created with transaction SFP.
Determine whether this statement is true or false.
 True
 False

7. The style used in a text module can be overridden when this text module is integrated into a PDF-based form.
Determine whether this statement is true or false.
 True
 False



Answers

1. A context contains all fields of an interface.

Answer: False

It is up to you to decide which of the interface fields should be integrated into the context.

2. A context must use exactly one interface.

Answer: True

You determine which interface to use on the *Properties* tab.

3. Deactivating fields makes sense for testing only. Active forms should have active context fields only.

Answer: False

Deactivating fields reduces the XML data stream that is sent to Adobe document services. Thus, performance can be improved.

4. Addresses from Business Address Services are automatically formatted when included in a PDF-based form.

Answer: True

The responsible function module that is automatically called is `ADDRESS_INTO_PRINTFORM`.

5. URLs of images are static.

Answer: False

You can dynamically determine a URL, or parts of it, by using fields of type `STRING`.

6. Text modules can be created with transaction SFP.

Answer: False

Text modules are created with transaction SMARTFORMS. They can be integrated into PDF-based forms using transaction SFP.

7. The style used in a text module can be overridden when this text module is integrated into a PDF-based form.

Answer: True

Deselect *Copy Style from Text Module* and set a different style.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Unit 4

Designer

Unit Overview

Adobe LiveCycle Designer is a powerful tool that lets you define both layout and logic of a form. This unit will explain the key functionalities of the tool itself, without yet going into details of a form design.



Unit Objectives

After completing this unit, you will be able to:

- List the various functions of Adobe LiveCycle Designer
- Use Adobe LiveCycle Designer as a graphical tool for designing forms
- Create a simple form layout with various objects
- Distinguish between master pages, content areas, pages (design view), and subforms
- Describe typical contents of master pages, content areas, pages (design view), and subforms

Unit Contents

Lesson: Adobe LiveCycle Designer: Overview	80
Exercise 3: Adobe LiveCycle Designer: Overview.....	99
Lesson: Adobe LiveCycle Designer: Structuring a Form	108
Exercise 4: Adobe LiveCycle Designer: Structuring a Form.....	115

Lesson: Adobe LiveCycle Designer: Overview

Lesson Overview

Once you have created the context of a form, you can continue to create the layout. Adobe LiveCycle Designer is the tool that supports you here. It is fully integrated into transaction SFP and lets you create and edit pages and their contents (like text or images). This lesson gives a general overview how Adobe LiveCycle Designer works.



Lesson Objectives

After completing this lesson, you will be able to:

- List the various functions of Adobe LiveCycle Designer
- Use Adobe LiveCycle Designer as a graphical tool for designing forms
- Create a simple form layout with various objects

Business Example

A travel agency wants to print invoices for their customers' flight bookings using the tool "Interactive Forms". The form designers first need to get an overview of how the graphical tool (Adobe LiveCycle Designer) works.

Designer: Basics

If you want to work with Adobe LiveCycle Designer, the following software must be installed on your PC:

- SAP GUI for Windows. Make sure that during installation Adobe LiveCycle Designer gets installed. You can check in the Windows directory C:\Program Files\Adobe\Designer 8.0 (for SAP NetWeaver 7.0).
- Adobe Reader. The most current version should always be used, in particular for interactive features. Check SAP Note 834573 for details.
- If you want to use interactive features, you will need – depending on the layout type of your forms, the release of SAP NetWeaver and its Support Packages – an additional component installed on your PC, the Active Component Framework (ACF). See SAP Note 766191 for details. If your forms are of layout type ZCI (Zero Client Installation), no ACF installation is required.
- Microsoft Windows 2000 or higher



SAP GUI for Windows



- Designer must be installed on your hard drive.
Default location for Windows:
C:\Program Files\Adobe\Designer 8.0
(depending on version)



Adobe Reader

- used to be called Acrobat Reader



Microsoft Windows

Figure 33: Designer: Technical Prerequisites

Make sure that you do not start to work on a form with a recent version of Adobe LiveCycle Designer and then continue with the same form, but an older version of Designer.

Important: Loss of Changes



- Transaction SFP has display mode, but Designer does not. Consequently, you can make changes in the layout – but these changes will be lost when you leave the layout area.
- The following note is displayed only once in the status line:
“Changes to the layout cannot be saved in display mode!”

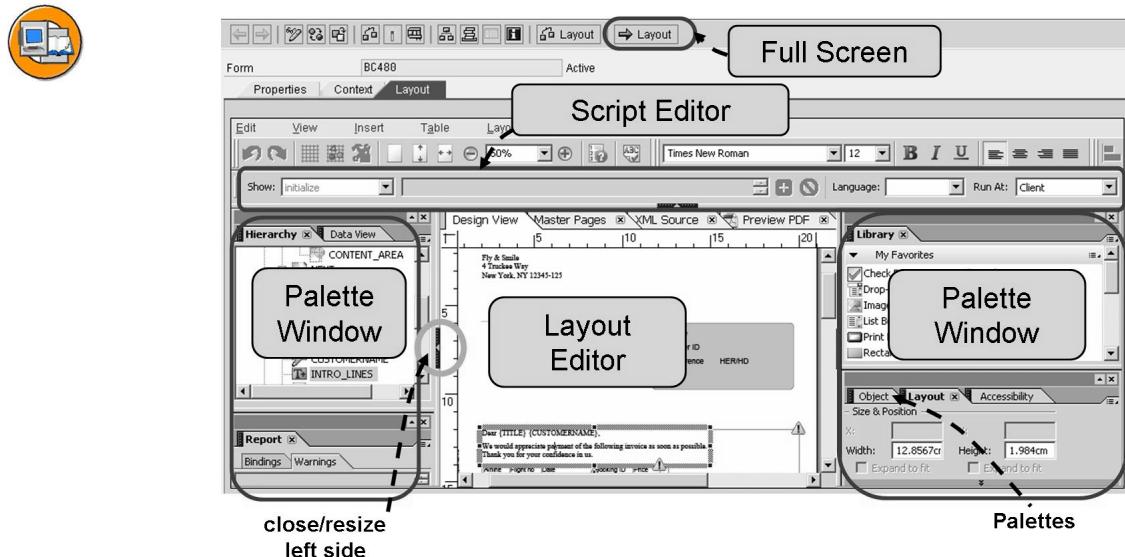


Figure 34: Designer: Overview

After choosing the *Layout* tab page, Designer (in which you determine the graphical layout) is called up. In SAPNetWeaver 2004s, the *Layout* button was added to transaction SFP and displays Designer in a full screen.

The Designer workspace consists of four main areas. All but the central one (the Layout Editor) can be closed by choosing *Palettes* → *Workspace* (*Palettes* → *Manage Palettes* in some versions).

In the top area, the Script Editor can be displayed. It allows you to enter scripts for calculations. You can choose between JavaScript and Adobe's FormCalc.

The subdivisions of the left and right areas are called **palette windows** with further subdivisions of **palettes**. It is up to you to decide which palette windows you want to display in which size. To return to the standard, choose *Palettes* → *Workspace* → *Reset Palette Locations*.

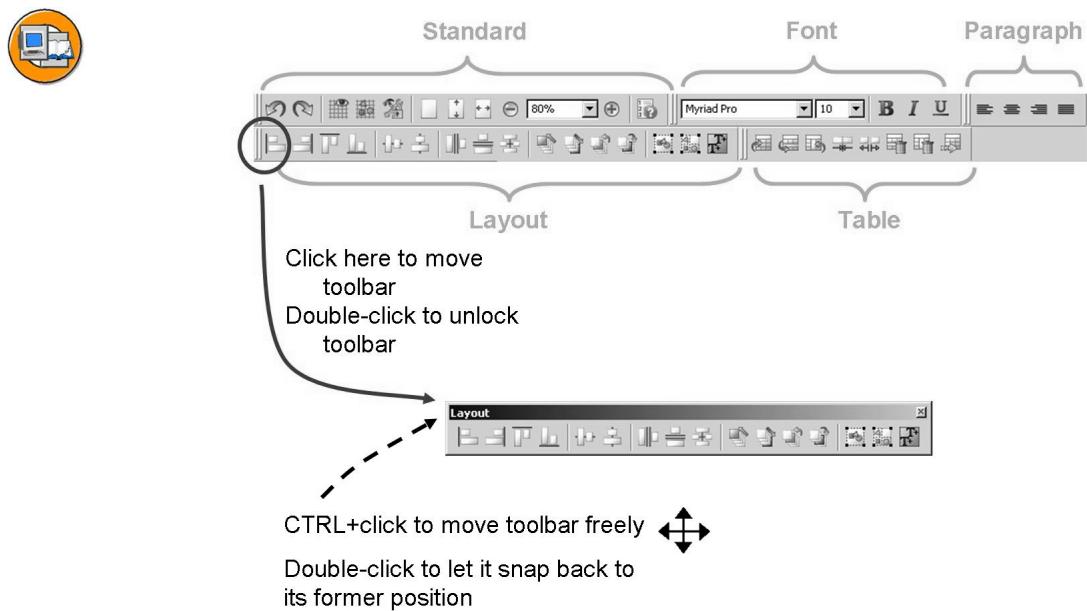


Figure 35: Designer: Toolbars

Designer has one menu toolbar and five button toolbars (Standard, Font, Paragraph, Layout and - since Adobe Live Cycle Designer 7.1 - Table).

You can move a toolbar by dragging its title bar. If you drop it, it will be docked to another toolbar, the *Layout Editor* window, or to a palette window. Holding CTRL while dragging a toolbar lets you freely move the toolbar anywhere in the workspace.

By choosing *Tools* → *Customize*, you can define which toolbars are displayed or hidden. Here, it is also possible to move individual items to different positions or to reset toolbars to the standard settings.

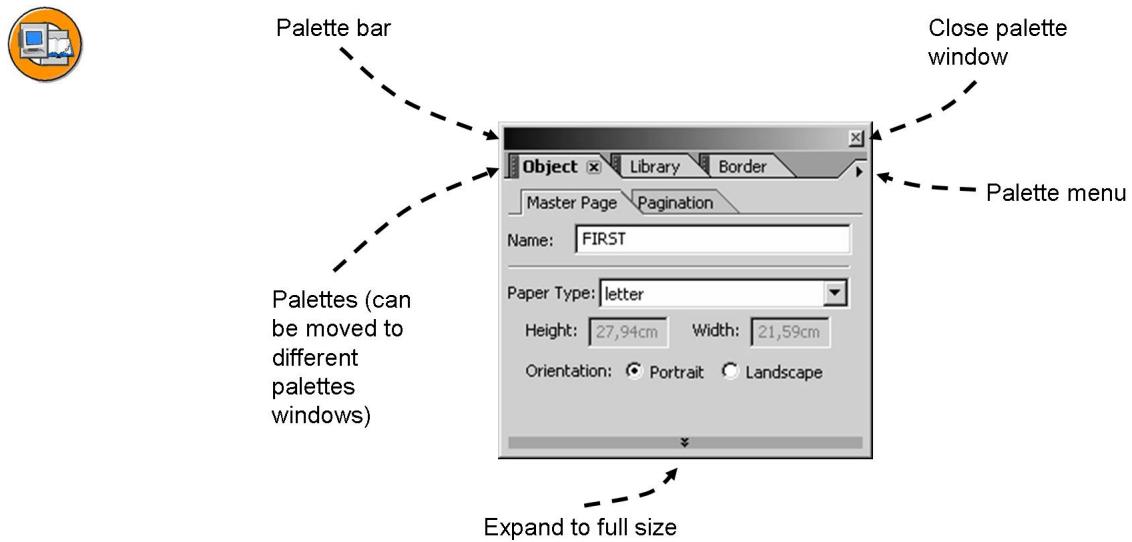


Figure 36: Palette Windows: Handling

Palettes can be closed and opened via the *Palettes* menu.

If you want to resize a palette window, drag any side of it.

You move a palette window by dragging its palette bar. It will be docked to the Layout Editor window or to another palette window. Holding CTRL while dragging a palette lets you freely move the palette anywhere in the workspace.

Double-clicking on the palette bar switches between the palette window's docked and its free position.

The palettes of any palette window can be dragged and dropped to another palette window or to a free position, or can be rearranged in a different order in the same palette window.

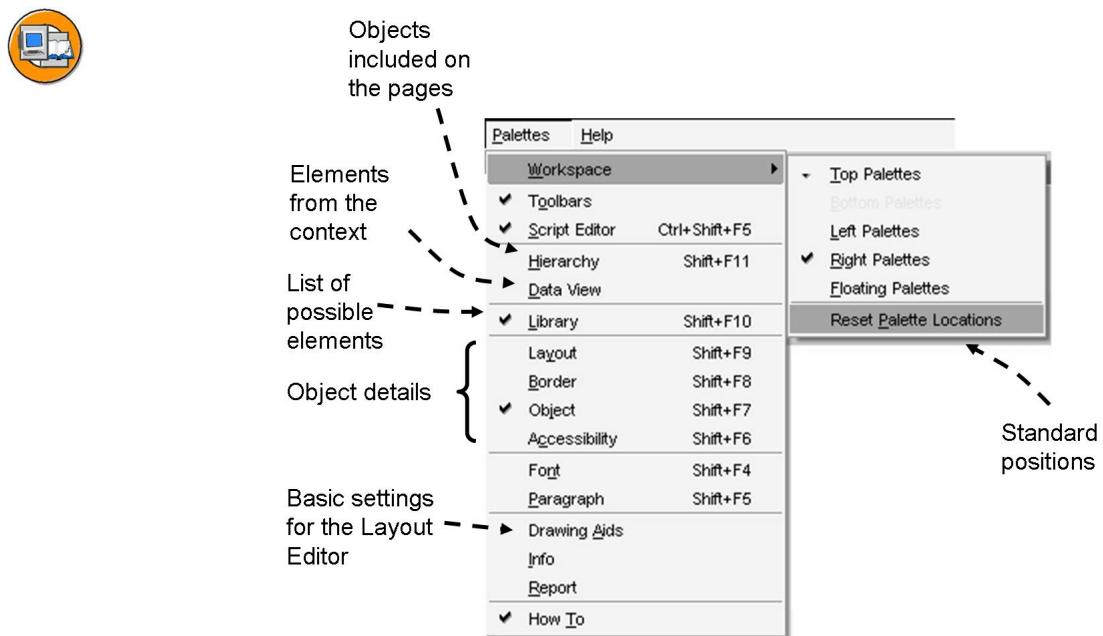


Figure 37: Palettes: Overview

The *Hierarchy* palette shows in a tree everything that has been included in the layout.

All active context nodes are displayed on the *Data View* palette. As the context is SAP-specific, you should be aware that Adobe documentation refers to it as data schema. Even though you can change the data schema directly, you should not (this option makes sense only if you work with Designer without SAP integration). Always change the form's context if you want to change the data source.

The *Library* palette lists all objects that you can include (by dragging and dropping) on the form pages. It can be customized to your needs.

On the *Font* and *Paragraph* palettes, you can make adjustments for, static texts, text fields, or their captions.

If you mark a field on a page that originates from the context, the *Info* palette displays the information entered in the context.

In the *Report* palette, you find information on fields in the layout and their mapping to context fields. Furthermore, warnings will be displayed here when the form is previewed locally, that is, with Designer's built-in preview.

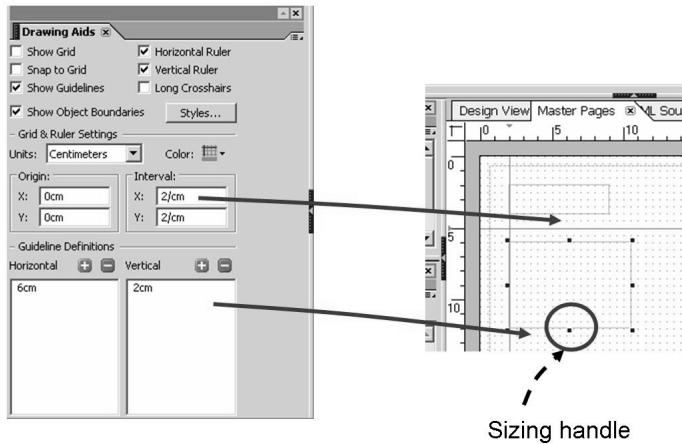


Figure 38: Drawing Aids: Settings for the Layout Editor

Before you start with a new form, determine the basic settings for the Layout Editor by choosing *Palettes* → *Drawing Aids*.

If you mark an object in the Layout Editor, its corners will be highlighted in blue. It is helpful to choose different colors for object boundaries and the grid.

What you set as the unit for grid and ruler will also be taken for the positioning and sizing of objects. If you change the unit, all existing measurements will be changed automatically.

Whatever you enter on the *Drawing Aids* palette will be saved in your form.

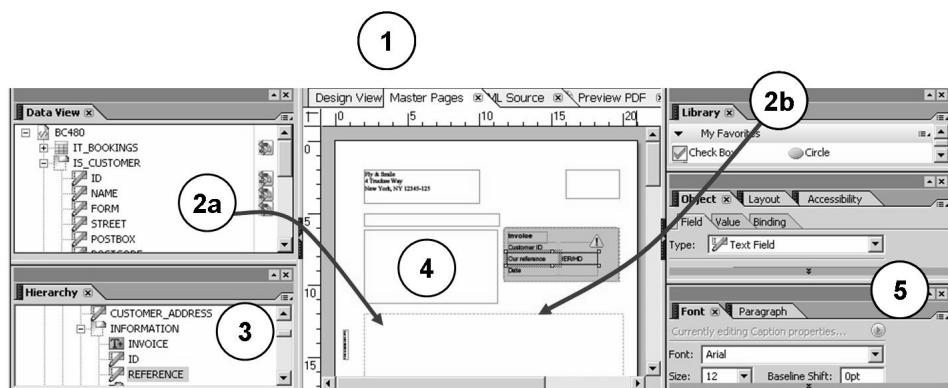


Figure 39: The Layout Editor: Including Objects

If you want to add an object to a page, the general steps are:

1. In the Layout Editor, choose the relevant page.
2. (a) If you want to include an object from the context, go to the *Data View* palette, drag the object to the page, and drop it at the desired position.
- (b) If you want to include other objects (like static texts or graphics), go to the *Library* palette, drag the object to the page, and drop it at the desired position. If you right-click on an object in the library, you can select *Keep drawing tool*. Once this is selected, when you drag and drop any of the objects from the *Library* tab page to the layout area, any subsequent click into the layout area will automatically create a new object of the same type.
3. Give the object a reasonable name. Select the *Hierarchy* tab, right-click on the newly inserted object, and choose *Rename*.
4. Resize the object if needed. You can do this on the *Layout* tab or in the Layout Editor by dragging the resizing handles at the corners of the object.
5. Set details for the object, for example, margins or borders. You can do so on the following palettes: *Layout*, *Border*, *Object*, *Font*, and *Paragraph*.

The Hierarchy Palette

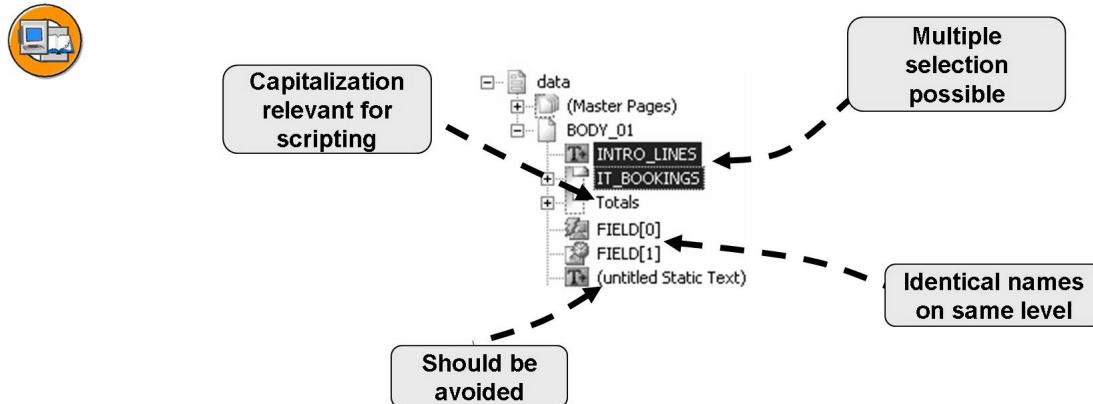


Figure 40: The “Hierarchy” Palette

All objects that you have included somewhere in the layout will be visible in the *Hierarchy* palette.

You can expand or compress the *Hierarchy* by clicking on the plus (+) or minus (-) signs.

If you want to set details for an element, select it and then select the relevant palette, for example, *Layout*, *Border*, or *Font*. The context menu will show you available options like *Rename* or *Delete*.

You can select more than one element at a time by holding down the CTRL key and left-clicking on the elements.

When renaming an object, you should bear in mind that capitalization matters for scripting. *Field* is not the same as *field* or *FIELD*.

If, within one page, two elements with the same name appear more than once within the same subform, they will be numbered consecutively in the *Hierarchy* tree, starting with 0. For instance, you might have *FIELD[0]* and *FIELD[1]*. Thus it is possible to address one of the fields directly. (However, for the sake of clarity you should strive to avoid such cases.)

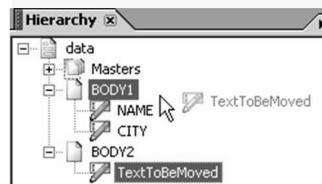
Does the position of an object in the *Hierarchy* affect its position on a page?

- Some objects can be positioned exactly in the Layout Editor at design time. Their positions will never change at runtime. For these objects, changing their positions in the *Hierarchy* palette does not change their positions in the layout – as long as you do not move them to a different page. (This is the case for boilerplate objects on a master page and in subforms of type *Position Content*.)
- For those objects whose position and/or call sequence is determined at runtime only, changing their position in the *Hierarchy* palette will change their position in the final document. Example: For example, of two master pages in the hierarchy, the uppermost one will be taken first by default. Also, if you swap table cells in the Hierarchy, they will be swapped in the resulting document as well.



To nest an object *into* another object (making it the last part of it) in the *Hierarchy* palette:

Drag the object over the object in which you want it nested.



To place an object *underneath* another object in the *Hierarchy* palette:

Drag the object below the other object. The triangle will indicate the place and level.

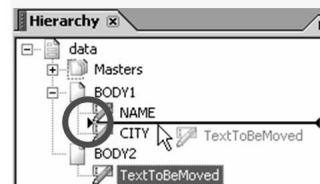


Figure 41: Nested Objects vs. Consecutive Objects

When you move objects in the *Hierarchy* palette, you must distinguish between **nested objects** and **consecutive objects**. If you nest one object in another one (for example in a body page or a subform), it will become part of it. If, at runtime, the enclosing object is not displayed (for example, due to coding), any objects nested in that hidden object will not be displayed either.

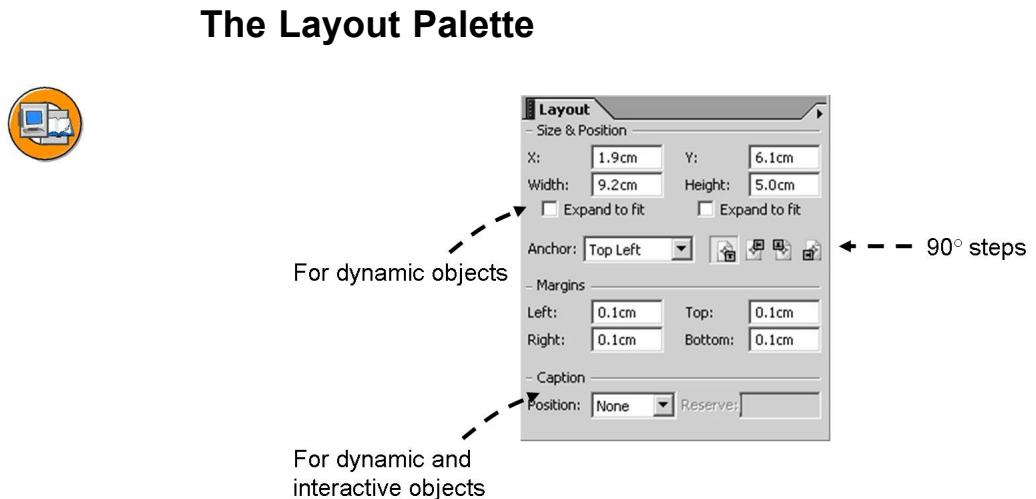


Figure 42: The Layout Palette

You can position and size an object by clicking the resize handles of the Layout Editor and moving them. You can achieve the same by typing in the coordinates and the width/height in the *Layout* palette.

For dynamic elements (like dynamic texts that come via the application program) you can select *Expand to fit* for the width and/or height to avoid the disappearance of lines.

You can set the margins, for instance, the space between text and the borders of the text object.

Dynamic and interactive objects (like text fields or checkboxes) will normally need to have a caption. You can determine its size and its position with regards to the object itself.

Objects can be rotated in 90° steps. You must specify around which anchor point the object should be rotated.

The Border Palette



Information

Our contact	Smith
Phone	+49 6227 777777
Fax	+49 6227 777775

Edges:

- Edit together
- dashed
- 0.1cm
- color: black
- round corners, radius: 0.5 cm

Background fill:

- solid
- gray – 25%

Information

Our contact	Smith
Phone	+49 6227 777777
Fax	+49 6227 777775

Edges:

- Edit together
- none

Background fill:

- Linear – to bottom
- red/light orange

Figure 43: Borders and Background Colors

On the *Border* palette, you can determine edges and/or background fills.

Edges can be edited together or individually.

For a background fill, you can choose between *none*, *solid* (one color) and various patterns for two colors.

For objects that are non-static (like a text field), you can also specify the border properties of the fillable areas. For example, you might choose to have a background color for a text field that differs from its caption color. To achieve this, select the object. In the *Object* palette, choose the *Field* tab. Then, in the *Appearance* list, choose the option *Custom....*

Arranging Objects

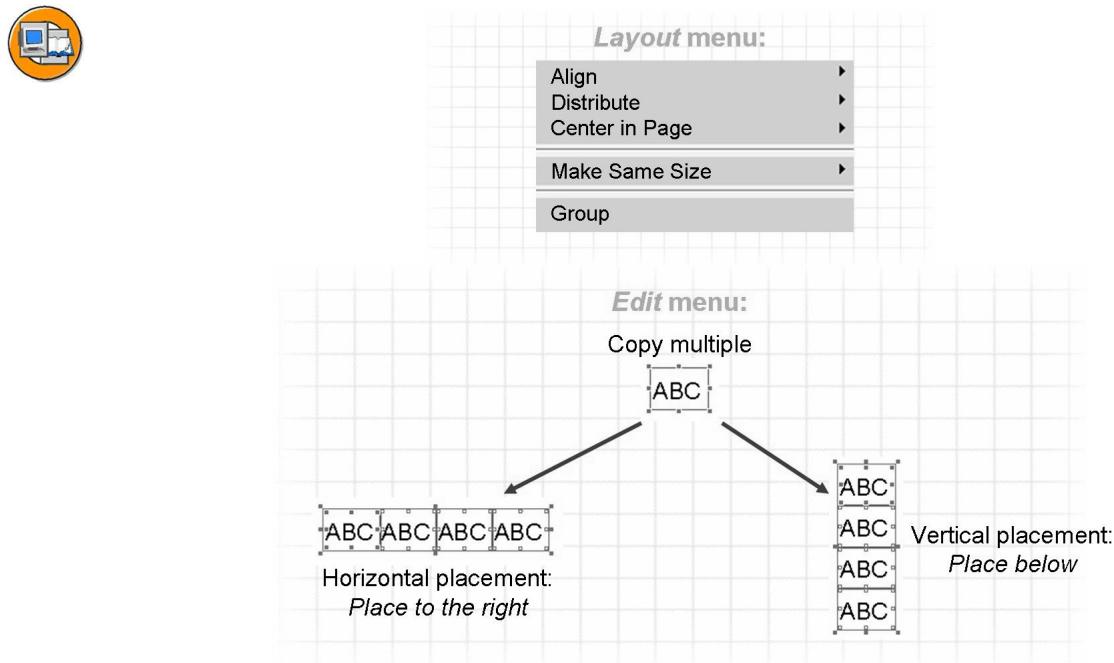


Figure 44: Arranging Objects

The following options are available for arranging the objects that you have inserted on a page. You can, for example, assign two objects the same x-coordinate or the same size. You access these functions by choosing the *Layout* menu.

If you want to multiply an object and arrange the copies so that they have all the same X position or Y position as the original, choose *Copy Multiple* from the *Edit* menu.

Re-using Layout Elements



1. Select element(s)
2. Drag and drop to the desired *Library* tab

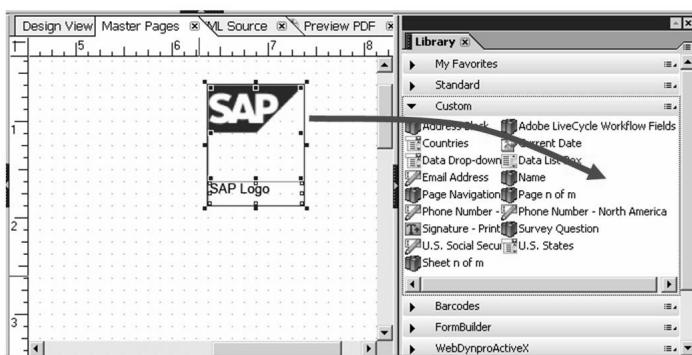


Figure 45: Preparing Layout Elements for Reuse

If you create an element and need to use it several times in your layout, it can be added to a tab page of the *Library* palette. You can then drag and drop your element from the *Library*, just like all predefined elements.

If you want to add an object to the *Library*, choose the tab where you want to include the object, select the object in the page, and right-click it. Then select *Add to Library* from the context menu.

In the following dialog box, enter a name and description for the object. Check that the tab group is the one you want and choose *OK*.

Alternately, you can drag and drop the layout element to the appropriate *Library* tab.

It is also possible to represent several layout elements by one *Library* object. To achieve this, mark the required objects before moving them to the palette.

If you enter a name that already exists in the selected *Library* tab, you can choose whether you actually want to override the existing one or cancel the operation. All standard objects that come with Designer can be restored to any *Library* tab by selecting *Restore Standard Objects* from the palette menu.

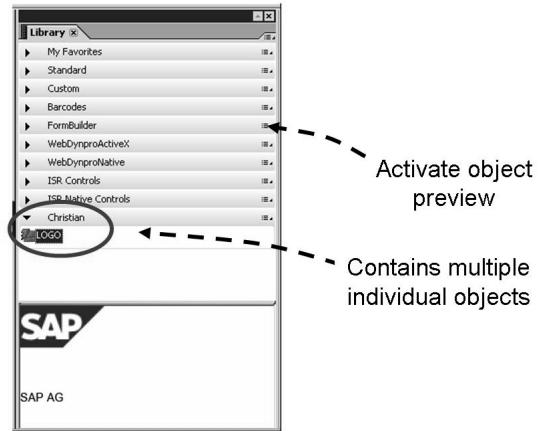


Figure 46: Library: Object Preview

In the palette menu of the *Library* palette, choose *Show Object Preview* to get an impression of what the object you are going to include on a page will look like. This is of particular interest if you include objects that consist of multiple parts, or if you include objects other than the predefined standard objects.

If you include an object that consists of several parts, you will actually include these individual parts, rather than one single object.

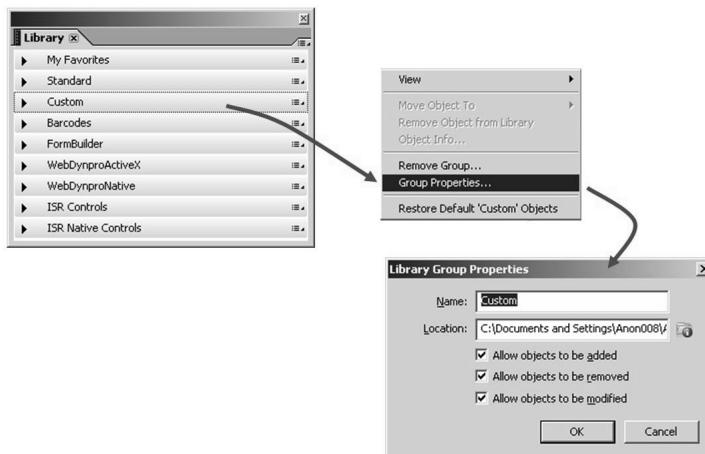


Figure 47: Sharing Library Objects

You can create your own groups of *Library* objects by right-clicking on an existing group (that is, the tab) and choosing *Add group*. A library with its objects can also be published on a server. In the *Library* palette, click the palette menu (that is, the black arrow in the upper right) and then specify the location in the *Library Group Properties* dialog box. Library objects will be saved as individual XML files (with the extension XFO) there. Note that your logon language determines which libraries will be shown; as per default, the XFO files are stored in separate directories for each language.

From the palette menu, you can also choose *Shared Library Location*. Here you can enter an XML file that lists tabs and their directories with XFO files. Thus you get a reference to a certain number of tabs without having to include them individually. An example for such an XML file could be:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectLibraryTabSet>
    <tab name="Extras 1" directory="\machine1\ABC" permission="adm"/>
    <tab name="Extras 2" directory=".\\XYZ" permission="adm"/>
</objectLibraryTabSet>
```

XML Data and Preview

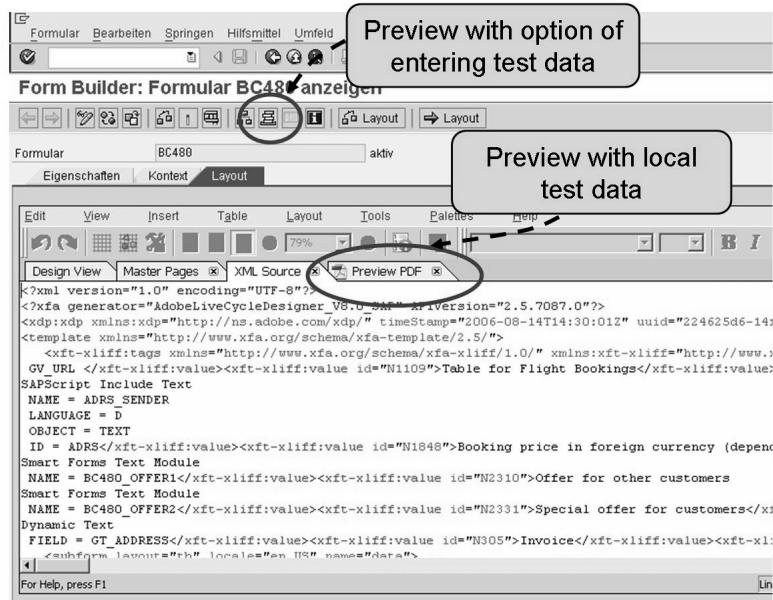


Figure 48: XML Data and Preview

Adobe LiveCycle Designer uses the XML Forms Architecture (XFA), an XML-based language, to model form templates. XFA supports scripting in JavaScript and Adobe's own script language, FormCalc.



Caution: XML data can be changed directly – but you should be aware that you can easily corrupt a form.

Since the XML data contains information on the data schema, using the clipboard to copy the layout between different forms can result in faulty forms. You should use the download option provided in transaction SFP instead. To do so, choose *Utilities → Downloading form....* This will download complete forms (with context and layout).

You can preview the form on the *Preview PDF* tab. Note that this preview does not allow you to enter test data, but you can determine an XML file to be used as the test data source.

If you do not see the *Preview* tab, your browser plug-in settings for Adobe Reader probably need to be reconfigured:

1. Close Adobe LiveCycle Designer.
2. Start Adobe Reader.
3. Choose *Edit → Preferences... → Internet*.
4. Select *Display PDF in browser* and *Check Browser settings when starting Reader*. You may disable the second setting again later on.
5. Close Adobe Reader.
6. Restart Adobe LiveCycle Designer.

In case you still do not see the *Preview* tab, a new installation of Adobe Reader might be necessary.

If you want to see the result in the spool preview from the SAP system, you have two options:

- If you want to view the result with data that you type in as you test it, choose *Form → Test* from the SAP menu (shortcut: F8). This will generate the form's function module and display the Function Builder test environment, where you can enter test data manually. This data can also be saved in the test data directory for future tests.
- You can activate the form and run an application program that calls your form.

Form Properties

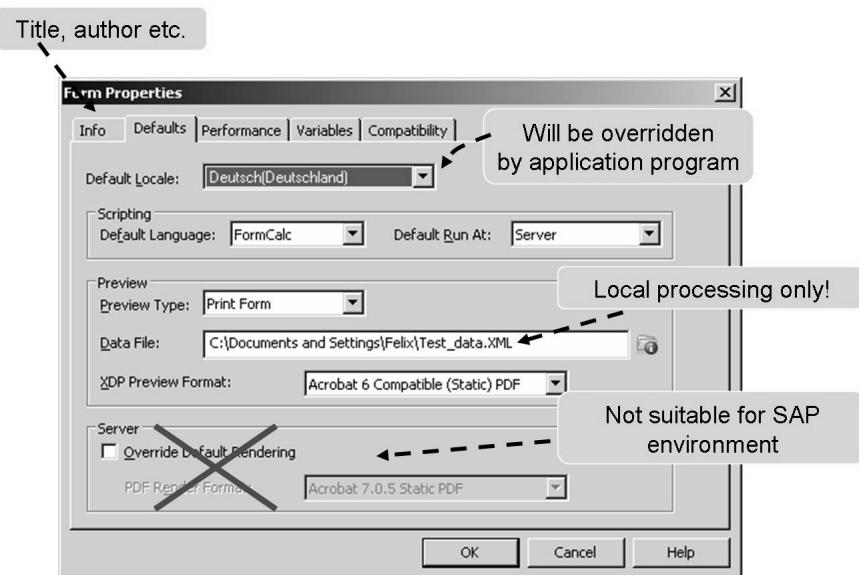


Figure 49: Form properties

When you start designing your form, you might want to begin with setting the form properties (however, you could also do it any time, as the form properties do not interfere with the layout). To set the form properties, choose *Edit → Form properties* in the Designer.

A **locale** is a combination of a language and country, for example English and United Kingdom. For a PDF-based form in an SAP application, you would typically set the locale dynamically in the ABAP program. However, for testing purposes you should choose a locale for your form.

A locale that you set will have implications on the time and date formats and on decimals formats.

All data retrieval for the form must be done with ABAP coding. This is usually either done in the application program, but can also be done in the code in the interface. However, you can also use scripting for calculations within the form. This scripting will be executed at runtime, triggered by Adobe document services. For this scripting, two programming languages are available: JavaScript (preferable for interactive scenarios) and Adobe's FormCalc (preferable for printing scenarios). In the form properties, you can set one of these two scripting languages as the default, but it is possible to change this for individual scriptings.

The preview type affects Designer's preview only, not SAP spool's preview, which is always non-interactive (unless, in the ABAP program, you set the *fillable* parameter of function module FP_JOB_OPEN to X or N).

The setting *Override Default Rendering* is not for the use of LiveCycle Designer within an SAP environment.

Entering a data file for the preview makes sense if you want to test your form with some data but you don't want to activate the form or type in the data every time. A file that you enter here will be used by Designer's preview. Be aware that this testing is done locally, that is, without invoking Adobe document services. Fonts might look different from what they will look like in a preview or printout when rendered by Adobe document services. Also, you might or might not have access to files or URLs that you will or will not have in a real scenario.



Local Test Data File

```
<data>
<IS_CUSTOMER>
<MANDT>800</MANDT>
<NAME>Rahn</NAME>
<FORM>Frau</FORM>
<STREET>NG Road 1</STREET>
<POSTBOX/>
<CITY>Walldorf</CITY>
</IS_CUSTOMER>

<IT_BOOKINGS>
<DATA>
<CARRID>AA</CARRID>
<CONNID>0017</CONNID>
</DATA>

<DATA>
<CARRID>AZ</CARRID>
<CONNID>0788</CONNID>
</DATA>
</IT_BOOKINGS>

<LOGO href = "\\\MyMachine\Uli.bmp"/>
</data>
```

A test data file must be structured as shown above. In particular, a leading `<data>` and trailing `</data>` is required. Field names are case sensitive. The individual rows of internal tables must be embraced by `<DATA>` and `</DATA>`.

You can use transaction SFP to create test data for Designer: In transaction SFP, choose *Utilities(M) → Settings*. Set *Trace Level* to *Very Detailed Trace*. This will attach a number of documents to the main PDF document. You can also specify that an extra copy of this main document including all of its attachment is saved locally.

These settings are user-specific and valid for one session. They are not restricted to a specific form or program. If there are several print requests, the corresponding files will be overwritten unless different names are specified. If, however, several forms are issued in one print job, in which case the runtime data will be numbered sequentially.

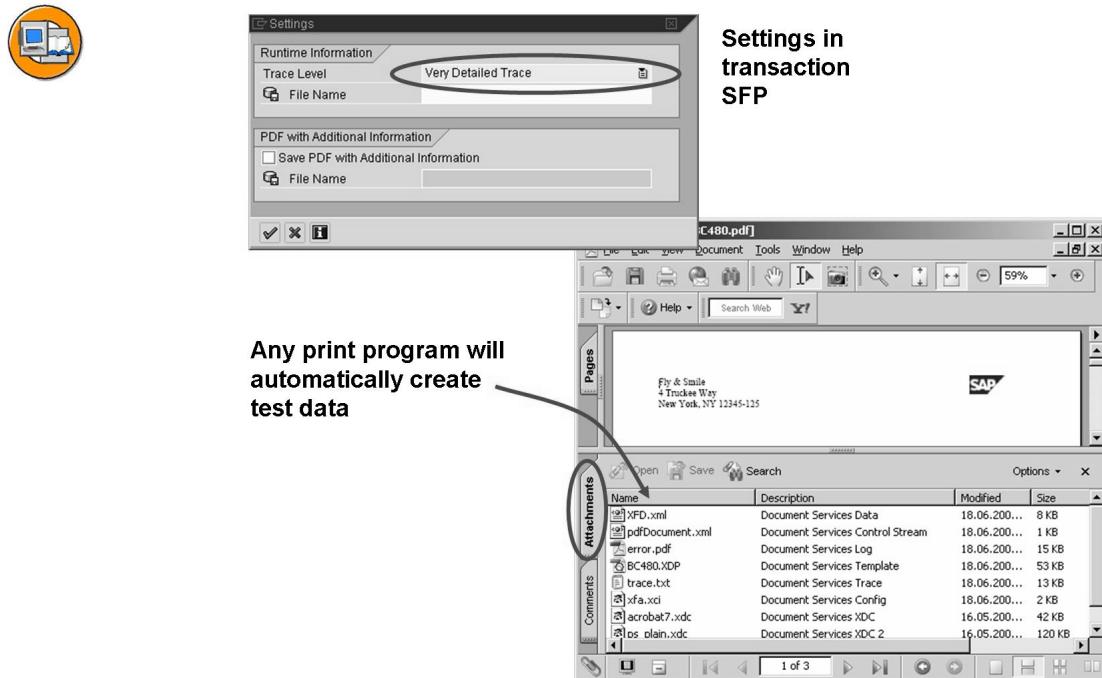


Figure 50: Creating Test Data Automatically

For the download and trace functionality, the user must have authorization for debugging. During background processing, only *Very Detailed Trace* works.

If you work with Web Application Server 6.40, Support Package 10 or higher, execute the following steps: In transaction SFP, choose *Utilities(M) → Settings*. Check *Save Runtime Data* and enter the destination file (preferably with extension XML). See SAP Note 779319 for details.

Exercise 3: Adobe LiveCycle Designer: Overview

Exercise Objectives

After completing this exercise, you will be able to:

- Add a simple layout to a form context
- Use various functionalities of Adobe LiveCycle Designer to include and manipulate form objects

Business Example

A travel agency wants to print invoices for their customers' flight bookings using the tool "Interactive Forms". The interface and the form context have been designed already, but there is no form layout yet. The form designers first need to get an overview of how Adobe LiveCycle Designer works.

Task 1:

Set up the workspace.

This exercise makes some suggestions on how to get to know Adobe LiveCycle Designer. Basically, you should play around with the tool. Feel free to invent your own tasks. Use your form, ZBC480_##, to integrate some dummy elements.

 **Note:** At the end of the exercise, your form will not yet have a proper layout.

1. Try out both the integrated and the full screen version of Designer
2. Display and hide the areas for the palette windows. Resize them.
3. Undock some palettes and move them away from their starting position.
4. Examine the menus of various palette menus.
5. Set centimeters (or inches, if you prefer) as the unit for the ruler.

Task 2:

Integrate and manipulate layout elements.

1. Switch between pages (design view) and master pages
2. Integrate various kinds of layout elements into the form layout, using the *Data View* and *Library* palettes. Include at least one static text.
3. Rename objects that you have integrated into the layout.

Continued on next page

4. Move objects in the *Hierarchy* palette. Observe the difference between nested and consecutive objects.
5. Change the content of a static text and set some details (for example, the justification or the font).
6. Rotate a static text.
7. Change the background color and the border of a static text.
8. Copy a static text four times, so that all five texts are arranged horizontally.
9. Select several elements at a time in the Hierarchy.
10. Undo some changes.

Task 3:

Reuse elements (templates).

1. Create a new tab for the *Library*.
2. Add the rotated static text to the new tab.
3. Drag and drop the template object from your tab into the layout.

Task 4:

Use the XML view and test data.

1. Select a text in the layout and have a look at its XML representation. Carefully (!) change its value (within the `<value> </value>` tags).
2. If in this class, you are working with a Windows Terminal Server, you should find an XML test data file in the BC480 directory with suitable test data. Its name: `Test_data.xml`. Locate this file with Windows Explorer and have a look at its content.

If you cannot locate the file, or if you want to try to create it yourself, do the following:

In the SFP settings, set the trace level to *Very Detailed Trace*. Once set, every SAP program that creates a PDF document (with the help of Adobe document services) will attach a number of files to the PDF, including an XML file with an XML data stream. (These settings are valid for the current session.) To create a test file suitable for your forms, run report `SAPBC480_DEMO` for form `BC480_FINAL`.

3. Set the XML test data file provided or the newly created XML file as the test data file of your form, `ZBC480_##`.

Continued on next page

4. From the *Data View*, integrate at least one context field into the layout. This field should also be in the XML file just created. Test your form in Designer's preview. The field should be populated with data.

Task 5:

Set form properties.

1. Set *English (US)* as the *Default Locale*, FormCalc as the default scripting language, Server as the default place for the execution of scripting, and *Print Form* as the *Preview Type*.
2. From the *Data View*, integrate the field *DATE* into a page. Go to the PDF preview of the Layout editor. If you now change the default locale to a different country/language and rerun the PDF preview, the date format should differ.

Solution 3: Adobe LiveCycle Designer: Overview

Task 1:

Set up the workspace.

This exercise makes some suggestions on how to get to know Adobe LiveCycle Designer. Basically, you should play around with the tool. Feel free to invent your own tasks. Use your form, ZBC480_##, to integrate some dummy elements.

 **Note:** At the end of the exercise, your form will not yet have a proper layout.

1. Try out both the integrated and the full screen version of Designer
 - a) In transaction SFP, choose the *Layout* tab to get to the integrated view of Designer.
 - b) For the full screen version, click on the *Layout* pushbutton in the SAP GUI or press Ctrl-F12.
2. Display and hide the areas for the palette windows. Resize them.
 - a) Palette windows are to the left and the right side of the Layout Editor. In the middle of each of the two borders, between the areas and the Layout Editor, you will find a small vertical bar with a triangle. Click on it to open or close an area for palette windows. Click on it (or anywhere on the border) and drag it to the left or right side to resize the area.
3. Undock some palettes and move them away from their starting position.
 - a) Press the CTRL key, click on a palette or a palette bar (which is typically blue, depending on your Windows settings), and drag it to a new place.
4. Examine the menus of various palette menus.
 - a) Click on the small triangle in the upper-right corner of the palette.
5. Set centimeters (or inches, if you prefer) as the unit for the ruler.
 - a) Locate the *Drawing Aids* palette. Choose the *Palettes* menu and select *Drawing Aids*.
 - b) In the *Grid and Ruler Settings*, enter centimeters (or inches) as the unit.

Continued on next page

Task 2:

Integrate and manipulate layout elements.

1. Switch between pages (design view) and master pages
 - a) If the *Master Pages* tab does not show at the top of the Layout Editor, right-click the *Design View* tab and choose *Master Pages*.
2. Integrate various kinds of layout elements into the form layout, using the *Data View* and *Library* palettes. Include at least one static text.
 - a) Integrate the elements by dragging and dropping them to the layout.
 - b) Static texts can be found on the *Standard* tab of the *Library* palette.
3. Rename objects that you have integrated into the layout.
 - a) In the *Hierarchy* palette, set the cursor on the element. Press F2 and enter a new name.
4. Move objects in the *Hierarchy* palette. Observe the difference between nested and consecutive objects.
 - a) Click and drag the object with the left mouse button. You can nest a text in a page (design view) or create two texts as consecutive elements. Nesting one text into another is not possible.
5. Change the content of a static text and set some details (for example, the justification or the font).
 - a) Static texts can be changed only in the Layout Editor. Set the cursor on the element and type in your text. You can set the text justification on the *Paragraph* palette and the font on the *Font* palette.
6. Rotate a static text.
 - a) The rotation of an object (for example, a static text) is achieved on the *Layout* palette, in the *Size & Position* area. You can select the anchor (fulcrum) and then set a rotation angle of 90 degrees, 180 degrees, or 270 degrees.
7. Change the background color and the border of a static text.
 - a) Use the *Border* palette.
8. Copy a static text four times, so that all five texts are arranged horizontally.
 - a) Choose *Edit* → *Copy Multiple....*

Continued on next page

9. Select several elements at a time in the Hierarchy.
 - a) Within the *Hierarchy* palette:

To mark random objects, press the CTRL key and click on the elements.

To mark an uninterrupted range of objects, set the cursor on the first element, keep the SHIFT key pressed, and click on the last element.
 - b) Alternatively, within the Layout Editor, keep the left mouse button pressed and draw a rectangle around the objects.
10. Undo some changes.
 - a) Press CTRL+Z. In some cases, you might have to unmark a marked object first.

Task 3:

Reuse elements (templates).

1. Create a new tab for the *Library*.
 - a) Right-click on an existing tab (for example, *Standard*) and choose *Add Group*. Enter the name of the new tab in the next window.
2. Add the rotated static text to the new tab.
 - a) Mark the text in the Layout Editor and drag it over to the new tab in the *Library* palette. Enter a suitable name and a description.
3. Drag and drop the template object from your tab into the layout.
 - a) Check that it is an exact copy of the original static text.

Task 4:

Use the XML view and test data.

1. Select a text in the layout and have a look at its XML representation. Carefully (!) change its value (within the `<value> </value>` tags).
 - a) Set the cursor on the element. Go to Designer's *XML* tab in the Layout Editor. If the *XML Source* tab is not displayed, right-click the *Design View* tab and choose *XML Source*.

You will find a tag similar to `<draw name="StaticText1" y="70.00mm" x="120.50mm">`, which marks the beginning of the static text. `<draw name="StaticText1" y="70.00mm" x="120.50mm">`.

Some lines further down, you will find something similar to `<value><text>Static Text</text></value>`.
`<value><text>Static Text</text></value>`.

Continued on next page

2. If in this class, you are working with a Windows Terminal Server, you should find an XML test data file in the BC480 directory with suitable test data. Its name: Test_data.xml. Locate this file with Windows Explorer and have a look at its content.

If you cannot locate the file, or if you want to try to create it yourself, do the following:

In the SFP settings, set the trace level to *Very Detailed Trace*. Once set, every SAP program that creates a PDF document (with the help of Adobe document services) will attach a number of files to the PDF, including an XML file with an XML data stream. (These settings are valid for the current session.) To create a test file suitable for your forms, run report SAPBC480_DEMO for form BC480_FINAL.

- a) By double-clicking on the XML file in Windows Explorer, the associated viewer will open.
 - b) To create your own XML test data, start transaction SFP.
 - c) Choose *Utilities* → *Settings*.
 - d) Set the trace level to *Very Detailed Trace*
 - e) Start transaction SA38 to run report SAPBC480_DEMO. On the selection screen of this report, enter the form **BC480_FINAL**. Execute the report (press F8). The resulting PDF will be displayed in the SAP GUI.
 - f) From Adobe reader (which is integrated into the SAP GUI), click on the tab *Attachments* on the left-hand side. You will find a document titled XFD. Right click on it and choose *Save Attachment*. Choose a suitable location where to save the file. Name it Mytest.XML and press *Save*.
3. Set the XML test data file provided or the newly created XML file as the test data file of your form, ZBC480_##.
 - a) In Designer, choose *Edit* → *Form Properties...*, choose the *Defaults* tab and the *Data File* field.
 4. From the *Data View*, integrate at least one context field into the layout. This field should also be in the XML file just created. Test your form in Designer's preview. The field should be populated with data.
 - a) As the form BC480_FINAL has the same context fields as your form, everything that you drag from the *Data View* of your form to the layout will have a test file entry.

Continued on next page

Task 5:

Set form properties.

1. Set *English (US)* as the *Default Locale*, FormCalc as the default scripting language, Server as the default place for the execution of scripting, and *Print Form* as the *Preview Type*.
 - a) Choose *Edit → Form Properties*.
2. From the *Data View*, integrate the field *DATE* into a page. Go to the PDF preview of the Layout editor. If you now change the default locale to a different country/language and rerun the PDF preview, the date format should different.
 - a) See previous step.



Lesson Summary

You should now be able to:

- List the various functions of Adobe LiveCycle Designer
- Use Adobe LiveCycle Designer as a graphical tool for designing forms
- Create a simple form layout with various objects

Lesson: Adobe LiveCycle Designer: Structuring a Form

Lesson Overview

Adobe LiveCycle Designer is simple to handle and is very similar to other graphics programs. However, you need to be familiar with the concepts of form printing. You are unlikely to associate anything complicated with the term “page”. However, in this lesson you will learn that a lot of information is connected to the concept and this information needs to be organized. For example, you must distinguish between page (design view) and master pages, you must reserve areas for dynamic output, and you must determine the page sequence.



Lesson Objectives

After completing this lesson, you will be able to:

- Distinguish between master pages, content areas, pages (design view), and subforms
- Describe typical contents of master pages, content areas, pages (design view), and subforms

Business Example

A travel agency wants to print invoices for their customers' flight bookings using the tool “Interactive Forms”. The form designers have a basic understanding of the graphical tool, but no ideas of the concepts behind it.

Master Pages

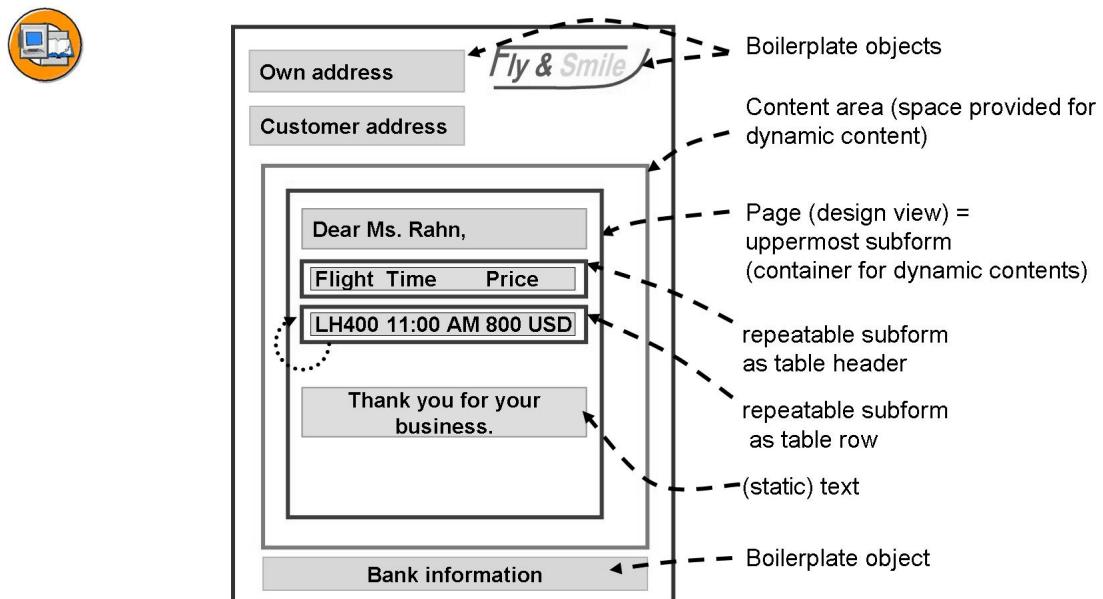


Figure 51: Objects on a Master Page

Every form design contains at least one **master page** that Adobe LiveCycle Designer creates automatically. You can put objects on a master page that will appear on any resulting page at runtime, for example, your company logo. Such objects that never change are called **boilerplate objects**.

Everything that is on an (abstract) master page will reappear on every page issued (design view). To some extent, the boilerplate objects of a master page could be compared to secondary windows in SAPscript or Smart Forms.

On a master page, you must include at least one **content area**. This defines the size to be used for dynamic output. Content areas can be included only on master pages.

Dynamic content is wrapped up in **pages (design view)**. A page (design view) will use the space provided by a content area. If, at runtime, this happens not to be sufficient due to a large amount of data, the body page will look for the next content area (which might involve an automatic page break).

You should think of a page (design view) as nothing more than a wrapper or a container for dynamic content. So you might have one single page (design view) in a form with lots of items at runtime; consequently, the printout would have many pages.

To some extent, the combination of a content area and a page (design view) included there could be compared to the main window in SAPscript or Smart Forms. (Note, however, that this is only a very rough equivalent!)

For further organising pages (design view) (or, though rarely, master pages), they can contain **subforms**.

It is a good idea to always start out with the static part of a form: the master pages.

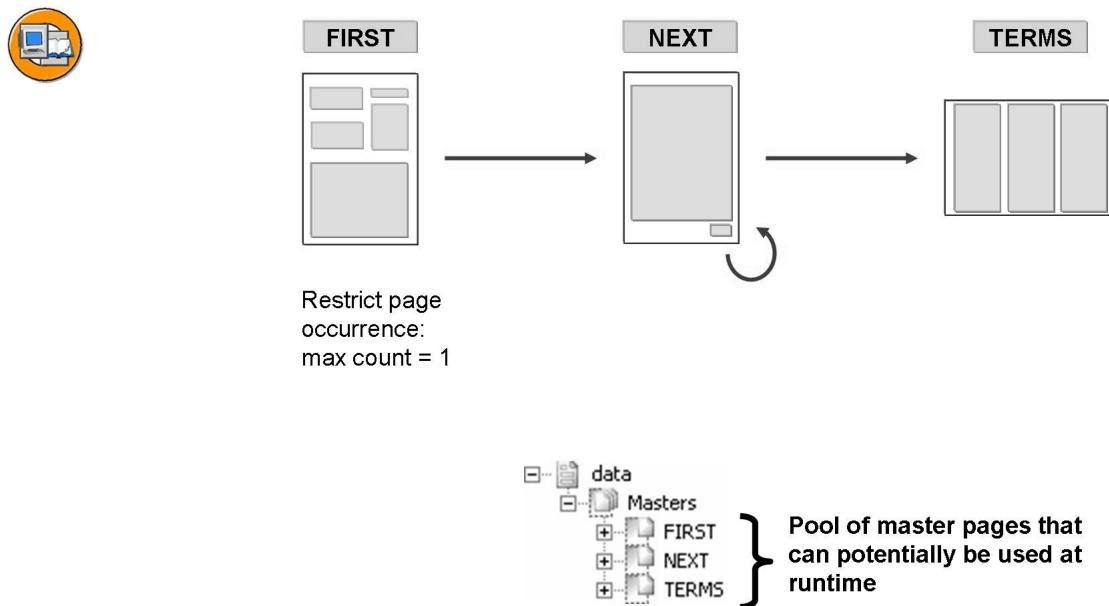


Figure 52: Inserting Several Master Pages

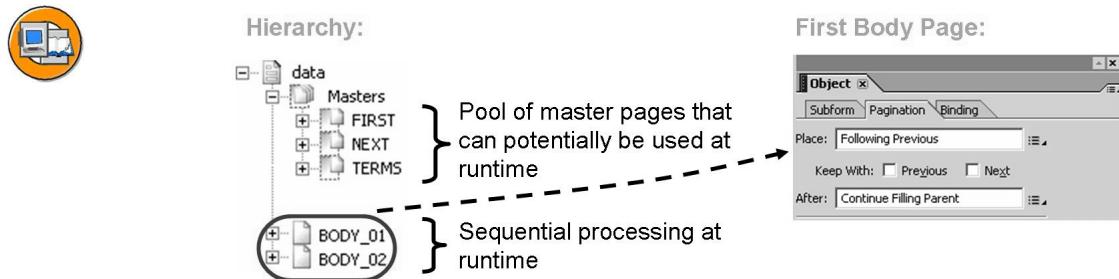
For every master page, go to the *Object* palette in order to set:

- The page size, for example, letter or A4
- The page orientation, for example, portrait or landscape
- Whether the page occurrence should be restricted: you can restrict the number of pages on which the first master page (with addresses, company logo and so on) will be used; you could restrict it to 1, for example. During runtime, this master page will then only be used for one output page. The second master page in the *Hierarchy* would by default automatically be taken for following output pages if more data is laid down than can be displayed on a single page. If you restrict the maximum page occurrence without having another master page, this setting will be ignored at runtime if more data needs to be displayed.



Caution: If you want to change properties of a master page, double-clicking on it in the *Hierarchy* palette is not sufficient, first select the *Master Pages* tab.

Pages (Design View)



When should you insert a new page (design view)?

- If you want to force a page break
(possibly using a different master page)

Figure 53: Pages (Design View) and Using Them

A page (design view) is a top-level subform. It serves as an organising unit for dynamic content and can be laid down only in a content area of a master page. Make sure to have its place set to *Top of Next Page* (which will take the next master page from the hierarchy) or *Top of Page...* if you want to start a new page.

Selecting the *Following Previous* option for the first page (design view) means that you start with the first master page in the *Hierarchy*.

If you set a page's (design view) place to *On Page <Master Page>*, there are two possible scenarios:

- If the preceding subform was already laid out on that master page, the page (design view) will follow on the same output page (provided there is enough space).
- If the preceding subform was laid out on a different master page, a page break will be inserted and a new output page will begin using the desired master page.

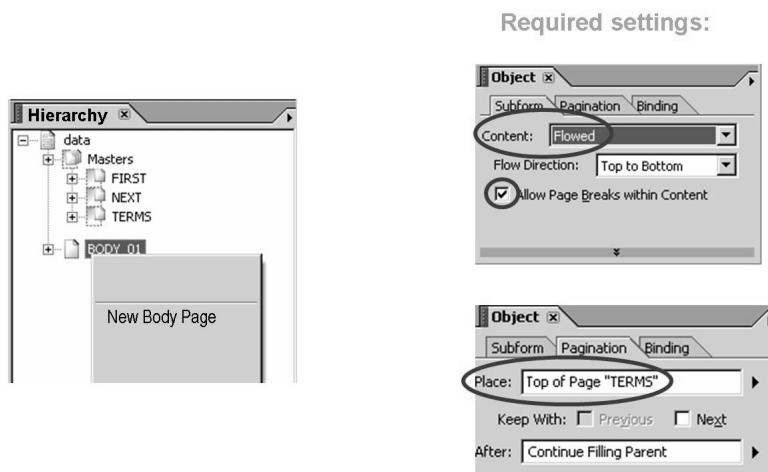


Figure 54: New Page: Inserting the Terms and Conditions

To display the terms and conditions on a separate page with a special layout, you need to define an extra master page (let's call it *TERMS*) and create a new page (design view). For the new page (design view) (which is, like all pages (design view), a subform), choose *Top of Page → TERMS* for *Place*.

Instead of creating a new page (design view), you could tell the previous subform to go to the top of page *TERMS* after it has been laid down. (However, we recommend creating a new page (design view) for each forced page break. This makes the *Hierarchy* palette clearer.)

Subforms



Subforms: containers for grouping several objects

- If of type *Positioned*, objects of subforms can be laid down at their exact position at design time. (Hierarchy position of objects is not relevant for layout position.)
- If of type *Flowed*, the objects will follow each other, depending on which space they require at runtime.
Body pages (being top level subforms) are typically of type *Flowed*.

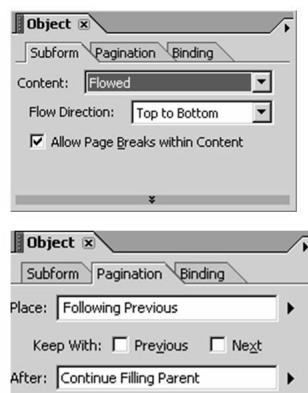


Figure 55: Subforms

In a very simple approach, subforms can be thought of as folders containing several objects. They can be used for the simple reason of keeping order in the *Hierarchy*, as it is possible to expand and compress subforms.

There are two types of subforms:

- If of type *Positioned*, objects of subforms can be laid down at their exact position at runtime, relative to the subform. For example, if a text field has been positioned at the top left corner of a subform of type *Positioned*, it will always be positioned at the top left corner of the subform, independent where on a page this subform is included. (The *Hierarchy* position of an object within a subform of type *Positioned* is irrelevant for its layout position.)
- If of type *Flowed*, the objects will follow each other, depending on the space they require at runtime. A page (design view) (as the topmost subform) is typically of this type.



Placing objects in a subform makes sense:

- If you want to visually group objects
- If you want to keep objects together (protect them against page breaks)
- If you want to output the elements repeatedly (as table row or table header)
- If you want to hide several elements at once
- If you want to influence the screen reader order

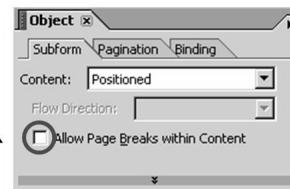


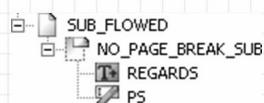
Figure 56: When to Include Subforms

Subforms also help to rearrange several objects at a different place in the form.

Be aware that nesting subforms will slow down the processing performance.



Subform/body
page of type
Flowed



Subform settings:
Do not allow
Page Breaks
within Content

Figure 57: Preventing Page Breaks

If you want to make sure that certain objects appear on one page, proceed as follows:

1. Mark the objects.
2. In the *Hierarchy*, choose *Wrap in subform* from the context menu.
3. Disallow a page break for this subform. You do this on the *Object* palette, *Subform* tab. Deselect *Allow Page Breaks within Content*.

This option is not available on a master page or if the subform itself is included in another subform of type *Positioned*.

You can also prevent a page break between several subsequent subforms. On the *Object* palette, *Pagination* tab, select *Keep With: Previous* and/or *Keep With: Next*.

- *Keep With Previous*: Keeps the subform on the same page as the previous subform, content area, or page (design view), depending on which one is selected in the *Place* box.
- *Keep With: Next*: Keeps the subform on the same page as the next subform, content area, or page (design view), depending on which one is selected in the *Place* box.

Some hints that might help to create readable forms:

- Even though you could have no pages (design view) at all and manage everything from master pages, it is strongly recommended not to do so.
- You should not nest a subform B in subform A if you want B to be on a different master page. If you do, the hierarchy tends to become unreadable.
- Although it is possible to select *Go to Next Page* or *Go to Page...* in the *After* field of an ordinary subform, for the sake of clarity you should not, as you would essentially force a page break. If a page break is necessary, you should create a new page (design view) (which is also a subform). This keeps the hierarchy clear.
- The options *Keep With: Previous* and *Keep With: Next* (to prevent a page break between two subforms) are not very useful for pages (design view). It should be used for normal subforms only.

Exercise 4: Adobe LiveCycle Designer: Structuring a Form

Exercise Objectives

After completing this exercise, you will be able to:

- Create master pages and include elements on them
- Create pages (design view) and use master pages
- Set different flow directions for the pages (design view)

Business Example

A travel agency wants to print invoices for their customers' flight bookings using the tool "Interactive Forms". The interface and the form context have been designed already, but there is no form layout yet. The first step for the form designers will be to create the structure of the form, namely the master pages and pages (design view).

Task 1:

Prepare the form for further enhancements.

Continue to work with your own form, ZBC480_##. ## stands for your two-digit group number (monitor number).

It might be a good idea to create a local safety copy of the form every now and then. To do so, choose *Utilities → Downloading form*.

If you have not finished one of the previous exercises, you can copy the template form BC480_CONTEXT_ONLY.

The reference solution of this exercise is form BC480_STRUCTURE.

1. Remove any redundant elements from the form layout. You should be left with just the minimum: one master page with your content area.

Task 2:

Define master pages.

Your form should have three master pages.

1. FIRST: size: A4; alignment: portrait format.

Make sure FIRST is used only once per form. The content area on this page should cover approximately two thirds of the page so that there is enough space for the address fields above and for the bank details below.

2. NEXT: size: A4; alignment: portrait format.

Continued on next page

NEXT should be processed as many times as needed.

3. TERMS: size: A4; alignment: landscape format.

Task 3:

Define pages (design view)

Make sure your form has two pages (design view).

1. Call the first one BODY_01. Define *Following Previous* as its place.
2. Call the second page (design view) BODY_TERMS. Define *Top of Page TERMS* as its place. Make sure it fits into the content area of the master page.

Task 4:

Explore subform types.

1. Try out both subform types for BODY_01. Integrate dummy elements (for example, static texts) and observe how their placements change.
2. In the end, select *Flowed* for BODY_01, *Flow Direction = Top to Bottom*. Delete all elements that you have included in the body page.

Task 5:

Test the result.

1. Activate the form and then test it by running report SAPBC480_DEMO. You should get two empty pages: one in portrait and one in landscape orientation.

Solution 4: Adobe LiveCycle Designer: Structuring a Form

Task 1:

Prepare the form for further enhancements.

Continue to work with your own form, ZBC480_##. ## stands for your two-digit group number (monitor number).

It might be a good idea to create a local safety copy of the form every now and then. To do so, choose *Utilities* → *Downloading form*.

If you have not finished one of the previous exercises, you can copy the template form BC480_CONTEXT_ONLY.

The reference solution of this exercise is form BC480_STRUCTURE.

1. Remove any redundant elements from the form layout. You should be left with just the minimum: one master page with your content area.
 - a) The easiest way to reduce the form layout to the minimum is to go to the *Hierarchy* palette and remove everything but one master page with its content area.

Task 2:

Define master pages.

Your form should have three master pages.

1. FIRST: size: A4; alignment: portrait format.

Make sure FIRST is used only once per form. The content area on this page should cover approximately two thirds of the page so that there is enough space for the address fields above and for the bank details below.

- a) Go to the *Hierarchy* palette and set the cursor on the master page. Press F2 to rename it.
 - b) Go to the *Object* palette and select the page size and orientation. Also, select the *Restrict Page Occurrence* option and set *Max Count* to 1.
 - c) In the Layout Editor, select the *Master Pages* tab. Resize the content area by clicking and dragging its upper and lower borders to suitable heights.
2. NEXT: size: A4; alignment: portrait format.

Continued on next page

NEXT should be processed as many times as needed.

- a) Go to the *Hierarchy* palette and set the cursor on the master page FIRST. Right-click on it and choose *Insert Master Page*. See previous step for further details.
3. TERMS: size: A4; alignment: landscape format.
 - a) See previous step.

Task 3:

Define pages (design view)

Make sure your form has two pages (design view).

1. Call the first one BODY_01. Define *Following Previous* as its place.
 - a) Go to the *Hierarchy* palette and set the cursor on the top node. Right-click on it and choose *Insert Page (Design View)*.
 - b) Set your cursor on the newly created page (design view) and go to the *Object* palette, *Subform* tab (every page (design view) is actually a subform). Choose *Following Previous* as its place.
2. Call the second page (design view) BODY_TERMS. Define *Top of Page TERMS* as its place. Make sure it fits into the content area of the master page.
 - a) See previous step.
 - b) Go to the *Pages (Design View)* palette to check the size. Resize the page (design view) by clicking and dragging its bottom right corner so that it fits into the content area.

Task 4:

Explore subform types.

1. Try out both subform types for BODY_01. Integrate dummy elements (for example, static texts) and observe how their placements change.
 - a) In the *Hierarchy* palette, set your cursor on the page (design view) and go to the *Object* palette, *Subform* tab.
2. In the end, select *Flowed* for BODY_01, *Flow Direction = Top to Bottom*. Delete all elements that you have included in the body page.
 - a) See previous step.

Continued on next page

Task 5:

Test the result.

1. Activate the form and then test it by running report SAPBC480_DEMO. You should get two empty pages: one in portrait and one in landscape orientation.
 - a) Start transaction SA38.
 - b) Enter **SAPBC480_DEMO** as the report.
 - c) Press F8 to execute.



Lesson Summary

You should now be able to:

- Distinguish between master pages, content areas, pages (design view), and subforms
- Describe typical contents of master pages, content areas, pages (design view), and subforms



Unit Summary

You should now be able to:

- List the various functions of Adobe LiveCycle Designer
- Use Adobe LiveCycle Designer as a graphical tool for designing forms
- Create a simple form layout with various objects
- Distinguish between master pages, content areas, pages (design view), and subforms
- Describe typical contents of master pages, content areas, pages (design view), and subforms



Test Your Knowledge

1. Which of the following statements about Adobe LiveCycle Designer are true?
Choose the correct answer(s).
 - A Toolbars can be moved around freely.
 - B Double-clicking on a palette will undock it.
 - C Every palette has a menu of its own.
 - D Changing the grid units will change the element positions.

2. Do you agree with the following statements on palettes?
Choose the correct answer(s).
 - A The *Hierarchy* palette displays those objects that have been included somewhere in the layout.
 - B The *Data View* palette displays elements from the form context.
 - C You can add elements to the *Data View* by right-clicking on it.
 - D The settings in the *Drawing Aids* palette are per form only. They are stored for every form individually.

3. Which of the following statements about form properties are true?
Choose the correct answer(s).
 - A You can select ABAP, JavaScript, or FormCalc as the default scripting language.
 - B The default locale will typically be overridden in the ABAP program.
 - C The XML data file will be taken for the local preview of Adobe Reader.
 - D The XML data file contains the definition of the form layout.

4. At runtime, every master page of a form is processed once.
Determine whether this statement is true or false.
 - True
 - False

5. Every master page must have exactly one content area.
Determine whether this statement is true or false.
 - True
 - False

6. A page (design view) can be laid down anywhere on a master page.

Determine whether this statement is true or false.

- True
- False

7. Dynamic content that can "grow" (like tables) should be included in pages (design view).

Determine whether this statement is true or false.

- True
- False

8. To prevent elements from being split by a page break, wrap them in a subform and uncheck *Allow Page Breaks within Content*.

Determine whether this statement is true or false.

- True
- False



Answers

1. Which of the following statements about Adobe LiveCycle Designer are true?

Answer: A, B, C

See relevant passages in the lesson “Designer: Overview”.

2. Do you agree with the following statements on palettes?

Answer: A, B, D

See relevant passages in the lesson “Designer: Overview”.

3. Which of the following statements about form properties are true?

Answer: B, C

See relevant passages in lesson “Designer: Overview”.

4. At runtime, every master page of a form is processed once.

Answer: False

Master pages can be processed from zero to an indefinite number of times.

5. Every master page must have exactly one content area.

Answer: False

Every master page must have at least one content area. Otherwise there is no space to insert dynamic content.

6. A page (design view) can be laid down anywhere on a master page.

Answer: False

Pages (design view) can be laid down only in content areas (which, in turn, are on master pages).

7. Dynamic content that can "grow" (like tables) should be included in pages (design view).

Answer: True

Page breaks can only be processed in pages (design view).

8. To prevent elements from being split by a page break, wrap them in a subform and uncheck *Allow Page Breaks within Content*.

Answer: True

This subform must be of type *Flowed*.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Unit 5

Layout

Unit Overview

Once you know how to find your way around in Adobe LiveCycle Designer, you can start to actually design a form. We will start out with simple static elements, like static texts or static images. Their contents are determined by you at design time.

We will next continue with dynamic elements, like business data as represented in fields or texts. The dunning level or a recipient's address, for example, belong to this group, as their values can only be determined at runtime. Typically (though not exclusively), the values of dynamic form elements will be calculated in the ABAP application program.

Finally, we will look at tables, which are special dynamic elements.



Unit Objectives

After completing this unit, you will be able to:

- Insert static elements into a form: images, text and graphic objects
- Set object properties for static form elements
- Insert dynamic elements into a form: text fields, image fields, date/time fields, floating fields
- Set the data binding (the connection between the layout fields and the business data)
- Apply patterns (picture clauses) to influence field output
- Insert tables into a form
- Format tables
- Set a header for a table
- Create data-driven page breaks
- Create control levels
- Create nested tables

Unit Contents

Lesson: Static Form Elements	129
------------------------------------	-----

Exercise 5: Static Form Elements.....	135
Lesson: Dynamic Form Elements	143
Exercise 6: Dynamic Form Elements	161
Lesson: Tables	171
Exercise 7: Tables.....	181

Lesson: Static Form Elements

Lesson Overview

It is hard to imagine a form without static elements that are always processed independent of the business data. This lesson will introduce you to the static elements available in Adobe LiveCycle Designer, including images (for example for company logos or signatures that have been scanned in), text (like your company's address), and geometric objects.



Lesson Objectives

After completing this lesson, you will be able to:

- Insert static elements into a form: images, text and graphic objects
- Set object properties for static form elements

Business Example

A travel agency wants to print invoices for their customers' flight bookings using the tool "Interactive Forms". The form needs to be refined. For example, the company's logo and address need to be included.

Static Images

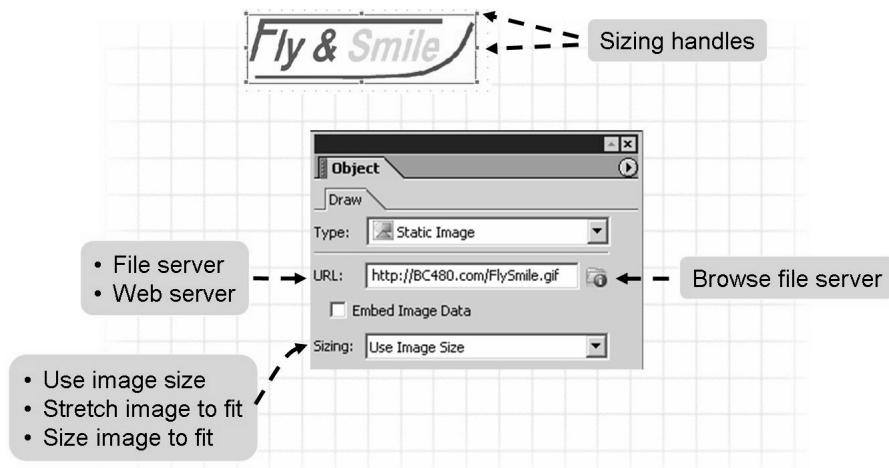


Figure 58: Inserting Static Images

As with all objects, you must first choose the page where you want to include the object. In our scenario, the company logo is to appear on all pages. Such objects are called **boilerplate** objects.

You include objects from the *Library* palette on a page by dragging and dropping them.

You can include images of the most popular types: Windows Bitmaps (*.bmp), JPEGs (*.jpg), TIFFs (*.TIF), PNGs (*.png), and GIFs (*.gif – animated GIFs are not supported) and (with Adobe LiveCycle Designer 7.1 or higher) EXIF.

For a static image, you must specify its location via a valid URL. This address can be a file name on a file server or in the intranet or Internet (as of SAP NetWeaver 2004, you cannot access the MIME Repository via a URL).

If the image file is located on a file server, you can browse your file system by clicking on the folder icon to the right of the URL address. After selecting an image, make sure to specify the complete path so that it will be found at runtime. Addresses such as C:\image.gif might work when carrying out a test in the Designer preview. However, as soon as you test the form with an application program, you will notice that C:\ refers to the server's file system on which the Adobe Document Services are executed. This is generally not the computer that you use for designing a form. Also, make sure that you enter a publicly accessible file location.

When you specify a URL, you must prefix the address with http:// or https://. Make sure this address is accessible for Adobe document services.

To avoid incorrect image addresses at runtime, you might select *Embed image*. This way you can include an image which, at its original location, is accessible to you only.



Caution: Even if you choose this option, you still have to leave the URL that you have typed in. It will not be evaluated any more; it merely serves as a reference to where the image was found originally. If you change the URL after having embedded a picture, the picture will actually be reloaded!

You have three options for the size of an image:

- *Use Original Size* – The original size of the image will be taken. Sizing handles cannot be used.
- *Scale Image Proportionally* – If you resize the placeholder for the image, the image will automatically be resized to the maximum size possible in the placeholder. The image proportions will be preserved.
- *Scale Image to Fit Rectangle* – You can determine the image size yourself. Please note that image proportions will not be preserved.

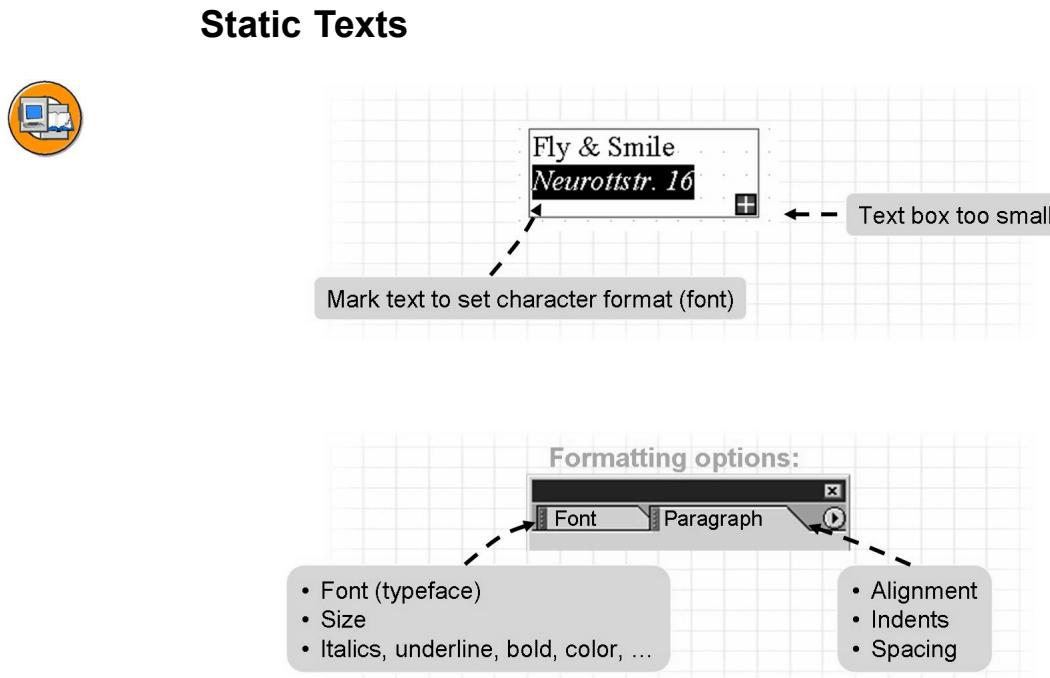


Figure 59: Inserting Static Text

In the placeholder for static text, just type your text. You need not worry about line breaks, as word wrapping is automatic. The editor is a WYSIWYG (**What You See Is What You Get**) editor, which means that the text with its layout and its line breaks looks very similar to what it will look like in the printout.

If you want to force a line break, you have two options:

- Pressing **RETURN** will insert a new paragraph. Two paragraphs are completely independent of each other as they can have individual paragraph formats.
- Pressing **SHIFT+RETURN** will insert a line break only. This means that the current paragraph is not split into two; it will remain one unit. All lines (independent of the number of line breaks) within one paragraph will have the same paragraph format.

If you include a static text, make sure the placeholder in the Layout editor is big enough. A small red plus sign will appear in the bottom right corner if it is not.

The same formatting options are available for static text that are used in common word processing programs. That is, you can choose between settings for the fonts and the paragraph.

To change the font or its appearance (like italics, underline, bold face, or color), mark the relevant text passage first and then go to the *Font* palette to determine the details.



Caution: The Adobe LiveCycle Designer preview will show the text with the fonts as they are installed on the PC. These fonts may be completely different than the ones that are available during runtime (for Adobe Document Services). To avoid confusion, use only those fonts that are installed in the operating system of Adobe document services. Safe bets are typically Times, Arial, and Courier.

Furthermore, for performance reasons, it is best to use only a small number of fonts per document.

If you want to change the paragraph format of a single paragraph, set the cursor there and go to the *Paragraph* palette. If several paragraphs need to be formatted at a time, you must mark them first. Among the options for the paragraph settings you will find options for horizontal text alignment (left-aligned, right-aligned, centered, and justified), vertical text alignment (top, middle, bottom), indents, and spacing.

Geometric Objects



Figure 60: Inserting Geometric Objects

You might want to include some graphical elements to enhance your form's layout. You can choose between lines, rectangles, and circles.

- A line might be drawn in any angle by click-dragging on one of its handles. On the *Object* palette, you will also find buttons to set the line to horizontal, vertical, or 45-degree diagonal position.
- You can set the line style, corners, and background colors for rectangles as you can for text fields.
- A circle might also be drawn in parts (arcs).

Typically, but not necessarily, these graphical elements are placed on a master page, so they act as boilerplate objects.

Exercise 5: Static Form Elements

Exercise Objectives

After completing this exercise, you will be able to:

- Insert static elements into a form: images, text and graphic objects
- Set object properties for static form elements

Business Example

A travel agency wants to print invoices for their customers' flight bookings, using the "Interactive Forms" tool. The form layout has been structured to comprise master pages and pages (design view), but it needs to be refined. For example, the company's logo and address need to be included.

Task 1:

Extend your form ZBC480_##. ## stands for your two-digit group number (monitor number).

It might be a good idea to create a local safety copy of the form every now and then. To do so, choose *Utilities* → *Downloading form* from SFP's initial screen.

If you have not finished one of the previous exercises, you can copy the template form, BC480_STRUCTURE.

The reference solution of this exercise is form BC480_STATIC.

1. Include a company logo as a static image so that it appears in the top right area of the first print page. On which page and in which area of the form would be the best place to include the image? You can search the Internet for an image you like and save it locally. If you are using a Windows Terminal Server, you will find some graphics in the BC480 directory. Name the image *LOGO*.
2. Try out different settings for the sizing.
3. Embed the image and look at the resulting XML data.

Task 2:

Integrate the sender address as a static text on master page FIRST.

1. Create a static address, *SENDER_ADDRESS*, as the address of the travel agency. Position it to the left of the company logo. Enter an address. (For perfectionists: This address should be the same as the small one that will later be displayed in the envelope. In the context, you selected SAPscript

Continued on next page

text FLY_AND_SMILE_SMALL, Object TEXT, ID ADRS for that purpose. You might want to read this text in transaction SO10 and copy its content over to the form.)

Task 3:

Create an “Invoice” text on master page FIRST.

1. Create a static text with “Invoice” in it. Place it below the company logo, but above the content area. Use a bold, 14pt font.
2. Wrap this text in a subform, *INFORMATION*. In a later exercise, this subform will also contain the customer number, the invoice reference number, and the date. To make arranging these texts easier, set the subform type to *Flowed*.
3. Shade the subform and create round borders.

Task 4:

Integrate static texts in page (design view) BODY_01.

1. Insert the opening line (“Dear sir/madam”).
2. Insert the closing (“Yours sincerely”).
3. Insert a postscript, “PS: A discount has been granted”.
4. For all three texts, select a paragraph format and a character format of your choice.

Task 5:

Prevent page break.

1. Prevent a page break between the closing and the *PS* text. Wrap these two texts into a suitable subform named *FINALLY*.

Task 6:

Test the result.

1. Activate your form and test it with report SAPBC480_DEMO. Look closely at the resulting fonts.

Task 7:

Optional: Rotate text.

1. On the left border of the first master page, create a static text with the content “Invoice template XYZ-ABC”. Rotate it by 90 degrees to a vertical position. Use a 5pt font.

Continued on next page

Task 8:

Optional: Create terms and conditions.

1. On the last master page, create a static text with the terms and conditions.

Task 9:

Optional: Include geometric objects.

1. Integrate some additional geometric objects to enhance the visual design of the invoice.

Solution 5: Static Form Elements

Task 1:

Extend your form ZBC480_##. ## stands for your two-digit group number (monitor number).

It might be a good idea to create a local safety copy of the form every now and then. To do so, choose *Utilities* → *Downloading form* from SFP's initial screen.

If you have not finished one of the previous exercises, you can copy the template form, BC480_STRUCTURE.

The reference solution of this exercise is form BC480_STATIC.

1. Include a company logo as a static image so that it appears in the top right area of the first print page. On which page and in which area of the form would be the best place to include the image? You can search the Internet for an image you like and save it locally. If you are using a Windows Terminal Server, you will find some graphics in the BC480 directory. Name the image *LOGO*.
 - a) The company logo is a read-only element (a boilerplate object). A good place would be directly on master page FIRST. In the Layout Editor, select the *Master Pages* tab. In the *Library* palette on the *Standard* tab, drag an image field onto the FIRST master page. Specify the source location of the image on the *Object* palette, *URL* field.
 - b) Go to the *Hierarchy* palette, click on the newly inserted image, and press F2 to enter **LOGO** as the name.
2. Try out different settings for the sizing.
 - a) On the *Object* palette, you will find the options *Scale Image Proportionally*, *Scale Image to Fit Rectangle*, and *Use Original Size*. You will see the difference only if you resize the image.
3. Embed the image and look at the resulting XML data.
 - a) On the *Object* palette, select the *Embed Image Data* option.
 - b) In the Layout Editor, go to the *XML Source* tab. You should see the unreadable hexadecimal content of the image.

Task 2:

Integrate the sender address as a static text on master page FIRST.

1. Create a static address, *SENDER_ADDRESS*, as the address of the travel agency. Position it to the left of the company logo. Enter an address. (For perfectionists: This address should be the same as the small one that will

Continued on next page

later be displayed in the envelope. In the context, you selected SAPscript text FLY_AND_SMILE_SMALL, Object TEXT, ID ADRS for that purpose. You might want to read this text in transaction SO10 and copy its content over to the form.)

- a) In the Layout Editor, select the *Master Pages* tab. In the *Library* palette on the *Standard* tab, drag a static text onto the FIRST master page.
- b) Click on the newly created text in the *Hierarchy* and press F2 to rename it.
- c) To check the SAPscript text, run transaction SO10. To enter the text in the form, click on the static text element in Designer's Layout Editor and type in the text.

Task 3:

Create an “Invoice” text on master page FIRST.

1. Create a static text with “Invoice” in it. Place it below the company logo, but above the content area. Use a bold, 14pt font.
 - a) In the Layout Editor, go to the *Master Pages* tab.
 - b) Drag a static text from the *Library* palette, *Standard* tab to the master page FIRST. Position it with the mouse.
 - c) Type in the text.
2. Wrap this text in a subform, *INFORMATION*. In a later exercise, this subform will also contain the customer number, the invoice reference number, and the date. To make arranging these texts easier, set the subform type to *Flowed*.
 - a) In the *Hierarchy* palette, right-click on the newly inserted text and choose *Wrap in Subform*.
 - b) On the *Object* palette, *Subform* tab, choose *Flowed*.
3. Shade the subform and create round borders.
 - a) Use the *Border* palette. Make sure to set the radius to more than 0 cm.

Task 4:

Integrate static texts in page (design view) BODY_01.

1. Insert the opening line (“Dear sir/madam”).
 - a) In the Layout Editor, go to the *Design View* tab.
 - b) From the *Library* palette, *Standard* tab, drag a static text to the page (design view) BODY_01.
 - c) Type in the text.

Continued on next page

2. Insert the closing (“Yours sincerely”).
 - a) See previous steps.
3. Insert a postscript, “PS: A discount has been granted”.
 - a) See previous steps.
4. For all three texts, select a paragraph format and a character format of your choice.
 - a) Mark the text to be formatted. Set the formatting details on the *Paragraph* and *Font* palettes.

Task 5:

Prevent page break.

1. Prevent a page break between the closing and the *PS* text. Wrap these two texts into a suitable subform named *FINALLY*.
 - a) Mark both texts. Right-click on them in the *Hierarchy*. Select *Wrap in Subform*.
 - b) In the *Hierarchy*, click on the newly created subform. Go to the *Object* palette, *Subform* tab and deselect the *Allow Page Breaks within Content* option.

Task 6:

Test the result.

1. Activate your form and test it with report SAPBC480_DEMO. Look closely at the resulting fonts.
 - a) Press Ctrl+F3. In a second SAP GUI mode, run transaction SA38. Enter **SAPBC480_DEMO** as the name of the report. Choose *Execute* (F8).
 - b) Enter **ZBC480_##** on the selection screen and press F8 again.
 - c) Exotic fonts are most likely not installed on Adobe document services. You can select them in the Designer, but they will be replaced by normal fonts when handled by Adobe document services.

Continued on next page

Task 7:

Optional: Rotate text.

1. On the left border of the first master page, create a static text with the content "Invoice template XYZ-ABC". Rotate it by 90 degrees to a vertical position. Use a 5pt font.
 - a) Create a static text like you did before.
 - b) Go to the *Layout* palette, *Size & Position* area. You can select the anchor (fulcrum) and then set the rotation angle of 90 degrees.

Task 8:

Optional: Create terms and conditions.

1. On the last master page, create a static text with the terms and conditions.
 - a) Proceed as before.

Task 9:

Optional: Include geometric objects.

1. Integrate some additional geometric objects to enhance the visual design of the invoice.
 - a) Geometric objects are read-only boilerplate objects. They are typically included on master pages.
 - b) From the *Library* palette, *Standard* tab, drag a line, a circle, or a rectangle to a page.



Lesson Summary

You should now be able to:

- Insert static elements into a form: images, text and graphic objects
- Set object properties for static form elements

Lesson: Dynamic Form Elements

Lesson Overview

Virtually all business printing scenarios require the use of dynamic data – otherwise a photocopier would be your tool of choice, and not Adobe LiveCycle Designer/transaction SFP. This lesson will introduce you to the dynamic elements that can be included on a page, such as text fields or image fields. You will also learn how these elements are populated with dynamic data.



Lesson Objectives

After completing this lesson, you will be able to:

- Insert dynamic elements into a form: text fields, image fields, date/time fields, floating fields
- Set the data binding (the connection between the layout fields and the business data)
- Apply patterns (picture clauses) to influence field output

Business Example

A travel agency wants to print invoices for their customers' flight bookings using the tool "Interactive Forms". The form needs to be refined. For example, the current date and customer's address need to be included

Integrating Dynamic Fields



Drag & Drop from Data View → automatic link between form context and layout

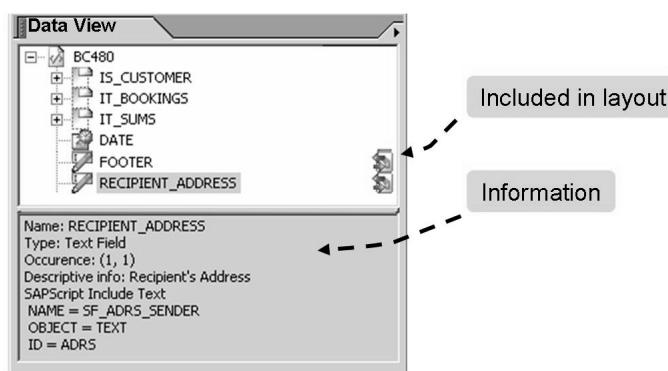


Figure 61: Inserting Fields with the Data View

There are two ways to insert a dynamic field:

1. Go to the *Data View* (which results from the form context), and drag a field to the desired position in the *Layout Editor*.
2. Drag a field from the *Standard* library to the desired position.

Dynamic elements can correspond to a number of elements from the context. For example, an image field can refer to a field of type *graphic content* or *graphic reference*. That is why it is best to use the first method, as this will automatically create the appropriate link between the context element and the layout element: the so-called data binding.

An element from the context that has been included somewhere in the layout will be displayed in the *Data View* with a special symbol to the right of its name. Note, however, that it is possible to include a field in the layout several times.

If you select an element in the *Data View* that has been included somewhere in the layout, its details will be shown in the *Layout*, *Border*, and *Object* palettes (provided you have opened these palettes). For some types of fields, the *Font* and *Paragraph* settings will also be displayed. If an element has been included several times, you might find the entry *mixed* in some fields.

You can display detail information for fields from the *Data View* by right-clicking on them and selecting *Show Info*. For example, descriptive texts will be displayed for context fields with reference to the ABAP Dictionary. For external texts, their types (SAPscript, text module, ...) and their key identifiers will be shown.

You set details for the dynamic element on the *Object* palette. What you can actually enter there depends on the type of the element.

Text Fields

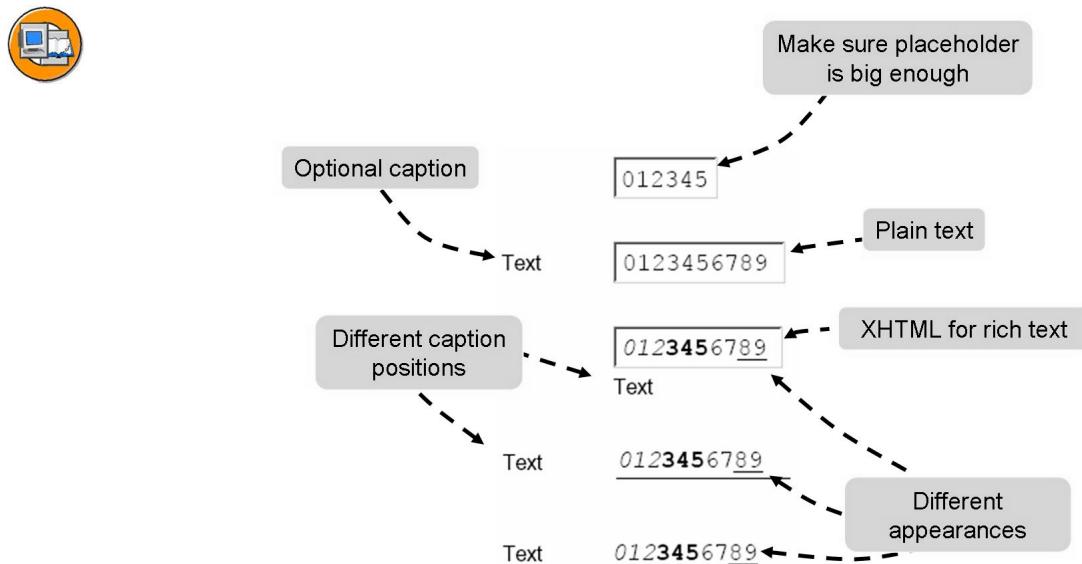


Figure 62: Text Fields

A text field can refer to a simple string variable, a text module, a SAPscript text, a dynamic text, or an address from the Business Address Services. The exact type of it is irrelevant for Adobe LiveCycle Designer; all types have the same handling. Among the details you can set for a text field, you will find:

- *Layout* palette
 - *Width and Height*

A text fields might correspond to a text module or a SAPscript text whose exact length might be unknown at design time. If you do not know for sure the height and width, choose *Expand to fit* in the *Layout* palette under *Size & Position*. If you know the text consists of a single line, it will be sufficient to allow the text to grow in its horizontal dimension. Be aware that whatever you enter as the width of this text will be overwritten at runtime, which means that the width you enter is merely a rough estimate of how wide you think the text will be (it will be considered the minimum dimension).

- *Caption*

This is a static text as an accompaniment to the dynamic field content. From SAP NetWeaver 2004s on, this caption is automatically taken from the ABAP Dictionary if you drag an element from the *Data View* which has a Dictionary reference. A caption should not be confused with a tooltip that helps visually impaired people to read a form. A tooltip can be included independent of a caption.

If you set a caption, you must choose one of four positions for the text field: top, bottom, left, and right. You must also determine how much space should be reserved for the caption.

- *Object* palette

- *Field* tab

- *Appearance*

You would typically choose a specific appearance in interactive forms only, so that the user of the form can distinguish more easily whether text is ready for input or static. This distinction is usually irrelevant for print forms.

- *Allow multiple lines*

If selected, make sure to provide enough space. You might want to set the size properties to *Expand to fit*.

- *Allow plain text only*

Irrelevant

- *Limit length*

Relevant only if *XHTML* is deselected on the *Binding* tab. This allows you to determine the maximum length of the resulting text.

If you integrate context fields that have a reference to the ABAP Dictionary, you might find that the length specified in the Dictionary will be copied.

If you are acquainted with output options from SAPscript or Smart Forms, the *Limit Length* setting is the equivalent of (*OutputLength*), like &symbol(8)&.

- *Value* tab

- The *Type* is relevant for interactive scenarios only.

- The default value will be taken for fields with no or corrupt data binding. It does not work for empty fields; they will remain empty.

- *Binding* tab

- *Default binding*

You determine the link to the *Data View* (that is, the form context) here.

- *Data format*

Select *XHTML* for *Data Format* if you want to preserve the formatting of your text, including line breaks or font settings. If you drag a text field from the Data View that is a formatted text (address node, Smart Forms text, SAPscript text, or dynamic text), this option will automatically be selected.

If you select XHTML, the only text formats you can alter in the Font and Paragraph palettes are the formats not explicitly set in the text. If, for example, you have already explicitly set a font color in the text module being used, you can no longer override the color.

You should not choose XHTML if the text contains no formatting. Otherwise, the text will look right, but performance will be poorer.

If you want to set the font or paragraph for the caption and the field separately, you must click on the small triangle on the *Font* or *Paragraph* palette and select the *Edit Caption* or *Edit Value* option. The default is *Edit Caption and Value*.

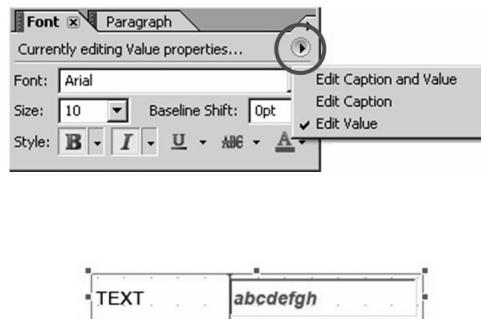


Figure 63: Editing Text Caption and Text Value

Data Binding

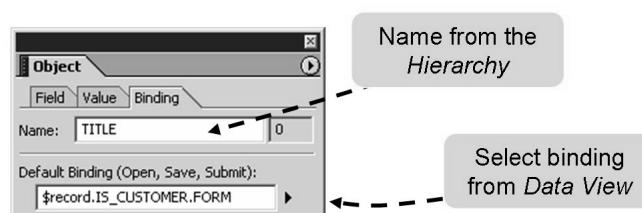


Figure 64: Data Binding

All dynamic fields (such as text fields or image fields) must have a data binding. This determines the link between the field in the layout and the field from the *Data View* (that is, the form context). In other words, it makes sure the proper data is displayed at runtime.

Data Binding



- Link between objects and data connection (= SFP context)
- Created automatically when field is dropped to layout from the *Data View*
- Can be set manually: in the *Object* palette, *Binding* tab
- Case-sensitive
- Absolute names must start with *\$record*.
- Relative names are possible for elements of subforms

If you integrate dynamic fields into your layout by dragging them from the *Data View*, the correct data binding will be inserted automatically. However, there are cases when you have to set the data binding manually:

- If you insert a dynamic field from the *Library* palette
- If you copy and paste an existing element (the binding will not be duplicated)
- If you include elements in subforms and later move them to another subform or unwrap the subform.

The easiest way to set the data binding manually is to click on the small triangle to the right of the *Data binding* input field and then select the appropriate field from the context.

For data binding, **absolute** names are possible for all elements, and **relative** names are possible for those elements that are included in subforms. Absolute names start with *\$record*. *\$record* represents the current record of a collection of data (for example, the customer for whom the form is being printed).

If you set the binding to *Normal* (also called the implicit data binding), the name of the object must correspond to a field from the *Data View* (that is, the context). Be careful when renaming objects with binding *Normal*. Unless the corresponding element from the *Data View* (that is, the form context) is renamed as well, you will corrupt the data binding.

For elements from the *Data View* that you want to include several times in your layout, the data binding must be set to *Global*.

A faulty data binding or binding set to *None* will lead to wrong or empty data.

Let us assume there is a top-level structure called *SUB1* in the *Data View* (context) with one field of name *OBJ1*. If, in the *Hierarchy*, a subform is called *SUB1* and an object that is included in that subform is called *OBJ1*, all of the following data bindings would be possible:

- | | |
|--|--|
| <ul style="list-style-type: none"> - SUB1: <i>normal</i> - SUB1: <i>normal</i> - SUB1: <i>\$record.SUB1</i> - SUB1: <i>\$record.SUB1</i> | <ul style="list-style-type: none"> OBJ1: <i>normal</i> OBJ1: <i>OBJ1</i> OBJ1: <i>OBJ1</i> OBJ1: <i>\$record.SUB1.OBJ1</i> |
|--|--|

The second and third options shown for OBJ1 are relative because they do not start from the top node.

An easy way of updating the data binding of a dynamic element that has been included (and changed) in the layout is dragging and dropping the appropriate context element from the *Data View* to the field. You can also update name, caption, and type in this way.

From Adobe LiveCycle Designer 7.1 on, you can set fields as the source for certain properties, like captions or tooltips. Instead of setting a fixed caption, you might want to have different values, depending on the data source. Whenever you can bind a property to a field from the *Data View*, the property will be underlined. Click on the property to set the binding. Alternatively, drag an element from the *Data View* and drop it on the property.

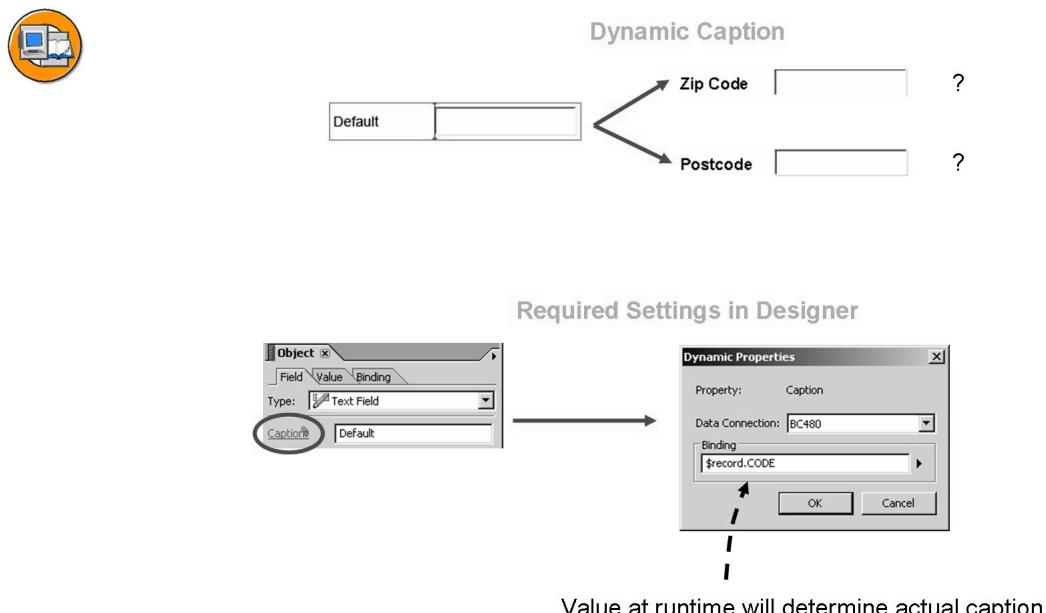


Figure 65: Dynamic Properties

If you enter a default property and also a dynamic property, the default will be taken at runtime only if the data binding is incorrect.

Changing an object property via context binding should be preferred to scripting with if-then-else commands. (You can still override a dynamic property via scripting, though.)

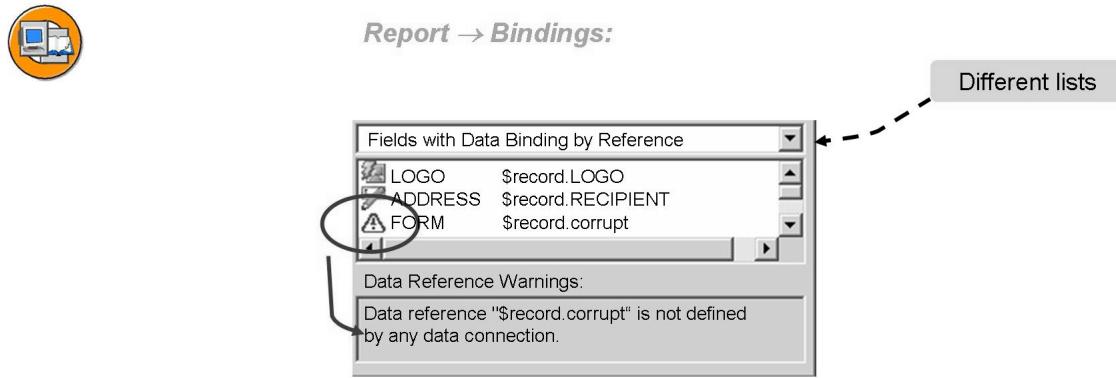


Figure 66: Checking the Data Binding

In the *Report* palette, *Binding* tab, you will find different lists for the fields in the layout:

- *Fields with Normal Data Binding*
- *Fields with Data Binding by Reference*: All fields with relative or absolute path names to their corresponding fields from the *Data View* (context).
- *Fields with no Data Binding*: Type *None* makes sense only if you have scripting to determine the field value.

If the binding for a field is faulty, you will get a caution traffic sign left to its name. If you click on it, an explanation will be shown.

Finally, you can display *Unbound Data Connection Nodes*, which are all fields from the context that have not been included anywhere in the layout. If you find too many fields here, you might want to think about redesigning your context and possibly your interface.

Image Fields

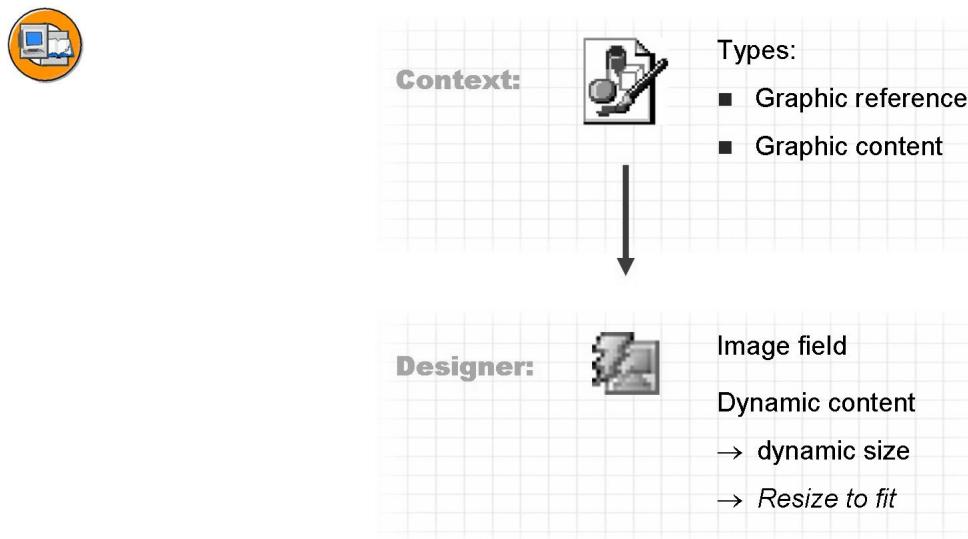


Figure 67: Inserting Image Fields

Image fields should have their equivalents in the context: either type *Graphic Content* (a variable that holds the data) or *Graphic Reference* (a variable that points to the address where the image is located). Make sure the data binding is set accordingly.

As with dynamic texts, on the *Layout* palette you can check *Expand to fit* for the width and height of the image. The size that you determine in the layout will then be considered the minimum size of an image. At runtime, it might be considerably larger or smaller.

Even though you can set a URL and even select *Embed Image Data* for image fields, these options will be ignored unless you set the binding to *None*. However, in this case, it would make more sense to choose type *Static image*.

Numeric Fields

What has been said about image fields and text fields (apart from their fomatting) is basically also true for numeric fields and date/time fields. Some more details:

There are three types of numeric fields:

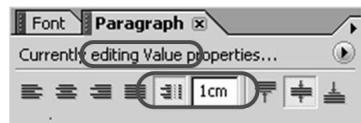
- Decimal fields
- Float
- Integer

The type is defined on the *Object* palette, *Binding* tab.

As decimals and currency amounts are treated in a special way in an SAP system, there is also a special type in Designer called decimal field; it is available in the *Library* on the *Standard* tab. Always integrate your fields by dragging them over from the *Data View*, as the correct type (and potentially some required scripting for correct decimals) will be inserted automatically.

If you want several numeric fields (typically within one table column) to be aligned so that their decimal signs are always at the same horizontal position, proceed as follows:

1. Mark all numeric fields in question.
2. Go to the *Paragraph* palette and make sure that *Edit attributes* is set.
3. Click on *Radix alignment* and enter how much space in the right corner of the value area should be reserved for the decimals.



Numeric field	12.345,67
Numeric field	12,34
Numeric field	0,12

Decimal separator depends on locale

Figure 68: Radix Alignment for Numeric Fields

Date and Time Fields

The *Standard Library* allows you to integrate date/time fields. On the *Object* palette, *Binding* tab, you determine one of three types:

- Date
- Time
- Date and Time

ABAP has types d and t, and there are types DATS and TIMS as the equivalent ABAP Dictionary types. If you use ordinary date or time fields from the context, they will correspond to one of these SAP types, and their format will be automatically converted to the correct input format so that it can be interpreted correctly at runtime for the PDF-based form. (Note that there are no standard ABAP or Dictionary types that combine both date and time.)

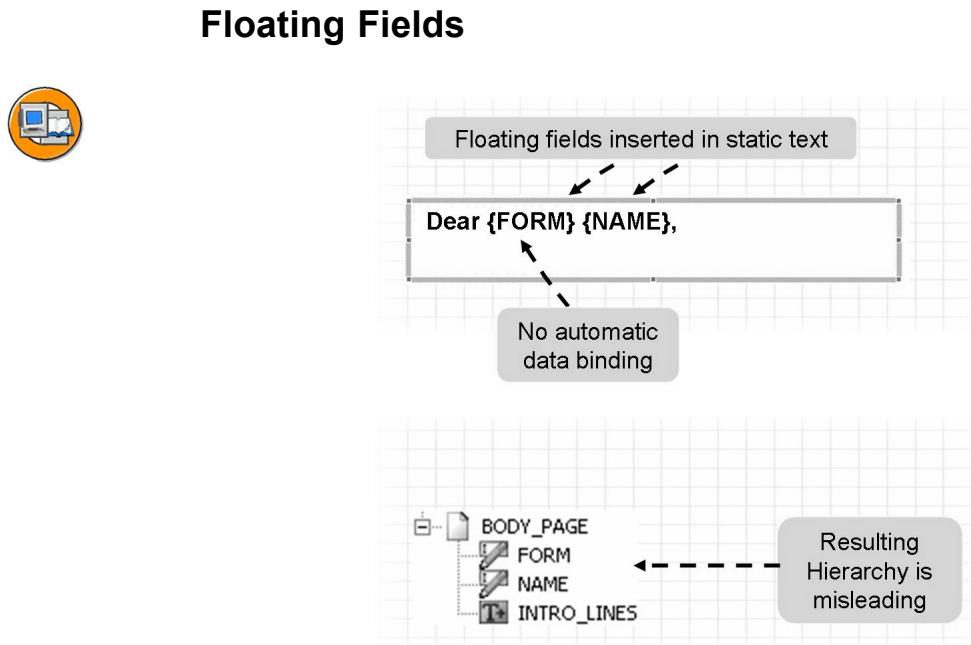


Figure 69: Static Texts with Floating Fields

Sometimes it is necessary to include fields in static texts. For example, you want to avoid something like “Dear sir or madam” and rather address your customers with their real names. You achieve this by using floating fields (fields in body text):

1. In an existing static text, set the cursor to the position where the floating field should be inserted.
2. Choose *Insert → Floating Field* for every field that you want to integrate.
3. Within the static text, place the cursor on one of the text fields just created (make sure it is marked). Then, in the *Object palette, Binding tab*, enter the appropriate data binding. It is safest to do this by clicking on the small triangle to the right of the *Binding* field and then choosing a field from the form context.
4. Do not be confused that in the *Hierarchy*, the floating fields appear as individual elements; it is not discernible that they have been included in a static text. To make things a bit clearer, you might want to specify a meaningful name for the fields. You can do so by right-clicking on them and choosing *Rename* from the context menu.

Floating fields are automatically text fields. If you want to have a different type, go to the *Object palette, Field tab* and change the type there. The type is also adjusted if you update the binding. The type can also be changed automatically if you update the data binding.

You should not worry that on the *Object palette, Field tab*, the value for *Presence* is automatically set to *Hidden (Exclude from Layout)*. Your floating text field will appear, but not as a separate layout element.

Page numbers are also inserted as floating fields, together with some scripting. Set the cursor into a static text, then choose *Insert → Current Page Number* or *Number of Pages*.

Manipulating the Output Format

Formatting Options



- Formatting output of fields can be performed:
 - With patterns (date formats, omitting of leading zeros, ...)
 - Via checking *Limit length* on the *Field* tab of the *Object* palette (output length)
 - In the ABAP program (offsets, compression of blanks)

You cannot use output options for character fields as you might know them from SAPscript or Smart Forms, like specifying an offset (*name+5*). Instead, use the ABAP program and/or ABAP coding at the form's initialization to format your fields according to your needs.



2010-12-15 or Sonntag, den 15. Dezember 2010?

2127778899 or +1 (212) 777-8899?

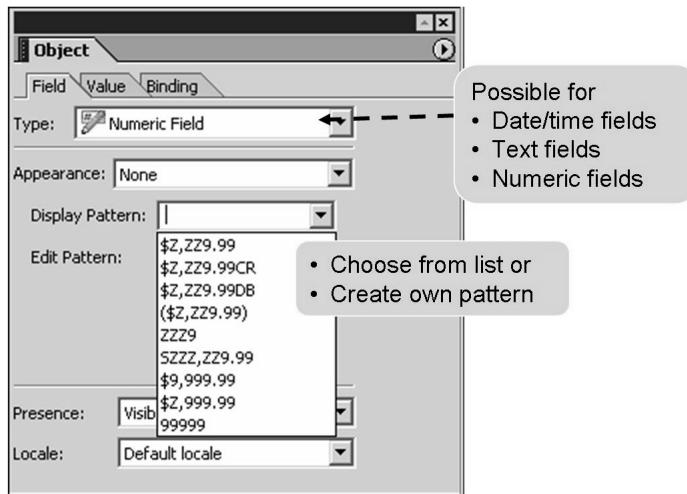


Figure 70: Display Patterns

The display pattern describes how data at runtime will be displayed in the document. There is a set of rules-based specifications to govern the formatting and parsing of **date**, **time**, **numeric data**, and **text data**.

Example: The current date (which in ABAP has the format YYYYMMDD, for example 20101215) can be displayed as “2010-12-15”, “Sonntag, den 15. Dezember 2010” or “Sunday, December 15, 2010”.

In some parts of the documentation, you will also find the term *picture clauses* instead of display pattern.

Display patterns are optional and should be used only when necessary. If you do not use them, a standard format will be used based on the locale (the combination of language and country). You specify a display pattern on the *Field* tab of the *Object* palette. The drop-down list lets you select from some typical display patterns, but you are free to type in your own.

The *Edit Pattern* on the *Field* tab and the *Validation Pattern* on the *Value* tab are for interactive forms only. They are irrelevant for print forms.

The *Data Pattern* on the *Binding* tab defines how a data field should be read. As all date/time fields coming from the context are automatically parsed correctly, you should not manually set the *Data Pattern*.

Display patterns consist of

- Special symbols, dependent on the type of the field. For example, MMMM stands for the full month name. These symbols will be resolved at runtime according to the appropriate locale. For instance, MMMM could result in January, Januar, or Janvier.
- Literals. To include a phrase in a pattern, delimit the text string with single quotation marks ('...'). For example, 'Your payment is due no later than' MM-DD-YY.
- The comma (,), dash (-), colon (:), slash (/), period (.), and space () are treated as literal values and can be included anywhere in a pattern.

Be aware that display patterns cannot be translated.

All patterns are case-sensitive.

After typing in the desired display pattern, make sure to press Enter. Adobe LiveCycle Designer will then check whether the pattern is syntactically correct.

Make sure the field for which you define the pattern has enough space, in particular if you use a long display pattern. To find out if enough space has been provided, activate the form and run a test program.

Sometimes data can come in various forms, so that one single simple pattern would not work. For example, number plates vary a lot from state to state. That is why you can also write patterns with several options. At runtime, the pattern that matches the data will be taken. The syntax for defining a number of acceptable patterns is as follows: <category_name>{<pattern>}|<category_name>{<pattern>}. You can specify an unlimited number of patterns. <category_name> can be *date*, *time*, *num*, or *text*. For example, *text* {AA-99} | *text* {AAA-999} *text* {AA-99} | *text* {AAA-999}



Date Pattern Symbols

D	One- or two-digit (1-31) day of the month
DD	Zero-padded two-digit (01-31) day of the month
M	One- or two-digit (1-12) month of the year
MM	Zero-padded two-digit (01-12) month of the year
MMM	Abbreviated month name
MMMM	Full month name
EEE	Abbreviated weekday name
EEEE	Full weekday name
YY	Two-digit year
YYYY	Four-digit year

The above table shows some date pattern symbols. For a full list, consult the Adobe LiveCycle Designer online manual.



Time Pattern Symbols

h	One- or two-digit (1-12) hour of the meridian (AM/PM)
hh	Zero-padded two-digit (01-12) hour of the meridian (AM/PM)
H	One- or two-digit (0-23) hour of the day
HH	Zero-padded two-digit (0-23) hour of the day
M	One- or two-digit (0-59) minute of the hour
MM	Zero-padded two-digit (00-59) minute of the hour
S	One- or two-digit (0-59) second of the minute
SS	Zero-padded two-digit (00-59) second of the minute
A	Meridian (AM or PM)
Z	Abbreviated time-zone name, for example, GMT, GMT+05:00, GMT-00:30, EST, PDT

The above table shows some time pattern symbols. For a full list, consult the Adobe LiveCycle Designer online manual.



Pattern Examples (Date/Time Fields)

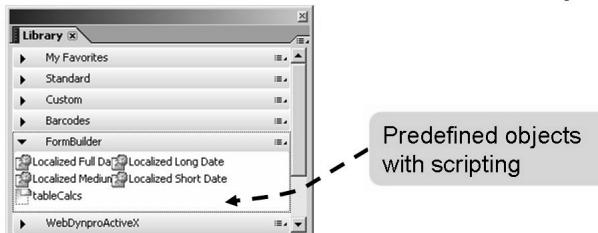
Custom pattern	Input value (ABAP)	Resulting formatted value	
		Locale en_US	Locale de_DE
MMM DD, YYYY	20101215	Dec 15, 2010	Dez 15, 2010
EEEE, 'der' D. MMMM YYYY	20101215	Sunday, der 15. December 2010	Sonntag, der 15. Dezember 2010
h:MM A	091030	9:10 AM	9:10 AM

The table above shows some custom patterns that you can define for formatting date and time values using a display pattern. Note in particular that only the special symbols (DD, EEEE, and so on) are locale-dependent. The sequence of the individual parts of the patterns remains unchanged in all languages, and everything you include in single quotes will remain unchanged as well.



Different languages/countries
require different display patterns

→ use automatic scripting!



	en_US	de_DE
Short	12/15/10	15.12.10
Medium	Dec 15, 2010	15.12.2010
Long	December 15, 2010	15. Dezember 2010
Full	Sunday, December 15, 2010	Sonntag, 15. Dezember 2010

Figure 71: Predefined Formatting of Date Fields

Patterns cannot be adjusted when a form is translated. As a consequence, setting a fixed pattern will result in extra effort. On the other hand, in some cases you would like to set some features of the date. In a table, for example, you might want to make sure a date field fits into a cell. To combine the flexibility of a localized date format with a specific date length, a script is needed. That's why Designer, as part of an SAP GUI, comes with four predefined date objects, which are assembled in the *Library* palette on a tab named *FormBuilder*.

You insert a predefined date object just as you would insert a standard object or one of your own custom objects: by dragging and dropping it to its position on any page.

The FormCalc scripting that is executed at event *initialize* is as follows:

```
$.format.picture = DateFmt(1, $.locale); // for display  
$.ui.picture = DateFmt(1, $.locale); // for user input
```

Instead of digit 1 for short format, you would have 2 for medium, 3 for long, and 4 for full.



Hint: A pattern that you might use in addition to the coding will be overridden by the scripting.

Country-Specific Formatting with No Patterns



- If you use neither display patterns nor coding, medium format will be used for dates.
- If there are no pattern entries, data is formatted according to the specified locale.
- Prerequisite: The locale is specified in the function module's interface and/or by the ABAP command SET COUNTRY..

A locale is a combination of a country and a language. Be sure to set it in your ABAP program (even though there are various places in Designer where you could set it for the form or parts of it). For example, you have customers in various countries and you would like to print a date on an invoice according to the rules of your customer's country.

The way you determine a locale depends on the interface type used:

- ABAP Dictionary interface type: Set the *langu* and *country* fields of structure /Ibcdwb/docparams. This structure is an import parameter of the form's generated function module.
- Smart Forms compatible interface type): Use ABAP command SET COUNTRY to determine the country version of the form.

Use the *langu* field of structure *control_parameters* to determine the language. This structure is an import parameter of a form's generated function module if a Smart Forms-compatible interface is used.



Numeric Pattern Symbols

9	A single digit, or a zero digit if empty or a space
8	Currency-specific or unit-specific output of decimal places (takes ABAP Dictionary references into account)
z	A single digit, or nothing if empty, a space, or the zero digit
Z	A single digit, or a space if empty, a space, or the zero digit
s	A minus sign if the number is negative, and nothing otherwise
S	A minus sign if the number is negative, and a space otherwise
\$	The currency symbol of the ambient locale
,	The grouping separator of the ambient locale
.	The decimal radix of the ambient locale

The above table shows some numeric pattern symbols. For a full list, consult the Adobe LiveCycle Designer online manual.

To evaluate currency-specific or unit-specific information on decimal places, however, you must explicitly use display pattern 8. (Background: ABAP distinguishes between internal and external representations of currencies and quantities.) For example, the internal digit of 1000 might be 10.00 USD or 1000 JPY when printed.)

ABAP Dictionary information about decimal places and output lengths are evaluated automatically: they are represented by the options *Limit Leading Digits* and *Limit Trailing Digits*. Make sure to uncheck these options if you want your own display pattern to work.

You must give the maximum number of decimals that you expect. So if you have a field (type *decimal field*) that at run time can have amounts with, say, USD and JPY, a suitable display pattern could be zzz,zzz,zz9.88.



Pattern Examples (Numeric Fields)

Custom pattern	Input value	Formatted value	
		en_US	de_DE
S999.9	1.55	“001.6”	“001,6”
	155	“155.0”	“155,0”

Custom pattern	Input value	Formatted value	
		en_US	de_DE
SZZ9.99	15.5	“15.50”	“15,50”
	-1.55	“- 1.55”	“- 1,55”
szz9.99	155	“155.00”	“155,00”
	-155	“-155.00”	“-155,00”
\$9,999.99	1.55	“\$0,001.55”	“€0.001,55”

The above table shows some custom patterns that you can define for formatting numeric values. The formatted results are delimited by double quotation marks so that you can see where spaces would appear in the result. The quotation marks are not actually part of the result.



Text Pattern Symbols

A	Single alphabetic character
X	Single character
O or 0	Single alphanumeric character
9	Single digit

Exercise 6: Dynamic Form Elements

Exercise Objectives

After completing this exercise, you will be able to:

- Insert dynamic elements, including text fields, image fields, date fields, and floating fields, into a form
- Set the data binding
- Apply patterns

Business Example

A travel agency wants to print invoices for their customer's flight bookings using the tool "Interactive Forms". The form has been set up, but it lacks all data that is relevant for the business process. The form designers will now have to integrate dynamic content, such as the customer's address or the date.

Task 1:

Change the image type of the company logo.

Continue to work with your own form, ZBC480_##. ## stands for your two-digit group number (monitor number).

It might be a good idea to create a local safety copy of the form every now and then. To do so, choose *Utilities* → *Downloading form* from SFP's initial screen.

If you have not finished one of the previous exercises, you can copy the template form instead: BC480_STATIC. The reference solution of this exercise is form BC480_DYNAMIC.

In between the steps of this exercise, you might want to activate your form and test it with report SAPBC480_DEMO (or with Designer's preview, provided you have set an XML test data file in the form properties).

You will find a detailed testing scenario at the end of this exercise.

1. Change the type of the image *LOGO* (which so far was static). Set it to *Image field*. Create the correct data binding to the context variable *LOGO*.

Task 2:

Fill the *INFORMATION* box on master page *FIRST*.

Integrate the following elements (vertically arranged and left-aligned). Make sure the subform is wide enough.

1. The customer number as a text field with caption, but without a frame. Refer to the context field *IS_CUSTOMER-ID*.

Continued on next page

2. “Our reference” as a text field *REFERENCE* with caption, but without a frame. Set a default value of your choice.
3. The date. Refer to the *DATE* field of the context.

Task 3:

Integrate the customer address.

1. In the top left area of master page *FIRST*, below the travel agency's address, include the customer's address. (Reminder: this address – formatted according to the rules of the customer's country – should be in the context field *CUSTOMER_ADDRESS*.)

Task 4:

Output the clerk's name.

1. Below the closing lines of the invoice, insert the clerk's name. You will find it in the context constant *GC_CLERK*.

Task 5:

Optional: Integrate travel agency's small address for the envelope window.

1. On master page *FIRST*, between the travel agency's full address and the customer address, include the travel agency's small address intended for the envelope window. Refer to context field *ENVELOPE_ADDRESS*. Make sure space is sufficient.

Task 6:

Display page numbers.

1. On the top right margin of master page *NEXT*, create a static text *PAGE* with the content “Page of”. Integrate the current page number and the total number of pages, so that at runtime, the result will look similar to “Page 2 of 3”.



Note: You can view the result with report SAPBC480_DEMO only once you have integrated the bookings (which you will do later). Until now, no page break will be processed.

Task 7:

Personalize the discount note.

1. Up until now, always the same sentence was printed for the discount (element *PS*). With the help of a floating field, include the actual discount rate. (The relevant context field is *IS_CUSTOMER-DISCOUNT*.)

Continued on next page

Take care to integrate a numeric field.

Make sure leading zeros are omitted (with the help of a display pattern).

Task 8:

Optional: Personalize the introductory greeting.

1. Change your existing introductory greeting (“Dear sir/madam”) in page (design view) BODY_01 so that the customer is personally addressed with form and name. To achieve this, integrate two floating fields and refer to the context fields *IS_CUSTOMER-FORM* and *IS_CUSTOMER-NAME* from the context. It is up to you to set the data binding manually.

Task 9:

Integrate texts with vacation suggestions.

1. Integrate the alternative *OFFER* into the layout as the last field of page (design view) BODY_01, so that the travel agency's customers get some hints on where they should spend their next vacations.
2. Make sure the closing, the clerk's name, the text *PS* (discount note) and the alternative *OFFER* are protected against page break.

Task 10:

Make sure the date is displayed in a country-specific format.

1. From the *INFORMATION* subform on master page *FIRST*, remove the “plain” date field included in a previous task.
2. Use one of the special objects in the *Library* palette, *Form Builder* tab (which consists of a neutral date field plus scripting) to display the date in localized medium format.

Task 11:

Perform a final test.

You have finally come to the point where you can harvest the fruits from the preceding lessons and evaluate the output.

1. Activate your form and test it with report SAPBC480_DEMO.

Is all the customer information printed correctly? In particular, have the form and the name been integrated correctly? Are the page breaks OK?

Test with a German customer (e.g. number 1) and an English customer (e.g. number 5). You should get different vacation tips.

Continued on next page

Test with customers with different discounts, for example numbers 2 and 3. You should get different *PS* texts.

2. On the selection screen of the report, enable the advanced functions (via the button). On the *Data source* tab, you can select a URL for the company logo. Try both choices.

On the *Language/Country* tab, you can set the sending country. You might choose *DE* and *GB*, for instance. Your invoices for customers 1 and 5 should be mailed as domestic mail in one case, and as international mail in the other. (You can evaluate this in the customer address field.)

Solution 6: Dynamic Form Elements

Task 1:

Change the image type of the company logo.

Continue to work with your own form, ZBC480_##. ## stands for your two-digit group number (monitor number).

It might be a good idea to create a local safety copy of the form every now and then. To do so, choose *Utilities → Downloading form* from SFP's initial screen.

If you have not finished one of the previous exercises, you can copy the template form instead: BC480_STATIC. The reference solution of this exercise is form BC480_DYNAMIC.

In between the steps of this exercise, you might want to activate your form and test it with report SAPBC480_DEMO (or with Designer's preview, provided you have set an XML test data file in the form properties).

You will find a detailed testing scenario at the end of this exercise.

1. Change the type of the image *LOGO* (which so far was static). Set it to *Image field*. Create the correct data binding to the context variable *LOGO*.
 - a) On the *Object* palette, tab *Field*.
 - b) On the *Object* palette, tab *Binding*.

Task 2:

Fill the *INFORMATION* box on master page *FIRST*.

Integrate the following elements (vertically arranged and left-aligned). Make sure the subform is wide enough.

1. The customer number as a text field with caption, but without a frame. Refer to the context field *IS_CUSTOMER-ID*.
 - a) Drag the field from the *Data View*. On the *Object* palette, *Field* tab, set the appearance to *None*.
2. “Our reference” as a text field *REFERENCE* with caption, but without a frame. Set a default value of your choice.
 - a) See previous step. You determine the default value on the *Object* palette, *Value* tab.
3. The date. Refer to the *DATE* field of the context.
 - a) Drag the field from the *Data View*. On the *Object* palette, *Field* tab, set the appearance to *None*.

Continued on next page

Task 3:

Integrate the customer address.

1. In the top left area of master page *FIRST*, below the travel agency's address, include the customer's address. (Reminder: this address – formatted according to the rules of the customer's country – should be in the context field *CUSTOMER_ADDRESS*.)
 - a) From the *Data View*, drag the *CUSTOMER_ADDRESS* field over to the master page *FIRST* in the layout.
 - b) On the *Object palette*, *Field* tab, set the appearance to *None*. On the *Layout palette*, set the caption to *None*.
 - c) Resize the field, either in the Layout Editor or on the *Layout* palette.

Task 4:

Output the clerk's name.

1. Below the closing lines of the invoice, insert the clerk's name. You will find it in the context constant *GC_CLERK*.
 - a) From the *Data View*, drag the *GC_CLERK* field over to the page (design view) *BODY_01*.

Task 5:

Optional: Integrate travel agency's small address for the envelope window.

1. On master page *FIRST*, between the travel agency's full address and the customer address, include the travel agency's small address intended for the envelope window. Refer to context field *ENVELOPE_ADDRESS*. Make sure space is sufficient.
 - a) From the *Data View*, drag the *ENVELOPE_ADDRESS* field to the layout.

Continued on next page

Task 6:

Display page numbers.

1. On the top right margin of master page *NEXT*, create a static text *PAGE* with the content “Page of”. Integrate the current page number and the total number of pages, so that at runtime, the result will look similar to “Page 2 of 3”.



Note: You can view the result with report SAPBC480_DEMO only once you have integrated the bookings (which you will do later). Until now, no page break will be processed.

- a) Create a static text with the help of the *Library*. To integrate the page numbers, set the cursor in the static text and choose *Insert → Current Page Number or Number of Pages*.

Task 7:

Personalize the discount note.

1. Up until now, always the same sentence was printed for the discount (element *PS*). With the help of a floating field, include the actual discount rate. (The relevant context field is *IS_CUSTOMER-DISCOUNT*.)

Take care to integrate a numeric field.

Make sure leading zeros are omitted (with the help of a display pattern).

- a) Set the cursor into the static text *PS*. From the *Insert* menu, choose *Floating field*.
- b) To set the data binding, mark the field in the *Hierarchy*, then go to the *Object* palette, *Binding* tab. From the input help for the default binding, select *IS_CUSTOMER.DISCOUNT*. To change the type of the field, go to the *Object* palette, *Field* tab.
- c) The display pattern is “zz”. You set it on the *Object* palette, *Field* tab.

Continued on next page

Task 8:

Optional: Personalize the introductory greeting.

1. Change your existing introductory greeting (“Dear sir/madam”) in page (design view) BODY_01 so that the customer is personally addressed with form and name. To achieve this, integrate two floating fields and refer to the context fields *IS_CUSTOMER-FORM* and *IS_CUSTOMER-NAME* from the context. It is up to you to set the data binding manually.
 - a) Set the cursor into the static text. From the *Insert* menu, choose *Floating Field*.
 - b) Determine the data binding on the *Object* palette, *Binding* tab: enter *\$record.IS_CUSTOMER.FORM* *\$record.IS_CUSTOMER.NAME*. If you enter it manually, make sure not to write *\$record.IS_CUSTOMER-FORM* (with a hyphen).

Task 9:

Integrate texts with vacation suggestions.

1. Integrate the alternative *OFFER* into the layout as the last field of page (design view) BODY_01, so that the travel agency's customers get some hints on where they should spend their next vacations.
 - a) Drag the *OFFER* field from the *Data View* to the layout.
2. Make sure the closing, the clerk's name, the text *PS* (discount note) and the alternative *OFFER* are protected against page break.
 - a) You should already have a subform that contains the closing and the *PS*. Both the clerk and the alternative should now also be in this subform.

Task 10:

Make sure the date is displayed in a country-specific format.

1. From the *INFORMATION* subform on master page *FIRST*, remove the “plain” date field included in a previous task.
 - a) Locate the field *DATE* in the *INFORMATION* subform on master page *FIRST* and delete.

Continued on next page

2. Use one of the special objects in the *Library* palette, *Form Builder* tab (which consists of a neutral date field plus scripting) to display the date in localized medium format.
 - a) Drag the element from the *Library*, *Form Builder* tab to the *INFORMATION* box.
 - b) Make sure the field (including its caption) is long enough. Resize the field if needed.
 - c) On the *Object* palette, *Field* tab, set the appearance to *None*.
 - d) On the *Object* palette, *Binding* tab, set the default binding to *\$record.DATE*.

Task 11:

Perform a final test.

You have finally come to the point where you can harvest the fruits from the preceding lessons and evaluate the output.

1. Activate your form and test it with report SAPBC480_DEMO.

Is all the customer information printed correctly? In particular, have the form and the name been integrated correctly? Are the page breaks OK?

Test with a German customer (e.g. number 1) and an English customer (e.g. number 5). You should get different vacation tips.

Test with customers with different discounts, for example numbers 2 and 3. You should get different PS texts.

- a) Press CTRL+F3, then open a second mode and start transaction SA38 to run the report. Press F8 to execute.

2. On the selection screen of the report, enable the advanced functions (via the button). On the *Data source* tab, you can select a URL for the company logo. Try both choices.

On the *Language/Country* tab, you can set the sending country. You might choose *DE* and *GB*, for instance. Your invoices for customers 1 and 5 should be mailed as domestic mail in one case, and as international mail in the other. (You can evaluate this in the customer address field.)

- a) ...



Lesson Summary

You should now be able to:

- Insert dynamic elements into a form: text fields, image fields, date/time fields, floating fields
- Set the data binding (the connection between the layout fields and the business data)
- Apply patterns (picture clauses) to influence field output

Lesson: Tables

Lesson Overview

In many scenarios, you will have data that is structured in rows and columns. For example, it is hard to imagine invoices or order confirmations without table-like presentations of data. In a printing scenario with PDF-based forms, the data would typically be collected in internal tables and then passed on to the generated function module of the form. The line-by-line processing is then done by Adobe document services.

This lesson will show you how to create tables and set details such as headers or borders.

 **Note:** The kinds of tables described in this lesson require Adobe LiveCycle Designer with at least version 7.1.



Lesson Objectives

After completing this lesson, you will be able to:

- Insert tables into a form
- Format tables
- Set a header for a table
- Create data-driven page breaks
- Create control levels
- Create nested tables

Business Example

A travel agency wants to print invoices for their customers' flight bookings using the tool "Interactive Forms". The form needs to be able to print a list of items.

Table Types

Basically, tables are special types of subforms that are nested. Every table consists of at least one subform for the table and one subform for a row. The individual cells are ordinary layout elements, most frequently static texts or in particular text/date/numeric fields. If a row subform can be repeated at runtime, the table will grow vertically. Such a table is called a **dynamic table**. If the number of rows is determined at design time, the resulting table is called a **static table**.



Table rows = special kinds of subforms

Repeated rows based
on available number of
datasets →

Dynamic table

ID	No.	Type
AA	17	747
LH	400	A380
UA	123	A340



Fixed number of
rows →

Static table

.....
.....
.....
.....



Wolfgang	Amadeus	Mozart
27	January	1756
5	December	1791

Figure 72: Dynamic and Static Tables

Creating a Dynamic Table

Even though you could create a table manually by using the *Object* and *Hierarchy* palettes only, it is by far easier to do so by dragging and dropping a table from the *Data View* to the Layout Editor. (The table will typically be an internal table from the form interface that is used in the context.) You can then remove the fields that you do not want to have included in the form table – but it might be a good idea to either reduce the line type of the table in the interface or to go to the context and set the superfluous fields of the internal table to inactive. Both ways will reduce the amount of data transferred to Adobe document services at runtime.

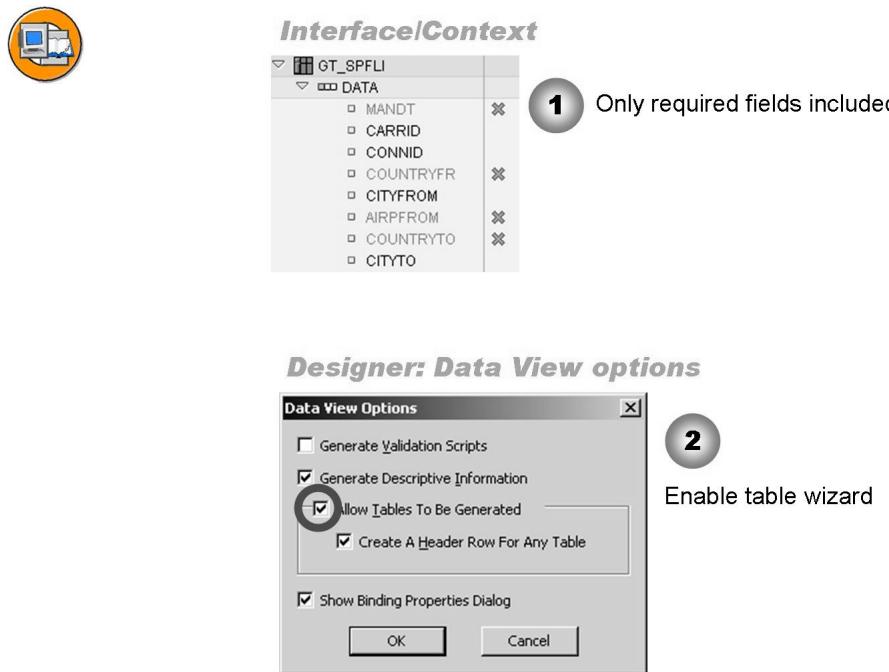


Figure 73: Creating a Dynamic Table: Preliminary Steps

You can create accessible tables by dragging an internal table from the *Data View* to the layout. In order to achieve this, go to the palette menu of the *Data View*, choose *Options...*, then *Allow Tables To Be Generated*. If this option is unchecked, Adobe LiveCycle Designer creates nested flowed or positioned subforms. You should generally have dynamic tables generated, unless you want to create multiline tables or tables that extend horizontally instead of vertically.

Drag the internal table from the *Data View* to the layout. Make sure to include it in a page (detail view) or a subform of type *Flowed* which allows page breaks – unless you know in advance that the maximum table length will not exceed one page.

Once you have it included the table, you can determine cell widths and heights by click-dragging on the cell borders. If you want to add or remove cells, set the cursor as required and use the *Table* toolbar or the *Table* menu (which has more functions).

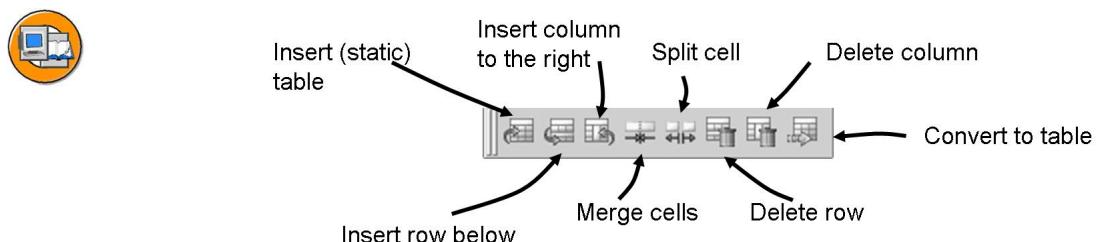


Figure 74: The Table Toolbar

For example, if you want to add another row (like a header row, for instance), set the cursor on one row in the *Hierarchy* or on a cell in the *Layout*, then choose *Table → Insert → Row Above*.

After determining the number of rows and cells you need to do some adjustments:

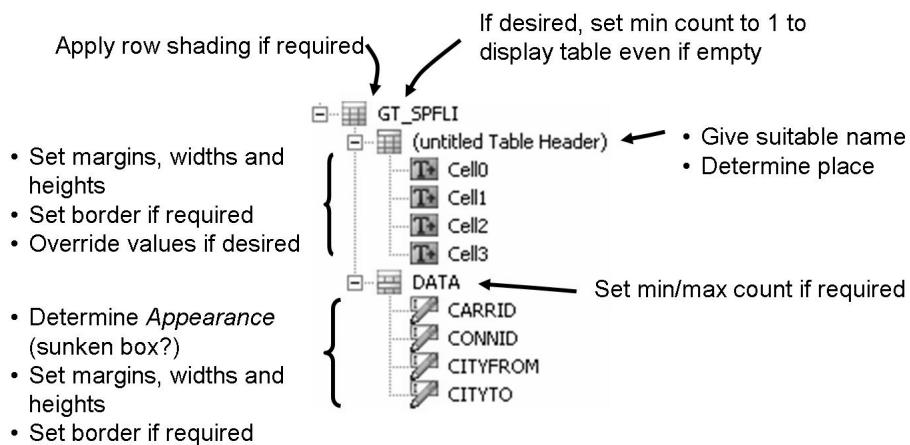


Figure 75: Manual Adjustments After Table Has Been Inserted

Settings for the table (in the example above, GT_SPFLI)

- On the *Object* palette, you can apply row shading if desired.
- On the *Binding* tab (*Object* palette), you can set *Min Count* to 1. This ensures that the table header will be printed even if the table contains no data. If *Min Count* is set to 0, the table will not be printed if it does not contain at least one row of data.
- As the table is a subform, you can protect it against page break. You do this on the *Object* palette, *Table* tab by selecting or deselecting the *Allow Page Breaks within Content* checkbox.
- If you want your table to have an outer border only, you would set it on the *Border* palette of the table.
- If you want to apply zebra stripes to your table, select the table in the *Hierarchy* palette, then go to the *Object* palette, *Row shading* tab, and check *Apply Alternating Row Shading*.

Settings for the Header Row

- You should give the header a name so that it can easily be identified when used for conditional breaks or scripting.
- On the *Object* palette, *Pagination* tab, you can determine whether the header should be used on the first and/or the subsequent pages.

Settings for the Header Fields

- In a typical scenario, you would use an internal table with Dictionary reference in your form. In that case, the cells values (column headers) will be taken from the data element texts. If you would like to change them, or if you do not have a Dictionary reference in the first place, you can enter your own values.
If you need dynamic column headers, you can change the original static texts from the header row into text fields.
- You can set margins, column widths and row heights on the *Layout* palette. The widths and heights can be changed in the Layout Editor, too, by clicking on a cell border and dropping it at the desired position. If you change the width of a header cell, the width of the corresponding data cell will be changed automatically, and vice versa.

Settings for the DATA (Body) Row

- You can set the minimum number of body rows to be printed on the *Object* palette, *Binding* tab. If at runtime not enough data is available, empty rows will result. Likewise, you can set the maximum number of rows to be rendered. If you set the minimum and maximum to the same size, you will essentially create a static table.
- You can determine conditional breaks – see further down for details.

Settings for the DATA Fields

- Depending on your *Library* settings, you might get appearances like *Sunken box* for your data fields which you might want to avoid for print forms.
- The settings of borders, margins, widths, and heights are basically the same as for header fields.

Conditional Breaks

Quite frequently, you want to have additional page breaks or intermitting headers if a condition is met. An example for such a data-driven break would be: If, within a table of flight bookings, the next dataset has a different airline, a separating empty row should be inserted.



Condition for typical scenario: table has been sorted

ID	No.	From	To
AA	0064	SAN FRANCISCO	NEW YORK
AZ	0788	ROME	TOKYO
AZ	0790	ROME	OSAKA
JL	0407	TOKYO	FRANKFURT
LH	0400	FRANKFURT	NEW YORK
LH	0401	NEW YORK	FRANKFURT
LH	2402	FRANKFURT	BERLIN
SQ	0015	SAN FRANCISCO	SINGAPORE
SQ	0988	SINGAPORE	TOKYO
UA	3516	NEW YORK	FRANKFURT
UA	3517	FRANKFURT	NEW YORK

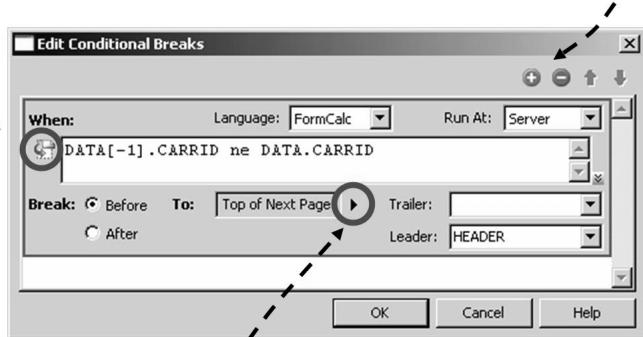
Data-driven (empty) header line inserted or page break when airline changes

Figure 76: Example of Data-Driven Breaks



Insert sample coding for table field

Insert/delete condition



List of available places to jump to

Figure 77: Setting Conditional Breaks

You set conditional breaks on the *Object* palette, *Pagination* tab of the *DATA* row (body row). By clicking on the plus sign, you add one condition. For every condition, you can choose between FormCalc and JavaScript as the scripting languages.

You can enter any logical expression that evaluates to true or false. The most frequent usage is probably to compare a particular cell value with one row with the cell in the previous or next row. For such a case, Designer can suggest some coding for all table fields, which you can then adapt to your needs.



Hint: Please note that conditional breaks based on row comparison typically make sense only if you have sorted the table before.

The places that you can jump to if the condition is met include all those that are available for normal subforms. Furthermore, you have the option to not insert a break, but just an additional header. If you want to have a header when the condition is met, select it as the leader.



Hint: The header (heading) that you set for a conditional break will not be processed for a regular page break that occurs when data does not fit on one page – unless you explicitly say so. Conversely, if for a header you set *Include Header Row in Subsequent Pages* on the *Pagination* tab, this header will be shown only during normal page break, but not by a data-driven page break.

In other words: you might have several headers, some for regular page breaks, some for data-driven breaks.

Control Levels

Tables are often not output in exactly the same structure in which they are filled. For example, it should be possible to group data records that have the same value in a sort field. Grouped data records that have certain identical values are called control levels. The easiest way to achieve this is by creating these control levels in the context and then simply drag & drop the table from the *Data View* to the layout.



Without control levels

AZ	0555	Rome	Frankfurt
AZ	0788	Rome	Tokyo
AZ	0789	Tokyo	Rome
AZ	0790	Rome	Osaka
LH	0400	Frankfurt	New York
LH	0401	New York	Frankfurt

With control levels

AZ	0555	Rome	Frankfurt
	0788	Rome	Tokyo
	0789	Tokyo	Rome
	0790	Rome	Osaka
LH	0400	Frankfurt	New York
	0401	New York	Frankfurt

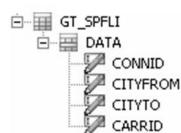


Figure 78: Control Levels: Example and Hierarchy View

A control level contains all records of the internal table that have the same value in the sort field. In the example above, all records of an airline carrier belong to one control level. A table with control levels can look like a nested table.

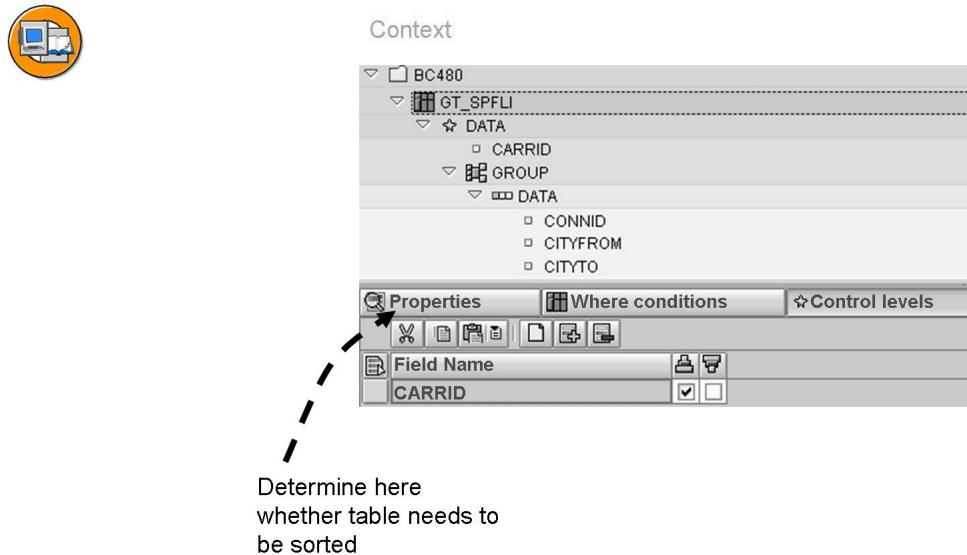


Figure 79: Setting Control Levels in the Context

In the form context, you enter the control levels on the *Control Levels* tab of a table. For every control level you can determine whether sorting (if desired) should be in ascending or descending order.

Control levels make sense only if the data is sorted according to these levels. You can sort the table yourself (in the ABAP coding of the print program or the initialization coding of the form), in which case you select *Already sorted* on the *Properties* tab of the table. Or you can have the table sorted automatically before form rendering. Deselect *Already sorted* to have the table sorted automatically. Mind, however, that the latter option is possible only if the table is either defined as a global interface field or as a call-by-value import parameter.

Nested tables



ID	No.	Departure	Destination	Dates
AA	0017	New York	San Francisco	March 29 August 15 November 6
LH	400	Frankfurt	New York	February 28 December 24
SQ	158	Singapore	Tokyo	January 6 January 17 April 11 December 5

Figure 80: Nested Table: Example

Nesting tables makes sense if, for example, you want to have a list of all flight connections and for every connection the list of its flight dates. If the form context has a nested table, you can easily drag it in LiveCycle Designer from the *Data View* to the layout to create a nested table there. How do you get a nested table into the context?

- You can create a nested table in the interface, using a nested Dictionary type or a nested local type.
- In the context, you can nest one table into an other. In that case, you must specify WHERE conditions saying how these two tables should be combined into one. You set these WHERE conditions for the inner table.



Interface: 1 internal table with a nested type

Variable Name	Type Assignment	Type Name
NESTED_ITAB	TYPE	NESTED_ITAB_TYPE

Context

Context	Generated
BC480	
NESTED_ITAB	
DATA	
FIELD_1	
FIELD_2	
INNER_ITAB	🔒

Figure 81: Nested Table That is Nested in the Interface



Interface: 2 independent internal tables

Variable Name	Type Assignment	Type Name
OUTER_ITAB	TYPE	FLAT_ITAB_TYPE1
INNER_ITAB	TYPE	FLAT_ITAB_TYPE2

Context

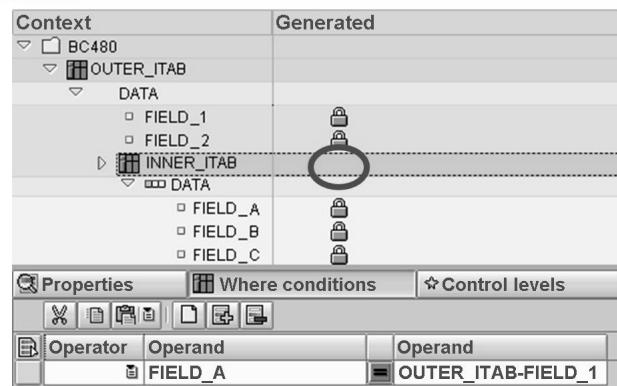


Figure 82: Combining Two Internal Tables Into One Nested Table

Exercise 7: Tables

Exercise Objectives

After completing this exercise, you will be able to:

- Integrate tables into a form
- Format tables
- Create data-driven page breaks
- Optional: integrate nested tables into a form

Business Example

A travel agency wants to print invoices for their customers' flight bookings, using "Interactive Forms based on Adobe software". One major part needs to be included: the list of items.

Task 1:

Prepare the development environment for the table.

Extend your form ZBC480_##. ## stands for your two-digit group number (monitor number).

It might be a good idea to create a local safety copy of your form every now and then. To do so, choose *Utilities(M) → Downloading form...* from SFP's initial screen.

If you have not finished one of the previous exercises, you can copy the template form instead: BC480_DYNAMIC. If you do this, make sure to replace the interface BC480 with your own interface or your own copy of interface BC480.

The reference solution of this exercise is form BC480_TABLE.

1. Set Designer's options so that a proper table will be created (and not just subforms) before you drag the table to the layout in the next task. Also make sure that a header will be added automatically.

Task 2:

Integrate the bookings table and set details.

1. Integrate the table *IT_BOOKINGS* into the layout, between the introductory lines and the closing.
2. Set suitable column widths and heights.
3. For all table cells, remove the 3D frame: set the *Appearance* to *None*.
4. Set inside and outside borders for all cells.

Continued on next page

5. Apply zebra stripes to the table.
6. Set the header so that it will appear at each page break.
7. Insert a page break when the airline changes.

Task 3:

Test the result.

1. Activate your form and test it with report SAPBC480_DEMO.

Task 4:

Optional: Add PS with nested table.

Compose a second PS, advertising some of your airlines connections. You will need a nested table for that.

1. In the interface, create a global field *GT_SCARR* of type TABLE OF SCARR and another one, *GT_SPFLI*, of type TABLE OF SPFLI.
2. In the initialization, fill the new internal table *GT_SCARR* with the datasets of U. S. American airlines (AA, DL, NW, UA). Use database table SCARR. Fill the internal table *GT_SPFLI* with all datasets from the database table SPFLI.
3. Activate your interface.
4. Integrate the two internal tables as one nested table into the context of your form. The connecting field is *CARRID*.
5. From *GT_SCARR*, deactivate all fields except for *CARRID* and *CARRNAME*. From *GT_SPFLI*, deactivate all fields except for *CONNID*, *CITYFROM* and *CITYTO*.
6. Integrate the table *GT_SCARR* into the layout (below the discount PS).
7. Set table details like cell widths, heights, borders, headers, etc.
8. Prefix this PS with a static text like “We would like to draw your attention to the following interesting flight connections.”
9. Protect the text and the table against a page break.

Solution 7: Tables

Task 1:

Prepare the development environment for the table.

Extend your form ZBC480_##. ## stands for your two-digit group number (monitor number).

It might be a good idea to create a local safety copy of your form every now and then. To do so, choose *Utilities(M) → Downloading form...* from SFP's initial screen.

If you have not finished one of the previous exercises, you can copy the template form instead: BC480_DYNAMIC. If you do this, make sure to replace the interface BC480 with your own interface or your own copy of interface BC480.

The reference solution of this exercise is form BC480_TABLE.

1. Set Designer's options so that a proper table will be created (and not just subforms) before you drag the table to the layout in the next task. Also make sure that a header will be added automatically.
 - a) Go to the palette menu of the *Data View* palette and choose *Options....* Select the checkboxes *Allow Tables To Be Generated* and *Create A Header Row For Any Table*.

Task 2:

Integrate the bookings table and set details.

1. Integrate the table *IT_BOOKINGS* into the layout, between the introductory lines and the closing.
 - a) From the *Data View* palette, drag the table to the page (*Design View*). If necessary, reposition the table in the *Hierarchy* by drag & drop.
2. Set suitable column widths and heights.
 - a) To determine the column widths, click with your mouse on a cell border and drag it to the left or right. Alternatively, select a cell and enter the cell width manually on the *Layout* palette.
Check that the overall width of the table does not exceed the width of the surrounding page (*Design View*).
b) To set row heights, mark all subnodes of *DATA* in the *Hierarchy*. Go to the *Layout* palette and set a minimum height, for example 0.3 cm. Make sure to select the *Expand to fit* checkbox. Furthermore, remove the top and bottom margins on the *Layout* palette.

Continued on next page

3. For all table cells, remove the 3D frame: set the *Appearance* to *None*.
 - a) You can change the *Appearance* of several fields at a time only if they have the same type. That is why, in the *Hierarchy*, you have to mark all relevant text fields first (*CARRID*, *CONNID*, *BOOKID*, *FORCURKEY*). Then go to the *Object* palette and choose *None* for *Appearance*.
 - b) In the *Hierarchy*, mark *FLDATE*, go to the *Object* palette and choose *None* for *Appearance*.
 - c) Repeat this step for *FORCURAM*.
4. Set inside and outside borders for all cells.
 - a) In the *Hierarchy*, mark all header cells and body cells. Go to the *Border* palette and set edges to your liking.
5. Apply zebra stripes to the table.
 - a) In the *Hierarchy*, mark the table *IT_BOOKINGS*. On the *Object* palette, *Row Shading* tab, select the *Apply Alternating Row Shading*. Set colors to your liking.
6. Set the header so that it will appear at each page break.
 - a) In the *Hierarchy*, mark the header row. Press F2 to rename it *HEADER*. Go to the *Object* palette, *Pagination* tab. Select the checkboxes for the two upper options.
7. Insert a page break when the airline changes.
 - a) In the *Hierarchy*, select the *DATA* node. Go to the *Object* palette, *Pagination* tab. Choose *Edit* to insert a conditional break.
 - b) On the following screen, click on the plus sign. Then enter the following:
`DATA [-1] . CARRID ne DATA . CARRID`
Alternatively, click on the pushbutton at the left-hand side to have the coding generated for you.
Select the *Break: Before* radio button. In the field *To*, select *Top of Next Page*. Then select *HEADER* as the *Leader*.

Continued on next page

Task 3:

Test the result.

1. Activate your form and test it with report SAPBC480_DEMO.
 - a) Press Ctrl+F3. In a second SAP GUI session, run transaction SA38. Enter **SAPBC480_DEMO** as the name of the report. Choose *Execute* (F8).
 - b) Enter **ZBC480_##** on the selection screen and press F8 again.

Task 4:

Optional: Add PS with nested table.

Compose a second PS, advertising some of your airlines connections. You will need a nested table for that.

1. In the interface, create a global field **GT_SCARR** of type TABLE OF SCARR and another one, **GT_SPFLI**, of type TABLE OF SPFLI.
 - a) Go to the *Interface* tab. Double-click on *Global Data* on the left side. On the right side, choose the icon with the white sheet twice to append two new rows.
 - b) In the first row of the two new ones, enter **GT_SCARR** as the *Variable Name*, **TYPE** as *Type Assignment*, and **TABLE OF SCARR** as *Type Name*.
 - c) In the second row of the two new ones, enter **GT_SPFLI** as the *Variable Name*, **TYPE** as *Type Assignment*, and **TABLE OF SPFLI** as *Type Name*.
2. In the initialization, fill the new internal table **GT_SCARR** with the datasets of U. S. American airlines (AA, DL, NW, UA). Use database table SCARR. Fill the internal table **GT_SPFLI** with all datasets from the database table SPFLI.
 - a) On the node *Code Initialization*, enter **GT_SCARR** and **GT_SPFLI** as output parameters. Insert the following source code:


```
SELECT * FROM scarr INTO TABLE gt_scarr
WHERE carrid = 'AA'
OR carrid = 'DL'
OR carrid = 'NW'
OR carrid = 'UA'.
SELECT * FROM spfli INTO TABLE gt_spfli.
```
3. Activate your interface.
 - a) Press Ctrl+F3.

Continued on next page

4. Integrate the two internal tables as one nested table into the context of your form. The connecting field is *CARRID*.
 - a) From the left side of the context editor, open the node *Global Data*. Drag *GT_SCARR* over to the right side.
 - b) Drag *GT_SPFLI* and drop it on the *DATA* structure of *GT_SCARR* on the right-hand side.
 - c) As a *Where condition* for *GT_SPFLI*, enter: *CARRID = GT_SCARR-CARRID*.
5. From *GT_SCARR*, deactivate all fields except for *CARRID* and *CARRNAME*. From *GT_SPFLI*, deactivate all fields except for *CONNID*, *CITYFROM* and *CITYTO*.
 - a) Mark the fields on the right-hand side, then right-click and choose *Deactivate*.
6. Integrate the table *GT_SCARR* into the layout (below the discount PS).
 - a) Drag it from the *Hierarchy* to the layout.
7. Set table details like cell widths, heights, borders, headers, etc.
 - a) See previous task.
8. Prefix this PS with a static text like “We would like to draw your attention to the following interesting flight connections.”
 - a) Drag a static text from the *Library* palette to the layout and change its content there.
9. Protect the text and the table against a page break.
 - a) In the *Hierarchy*, mark the text and the table *GT_SCARR*. Right-click and choose *Wrap in subform*. Press F2 and rename it *PS2_WRAPPER*. On the *Object* palette, *Subform* tab, disallow page breaks.



Lesson Summary

You should now be able to:

- Insert tables into a form
- Format tables
- Set a header for a table
- Create data-driven page breaks
- Create control levels
- Create nested tables



Unit Summary

You should now be able to:

- Insert static elements into a form: images, text and graphic objects
- Set object properties for static form elements
- Insert dynamic elements into a form: text fields, image fields, date/time fields, floating fields
- Set the data binding (the connection between the layout fields and the business data)
- Apply patterns (picture clauses) to influence field output
- Insert tables into a form
- Format tables
- Set a header for a table
- Create data-driven page breaks
- Create control levels
- Create nested tables



Test Your Knowledge

1. Static texts are filled in the ABAP program.
Determine whether this statement is true or false.
 True
 False

2. Static images must be located on the computer where Adobe LiveCycle Designer is located.
Determine whether this statement is true or false.
 True
 False

3. Text fields always correspond to character variables from the ABAP program.
Determine whether this statement is true or false.
 True
 False

4. The handling of Smart Form texts (= text modules) and SAPscript texts differs in Adobe LiveCycle Designer.
Determine whether this statement is true or false.
 True
 False

5. The size of an image field is always dynamic.
Determine whether this statement is true or false.
 True
 False

6. If you include a field from the context by dragging it from the *Data View* to the layout, you do not have to worry about the correct binding.
Determine whether this statement is true or false.
 True
 False

7. Floating fields can be inserted anywhere on a page (design view).
Determine whether this statement is true or false.
 True
 False

8. A display pattern is defined independent of the locale.

Determine whether this statement is true or false.

- True
- False



Answers

1. Static texts are filled in the ABAP program.

Answer: False

The form developer needs to type in the static text in the layout.

2. Static images must be located on the computer where Adobe LiveCycle Designer is located.

Answer: False

Static images must be located so that Adobe document services can access them, for example in your intranet or on a file server. If you embed them in your form, they can be located anywhere.

3. Text fields always correspond to character variables from the ABAP program.

Answer: False

A text field can refer to a simple string variable, a text module, a SAPscript text, a dynamic text, or an address from the Business Address Services.

4. The handling of Smart Form texts (= text modules) and SAPscript texts differs in Adobe LiveCycle Designer.

Answer: False

Designer does not evaluate any information about the origin of the text. The ABAP program passes on the correct data at runtime.

5. The size of an image field is always dynamic.

Answer: False

You determine whether or not the size should be dynamic by selecting or deselecting the *Expand to fit* option.

6. If you include a field from the context by dragging it from the *Data View* to the layout, you do not have to worry about the correct binding.

Answer: True

You have to manually set the binding if you drag the element from the *Library*.

7. Floating fields can be inserted anywhere on a page (design view).

Answer: False

Floating fields can be inserted only in static text.

8. A display pattern is defined independent of the locale.

Answer: False

A display pattern may have standard pattern symbols (such as \$, 9, or Z), but it is also possible to include literals, like 'Monday.'

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Unit 6

Scripting in the Form

Unit Overview

Although most of the data retrieval and calculation for PDF-based forms is done with ABAP coding (in the application program and in initialization of the form interface), some coding must be written that is executed during form rendering. This unit will introduce you to the principles of scripting and to Adobe's own scripting language, FormCalc.



Unit Objectives

After completing this unit, you will be able to:

- Name different events and what they can be used for
- Use the Script Editor
- Create scripting with FormCalc
- Access form objects, their values, and their properties

Unit Contents

Lesson: Scripting for Form Elements	194
Exercise 8: Scripting for Form Elements	207

Lesson: Scripting for Form Elements

Lesson Overview

Form printing with PDF-based forms means:

- Data is collected in an ABAP program (technically speaking, on the SAP Web Application Server ABAP).
- Data is passed to Adobe document services, which reside on the J2EE engine.
- Data is merged with form layout and logic.
- The PDF document is returned to the ABAP application program.

During form rendering, no communication with the SAP Web Application Server ABAP is possible. As a consequence, no ABAP coding can be written for the form itself. However, you can write scripting for all dynamic elements, using either Adobe's FormCalc or JavaScript.

This lesson will show you how you write scripting and introduce you to the basics of FormCalc syntax.



Lesson Objectives

After completing this lesson, you will be able to:

- Name different events and what they can be used for
- Use the Script Editor
- Create scripting with FormCalc
- Access form objects, their values, and their properties

Business Example

A travel agency wants to print invoices for their customers' flight bookings using the tool "Interactive Forms". The form needs to be refined.

Scripting – an Introduction

You can attach scripts to all kinds of non-static objects, that is, all objects apart from content areas, static texts, static images, and graphical elements (lines, circles, rectangles). However, scripting that is attached to a non-static object can access all other objects of the form, including the static ones.

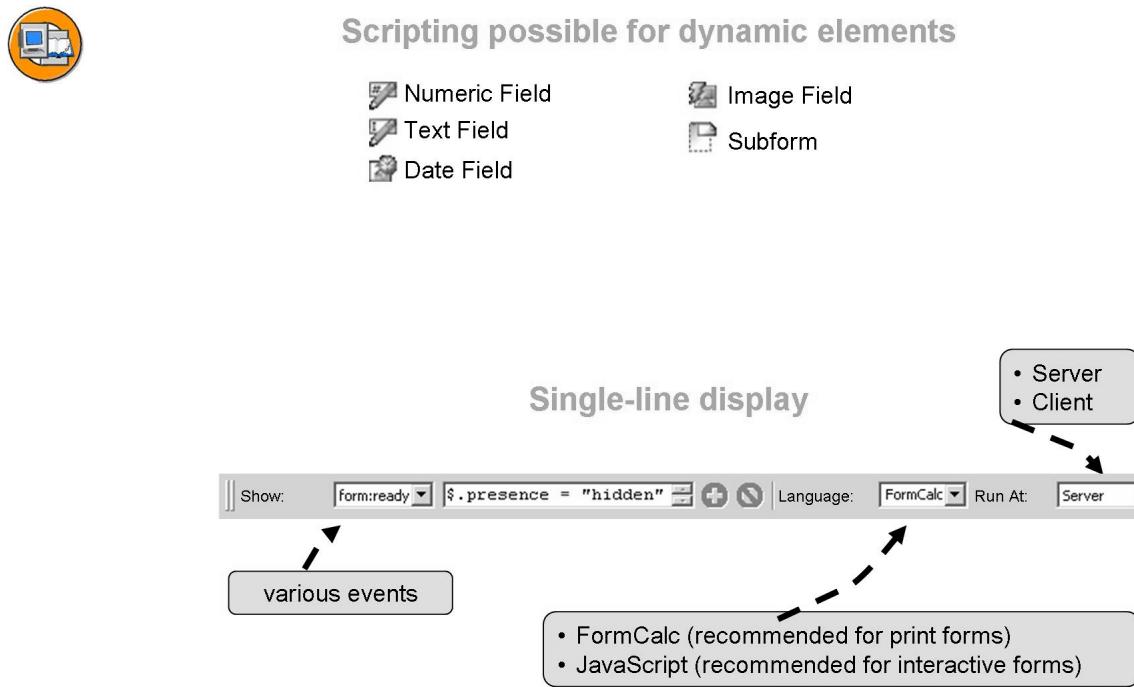


Figure 83: Script Editor I

To add scripting to an object, click on it and type the scripting in Designer's Script Editor.

You can choose between different events to determine when your coding will be executed.

For each event of an object, you can choose between JavaScript and Adobe's FormCalc. ABAP is not supported here. For print forms it is recommended to choose FormCalc for simple tasks, JavaScript for more complex tasks. If you intend to create coding for interactive (web-based) forms you should opt for JavaScript.

Scripting can be evaluated at the client (front end) or at the server. Since print forms do not require any user interaction, scripting should run on the server side.

To avoid setting the scripting language and the place for the coding execution for every scripting each time, it is best to set the preferred language and the place on the form properties. (This will still allow you to change it for individual scriptings.)



Multi-line display

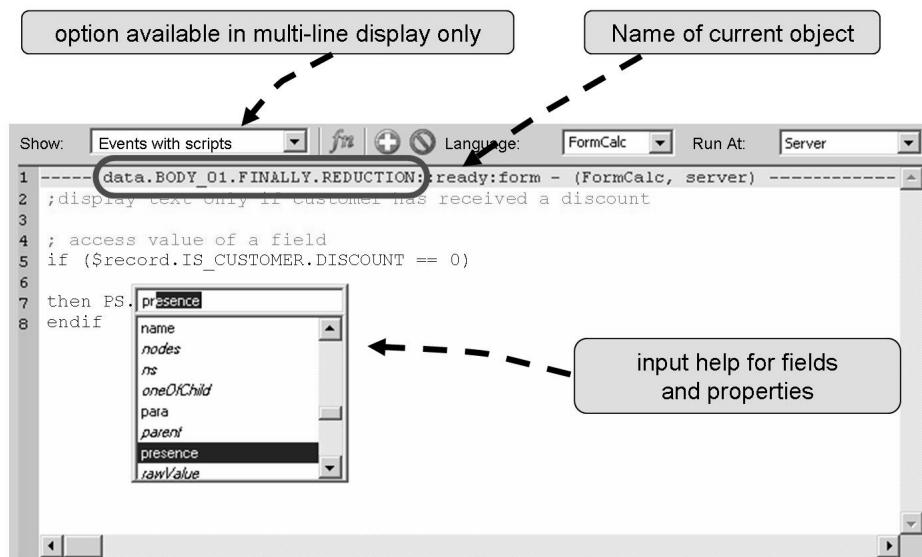


Figure 84: Script Editor II

If your coding is more complex or if for one object you have scripts for several events, it is best to enlarge the Script Editor a bit. Only then do you have the *Show Events with Scripts* option.

If you want to see all coding in a form, select *Show Events with Scripts*, then display the *Hierarchy*, and mark all elements. (Click on the top element, then shift-click on the bottom element.)

To change the scripting language for one event, right-click in the multi-line editor and select your preferred language. If you already have several events for one object and you want to change the script language for all of them, choose your preferred language from the *Language* drop-down list.

You can access subfields or properties of layout elements by typing a period. An input help dialog box will show you the available options. (You can switch off this input help by deselecting *Tools → Options → Workspace → Show Statement Completion Options*.)



Every dynamic form element can have scripting for every event.

Processing sequence:

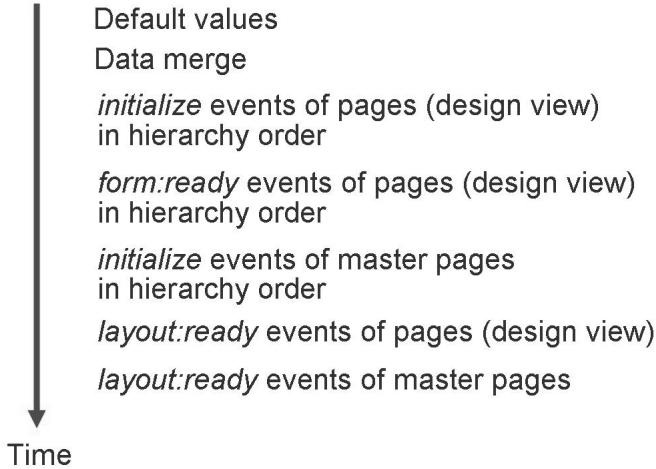


Figure 85: Events and Their Processing Order

You can choose between different events to determine when your coding will be executed. Among them you will find:

- **initialize** – When the object is initialized, after data from the data stream is merged. One usage of this event would be to dynamically determine the field value.
- **calculate** – Processed after data merge, but before display. You can query the field's value and edit it. The results from the ordinary data binding will thus be overridden. Calculation is always triggered again when a calculation-relevant value changes. Hence, self references must be avoided.
- **form:ready** – After the form and data have been merged and loaded, and any calculations and validation events have triggered. As this event is processed before pagination is finished, you can use it to dynamically hide objects. If the objects are positioned in a subform with type *Flowed*, no empty space will result. Instead, the following objects will take up the space of the hidden objects.
- **layout :ready** – After the application of the layout, that is, after page breaks have been inserted. You can use this event to access objects placed on master pages.



Scripting – Syntax Elements

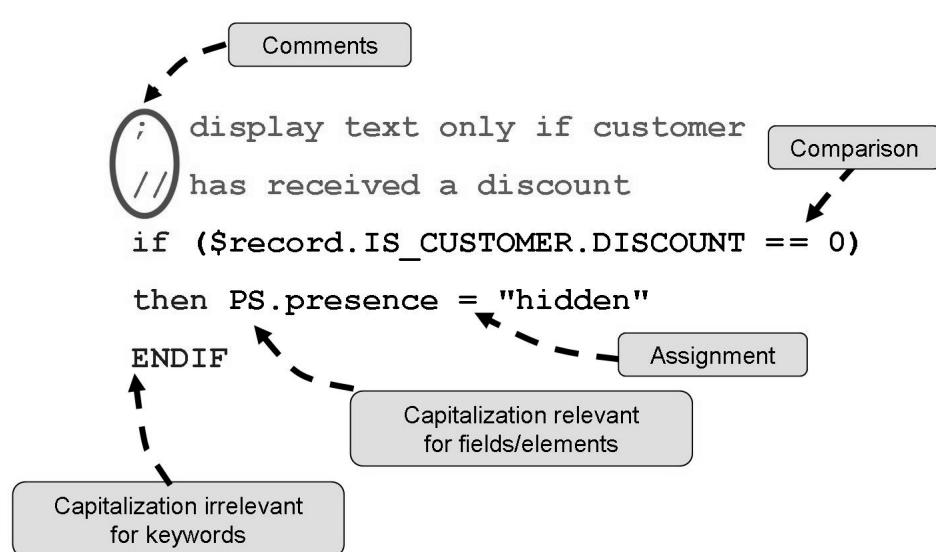


Figure 86: FormCalc Basics

A comment starts with a semicolon (;) or two slashes (//). It implicitly ends with the next line of coding.

Commands may stretch over several lines.

FormCalc has approx. 37 keywords (such as `if`, `do`, `or`). They are case-insensitive.

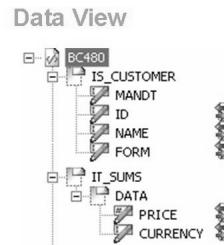
FormCalc has the usual calculation operators +, -, *, / and the comparison operators ==, <>, <, >, <=, >=. You can also use the English short forms for the comparison operators: eq, ne, lt, gt, le, ge. Programmers who are used to ABAP should take care not to confuse the assignment operator (=) with the equals operator (==).

Variables are case-sensitive.

Accessors

Elements from the form that you can use as fields in the coding are called accessors. More precisely: An accessor allows you to query or change a field value or an object property.

To access a field value from the form context (*Data View*), prefix the name with `$record`. Give all parts of the name as they appear in the *Data View*, separated by a period. The original ABAP representation is irrelevant!



```

; any field from Data View
$record.IS_CUSTOMER.ID = 7

;accessing entries of internal tables
$record.IT_SUMS.DATA.CURRENCY      = "USD" ;1st
$record.IT_SUMS.DATA[4].CURRENCY   = "USD" ;5th
$record.IT_SUMS.DATA[*].CURRENCY  = "USD" ;all
  
```

Prefix for accessing values
from Data View (i. e. form context)

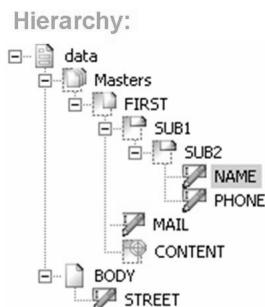
Figure 87: Accessing Context Field Values

You can access all context fields (that is, all fields from the *Data View*), even those that you have not included in your layout.

When accessing a field of an internal table, note the following:

- Take care to follow the path shown in the *Data View*. Due to the XML representation of internal tables, there is always a layer called *DATA* between the internal table and its fields.
- If you do not specify which dataset to take, the first one will be taken. (There is also a more general form of this rule: If two objects on the same hierarchy level happen to have the same name, any unspecific reference to them will actually be considered as being a reference to the first object of that name.)
- If you want to access a specific dataset of an internal table, include the number of that dataset in square brackets, like `$record.IT_SUMS.DATA[4].CURRENCY`. Note that unlike *sy-tabix* in ABAP, the first line has number 0, the second line 1, and so forth.
- If you want to access all datasets of the internal table, include an asterisk in square brackets, like `$record.IT_SUMS.DATA[*].CURRENCY`.

You can access a field value also by giving its name as shown on the *Hierarchy* palette. This is needed for all fields with no data binding, that is, for all fields that do not come from the form context. Furthermore, this method is required to change object properties at runtime.



```

;data.Masters.FIRST.SUB1.SUB2.NAME::ready:layout
$ = "Rahn"                                ;current object
PHONE = "+49 6221 150869"                  ;relative path
data.BODY.STREET = "Highway 66"             ;absolute path
  
```

Figure 88: Accessing Fields from the Hierarchy

Use \$ to access the current object.

You can access objects that are:

- On the same hierarchy level
- On the level of one of its enclosing subforms
- Direct children (if object to which scripting is attached is a subform)

by giving just their names. In the above example, within scripting for the subform *SUB1*, you might access the objects *SUB2* and *MAIL* by giving just their names.

An easy test as to whether an object is accessible via its name is to type it in the Script Editor and then type a period. If an input help appears, it is accessible directly.

You can access children anywhere along down a subform's hierarchy, provided you specify the traversed subforms or summarize them by two dots. For example, starting from *SUB1*, you could access *PHONE* by \$.. PHONE.

You can access all fields by giving their fully qualified paths and names. Start with the name of the master page or the corresponding page (design view), and insert all hierarchy levels on the way down to the object. If the hierarchy path contains unnamed subforms you should leave these out. An easy way of getting to the name of an object is to set the cursor there in the *Hierarchy*, go to its Script Editor, and have one random event displayed. The first line will always be a comment such as *data.Masters.FIRST.SUB1.SUB2.NAME::ready:layout*. The name of the element precedes the double colon. In this case, it would be *data.Masters.FIRST.SUB1.SUB2.NAME*.

In general, you should specify the minimal definition of the field for performance reasons.



Value Assignment

```
;event initialize of text field PHONE_NO
if ($record.IS_CUSTOMER.COUNTRY == "US")
then $ = "+49 6227 888888"
else $ = "+49 6227 777777"
endif

;event calculate of text field PHONE_NO
if ($record.IS_CUSTOMER.COUNTRY == "US")
then "+49 6227 888888"
else "+49 6227 777777"
endif
```

Note that in the `calculate` event, no assignment of the ordinary kind (like `$ = a`) is required if you want to set the value for the field for which the coding is written. Instead, the field will take the value of the last expression. In the above example, the text field `PHONE_NO` will have the value of “+49 6227 888888” if the `IS_CUSTOMER.COUNTRY` field equals “US”; “+49 6227 777777” otherwise.

Note that the event `CALCULATE` can be executed several times. Expressions with self references to the field for which the coding is attached, such as `$ = $ + 1`, might lead to undesired results.

For a value assignment in events other than `calculate`, an ordinary assignment like `$ = a` is required.

For your scripting, you can also define variables within the form without changing the form interface or context. You can do so in the form properties in the *Variables* tab. These variables are globally known within the form. They also appear in the *Hierarchy*, but they cannot be integrated in the form like ordinary text fields from the context, nor can they be used for the *Default Binding*.

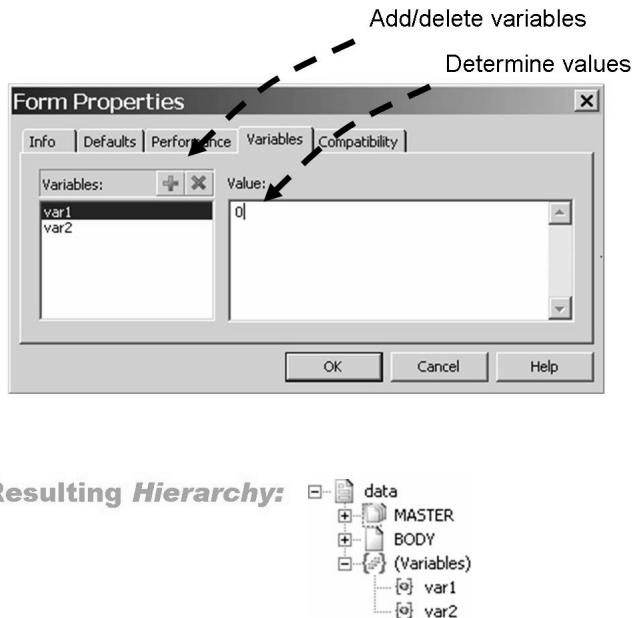


Figure 89: Variables

These variables can be text variables (with several lines) or numbers. Variable names are not case-sensitive.



Examples of Changing Object Properties

```

FOOTER.presence = "hidden"
StaticImage1.rotate = "90"
StaticText1.x = "12cm"
StaticText1.caption.font.fill.color.value = "200,0,0" ;RGB values

TextField1.caption.value.text = "Dynamic caption"
TextField1.assist.toolTip = "Better explanation"

$.break.after = "contentArea" ;page break after current subform
;can be processed only for subforms
  
```

Please note that to access the properties of a static object, you need to attach the relevant scripting to a non-static object, as static objects do not have events and hence no scripting.

If the values you want to set contain units, they must be enclosed in quotation marks, like “12cm”.

If you hide a subform, all of its content will be hidden as well.

Some properties include RGB values, which are combinations of three digits between 0 and 255 that specify how strong the red, green, or blue parts should be. For example, 0,0,0 represents black; 255,0,0 red; and 255,255,255 white.

For details, check the Object Reference of Designer's online help. Also, to find out what a suitable scripting command might be, it is quite helpful to create a dummy object with the desired settings and look at the XML presentation. For example, the caption of a text field *TextField1* might look like that:

```
<caption reserve="35mm">
<font typeface="Arial" size="12pt"/>
<para vAlign="middle"/>
<value>
<text>Here is the caption</text>
</value>
</caption>
```

So you can dynamically assign a caption like this:

```
TextField1.caption.value.text = "Dynamic caption"
```

If you wanted to change the caption's font dynamically, you would write:

```
TextField1.caption.font.typeface = "Times New Roman"
```

For more complicated calculations, JavaScript might prove easier to write than FormCalc. You need to know that accessing fields, their values, and properties differs between FormCalc and JavaScript:



	FormCalc	JavaScript
Current field or object	\$	this
Root of the context data	\$record	xfa.record
Value of field	<field> e.g. \$ = 7	<field>.rawValue e.g. this.rawValue = 7
Value of variable	<variable> e.g. count = 7	<variable>.value e.g. count.value = 7

Figure 90: Accessors in JavaScript Compared to FormCalc



fn ← – Input help for functions

<code>Mod(n1, n2)</code>	Modulus (remainder of division)
<code>Exists(v)</code>	Parameter is an accessor to an existing object?
<code>HasValue(v)</code>	A non-null, non-empty, or non-blank value?
<code>Concat(s1 [, s2 ...])</code>	Concatenation of two or more strings
<code>Substr(s, start, len)</code>	Substring
<code>Lower(s)</code>	String with lowercase characters only

Example:

```
var1 = Concat (Upper (var2) , var3)
```

Figure 91: FormCalc Functions

In assignments, you can use a wide range of functions. Among them, you will find logical functions, string functions, and arithmetic functions. For an overview of available functions, press the green “fn” button on the Script Editor toolbar.

`HasValue` will return 1 if the queried field has a value, 0 if not.

`Exists(v)` checks whether the given field is valid. If it returns 1, it is valid. One usage is to check whether an index entry exists for a repeated subform: `if (exists(TABLE.DATA[0].FIELD) == 0) if (exists(TABLE.DATA[0].FIELD) == 0).`

`Substr(s, start, len)` returns the part of a string *s* that starts with character number *start* and has up to *len* characters. The first character of a string has number 1.

Refer to the documentation for details on functions.

Function names are case-insensitive, but their arguments (that is, the variables) are case-sensitive.

Advanced Scripting

To find scripting errors for your FormCalc scripting, run Designer's preview. You will see the results in the *Report* palette, *Warnings* tab. Depending on the version of Adobe LiveCycle Designer, every time you run the preview, the new results will either replace the previous result, or they will be appended to the old list. In the latter case, for a better overview, you should clear this list every now and then by choosing *Clear Warnings* from the palette menu.



Run Designer's preview to test your scripting:

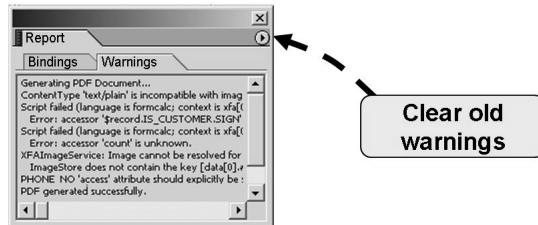


Figure 92: Finding Syntax Errors



Caution: The SAP syntax check (CTRL+F2) or the activation of the form (CTRL+F3) does not check the form and its scripting. It makes sure only that the interface and context are okay.

Testing with local test data (an XML file) is never enough. It is acceptable for quickly seeing what things look like, but (in particular with scripting that should be executed on the server) you might get wrong results. You must always run proper tests with an ABAP program.

If your scripting for one event contains more than one command, all commands that follow a syntax error will be ignored for that event. Any correct commands that precede this error will be executed normally.

For a more permanent control of errors, you could also use the trace functionality. Form Builder includes a settings dialog (SFP, *Utilities(M)* → *Settings*). Here, you can determine whether a trace file should be produced, in which depth, and in which directory of the user's PC it should be saved. Trace level 1 (*Short Trace*) is sufficient for tracking scripting errors. If you choose *Very detailed*, the trace will contain detailed rendering information from Adobe document services, too, and it will be attached to the resulting PDF. The trace settings are user-specific and valid for one session. If there are several print requests, the corresponding files will be overwritten unless different names are specified. This is not valid if several forms are issued in one print job, in which case the runtime data will be numbered sequentially.

For the trace functionality, the user must have the authorization for debugging. If you want to create a trace during background processing, you must choose level *mVery detailed* to have the trace information attached to the spool document. (The download functionality is not available during background processing.)



Scripting: Performance Aspects

- Avoid scripting if possible
 - Use ABAP coding in the program or the initialization node of the interface.
 - Use static object properties instead of setting them dynamically.
 - Use dynamic properties instead of scripting, if possible.
 - Use simple (relative) names instead of long names.
 - Use subforms for hiding elements without scripting.

As scripting tends to slow down the performance, try to avoid it if possible. For example, do not calculate the sums of an items list within the form, but in the ABAP program. Define the interface in such a way that the internal table with the items is passed on to the form along with a parameter that contains the sums.

If you want to hide some form elements if a field has no value, proceed as follows (no coding required):

1. Wrap the elements in a subform. This is best done by marking them in the *Hierarchy* and selecting *Wrap in Subform* from the context menu.
2. Set the subform's data binding to the field that has or has not a value.
3. On the *Object* palette, *Binding* tab, select the *Repeat Subform for Each Data Item* option, but deselect *Min Count* and *Max*.

Exercise 8: Scripting for Form Elements

Exercise Objectives

After completing this exercise, you will be able to:

- Use the Script Editor
- Create scripting with FormCalc, using different events
- Access form objects, their values, and their properties

Business Example

A travel agency wants to print invoices for their customers' flight bookings, using the "Interactive Forms" tool. The form has been created but still needs some final adjustments. Scripting is needed to achieve the desired result.

Task:

Extend your form ZBC480_##.

If you have not finished one of the previous exercises, you can copy the template form BC480_TABLE instead.

The reference solution of this exercise is form BC480_FINAL.

You can check your results with report SAPBC480_DEMO.

1. Calculate field value.

The *REFERENCE* field (which so far has had a standard value) should contain a combination of the first three letters of the clerk, the first three letters of the customer name, and the string "Demo". (You can locate the field on master page *FIRST*, in subform *INFORMATION*.)

2. Hide field dynamically dependent on condition.

The (*PS*) text with the discount should appear only if the customer has actually received a discount.

Take into consideration that *PS* is a static text, which cannot have scripting attached to it. You can write the scripting for the *DISCOUNT* field, for example.

To test the result of your scripting, you can run report SAPBC480_DEMO and enter customer numbers 2 and 3 on the selection screen. One is granted a discount, the other one is not.

3. Change field properties dynamically.

Continued on next page

The subform *INFORMATION* (which contains the heading “Invoice”, the customer number, the reference number, and the date) should be colored if the customer number equals 1.

4. **Optional:** Hide airline if no connections are available.

In the (optional) nested table, every airline gets an entry, even if it does not have any connections. Correct this error and hide the entire row if there are no connections.

Solution 8: Scripting for Form Elements

Task:

Extend your form ZBC480_##.

If you have not finished one of the previous exercises, you can copy the template form BC480_TABLE instead.

The reference solution of this exercise is form BC480_FINAL.

You can check your results with report SAPBC480_DEMO.

1. Calculate field value.

The *REFERENCE* field (which so far has had a standard value) should contain a combination of the first three letters of the clerk, the first three letters of the customer name, and the string “Demo”. (You can locate the field on master page *FIRST*, in subform *INFORMATION*.)

- a) FIRST.INFORMATION.REFERENCE::initialize

```
$ = Concat (Substr ($record.GC_CLERK, 1, 3),  
           Substr ($record.IS_CUSTOMER.NAME, 1, 3),  
           "Demo")
```

2. Hide field dynamically dependent on condition.

The (*PS*) text with the discount should appear only if the customer has actually received a discount.

Take into consideration that *PS* is a static text, which cannot have scripting attached to it. You can write the scripting for the *DISCOUNT* field, for example.

To test the result of your scripting, you can run report SAPBC480_DEMO and enter customer numbers 2 and 3 on the selection screen. One is granted a discount, the other one is not.

- a) BODY_01.FINALLY.REDUCTION::ready:form

```
if ($record.IS_CUSTOMER.DISCOUNT == 0) then  
    PS.presence = "hidden"  
endif
```

3. Change field properties dynamically.

Continued on next page

The subform *INFORMATION* (which contains the heading “Invoice”, the customer number, the reference number, and the date) should be colored if the customer number equals 1.

- a) FIRST.INFORMATION::ready:layout

```
if ($record.IS_CUSTOMER.ID == 1) then  
    $.border.fill.color.value = "240,160,0"  
endif
```

4. **Optional:** Hide airline if no connections are available.

In the (optional) nested table, every airline gets an entry, even if it does not have any connections. Correct this error and hide the entire row if there are no connections.

- a) BODY_01.FINALLY.PS2.PS2_WRAPPER.GT_SCARR.DATA:ready:form

```
if (exists(GT_SPFLI.DATA[0].CONNID) == 0) then  
    $.presence = "hidden"  
endif
```



Lesson Summary

You should now be able to:

- Name different events and what they can be used for
- Use the Script Editor
- Create scripting with FormCalc
- Access form objects, their values, and their properties



Unit Summary

You should now be able to:

- Name different events and what they can be used for
- Use the Script Editor
- Create scripting with FormCalc
- Access form objects, their values, and their properties



Test Your Knowledge

1. The initialize event is triggered once per form.
Determine whether this statement is true or false.
 True
 False

2. Accessing values or properties of master page elements is possible at event layout:ready.
Determine whether this statement is true or false.
 True
 False

3. You can mix both JavaScript and FormCalc in one form.
Determine whether this statement is true or false.
 True
 False

4. FormCalc's fields are case-sensitive.
Determine whether this statement is true or false.
 True
 False

5. What would scripting (attached to the field FIELD1) look like that at the event initialization sets the value of field FIELD1 to 7?
Choose the correct answer(s).
 A FIELD1 = 7
 B FIELD1 = 7;
 C FIELD1 == 7
 D 7

6. What would scripting (attached to the field FIELD1) look like that at the event initialization sets the value of field FIELD2 to 7?
Choose the correct answer(s).
 A FIELD2 = 7
 B \$record.FIELD2 = 7
 C \$.record.FIELD2 = 7;
 D Not possible.

7. A subform has three integer fields, named A, B, and C. Is the following scripting correct? if A = B then C = 7

Determine whether this statement is true or false.

- True
- False



Answers

1. The initialize event is triggered once per form.

Answer: False

The initialize event can be triggered for every dynamic element on any page (design view) or master page.

2. Accessing values or properties of master page elements is possible at event layout:ready.

Answer: True

Do not confuse layout:ready with form:ready. The latter is processed before master pages are applied.

3. You can mix both JavaScript and FormCalc in one form.

Answer: True

You can set the default scripting language on the form properties, but you can override them for every event.

4. FormCalc's fields are case-sensitive.

Answer: True

Fields are case-sensitive, keywords are not.

5. What would scripting (attached to the field FIELD1) look like that at the event initialization sets the value of field FIELD1 to 7?

Answer: A, B

A semicolon starts a comment.

Two consecutive equal signs are needed for comparisons, not for assignments.

A direct assignment, as in answer D, is possible in the calculate event only.

6. What would scripting (attached to the field FIELD1) look like that at the event initialization sets the value of field FIELD2 to 7?

Answer: A, B

In answer A, FIELD2 is taken as the Hierarchy name. Answer B would refer to the *Data View* name.

7. A subform has three integer fields, named A, B, and C. Is the following scripting correct? if A = B then C = 7

Answer: False

The correct scripting would be:

```
if (A == B) then C = 7 endif
```


Internal Use SAP Partner Only

Internal Use SAP Partner Only

Unit 7

Integration into ABAP Programs

Unit Overview

Designing the layout of a form is the most time-consuming part of administering a printing scenario. However, a form itself cannot be run; it can only be previewed with test data. An ABAP program is required to process it. This unit will walk you through the details of writing an ABAP application that integrates a PDF-based form.



Unit Objectives

After completing this unit, you will be able to:

- Use form interfaces
- Write an ABAP program for a simple printing scenario using the “Interactive Forms” tool

Unit Contents

Lesson: Integration of Forms into ABAP Programs	218
Exercise 9: Integration of PDF-Based Forms Into ABAP Programs (optional).....	227

Lesson: Integration of Forms into ABAP Programs

Lesson Overview

In this lesson, you will learn how to write the individual parts of an ABAP program that can trigger the processing of a PDF-based form.



Lesson Objectives

After completing this lesson, you will be able to:

- Use form interfaces
- Write an ABAP program for a simple printing scenario using the “Interactive Forms” tool

Business Example

A travel agency wants to print invoices for their customers' flight bookings, using the “Interactive Forms” tool. The form template is already available, but the application (an ABAP program) still needs to be written.



```
* (1) Data retrieval and processing
SELECT ... FROM ...
...

* (2) Find out name of generated function module
CALL FUNCTION 'FP_FUNCTION_MODULE_NAME'...

* (3) Start form processing
CALL FUNCTION 'FP_JOB_OPEN'...

LOOP AT ...
* (4) Call function module dynamically
    CALL FUNCTION <generated function module> ...
ENDLOOP.

* (5) End form processing
CALL FUNCTION 'FP_JOB_CLOSE'...
```

Figure 93: Sections of the ABAP Application Program

From the form printing perspective, every ABAP application program consists of the following parts:

1. The business data is selected from the database, which is by far the most comprehensive part. Additional calculations should also be done here, as the form should contain as little calculation as possible. The idea of Enterprise Services Architecture implies that no additional data is selected from within the form.

For performance reasons, you might want to find out which fields are actually used in the form. Call function module FP_FIELD_LIST. It will return a single-column, internal table with the names of the fields (in the case of structures, the individual fields will be listed).

2. The name of the function module generated for the form must be determined, as the name differs depending on the form and system.

Call the function module FP_FUNCTION_MODULE_NAME and pass the form name to it.

3. If you call a form with the new interface, you must explicitly open and close form printing. Call function modules FP_JOB_OPEN and FP_JOB_CLOSE to do so.
4. Within the bracket of FP_JOB_OPEN and FP_JOB_CLOSE, call the form's generated function module as many times as needed. Several PDF documents will result, which will be separate parts of one spool request. Using transaction SP01, you can print them individually or separately. Note, however, that it is not possible to mix Smart Forms and PDF-based forms in one job.



```

DATA:
  form          TYPE fpwbformname,
  fm_name       TYPE rs381_fnam,
  interface_type TYPE fpinterfacetype.

* (2) Find out name of generated function module
CALL FUNCTION 'FP_FUNCTION_MODULE_NAME'
  EXPORTING
    i_name        = form
  IMPORTING
    e_funcname    = fm_name
    e_interface_type = interface_type.

```

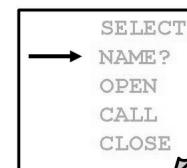


Figure 94: FP_FUNCTION_MODULE_NAME

Call the function module FP_FUNCTION_MODULE_NAME and pass the form name to it. You receive parameter *e_funcname*, which contains the name of the generated function module.

For interfaces, two different types exist: those that are Smart Forms-compatible and those that are not (the latter offer more options). If you do not want to restrict yourself to using forms with the new interface type only, you should check which kind of interface is implemented in the specific form. Parameter *e_interface_type* will have a value of 'S' if the Smart Forms-compatible interface is used, and a space otherwise.

Note that FP_FUNCTION_MODULE_NAME makes use of class-based exceptions. This means that you cannot query *sy-subrc*. Use TRY - CATCH - ENDTRY instead.



```

DATA:
  fp_outputparams  TYPE sfoutputparams.

...
* (3) Start form processing
CALL FUNCTION 'FP_JOB_OPEN'...
* set output parameters like printer, preview...
  CHANGING ie_outputparams = fp_outputparams ...
...
* (5) End form processing
CALL FUNCTION 'FP_JOB_CLOSE'...

```

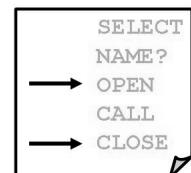


Figure 95: Starting and Ending Form Processing

If you call a form with the new interface, you must explicitly open and close form printing. Call function modules FP_JOB_OPEN and FP_JOB_CLOSE to do so.

If you call a form with the old (Smart Forms-compatible) interface, these two function modules must not be called. (Otherwise, the program will terminate with an error.) Starting and ending is implicitly achieved by calling the generated function module that uses the old interface type.

FP_JOB_OPEN has only one parameter: *ie_outputparams* of structure type *sfoutputparams*. It allows you to determine printer settings.

FP_JOB_CLOSE will tell you whether the output was produced successfully. You can also query the spool IDs.

If you are familiar with SAPscript, FP_JOB_OPEN and FP_JOB_CLOSE are roughly equivalent to function modules OPEN_FORM and CLOSE_FORM.



**Type SFPOUTPUTPARAMS
of changing parameter IE_OUTPUTPARAMS**

```
SELECT  
NAME?  
→ OPEN  
CALL  
CLOSE
```

<code>nodialog</code>	no printer dialog popup
<code>noprint</code>	no backend printing
<code>nopdf</code>	no PDF document, only PDL
<code>getpdf</code>	PDF as return parameter
<code>dest</code>	output device
<code>copies</code>	number of copies to be printed
<code>reqnew</code>	start a new spool job
<code>reqfinal</code>	spool request completed

Figure 96: Parameters of FP_JOB_OPEN

Structure type **SFPOUTPUTPARAMS**, which is used for changing parameter *ie_outputparams*, has many fields that allow you to determine printer settings, spool settings, settings for Adobe document services, and so on. They include:

- *nodialog*: If set to X, the printer dialog box will be suppressed.
- *noprint*: No backend printing. Printing from Adobe Reader preview will still be possible.
- *nopdf*: If set to X, no PDF document will be created, just the print file (PDL = Printer Definition Language). This improves print performance, but prevents you from viewing the result in transaction SP01 (Output Controller). By default, both a PDF document and print data (PDL) will be created. PDL can be used for PostScript, PCL, and Zebra printers.
- *getpdf*: If set to X, the */Ibcdwb/formoutput-pdf* field of the form's generated function module will contain the PDF data in hexadecimal form. This is helpful for further processing the PDF file. A preview will not be possible if *getpdf* = 'X'
- *dest*: The output device
- *copies*: Number of copies to be printed
- *reqnew*: If set to X, a new spool job will be started. If not set (default), the system will try to use an existing spool request. See SAP Note 85318 for details.
- *reqfinal*: If set to X, the spool job will be closed so that no other documents can use it.
- *connection*: Specify RFC destination (http/https connection) for Adobe document services.
- *XFP*: If set, external applications outside the SAP system can access and process the contents of forms. The XFP output contains all form data in XML format, but does not contain any layout information about the form. XFP output corresponds to XSF output for Smart Forms and RDI output for SAPscript.



```

DATA:
  fp_docparams  TYPE sfpdocparams,
  fm_name       TYPE rs381_fnam,
  fp_result     TYPE TYPE fpformoutput, ...

LOOP AT ...
  fp_docparams-langu = customer_language.

  * (4) Call function module dynamically
  CALL FUNCTION fm_name
    EXPORTING
      /1bcdwb/docparams  = fp_docparams
      it_bookings        = gt_bookings
    IMPORTING
      /1bcdwb/formoutput = fp_result
    EXCEPTIONS
      OTHERS             = 1.
  ENDLOOP.

```

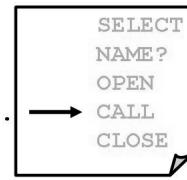


Figure 97: Calling the Generated Function Module

The form's generated function module is always called dynamically.

Most parameters of the function module are defined in the form interface, hence they vary a lot from form to form. They typically contain the business data.

There is only one (optional) standard import parameter: `/1bcdwb/docparams` of type `sfpdocparams`. It contains several fields:

- By setting the *langu* field, you determine the language in which the document should be processed. If the requested language should be unavailable, the logon language will be taken. If this is unavailable, too, the form's original language will be taken.
It is also possible to specify up to three replacement (alternative) languages, using parameters *replangu1*, *replangu2*, or *replangu3*, respectively.
- The *country* field determines the settings for decimal numbers, date, and time format. The combination of language and country is called a locale. Note that the ABAP command `SET COUNTRY` will be ignored if you use a Dictionary interface if *Country* is provided.
- The *Fillable* field determines whether a form can have interactive features. By default, it cannot. If you set *fillable* to 'X', interactive features are enabled and Adobe Reader grants the Reader rights (in particular, the user can enter data and save the form together with the data). If *fillable* equals 'N', the user can make entries in the form, but cannot save the form together with the entries.
- *daratab*: Table with archive indexes.

There is also only one standard exporting parameter of the form's generated function module: `/Ibcdwb/formoutput` of type `fpformoutput`. It contains three fields:

- `pdf`: Contains the PDF data in hexadecimal form if parameter `getpdf` of function module `FP_JOB_OPEN` is set to X. This is required if your program has a download functionality or if you want to pass on your PDF to Business Communication Service to fax or e-mail it.

Detailed information on Business Communication Services is available in the SAP Library, under “Generic Business Tools for Application Developers (BC-SRV-GBT)”. See also standard demo reports `BCS_EXAMPLE_...`

- `pdl`: Contains the PDF data in hexadecimal form if parameter `getpdl` of function module `FP_JOB_OPEN` is set to X.
- `pages`: Returns the number of pages that have been printed.



(1) Data retrieval

(2) Name of the generated function module?

(3) Call generated function module

```
DATA:
  ssf_name  TYPE tdsfname,
  func_mod_name TYPE rs38L_fnam.

SELECT ... FROM ...
...

CALL FUNCTION 'SSF_FUNCTION_MODULE_NAME'
  EXPORTING
    formname = ssf_name
  IMPORTING
    fm_name = func_mod_name.

LOOP AT ...
  CALL FUNCTION func_mod_name
    EXPORTING ...
    IMPORTING ...
ENDLOOP.
```

Figure 98: Reuse of Smart Forms Application Programs

If you have migrated a Smart Form to a PDF-based form, you might want to reuse your old application program without having to change it. This is possible because function module `SSF_FUNCTION_MODULE_NAME` (that is, the Smart Forms equivalent to `FP_FUNCTION_MODULE_NAME`) has been enhanced so that it can look up whether a Smart Form should be used or whether a migrated Smart Form, that is, a PDF-based form, should be used. Parameter `fm_name` will automatically contain the name of the appropriate generated function module. Since this generated function module is always called dynamically, no changes in the application program are necessary.

How do you tell SSF_FUNCTION_MODULE_NAME which function module to return? After migrating a Smart Form to a PDF-based form, run transaction SMARTFORMS and choose *Utilities → Migration → PDF-based Form → Switch Runtime*. This will create an entry in an administration table (STXFPCUST) with the name of the old Smart Form and its new PDF-based replacement. This entry will automatically be evaluated by SSF_FUNCTION_MODULE_NAME.

You could also create new PDF-based forms (with Smart Forms-compatible interfaces), and use them in old application programs. However, we recommend using the new technology for both the form and the application program.

If your form uses a Smart Form-compatible interface, the generated function module will have different parameters. Here are some of the most important parameters:

- *control_parameters*: A mixture of /Ibcdwb/docparams and of the equivalent of FP_JOB_OPEN's ie_outputparams.
- *output_options*: Spool settings
- *user_settings*: If set to X, the user's default settings are used for the spool.

Please note that some parameters that are possible in a Smart Forms-compatible interface can no longer be used in a meaningful way.

For details, see the Smart Forms documentation.

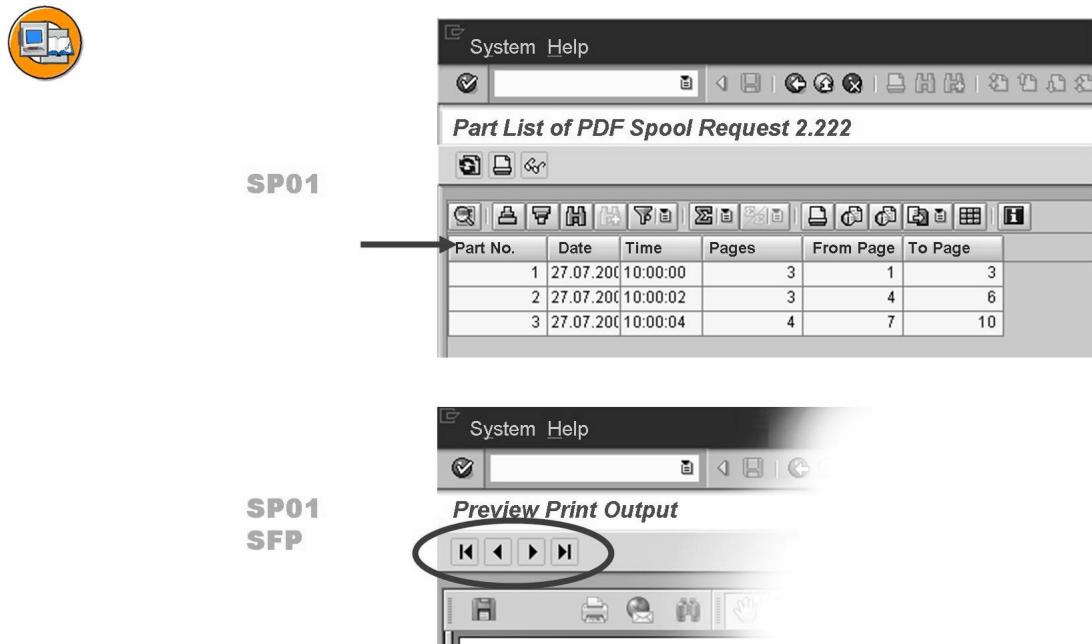


Figure 99: PDF Documents in the Print Preview

The print preview (of transactions SP01 and SFP) displays the PDF document in Adobe Reader.

 **Note:** The SAP GUI print preview makes uses an HTML control, which in turn calls an Internet browser. Adobe Reader is displayed in the SAP GUI only if you have enabled Adobe Reader to be displayed in your browser. You do this in the Adobe Reader settings. If this feature is not enabled, the SAP GUI print preview will remain empty but Adobe Reader appears in a separate window.

If, within one spool request, there are several PDF documents (results of several calls of the generated function module), they are kept separately as parts.

The individual parts within the print preview (of transactions SP01 and SFP) are accessible via arrows. You can go to the first, previous, next, or last part.

Exercise 9: Integration of PDF-Based Forms Into ABAP Programs (optional)

Exercise Objectives

After completing this exercise, you will be able to:

- Write an ABAP program for a simple printing scenario using the tool “Interactive Forms based on Adobe software”

Business Example

A travel agency wants to print invoices for their customers' flight bookings, using the “Interactive Forms” tool. As the form has been designed from scratch (without an SAP template), there is no application program yet to run the form; it needs to be written.

Task:

Create an ABAP program that can print your own PDF-based form.

1. In the ABAP Workbench (transaction SE80), copy program SAPBC480 TEMPLATE **with all includes** to ZBC480_##. Save it in package ZBC480_##. ## stands for your two-digit group number (monitor number).

The reference solution for this exercise is report SAPBC480 SOLUTION.

2. The template has a selection-screen and suitable data selection. The relevant parts in the source code where you have to insert your own coding are marked
*>>>>>>>>>>>>>>>

Generally, you might treat the exception handling lightly, as it is not the focus of this exercise.

At the suitable places, integrate the four required function modules. In particular, take care to create a correct mapping of the interface parameters of the generated function module. Use fields *gs_customer*, *gt_bookings*, *gt_sums*, *gv_image_url*, and *pa_send*.

The locale can be entered on the selection-screen. It is transferred to the relevant fields of *gs_docparams* (coding has been implemented for you), but you have to pass *gs_docparams* to the generated function module.

3. **Optional:** Specify some details for spool processing. For example, you might want to prevent the print dialog.
4. Activate and test your program.

Solution 9: Integration of PDF-Based Forms Into ABAP Programs (optional)

Task:

Create an ABAP program that can print your own PDF-based form.

1. In the ABAP Workbench (transaction SE80), copy program SAPBC480 TEMPLATE with all includes to ZBC480 ##. Save it in package ZBC480 ##. ## stands for your two-digit group number (monitor number).

The reference solution for this exercise is report SAPBC480 SOLUTION.

- a) Start transaction SE80. In the left area (the navigation area), display report SAPBC480 TEMPLATE.
 - b) Right-click the report to copy it.
2. The template has a selection-screen and suitable data selection. The relevant parts in the source code where you have to insert your own coding are marked

*>>>>>>>>>>>>>>>

Generally, you might treat the exception handling lightly, as it is not the focus of this exercise.

At the suitable places, integrate the four required function modules. In particular, take care to create a correct mapping of the interface parameters of the generated function module. Use fields *gs_customer*, *gt_bookings*, *gt_sums*, *gv_image_url*, and *pa_send*.

The locale can be entered on the selection-screen. It is transferred to the relevant fields of *gs_docparams* (coding has been implemented for you), but you have to pass *gs_docparams* to the generated function module.

- a) See source code.
3. **Optional:** Specify some details for spool processing. For example, you might want to prevent the print dialog.
 - a) See source code.
4. Activate and test your program.

- a) Press CTRL+F3, then F8.

Your coding should look similar to the following:

Excerpt from the TOP-Include:

```
DATA:  
      gt_customers      TYPE TABLE OF scustom,
```

Continued on next page

```

gs_customer           LIKE LINE OF gt_customers,
gt_bookings          TYPE ty_bookings, "table of sbook
gt_sums              TYPE flprice_t,
gv_image_url         TYPE string,
gv_fm_name           TYPE rs38l_fnam,

```

* parameters for calling the generated function module

```

gs_docparams          TYPE sfpdocparams,
gs_outputparams       TYPE sfpoutputparams,

```

Event START-OF-SELECTION:

Continued on next page

```
* Form processing could not be started
ENDIF.

LOOP AT gt_customers INTO gs_customer.
* Set form language and country (->form locale)
  gs_docparams-langu = pa_lang.
  gs_docparams-country = pa_cntry.

  PERFORM find_bookings_for_customer.

* >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
* Now call the generated function module
CALL FUNCTION gv_fm_name
  EXPORTING
    /1bcdwb/docparams = gs_docparams
    is_customer      = gs_customer
    it_bookings      = gt_bookings
    it_sums          = gt_sums
    iv_image_url     = gv_image_url
    iv_sending_country = pa_cntry
  EXCEPTIONS
    OTHERS           = 1.
  IF sy-subrc <> 0.
    MESSAGE e021.
  ENDIF.

ENDLOOP.

* >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
* Close spool job
CALL FUNCTION 'FP_JOB_CLOSE'.
```



Lesson Summary

You should now be able to:

- Use form interfaces
- Write an ABAP program for a simple printing scenario using the “Interactive Forms” tool



Unit Summary

You should now be able to:

- Use form interfaces
- Write an ABAP program for a simple printing scenario using the “Interactive Forms” tool

Internal Use SAP Partner Only

Unit 8

Tips and Tricks

Unit Overview

This unit will show you how to download and upload form objects, as well as how to import PDF documents. Furthermore, it gives information on how to create forms that are accessible for mobility impaired, visually challenged, or visually impaired persons.



Unit Objectives

After completing this unit, you will be able to:

- Download and upload a form
- Import a PDF file
- Name issues that are relevant in an accessible form

Unit Contents

Lesson: Download/Upload and Import	234
Lesson: Accessibility Aspects	240

Lesson: Download/Upload and Import

Lesson Overview

Typically, you do not work only on one form and one system. You frequently need a form somewhere else or, conversely, you might already have a PDF template that needs to be integrated into your printing scenario. This lesson will show you how to download and upload form objects and how to import existing PDF documents.



Lesson Objectives

After completing this lesson, you will be able to:

- Download and upload a form
- Import a PDF file

Business Example

Your company has PDF templates, such as tax forms, that need to be integrated into your printing scenario with the “Interactive forms” tool. Instead of re-creating the form with the help of a ruler, you want to import the form.

Downloading and Uploading Form Objects

Form objects are regular repository objects and, as such, they can be transported like, for instance, a report or a transaction. However, there are situations where going the regular transport way might be a bit too slow (or you might not have the rights to import objects yourself). For such cases, the download/upload option exists.

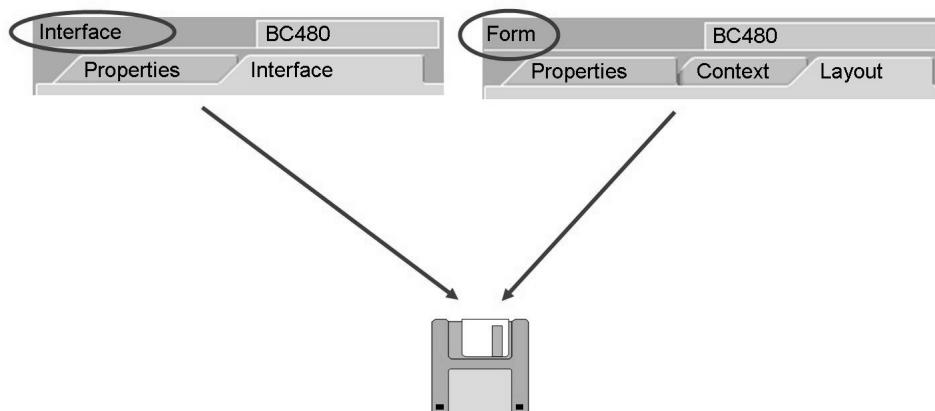


Figure 100: Downloading Form Objects

You can download interfaces and forms to your local PC and upload them from there. From transaction SFP, choose *Utilities* → *Downloading form object* or *Uploading form object*.

If you download a form, make sure to also download the interface used.

If you download a form, the current version (including all its languages) will be downloaded. You do not have to save or activate the form before downloading it.

You can also download parts of a context only. Select the relevant node (multiple selection is not possible) and chose *Utilities* → *Downloading subtree*.

You can also download the PDF document. You have two options:

- Within transaction SFP, go to the *Layout* tab, then to the Layout Editor, and select the *Preview PDF*. This displays the PDF form, but data is not merged. (Unless you have specified an XML test data file in the *Form Properties*.) To save this version, press CTRL+SHIFT+S and specify where you want to save the document.
- From the spool print view, either select *Form* → *Test* (shortcut: F8) in transaction SFP or run a print program. The result will be displayed in Adobe Reader. Press CTRL+SHIFT+S or select *File* → *Save a Copy* and specify where you want to save the document.

Be cautious when manually copying the XML source from within Designer. This method does not copy the context, so you will probably get a faulty form.

Importing PDF Documents

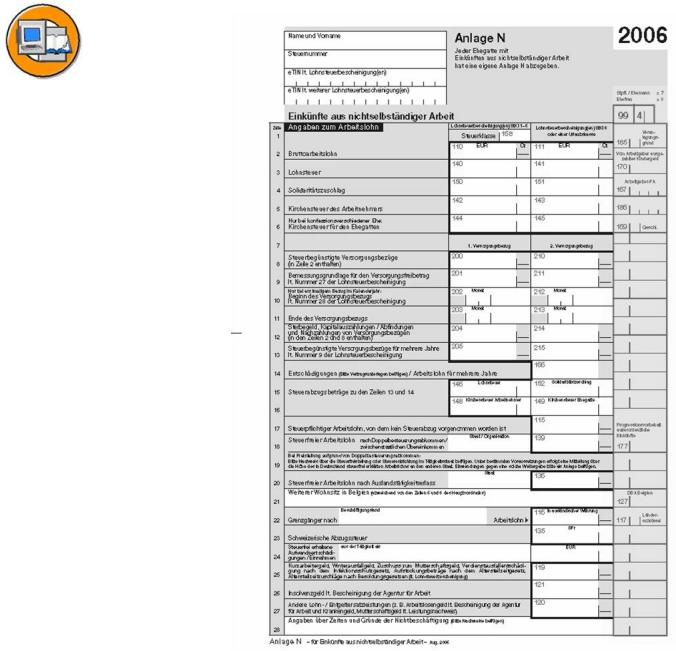


Figure 101: PDF Import Functionality of Designer

Due to the characteristics of PDF (page layout description) and XFA (declarative composition of XFA form objects), one-to-one mapping is not possible. Rather, the functionality must be considered as an assistive technology to speed up the process of converting an existing PDF form into an XFA form template to be used with Adobe Document Services. In general, the import tool will be capable of converting most PDFs with their layouts consisting of:

- Text
 - Form fields
 - Most kinds of images
 - Lines and rectangles

A number of restrictions exist, the most important being:

- Embedded fonts in the PDF cannot be transferred to the converted XFT.
 - JavaScript within the PDF is converted to comments in the XFT and will need reworking after conversion; most Acrobat built-in form actions are not converted.
 - Direct conversion of vector-based graphics, as well as some bitmap graphics, is not supported.

The PDF import is a tool that speeds up the form layout creation process substantially. Follow-up work could take several hours.

To import a PDF file in the Form Builder:

- In transaction SFP, choose the *Layout* tab.
- Choose *Tools → Import*.
- In the dialog box for opening files, from the Files of type drop-down list, select *Adobe PDF Files (*.pdf)* files.
- Navigate to the file that you want to open, select the file, and choose *Open*.

On the window that opens up when you start the import, a number of options can be set. Typically, the import works best with the default options.

Here are some details:

Conversion Type: You should always choose the *Optimize for editing* option.

Join text as:

- *Paragraph*: This setting makes it easy to maintain the converted form. Text that is close together on a line or in multiple lines is combined to create one text block with text that will re-wrap itself.
- *Text Block*: This is the same as *Paragraph* setting, but without text wrapping.
- *Line by line*: This is the same as *Text Block*, but will produce a single block for each line of text.
- *Single Words, Single Characters*: Not recommended (this mode will generate a conversion that most closely matches the original document, but it is hard to maintain, as each “visible” section of text may be composed of multiple text objects).

As for the *Tolerance for joining text*, lines, and paragraphs, a common setting of *High* should give good results in most cases. However, depending on the form, in special cases a lower tolerance may be needed to avoid combining text that should not be joined.

Page Range: If the form consists of several pages, of which only some are relevant for you, only select these pages.

Image conversion: In general, you should select this option to have images in the source PDF embedded into the resulting document.

Summary report: Select this option to get a summary report shown right after the import conversion is finished. The report can help you to identify problems in the result, for example, unknown fonts, encodings, and so on.

Trace information inclusion: The trace information is hardly readable for non-PDF experts, but blows up the size of the resulting XFT substantially. We recommend you always set this to *none*.

It is also possible to import Microsoft Word documents and RTF (Rich Text Format) documents.



Hint: You can find more information in the Adobe LiveCycle Designer Online Help under *Importing Documents from Other Applications*.



Lesson Summary

You should now be able to:

- Download and upload a form
- Import a PDF file

Lesson: Accessibility Aspects

Lesson Overview

Accessibility has been a very important topic in the software development of recent years. It aims at helping people that cannot use pointing devices, or whose vision is impaired, or who are even blind. This lesson focuses on the third group.

- **Note:** The following information does not represent any promise or obligation on SAP's part to make any aspect of the software accessible.



Lesson Objectives

After completing this lesson, you will be able to:

- Name issues that are relevant in an accessible form

Business Example

A travel agency wants to print invoices for their customers' flight bookings, using the tool "Interactive Forms". The form should be accessible.

Accessibility: definition



- Accessibility target groups are mobility impaired, visually challenged, and visually impaired persons.
- Adobe Reader provides several techniques for supporting accessibility target groups when "reading" PDF documents.

If you are sure the forms in your scenario get printed and never need to be accessed in electronic form, you might want to choose to produce untagged PDFs since they are considerably smaller. You can set field PDFTAGGED of structure SFPOUTPUTPARAMS (which is used in the function module FP_JOB_OPEN) to

- X if you want to have tagged PDFs
- "-" if you do want to generate untagged PDFs
- Space if you want to use the user's (GUI) settings

- **Note:** You can find the document "Creating Accessible Forms" in the Adobe LiveCycle Designer online help. However, this document mainly applies to interactive forms.

Here are some additional considerations for creating read-only PDF documents that provide as much help as possible for screen reader applications.

Basic principles



Basic Principles

	Problem	Solution
Mobility impaired	Unable to use pointing devices (for example, a mouse)	Requires use of tabs to move between fields in Adobe Reader; relevant for interactive forms only
Visually challenged	Can see, but not well	Use screen magnifier or zoom function in Adobe Reader
Visually impaired	Unable to see	<ul style="list-style-type: none"> • Requires use of screen reader • Requires generation of “tagged” PDF that provides: <ul style="list-style-type: none"> – Logical reading order – Navigation help – Alternate text descriptions for images and barcodes

Accessibility for the first two categories will be fulfilled automatically, but for the visually challenged, the form author will need to design the template according to some principles:

Logical reading order



- Take care to create a logical structure:
 - Title page
 - Headings
 - Sections and subsections
 - Multi-column text
 - Tables
- Set the alignment on every page for the page itself and for its subforms.
 - Identify units that belong together – such as the address, document information in your form, and structure your template – accordingly with subforms.

Adobe Document Services generate the reading order when rendering the PDF.

The default process is:

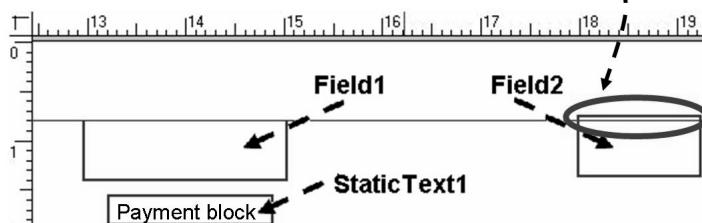
On each page:

- Subforms top to bottom, left to right
- Within each subform, top to bottom, left to right (subforms have an attribute that controls the layout order of their contents, but is set to top to bottom, left to right by default)
 - You can override the default layout order by integrating logical sections in subforms.



Layout in Designer:

Caution!
Slightly higher than Field1!



Resulting PDF with field values looks OK:

A
E
Payment block



Figure 102: Reading Order

In this example, there are three textual objects (labels *Field1*, *StaticText1*, and *Field2* are not part of the template, but have been inserted for better explanation):

Field1 with position X = 13cm and Y = 0.7cm

Field2 with position X = 18cm and Y = 0.65cm

StaticText1 with position X = 13.2cm and Y = 1.55cm

A screen reader would output: "E A Payment".

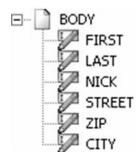
So there are actually three problems:

- “E” is read first, although you would expect it to be the last element in the reading order. This is because the Y position of *Field2* is the top of all text elements.
To correct this, make the Y position of *Field2* the same as the Y position of *Field1*.
- “Payment” is read after the value “A”, which it should describe. The reason is because *StaticText1* is positioned under *Field1*.
To correct this, specify “Payment” as the caption for *Field1*. Available caption positions are *Left*, *Right*, *Top*, and *Bottom*. In the above example, use the *Bottom* position.
- What is the meaning of “E”? If you do not want to show the field caption in the visual representation, then add a description either as tooltip or as custom screen reader text.

If you want to influence the screen reading order without re-arranging layout elements, you can include them in subforms:

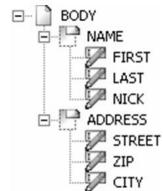


Without subforms



- | | | | |
|----------|------------|----------|----------|
| 1 | First Name | 2 | Street |
| 3 | Last Name | 4 | ZIP Code |
| 5 | Nickname | 6 | City |

With subforms



- | | | | |
|----------|------------|----------|----------|
| 1 | First Name | 4 | Street |
| 2 | Last Name | 5 | ZIP Code |
| 3 | Nickname | 6 | City |

Figure 103: Influencing the Reading Order

In the first example of the above figure, the elements were placed directly on the page (design view). Consequently, they are read out following the rule “First: top to bottom, second: left to right”. As the fields *FIRST* and *STREET* are on the same Y position (height), the content of *STREET* will be read out directly after *FIRST*. The same applies to *LAST* and *ZIP* and to *NICK* and *CITY*. Wrapping those elements that belong to one logical group into one subform helps.



Alternative Text descriptions

- Can be set for images and barcodes.
- Use the *Accessibility* palette to define alternative texts, and add a custom screen reader text.



Figure 104: Alternative Text Descriptions

The *Accessibility* palette allows you to specify custom text for an object that a Microsoft Active Accessibility (MSAA)-compliant screen reader reads as it passes through the form. If a custom screen reader text for the object is available for the object, the screen reader will read this text and not the tooltip, caption, or technical name.

You can also change the default order in which the screen reader searches for text to read on an object-by-object basis, and you can turn off screen reader text for any object.

For floating fields, the screen reader precedence should be set to *None*, as these fields are typically embedded into a static text. Thus, neither a caption nor any other additional text nor the technical name of the field should be read by the screen reader.

The same applies to fields representing text nodes or address nodes from the form context. Since they represent long texts, they should be self-explanatory. If you fail to set the screen reader precedence to *None*, the technical field name will be read by the screen reader, in addition to the text itself.

The support for navigation in PDF documents strongly depends on cooperation between screen reader vendors and Adobe. Analogous to HTML tables, Adobe provides information about table structures, such as cells and where a table begins and ends. Some screen reader applications also let you navigate between cells, get information on the current cell, or leave the table by using enhanced keyboard shortcuts. Adobe Reader 7.0 and the screen reader software JAWS 7.0®, for example, offer this kind of table support.



- Choose *View → Tab Order* to override activation order.
- Click elements in your preferred order.

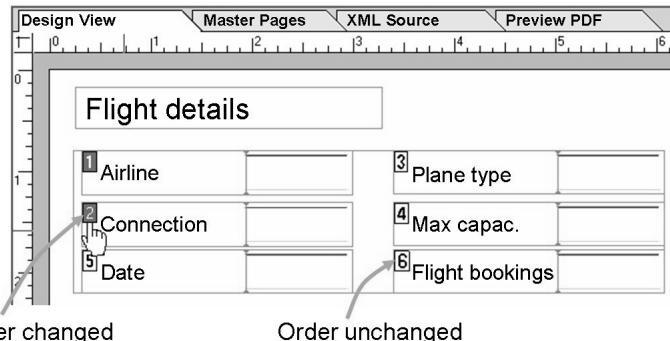


Figure 105: Setting the Tab Order

Users with vision or mobility impairments typically do not use a mouse to navigate through the form; they depend on the keyboard keys and a good tabbing order sequence to ensure that they have full access to all the fields on the form.

The default tabbing order for objects in a form is from left to right, top to bottom, starting from the upper-left corner. Tabbing order respects the existence of subforms, radio buttons, and content areas. For example, if two subforms exist side-by-side, and each subform contains a number of field objects, the tabbing sequence will go through the fields in the first subform before moving on to the next.

The tabbing order is also determined by the vertical position of the master page and the main subform on the page (design view). Whichever is highest receives the tab order first. That is, all objects contained within the content area on the master page or the main subform will be accessed first. Access then moves to the next highest object.

You can change the default tabbing order if you require a different sequence in your form. For example, you may want to change the tabbing order to move through objects in a column, from top to bottom, and then left to right.

Choose *View → Tab Order* and click the objects in the sequence in which you want them to be accessed by tabbing.



Lesson Summary

You should now be able to:

- Name issues that are relevant in an accessible form



Unit Summary

You should now be able to:

- Download and upload a form
- Import a PDF file
- Name issues that are relevant in an accessible form

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Internal Use SAP Partner Only

International Use SAP Partner Only

Unit 9

Appendix

Unit Overview

In the appendix, you will find some information on side issues, like styles, class-based exception handling (relevant for the ABAP application program), Customizing settings, fonts, and the reuse of SAPscript forms and Smart Forms.



Unit Objectives

After completing this unit, you will be able to:

- Describe the formatting concept used for text modules
- Name relevant parts of Smart Styles
- Create and maintain styles
- Explain the mechanisms involved in catching and throwing object-oriented exceptions
- Catch exceptions that might be raised within a program that calls PDF-based forms
- Explain the principles of how Customizing works for PDF-based forms
- Customize one example transaction in mySAP ERP so that PDF-based forms are used for a printing scenario
- Set up an http connection for Adobe document services
- Name basic steps needed to install new fonts
- Use tables in Adobe LiveCycle Designer versions 6.0 and 7.0
- Describe the basic concepts of form processing with SAPscript and Smart Forms
- Name the steps required to reuse or migrate existing form objects from SAPscript and Smart Forms

Unit Contents

Lesson: Styles.....	251
Lesson: ABAP Exception Handling.....	262
Lesson: Customizing	267
Lesson: Basic Administration	271
Lesson: Tables in Adobe LiveCycle Designer 6.0 or 7.0	276

Lesson: Migration.....282

Lesson: Styles

Lesson Overview

This lesson will show you how to create and maintain styles – in particular, Smart Styles. These styles can be used in text modules, which in turn can be included in form contexts.



Lesson Objectives

After completing this lesson, you will be able to:

- Describe the formatting concept used for text modules
- Name relevant parts of Smart Styles
- Create and maintain styles

Business Example

A travel agency wants to print invoices for their customers' flight reservations, using the tool "Interactive Forms". The invoice form uses text modules and SAPscript texts, which in turn use various text formats. As these formats must be administered in styles, the travel agency's form expert needs to know how to maintain such styles.

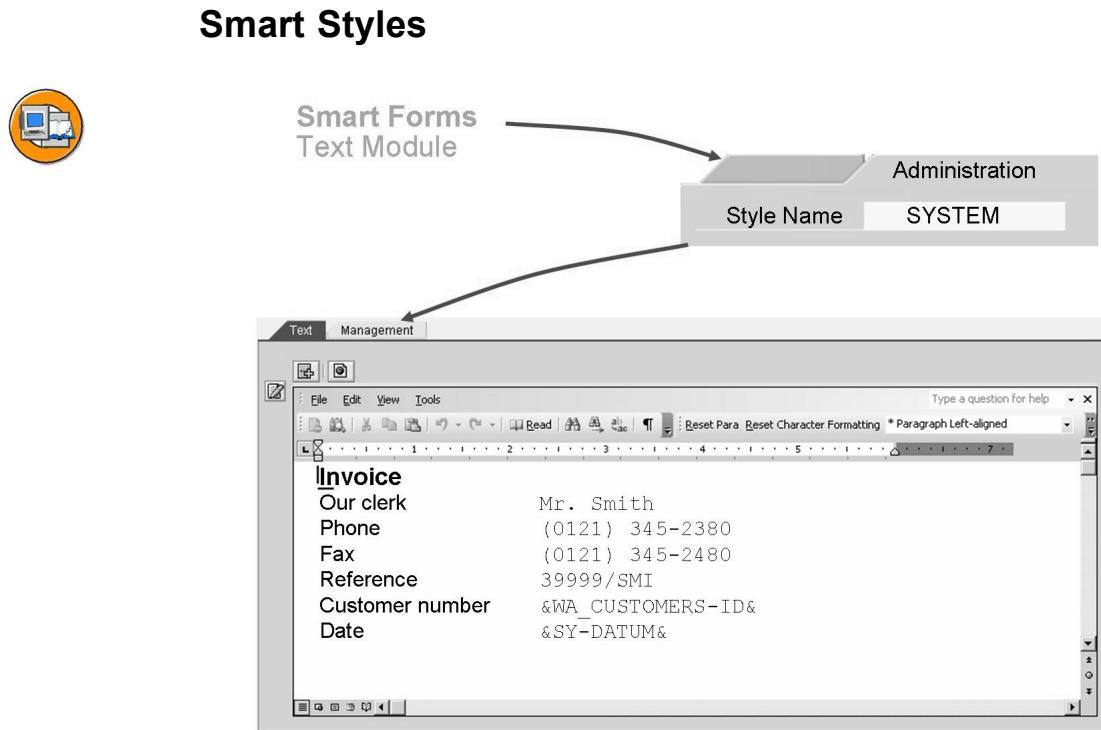


Figure 106: Styles: Used in Text Modules

Styles are collections of character and paragraph formats that are similar to format templates in common word processing programs. You cannot choose a format that has not been added to a style. This ensures that all text modules using the same style have a consistent text design.

If you want to format texts in a text module, select a character or paragraph format from the format lists in the editor. The formats offered in this list depend on the style (Smart Style) chosen.

Each text module must use one style. You enter a style on the *Management* tab. The default for new forms is the *System* style.

When you include a text module in the context of a PDF-based form, you can either use the style attached to the text module or override it by specifying a different style. In the first case, check the *Copy Style From Text Module* option on the *Properties* tab of the text node of the context. In the second case, make sure to enter the name of the style on the *Properties* tab.

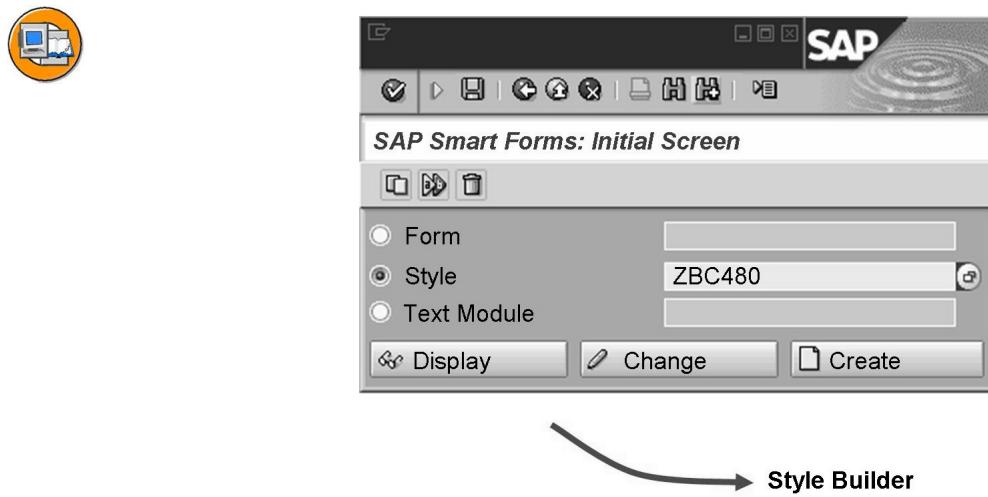


Figure 107: Smart Styles: Initial Screen

To create a Smart Style, you can either select *Style* on the initial screen of transaction SMARTFORMS, or start transaction SMARTSTYLES directly. Enter the name of a style. We recommend that you only change styles in your customer namespace, that is, styles beginning with Y or Z. If required, copy the SAP styles to your customer namespace. To do this, click the corresponding button or choose *Smart Styles → Copy* from the menu.

Like forms, styles are integrated into the SAP transport system. This is why the system prompts you for a package when you first save a style.

You can see the package assigned as well as other style-related information, such as who created or changed the style on the *Administration* tab of the style header data.

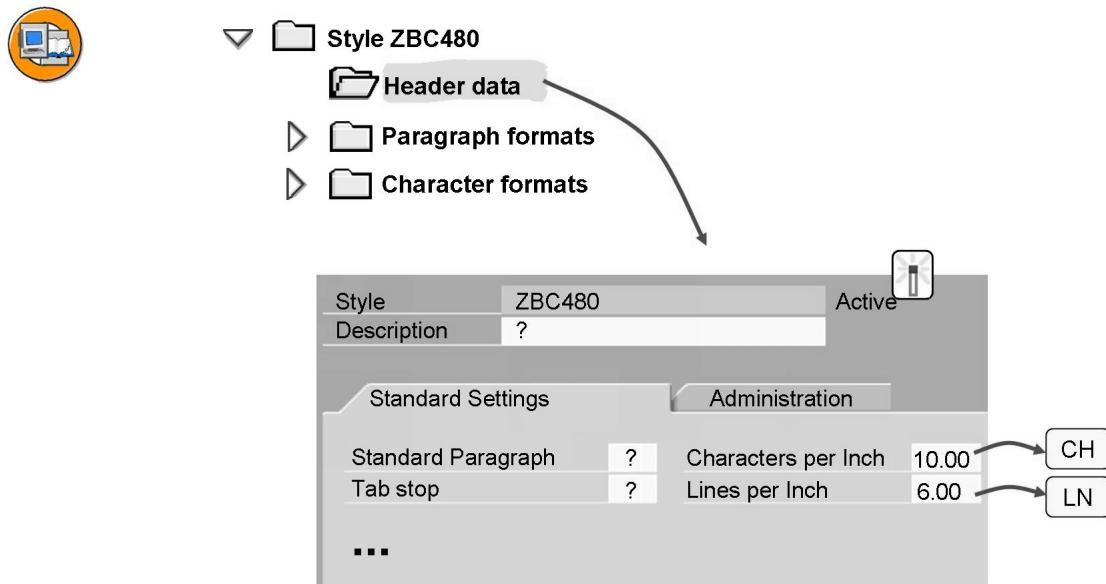


Figure 108: The Style Builder/Header Data I

The maintenance tool for Smart Styles, the Style Builder, consists of two areas:

- The navigation tree is on the left-hand side.
- Detailed information on the element you select in the tree is displayed on the right-hand side. You might also see a preview of the element there.

Like forms, styles can exist in an active and an inactive version. You activate a style by clicking the corresponding button or by choosing *Style → Activate*.

The header data has two tabs: *Standard Settings* and *Administration*. Part of the *Standard Settings* can be seen on the above slide.

- In the *Standard Paragraph* field, you specify which paragraph format is to be used to format the text if the text has no explicit formatting. In the format list of the editor, this standard paragraph is marked with an asterisk (*). You must specify the standard paragraph (and the description of the style) if you want to activate the style. Before you can make an entry here, you must have defined at least one paragraph format.
- The *Tab Stop* sets the distance between the standard tabs. (This setting is relevant only, if the text module that uses that style is included in a Smart Form. It will be ignored if the text module is included in a PDF-based form.)
- In the *Characters per Inch* field, you enter the size of the CH unit of measure (for horizontal size specifications) in the style. Similarly, you determine the LN unit of measure (for vertical size specifications) in the *Lines per Inch* field.

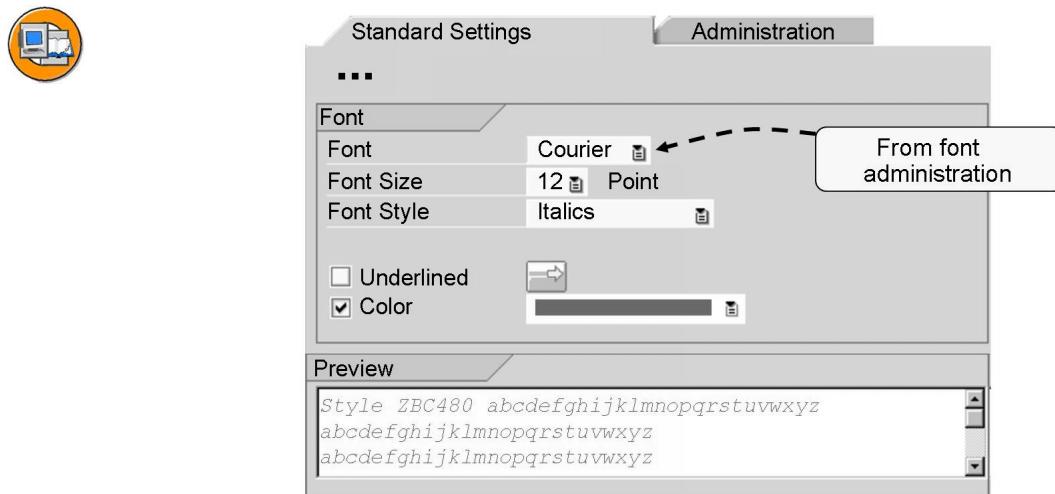


Figure 109: Header Data II

The lower part of the *Standard Settings* tab contains the remaining options. The selections you make in the *Font* box apply to all paragraph and character formats of the style, unless you explicitly specify individual font attributes for these formats.

The font families available depend on the SAP font administration settings (transactions SE73 and SPAD). Note that fonts available for Smart Styles might not be available when used in a text for a PDF-based form.

The font size determines the height of the font in points (pt). The default is 12pt.

You can choose between the following font styles: bold, italic, bold and italic, or none.

You can choose text to be underlined. Specific settings for *Underlined* with regard to spacing, thickness, or brightness will be ignored in the resulting PDF document. (This setting is relevant only if the style is used in a Smart Form.)

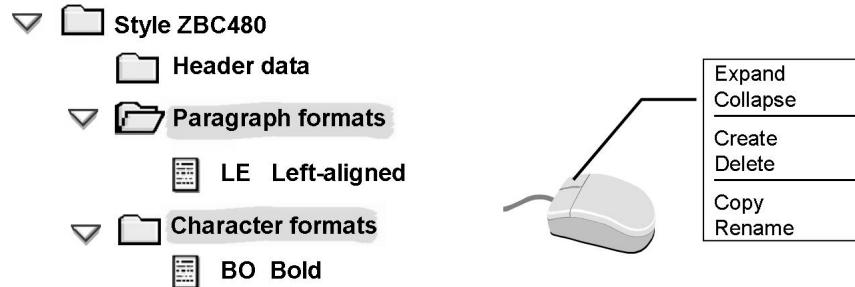
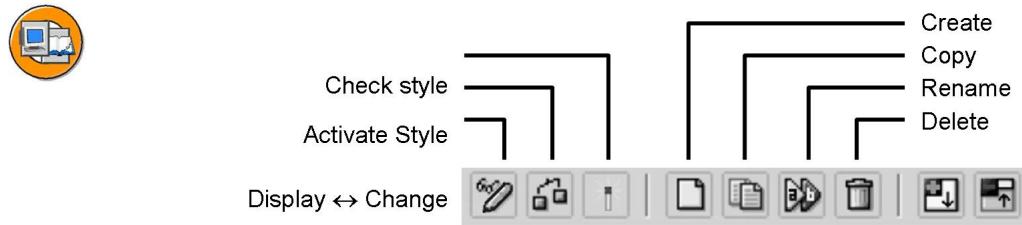


Figure 110: Editing Character and Paragraph Formats

Character and paragraph formats are displayed in the navigation tree. They always have a one- or two-character technical name and a description.

To create, copy, or rename a character or paragraph format, use the *Edit* menu, the buttons, or the context menu (right-click to view the context menu). The action you choose refers to the node currently selected in the tree. You can select nodes with a double-click.

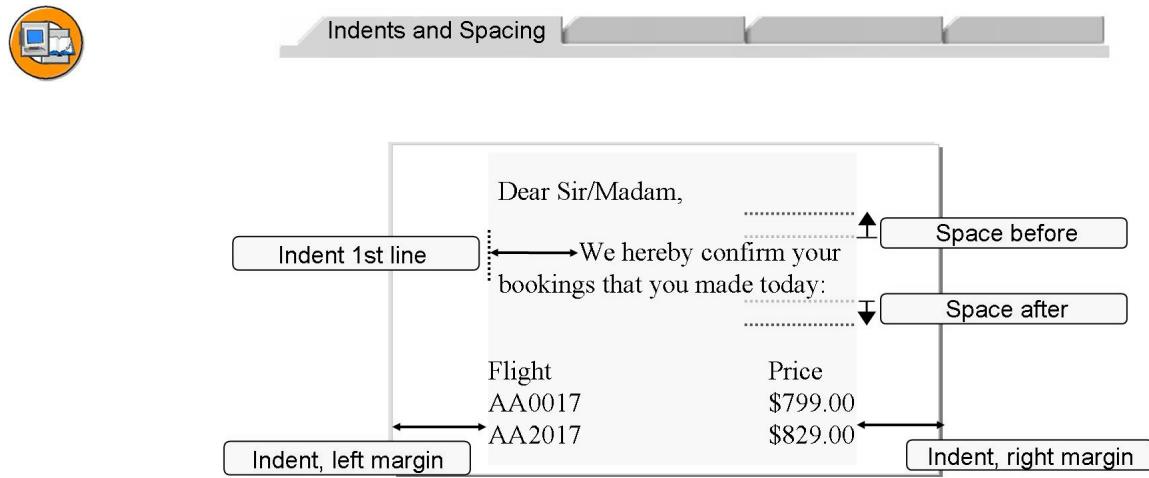


Figure 111: Paragraph Formats: Indents and Spacing

On the *Indents and Spacing* tab of a paragraph format, you can make the following entries:

- *Indent*:
 - *Left/Right Margin of Indent*: Amount of space between the window and the text margin.
 - In addition to the left margin, you can determine a greater indent for the first line of a paragraph.
- *Spacing*
 - *Space Before*: This is the space that is inserted before the first line of all paragraphs that make use of this paragraph format. Similarly, *Space After* defines the space that is left empty after the last line of all paragraphs that make use of this paragraph format.
 - *Line Spacing*: The line spacing is not adjusted automatically if you use a larger font size. To avoid overlaps, you must change the line spacing if required.
- *Text Flow*

These settings are ignored if a text module is inserted into a PDF-based form.

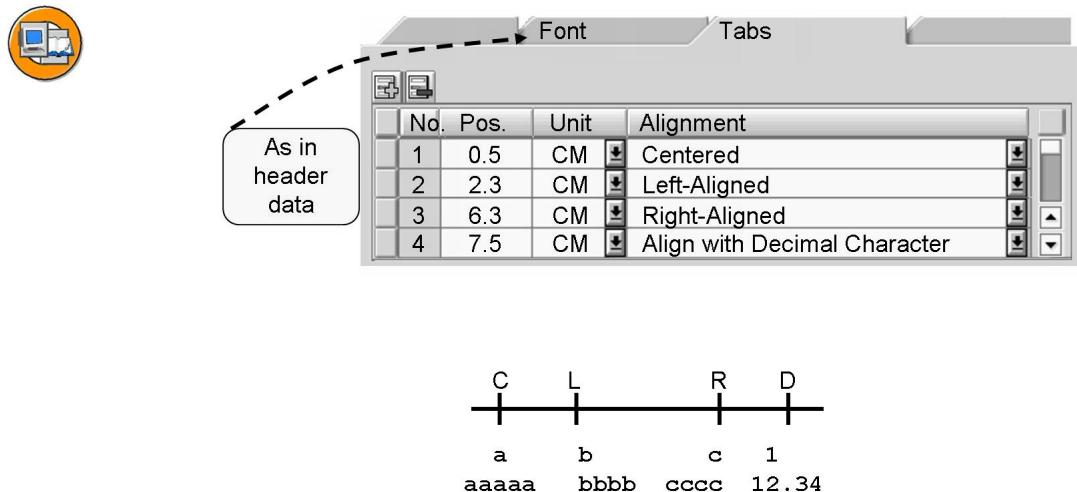


Figure 112: Paragraph Formats: Tab Pages

The *Font* tab has the same fields as the *Standard Settings* tab of the style. If you do not make any settings, the system uses the settings of the header data, including the *Font Family*, *Font Size*, *Font Style*, *Underlined*, and *Color*. You can override the header data entries by specifying a different font. Standard settings, such as the color or the font family, are not displayed in the preview of a paragraph format.



On the *Tabs* tab page, you define individual tab stops for the paragraph format:

- The *Align With Decimal Character* setting lets you align digits with decimal points printed at the tab stop position.
- The *Sign* alignment type works for Smart Forms only. In a PDF-based document, it will be a right-aligned tab.

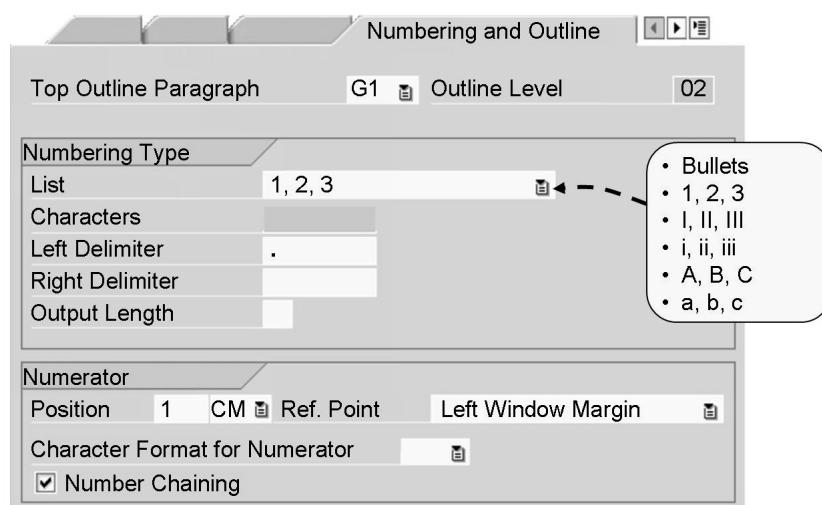


Figure 113: Paragraph Formats: Numbering and Outline I

You can define paragraphs that have outline characters (for example °) or are numbered automatically. To do this, go to the *Numbering and Outline* tab.

If you want to use multilevel numbering or outline (such as 1, 1.1, 1.2, 1.2.1, and so on), you must create a separate paragraph for each level (we recommend that you use the *Copy* function to do this). As the *Top Outline Paragraph*, enter the top paragraph in the hierarchy that controls all other outline or numbering levels. The individual levels are inserted into the navigation tree as subnodes of the top outline paragraph, and the outline level is automatically entered in the corresponding field on the tab.

Choose a numbering type:

- *List Character*: The list character printed at the beginning of the paragraph is the one that you enter in the *Characters* field. You can use up to eight digits.
- Arabic numbers, Roman numerals (lowercase or uppercase), or letters (lowercase or uppercase). You can also specify a *Left* and a *Right Delimiter*, such as an angle bracket to define numberings of the form "a), b), c)," and so on.

In the *Position* field, you enter the space between the numbering/outline character and the left window margin. Make sure that the paragraph margin is not affected by the numerator margin. To avoid overlaps, define a paragraph margin that is large enough.

If you select the *Number Chaining* option, the system uses the previous numerator for multilevel numberings. For example, if level G1 = 1, then level G2 (with number chaining) = 1.2.



Text

	Outline Level	Paragraph	Top Paragraph
1	1	G1	G1
1.1	2	G2	G1
1.1.a)	3	G3	G1
1.1.b)	3	G3	G1
1.2	2	G2	G1
2	1	G1	G1

Paragraph Definition

List	Left Delimiter	Right Delimiter	Number Chaining	Position
1, 2, 3				
1, 2, 3	.		<input checked="" type="checkbox"/>	0.5 cm
a, b, c	.)	<input checked="" type="checkbox"/>	1 cm

Figure 114: Paragraph Formats: Numbering and Outline I

This is an example of a three-level outline. G1 is the top level.

Outlines and numberings are not shown in exact WYSIWYG mode in the preview of the text module editor. For a real WYSIWYG display, you must apply the style to the text module, include the text module in a PDF-based form, and preview the form.

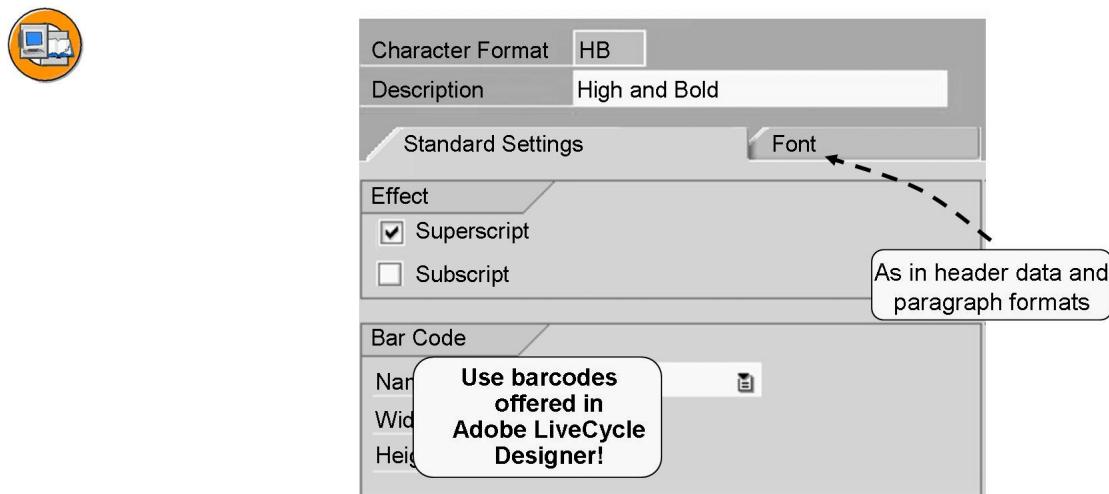


Figure 115: Character Formats

The character format has two tabs: *Standard Settings* and *Font*. The *Font* tab has the same fields as the *Standard Settings* tab of the style or the *Font* tab of a paragraph. The entries you make here override the settings of the header data or the paragraph format used in the text, including *Font Family*, *Font Size*, *Font Style*, *Underlined*, and *Color*. Note that font-related settings made in the standard settings or a paragraph format are not displayed in the preview of a character format.

On the *Standard Settings* tab, you can set the attributes *Superscript* and *Subscript*.

Barcodes must be printed using the Adobe LiveCycle Designer barcode feature, which is much more flexible than the traditional barcode technology implemented in transactions SE73/SPAD. (These barcodes will be simulated as parallel bars only.)

SAPscript Texts and Styles

For SAPscript texts, SAPscript styles can be assigned statically for the entire text, or dynamically using the / :style command. All style assignments for a SAPscript text are ignored when it is included in a PDF-based form. If you want to include a SAPscript text with a special style into the context of a PDF-based form, make sure to enter the name of the style on the *Properties* tab.



Hint: If you include SAPscript texts in PDF-based forms, you can use only Smart Styles.



Lesson Summary

You should now be able to:

- Describe the formatting concept used for text modules
- Name relevant parts of Smart Styles
- Create and maintain styles

Lesson: ABAP Exception Handling

Lesson Overview

In a typical ABAP program for a print scenario with PDF-based forms, function modules make use of two exception concepts: the old one (with system field *sy-subrc*) and the new one, which is object-oriented. This lesson will show you the basics of these class-based exceptions.



Lesson Objectives

After completing this lesson, you will be able to:

- Explain the mechanisms involved in catching and throwing object-oriented exceptions
- Catch exceptions that might be raised within a program that calls PDF-based forms

Business Example

A travel agency wants to print invoices for their customers' flight reservations, using the tool "Interactive Forms". The form template is already available, but the application (an ABAP program) still needs to be written. One of the function modules that are required for the program uses the class-based exception concept. The programmer wants to learn about its basics.

In SAP Web Application Server 6.10, a new exception concept was introduced. Coding (typically function modules, subroutines, and methods) that makes use of this concept does not set *sy-subrc* any more. Instead, an object is created that can contain any kind of information. Typically, it contains an error text plus information on which a message should be displayed.

The programmer of a function module chooses either the old or the new exception concept.



```

PARAMETERS: pa_int1 TYPE i DEFAULT 1,
            pa_int2 TYPE i DEFAULT 2.
DATA:       result  TYPE p DECIMALS 2,
           text    TYPE string,
           r_error TYPE REF TO cx_root.

TRY.
  result = pa_int1 * pa_int2. WRITE result.
  result = pa_int1 / pa_int2. WRITE result.

  CATCH cx_sy_arithmetic_overflow INTO r_error.
  text = r_error->get_text( ).
  MESSAGE text TYPE 'E'.

  CATCH cx_root.
  MESSAGE 'Unclassified error' (e01) TYPE 'E'.

ENDTRY.
...

```

No exception

Figure 116: Class-Based Exception Handling I

Code passages where such a new exception might be raised must be surrounded by TRY..ENDTRY.. If an exception is raised (also known as thrown), the program flow will automatically continue with the first CATCH command that fits. If no suitable CATCH command is found in the current processing block or a surrounding one, the program will terminate with a dump. If no exception is raised, no CATCH clause will be evaluated.

The CATCH syntax consists of the keyword CATCH plus the name(s) of exception classes. Optionally, you can add INTO and the name of a suitably typed reference variable. This variable will point to the created exception object so that you can access it to get details about the exception.

All exception classes provide the method `get_text`. Call this method to query a short description of the exception.

If you have multiple CATCH clauses, make sure to give them in ascending order, that is, listing subclasses first and superclasses last.

ABAP's root exception class is called `CX_ROOT`. All exception classes are direct or indirect subclasses of `CX_ROOT`.



```

PARAMETERS: pa_int1 TYPE i DEFAULT 999999999999,
             pa_int2 TYPE i DEFAULT 999999999999.
DATA:      result  TYPE p DECIMALS 2,
             text    TYPE string,
             r_error TYPE REF TO cx_root.

TRY.
  result = pa_int1 * pa_int2. WRITE result.
  result = pa_int1 / pa_int2. WRITE result.

Overflow  CATCH cx_sy_arithmetic_overflow INTO r_error.
          text = r_error->get_text( ).
          MESSAGE text TYPE 'E'.

          CATCH cx_root.
          MESSAGE 'Unclassified error' (e01) TYPE 'E'.

ENDTRY.
...

```

Figure 117: Class-Based Exception Handling II

If in the above example, the calculation of `pa_int1 * pa_int2` leads to an arithmetic overflow; an exception of class `CX_SY_ARITHMETIC_ERROR` will be raised. Hence, the program will jump to the first `CATCH` clause and execute it.



```

PARAMETERS: pa_int1 TYPE i,
             pa_int2 TYPE i DEFAULT 0.
DATA:      result  TYPE p DECIMALS 2,
             text    TYPE string,
             r_error TYPE REF TO cx_root.

TRY.
  result = pa_int1 * pa_int2. WRITE result.
  result = pa_int1 / pa_int2. WRITE result.

Division by zero/ Other exception CATCH cx_sy_arithmetic_overflow INTO r_error.
          text = r_error->get_text( ).
          MESSAGE text TYPE 'E'.

          CATCH cx_root.
          MESSAGE 'Unclassified error' (e01) TYPE 'E'.

ENDTRY.
...

```

Figure 118: Class-Based Exception Handling III

If in the above example, the calculation of `pa_int1 / pa_int2` fails because `pa_int2` equals zero, an exception of class `CX_SY_ZERODIVIDE` is raised. As no CATCH clause for that exception class exists, the program jumps to the first CATCH clause that is available for a superclass of `CX_SY_ZERODIVIDE`. In this case, the program jumps to the second CATCH clause, as `CX_ROOT` is the superclass of all exception classes, in particular of `CX_SY_ZERODIVIDE`.



Function module FP_FUNCTION_MODULE_NAME Active		
Attributes	Import	Export
Changing	Tables	Exceptions
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/> Exceptn Classes
Exception	Short text	Long txt
<code>CX_FP_API_REPOSITORY</code>	Exception API (Repository)	
<code>CX_FP_API_USAGE</code>	Exception API (Use)	
<code>CX_FP_API_INTERNAL</code>	Exception API (Internal)	

```

DATA: r_error          TYPE REF TO cx_fp_api,
      error_string    TYPE string.

TRY.
  CALL FUNCTION 'FP_FUNCTION_MODULE_NAME'
    EXPORTING ...
    IMPORTING ...
  CATCH cx_fp_api INTO r_error.
    error_string = r_error->get_text( ).
    MESSAGE error_string TYPE 'E'.
ENDTRY.

```

Figure 119: Exceptions: FP_FUNCTION_MODULE_NAME

When calling a global method or a function module, check whether the old exception concept is used (`sy-subrc`) or the new one (class-based exceptions). In the latter case, surround the call of that method or function module with suitable TRY...CATCH...ENDTRY... source code.

In the case of function module `FP_FUNCTION_MODULE_NAME`, exceptions of classes `CX_FP_API_REPOSITORY`, `CX_FP_API_USAGE` and `CX_FP_API_INTERNAL` are possible. They have these superclasses: `CX_ROOT` > `CX_STATIC_CHECK` > `CX_FP_EXCEPTION` > `CX_FP_API`. If you do not need to be specific about exactly which exception is raised, one CATCH clause for any of these superclasses would be fine.

All exception classes specific to the processing of PDF-based forms are subclasses of `CX_FP_EXCEPTION`.



Lesson Summary

You should now be able to:

- Explain the mechanisms involved in catching and throwing object-oriented exceptions
- Catch exceptions that might be raised within a program that calls PDF-based forms

Lesson: Customizing

Lesson Overview

This lesson gives you some examples of how to customize your applications so that PDF-based forms are used (instead of SAPscript forms or Smart Forms).



Lesson Objectives

After completing this lesson, you will be able to:

- Explain the principles of how Customizing works for PDF-based forms
- Customize one example transaction in mySAP ERP so that PDF-based forms are used for a printing scenario

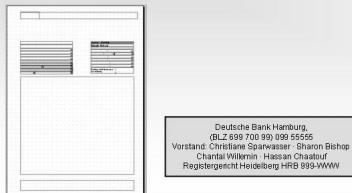
Business Example

A company wants to use the tool “Interactive Forms” for printing scenarios in mySAP ERP. They need to know which forms are provided by SAP, which ABAP programs go with them, and where in Customizing to determine the details.

Dunning



- 1 Form F150_DUNN_PDF**
Text modules
Graphics if desired
- } Copy and adjust
Create



Activate

- 2 Customizing: SAPscript or Smart Forms or PDF?**

SAP Customizing Implementation Guide

- Financial Accounting
- Financial Accounting Global Settings
- Business Transaction Events

Figure 120: Customizing: Dunning I

To adjust form and texts, copy the form F150_DUNN_PDF in the Form Builder (transaction SFP) to your customer namespace and then adjust and activate it. Adjust the inserted texts (such as the address) and add a company logo if desired.

Ensure that a PDF-based form is used instead of SAPscript or Smart Forms. In the Implementation Guide, choose *Financial Accounting → Financial Accounting Global Settings → Business Transaction Events*. In the menu, choose *Settings → P/S Modules → of an SAP Application*. Change the function module to FI_PRINT_DUNNING_NOTICE_PDF for the Business Transaction Event 1720 with the Application Indicator FI-FI. Save your entries.

**3**

Customizing: Which Form?

SAP Customizing Implementation Guide

- **Financial Accounting**
- **Accounts Receivable and Accounts Payable**
- **Business Transactions**
- **Dunning**
- **Printout**
- **Assign Dunning Forms**

Figure 121: Customizing: Dunning II

To enter your PDF-based form in Customizing, go to the Implementation Guide (transaction SPRO), and follow the path shown in the figure above. Select the desired procedure (for example, *Four-level dunning, every two weeks*), then choose *Forms for normal or legal dunning procedure* in the tree and enter the company code. Enter your copy of the dunning form for the desired dunning level. Save your data and ignore the warning saying that the form does not exist or is inactive (this warning is displayed because the system only checks for SAPscript forms).

Request Printouts (Public Sector)



Forms:

- FM_PAYMENTREQUEST
- FM_ACCEPTANCEREQUEST



Copy, adjust, activate

SAP Customizing Implementation Guide

- Public Sector Management
- Funds Management Government
- Funds Management-Specific Postings
- Requests
- Additional details
- Request Printout: Assign own forms

Form Number	Form Type	PDF-based form
consecutive number per form type, e.g. 1	01 (Payment request)	FM_PAYMENTREQUEST
consecutive number per form type, e.g. 1	02 (Acceptance request)	FM_ACCEPTANCEREQUEST

Figure 122: Customizing: Request Printouts (Public Sector)

For the Public Sector, a number of payment request forms come with mySAP ERP as PDF-based forms. Among them, you find the payment request (form FM_PAYMENTREQUEST) and the acceptance request (form FM_ACCEPTANCEREQUEST). If you want to make adjustments, copy the form to your customer namespace and then modify the copy as required.

In the SAP Customizing Implementation Guide (transaction SPRO), follow the path shown to find the Customizing table and enter your form copy.

Request printouts have a special feature: For one given form type, more than one form can be entered in the customizing table. You can even mix SAPscript and PDF. At application runtime, the user will get a dialog box with a choice of possible forms.



Lesson Summary

You should now be able to:

- Explain the principles of how Customizing works for PDF-based forms
- Customize one example transaction in mySAP ERP so that PDF-based forms are used for a printing scenario

Lesson: Basic Administration

Lesson Overview

This lesson will teach you how to handle basic installation issues connected with printing scenarios with PDF-based forms. See the ADS Installation Guide and the Configuration Guide for details. See <http://service.sap.com/instguides>.



Lesson Objectives

After completing this lesson, you will be able to:

- Set up an http connection for Adobe document services
- Name basic steps needed to install new fonts

Business Example

A travel agency wants to print invoices for their customers' flight bookings using the tool "Interactive Forms". The installation of the technology required (SAP NetWeaver Web Application Server with ABAP, J2EE engine, and Adobe Document Services) is done by Basis administration experts. However, the form designers and application programmers involved with the tool want to have at least a rudimentary understanding of the technology.

Setting up an http Connection for Adobe Document Services

For a print scenario with PDF-based forms, you must have installed Adobe Document Services on an SAP NetWeaver Application Server with ABAP and J2EE. To call Adobe document services from an ABAP application, an http connection must be created as follows:

1. Log on to your SAP Web AS central instance host.
2. Call transaction *SM59*.
3. Choose *Create* and name the connection ADS.

You can also create several http connections with random names, but these must be specified explicitly in the ABAP program: Field *connection* of changing parameter *ie_outputparams* of function module *FP_JOB_OPEN*.

4. Enter at least the following:
 - *RFC destination*: **ADS**
 - *Connection Type*: **G**
 - *Description*: <your description>
5. Choose ENTER.
6. Choose the *Technical settings* tab and enter at least the following:
 - *Target Host*: Enter the name of the host where your SAP J2EE engine is located.
 - *Service No.*: Enter the SAP J2EE http port number. The following naming convention applies: 5<J2EE_instance_number>00 (50000, for example, if your J2EE instance is 00).
 - *Path Prefix*: Enter exactly the string **/AdobeDocumentServices/Config?style=rpc** or **/AdobeDocumentServicesSec/Config?style=rpc** if you want to use SSL
 - When you press ENTER, a warning is displayed: “Query string not allowed”. Ignore this warning.
7. Choose the *Logon/Security* tab and configure the security account to your security requirements.
8. Save your settings.
9. Choose *Test Connection*.
10. A screen is displayed. The field *status_reason* should have the value OK if the test is successful.

Administering Fonts

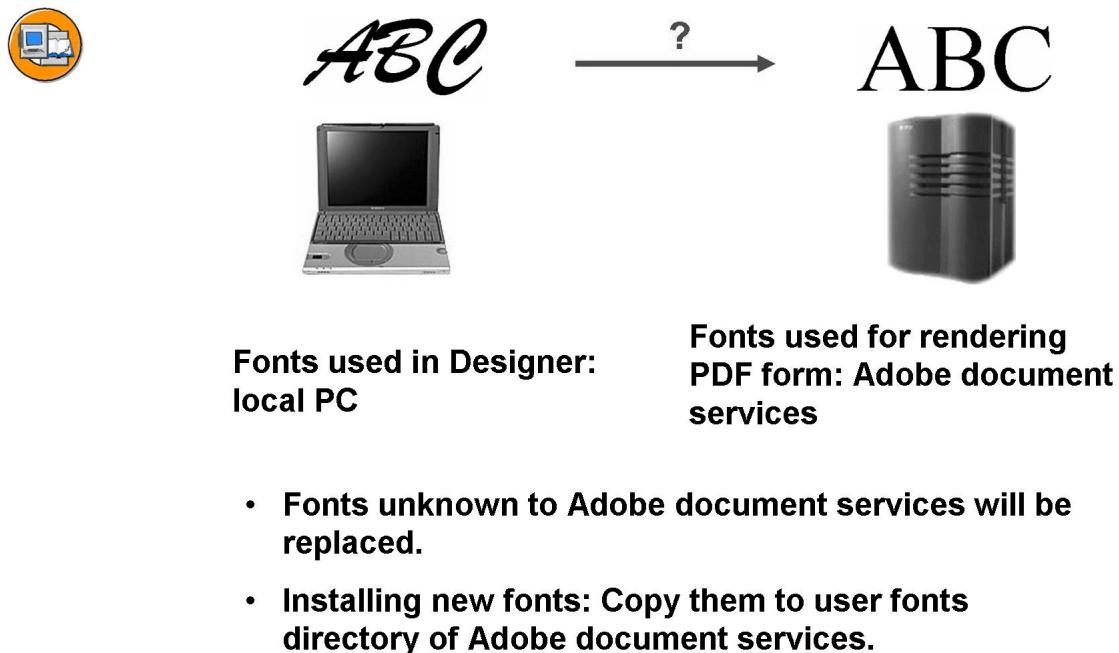


Figure 123: Installing New Fonts

Adobe document services require access to fonts that are installed with the Font Manager Module. This module contains a number of Adobe-bundled fonts installed in the directory `/usr/sap/<SAPSID>/<Installation>/j2ee/os_libs/adssap/FontManagerService/fonts/adobe`.

You can also add fonts obtained from other vendors. The types of fonts you can add are OpenType® (.otf), TrueType® (.ttf), and PostScript® Type 1 (.pfb/.pfm).

To add fonts:

1. Create a subdirectory called fonts/ below the `/usr/sap/<SAP-SID>/SYS/global/AdobeDocumentServices/FontManagerService` directory. Replace <Installation> with JC<xx> if your system is a SAP Web AS J2EE system, or with DVEBMGS<xx> if your system is a SAP Web AS ABAP + J2EE system (J2EE Add-In).
2. Create a subdirectory named customer/ below the fonts/ directory created in the previous step.
3. Copy your fonts into the `/usr/sap/<SAPSID>/SYS/global/AdobeDocumentServices/FontManagerService/fonts/customer` directory.
4. Restart the service "Document Services Font Manager". Restart the application com.adobe/AdobeDocumentServices.

An important file for the administration of fonts is /usr/sap/<SAP-SID>/SYS/global/AdobeDocumentServices/lib/xfa.xci. Here you can determine for the different printer types (PS, PCL, PDF, Zebra) whether fonts will be embedded. The following fonts are never embedded: Courier, Helvetica, Times, Arial, Times New Roman.

Fonts that are not available for Adobe Document Services at render time will be replaced according to the substitution rules in the file /usr/sap/<SAP-SID>/SYS/global/AdobeDocumentServices/lib/xfa.xci. At runtime, you can specify a different file to be used with the help of the field *XDCNAME* of parameter IE_OUTPUTPARAMS of function module FP_JOB_OPEN. Furthermore, in table FP FONTREPL, you can specify replacement fonts for specific forms and languages.

Font replacement at render time is not to be confused with font replacement as design time. Adobe LiveCycle Designer has its own font replacement mechanism. You can set the rules by choosing *Tools* → *Options* → *Document Handlung* → *Modify Font Substitutions*.



Lesson Summary

You should now be able to:

- Set up an http connection for Adobe document services
- Name basic steps needed to install new fonts

Lesson: Tables in Adobe LiveCycle Designer 6.0 or 7.0

Lesson Overview

This lesson shows you how to use tables in Adobe LiveCycle Designer versions 6.0 and 7.0. If you have Designer version 7.1 or higher, these old tables will automatically be displayed as new ones.



Lesson Objectives

After completing this lesson, you will be able to:

- Use tables in Adobe LiveCycle Designer versions 6.0 and 7.0

Business Example

A travel agency wants to print invoices for their customers' flight reservations, using the tool "Interactive Forms". The form needs to be able to print a list of items.

Create a Table



The screenshot shows the 'Interface/Context' palette on the left, which lists the 'IT_BOOKINGS' object with its 'DATA' section expanded, showing fields: MANDT, CARRID, CONNID, and FLDATE. A circled '1' with the text 'Only required fields included?' points to this list. Below it, the 'Data View' palette shows a tree structure with 'BC480' and 'IT_BOOKINGS'. A circled '2' with the text 'Drag to body page of layout' points to the 'IT_BOOKINGS' node. An arrow points from the 'Data View' palette to a table structure in the 'Layout Editor' below. The table has four columns labeled CARRID, CONNID, and two empty columns. The bottom row contains the label FLDATE followed by an empty column.

Figure 124: Creating the Booking Table

Even though you could create a table manually by using the *Object* and *Hierarchy* palettes only, it is by far easier to do so by dragging and dropping a table from the *Data View* to the Layout Editor. (The table will typically be an internal table from

the form interface that is used in the context.) You can then remove the fields that you do not want to have included in the form table. However, it might be a good idea to either reduce the line type of the table in the interface or to go to the context and set the superfluous fields of the internal table to inactive. Both methods reduce the amount of data transferred to Adobe Document Services at runtime.

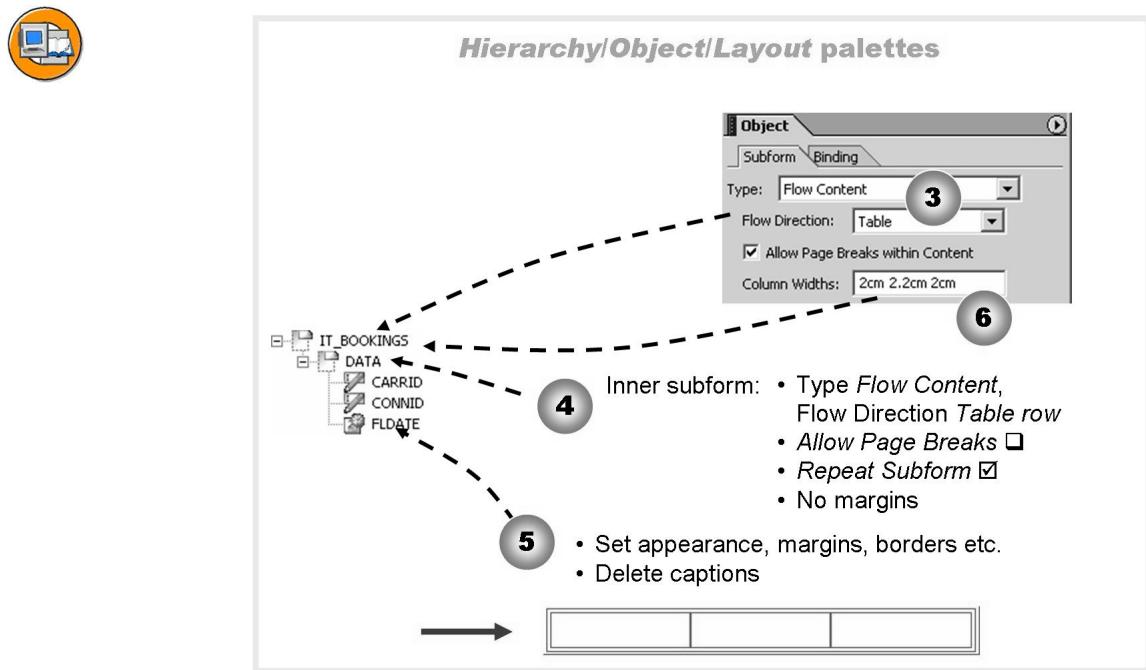


Figure 125: Creating the Booking Table: Cells

If you drag and drop an internal table from the *Data View* to the layout, Adobe LiveCycle Designer actually creates one subform (called like the internal table it represents), a second subform nested in the first one, and the individual fields. This is due to the XML representation of the data.

Here are the necessary steps to make sure the internal table is printed as a form table:

- Check that the **outer subform** (*IT_BOOKINGS* in the example above) is of type *Flow Content* with *Flow Direction* set to *Table*. (This specific option is not available in the standalone version of Adobe LiveCycle Designer.)

Deselect *Repeat Subform for Each Data Item*.

If all of the content should be printed on one page, deselect the *Allow Page Breaks within Content* option. If, at runtime, only little data needs to be processed, this flag will be ignored. If, however, more data is available than fits on the current page, an automatic page break will be inserted, and the table will start on the following page.

If you want your table to have an outer border only, you would set it on the *Border* palette of the outer subform.

- Make sure the **inner subform** (always *DATA*) is of type *Flow Content* with *Flow Direction* set to *Table Row*. (This specific option is not available in the standalone version of Adobe LiveCycle Designer.) Furthermore, deselect the *Allow Page Breaks within Content* option.

Select *Repeat Subform for Each Data Item*.

For the margins (*Layout* palette), select 0cm for top, bottom, left, and right.

The width and height should be set to *Expand to fit*, so that content of various sizes can be integrated.

If you want to have your table rows separated by horizontal lines, set them on the *Border* palette of the inner subform.

- **Individual cells** (in the example above: *CARRID*, *CONNID*, *FLDATE*): If your internal table has more fields than you actually want to print in the form table, delete them from the *Hierarchy*. (Note, however, that without scripting it is not possible to process the same internal table twice; you cannot “save” some fields for a second round.)

For print forms, you would typically avoid 3D frames. However, by default, all fields will be included in the layout with *Appearance* set to *Sunken Box*, which will result in 3D frames. To change this, choose *Object → Field → Appearance* and select the *None* entry.



Hint: In the *Hierarchy*, you can select and change multiple table fields at the same time. Either hold the CTRL key pressed down and select the fields one after another, or select the first table field, press the SHIFT key, and select the last table field. Note, however, that you can change the *Appearance* of several fields at a time only if they have the same type, for example, if they are all text fields.

You must also do away with the captions; otherwise they would be repeated for every item line. Select the fields in the *Hierarchy* (you can select them all at once), choose *Layout → Caption → Position* and select *None*.

The widths of the individual cells are irrelevant. They are determined centrally for the entire table. For the height, enter a minimum and check the *Expand to fit* option.

If you want to have a table with cell borders (frames) and/or shadings, you would set them on the *Border* palette of the individual cells.

Finally, select a margin that suits your design ideas, for example, 0.1 cm.

The best way to determine the individual cell widths is to set them on the *Object* palette, *Subform* tab, *Column Widths* field. In the “Colum Widths” field, enter 2cm 2.3cm 3cm 3.5cm 2cm 4cm. Note that inch will always be the default measurement unless you explicitly give a different measurement for every column. The unit that you might have specified on the *Drawing Aids* palette is used for rulers and guidelines only. Specify a value for every cell in the row. Make sure you do not insert spaces between the values and the units. (You can also go to the individual fields and change their widths on the *Layout* palette, but this will make inserting the heading more complicated.)

Test your form with data to find out whether cells are wide enough.

Creating Headers for Tables



Hierarchy palette/Object palette/Layout editor

- 1 Insert subform, call it HEADER



- 2 HEADER: Subform type Flow Content,
Flow Direction: Header

Library palette

- 3 Include one static text per column, set details

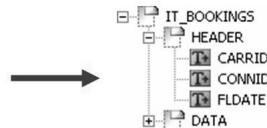


Figure 126: Booking Table: Creating Subform for Header

To create headers for the table, the following steps are required:

1. In the *Hierarchy*, select the outer subform (*IT_BOOKINGS* in the example above) and choose *Insert Subform* from the context menu. Give it a meaningful name, for example, *HEADER*. To do so, click the subform in the *Hierarchy* and then press F2; the name field is now ready for input. Change the name and press Enter.

Make sure the new subform is positioned above the *DATA* subform.

2. Set the subform type to *Flow Content*, and set the *Flow Direction* to *Header*.
3. Insert one static text into the subform for every column. Enter the texts and set details like height, shading, or frames.

Be sure to set the same margins that you have set for the table fields.



Object palette → Binding tab

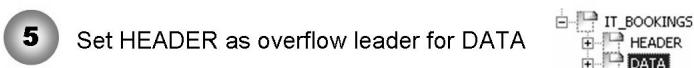
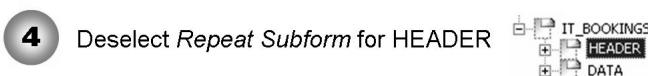


Figure 127: Setting Subforms as Headers

Typically, you would want the headers to be repeated after a page break. To achieve this, the following is required:

1. For the *HEADER* subform, go to the *Object palette*, *Binding tab*, and deselect the *Repeat Subform for Each Data Item* option.
2. For the *DATA* subform, go to the *Object palette*, *Binding tab*, and set *Overflow Leader* to *HEADER*. Note that for *HEADER* itself, nothing is said about it being an overflow header.



Lesson Summary

You should now be able to:

- Use tables in Adobe LiveCycle Designer versions 6.0 and 7.0

Lesson: Migration

Lesson Overview

This lesson will show you which parts of SAP's traditional form tools (SAPscript and Smart Forms) can be reused in a printing scenario with "Interactive Forms". It will also show you which steps are necessary if you want to migrate from one of these solutions to the new one.



Lesson Objectives

After completing this lesson, you will be able to:

- Describe the basic concepts of form processing with SAPscript and Smart Forms
- Name the steps required to reuse or migrate existing form objects from SAPscript and Smart Forms

Business Example

A travel agency has SAPscript forms and Smart Forms and is wondering which of these components can be re-used, which could be converted, and which must be migrated and manually adapted.

Re-using SAPscript and Smart Forms – Overview

The reuse of SAPscript forms as PDF-based forms is quite difficult because their concepts differ greatly. In particular, SAPscript has no clear separation between the form logic and the ABAP program. For example, printing a table of items requires you to call a function module (WRITE_FORMS) several times, which will then trigger data output in the form. If, within the form, you want to perform ABAP calculations, you actually have to do this outside of the form (in special form routines). That is why there is so much communication between form and program.

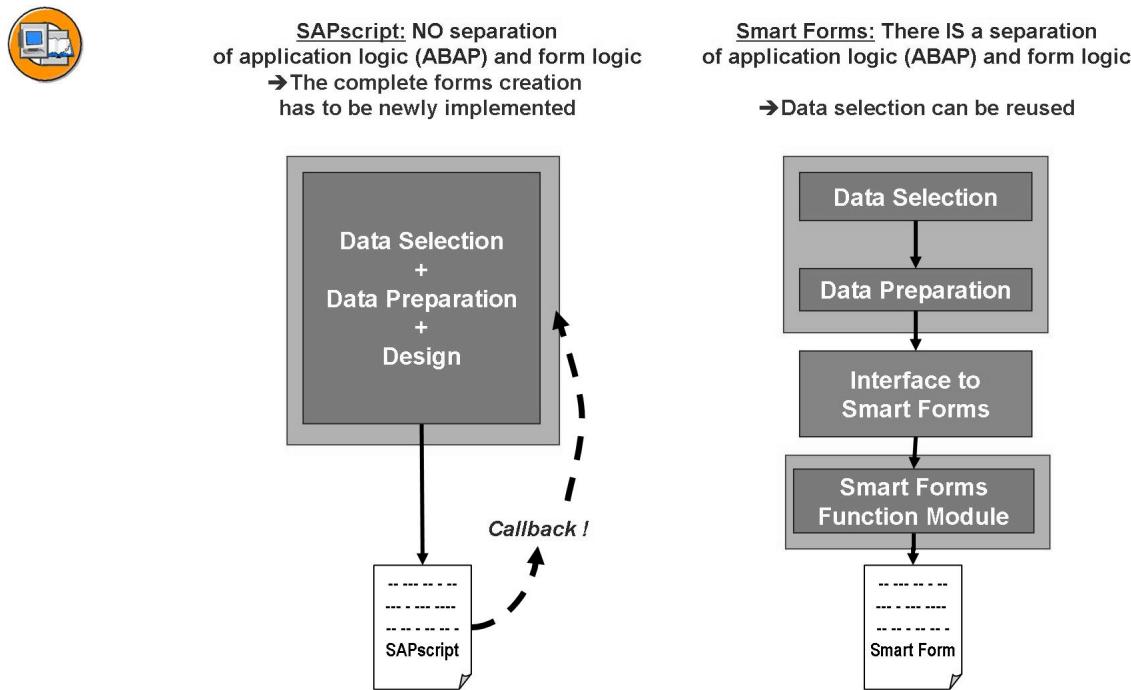


Figure 128: Forms Generation Process Without Adobe

Smart Forms have had a clear separation of form logic and program from the beginning. The program's task is basically to collect the data and then pass it on to the interface of the Smart Form. All form logic is done within the form. As this idea of an interface has been taken up for PDF-based forms, it is much easier to reuse Smart Forms and their programs than it is to reuse SAPscript components.



Objects	SAPscript	Smart Forms
Forms	Manual conversion	Automatic conversion supported (loss of details!)
Programs	Manual conversion	Re-use possible (with or without changes)
Texts	Re-use possible except for: • SAPscript fields • SAPscript commands	Reuse possible except for: • Smart Forms system fields • Output options
Styles	Migration to Smart Style possible	Reuse in Smart Forms texts possible
Graphics	RSTXLDMC: Re-use not possible SE78: migration to MIME Repository objects supported	migration to MIME Repository objects supported

Figure 129: SAPscript and Smart Forms: Possible Reuse

Both SAPscript texts and Smart Forms texts (text modules) can be reused in PDF-based forms. Note, however, that commands within SAPscript text will generally be ignored, as will output options. Furthermore, make sure all fields in the texts are known in the form context. The use of unknown fields (including all SAPscript or Smart Forms fields) leads to program termination at runtime.

Graphics that have been imported with report RSTXLDMC cannot be reused. However, if you still have the original files on a file server, all you have to do to make them accessible for PDF-based forms is to copy them to a place that can be accessed by Adobe document services.

Graphics imported with transaction SE78 can be migrated into the MIME Repository. Run report RSXFT_MIGRATE_BDS_GRAPHICS to do this. Note that as of SAP NetWeaver 2004, Adobe Document Services cannot access the MIME Repository directly.

Migrating SAPscript Forms



- Recommendation: Create a new PDF-based form from scratch.
- Split up big forms into small forms (representing only one business case each)

Even though there is a migration tool from SAPscript to Smart Forms and one from Smart Forms to PDF-based forms, this double migration requires too many manual adjustments. Hence, it is easier to create a new PDF-based form.

In SAPscript (and in Smart Forms), it is common practice to have one big form for different business scenarios and use conditions within the form to determine the actual type of business scenario at runtime. For example, SAPscript form RVORDER01 is used for Inquiry/Quotation/Order Confirmation. You should avoid using this technique in PDF-based forms.

Comparison of SAPscript and PDF-Based Forms

SAPscript	PDF-based form
MAIN window	Page (design type) of type <i>Flow Content</i> that is included in the content area of a master page
Multiple MAIN windows (for text in columns)	Multiple content areas with text fields; choose <i>Object → Field → Allow Multiple Lines</i> .
TOP area of MAIN window	Table header
BODY area of MAIN window	Normal subform of type <i>Flow Control</i> if part of a table: Table/table row
Secondary window	Objects on master page
Text elements, w/o symbols	Static texts
Text elements, with symbols	Static text with floating fields
Tabs	In a static text, tabs can be used, but position or type are not defined; use in subform with <i>Position Content</i> instead
Include texts	Include texts (alternative: text modules)
Include images	Static images without embedding (think about embedding, though)
Units pt, ch, ln	Convert to uniform units in SAPscript and use mm, cm, or in for new form

Comparison of SAPscript and PDF-Based Forms

SAPscript	PDF-based form
/: define	Global Variable (<i>Edit → Form Properties → Variables</i>)
/: new-page	Insert new page (design view) with <i>Place: Top of next page</i>
/: new-page XYZ	Insert new page (design view) with <i>Place: Top of page XYZ</i>

/: if	<ul style="list-style-type: none"> Content element with condition Content alternative Subform with min count = 0 Form element with scripting
/: case	see /: if (nested conditions required)
/: box	Settings on <i>Border</i> palette
/: set date mask	Palette <i>Object</i> → <i>Field</i> → <i>Display Pattern</i> or scripting; use one of four predefined date objects that come with scripting.
/: set time mask	Palette <i>Object</i> → <i>Field</i> → <i>Display Pattern</i> or scripting
/: set country	In print program: field <i>Country</i> of parameter <i>/Ibcdwb/docparams</i> of generated function module
/: protect	Subform with <i>Allow Page Breaks within Content</i> unchecked; to prevent content of several subsequent forms from page breaks, use subforms with <i>Keep with Previous/Next</i>
/: perform	N/A (you can only have scripting, but no ABAP calls)

Comparison of SAPscript and PDF-Based Forms: Symbols

SAPscript	PDF-based form
General symbols (&symbol&)	(Floating) fields; make sure to set the binding correctly
Program symbols	Interface parameters that are copied into context
System symbols	Selection available in interface
Text symbols	see /:define
Standard symbols	N/A (if absolutely necessary, read table TTDTG in the print program and transfer the value as an interface parameter)
&page&	Floating field with scripting at event layout:ready:\$ = \$layout.page(ref(\$)) Script (JavaScript) can be inserted automatically: <i>Insert</i> → <i>Current Page Number</i>
&sapscript-formpages&	Floating field with scripting at event layout:ready:\$ = \$layout.pageCount() Script (JavaScript) can be inserted automatically: <i>Insert</i> → <i>Current Page Number</i>

&sapscript-jobpages&	N/A (can be queried in print program only, but not evaluated in form)
&nextpage&	Scripting; make sure there is a next page
Are we on the last page? (&nextpage& eq 0 in all secondary windows of last page)	Scripting at event layout:ready if (\$layout.pageCount() == \$layout.page(ref(\$))) then ... endif

Comparison of SAPscript and PDF-Based Forms: Function Modules of Print Program

SAPscript	PDF-based form
OPEN_FORM	FP_JOB_OPEN; parameters differ!
CLOSE_FORM	FP_JOB_CLOSE; parameters differ!
START_FORM	Generated function module; parameters differ!
END_FORM	N/A
WRITE_FORM	N/A (order of texts is determined in the form)
WRITE_FORM_LINE	Dynamic text in content; pass internal table of line type TLINE to form
CONTROL_FORM	N/A

Migrating Smart Forms

Migration of Smart Forms: Prerequisites



- Make sure all referenced objects (Dictionary types, Smart Styles, text modules, SAPscript texts etc.) are available before conversion.
- Split up big forms into small forms (representing only one business case each)

The following example covers only one scenario. If you have Smart Forms that contain several business scenarios, proceed as follows:

1. Create temporary copies of the original Smart Form (one Smart Form for each business form).
2. For each business form, eliminate conditions that do not belong to the current business form (including corresponding subtrees).
To do this, use the Smart Form Builder and FormInfo functionality (which was added for SAP NetWeaver '04) as a graphical wizard.
3. Eliminate conditions from the form interface as well (this simplifies and accelerates data retrieval).
4. Save and activate the new forms.

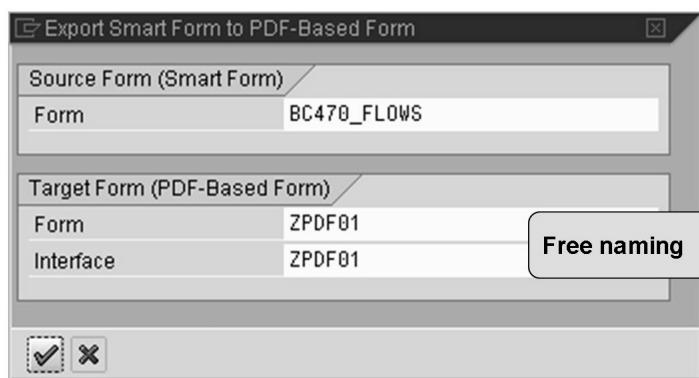


Figure 130: Migration of Smart Forms: Steps

To start the conversion from the initial screen of transaction SMARTFORMS choose *Utilities* → *Migration* → *Interactive Form* → *Export*.

Enter the name of a Smart Form, the name of a PDF-based target form, and the name of a target interface in the dialog box displayed.

If a target object (form or interface) exists already, you can decide whether to overwrite or rename it.

The names of the target interface and the target form are copied by default from the Smart Form. However, you can change them.

The names of the target form and the target interface can, but need not, be the same.

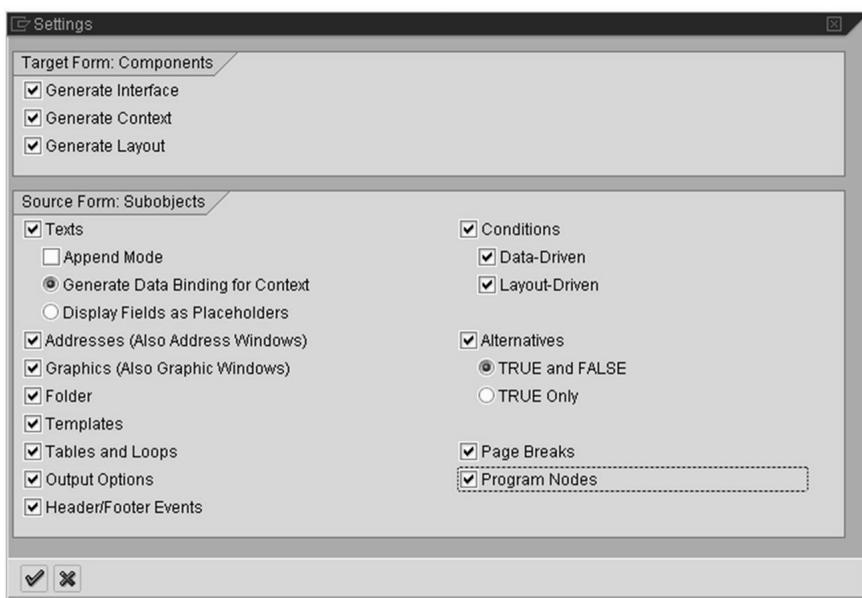


Figure 131: Migration of Smart Forms: Settings

The conversion tool is highly configurable and offers a range of conversion options. Online Help describes each option in detail.

Generate Interface: If this option is selected, the interface object is created or overwritten (after confirmation).

If you do not want to migrate the interface object (if it has already been migrated and was modified manually, for example), the field can be reset.

Texts: If this option is selected, all text nodes (text elements, text modules, and SAPscript texts) of the Smart Form are migrated. Depending on the layout setting, the nodes are copied to the layout.

Text elements are displayed as static texts in the layout. The text elements are converted from ITF format to XHTML.

Text modules are displayed as fields (XFA element `<field>`) in the layout that refers to the context elements.

Information about the text module is stored in the context. The text modules are converted at runtime from ITF format to XHTML.

SAPscript texts are displayed as fields (XFA element `<field>`) in the layout that refer to the context elements. Information about the SAPscript text is stored in the context. The SAPscript texts are converted from ITF format to XHTML at runtime.

Append Mode: If this option is selected, the append modes of the texts are reproduced in the layout. Since this feature is not currently supported by Adobe Document Services, it is reproduced by means of scripting.

Each text is packed into a <subform>, into which a special field, *append mode* __, is also packed.

The value of the field is analyzed in the generated scripting at runtime, and the text is appended to the previous text, depending on the mode. The following append modes exist:

- A – Append directly
- N – New line
- P – New paragraph

Since the append mode is a special feature of Smart Forms that is supported neither by the Adobe LiveCycle Designer nor by Adobe Document Services, we recommend that you perform the migration without the append mode.

Data Binding or Placeholder:

Create Data Binding to Context

If fields appear in a text, they receive the values from the data stream whose structure describes the form context. The mechanism that is used to link the fields from the layout with the nodes from the context is called data binding. If this option is selected, all fields of the static texts (text elements) are linked with context nodes.

Exceptions are *sfsy* fields (Smart Forms system fields such as *sfsy-page* and *sfsy-formpages*) and some *sy* fields (such as *sy-datum* and *sy-uzeit*). In the layout, they are analyzed directly by Adobe Document Services (without link to context). Since the syntax for fields in Smart Forms differs from the syntax in the XFA template (layout), the field names are renamed if necessary (&*mystruct-myfields*& becomes *mystruct.myfield*, for example).

Display Fields as Placeholders

This option is used if you do not want automatic data binding. The fields from text elements are not linked with context nodes. Instead, they are displayed as placeholders ({MYSTRUCT-MYFIELD}). The field name remains unconverted and is enclosed in curly brackets.

After the migration, you must replace these placeholders with fields that are linked to the context.

Addresses are displayed as fields in the layout that are linked to an address node in the context. At runtime, the addresses are read from the SAP system and copied to the form as plain text. That is why you should format the layout manually after migration.

Folder: If this option is selected, folder nodes of a Smart Form are displayed as folder nodes in the context and as XFA <subform> in the layout.

If this option is deactivated, the folder nodes from the Smart Form are ignored, but the subnodes of the folder are not affected (which means that a subnode is migrated if the corresponding migration flag allows this). Texts of a folder are migrated, for example, if the *Texts* option is set, even if the *Folder* option is deactivated.

Templates: If this option is selected, template nodes of a Smart Form are displayed as folder nodes in the context, and as subforms with the underlying cells in the layout.

If this option is deactivated, the template nodes (with the whole cell distribution) from the Smart Form are ignored, but the subnodes of a template are not affected (which means that a subnode is migrated if the corresponding migration option allows this). Texts of a template cell, for example, are migrated if the *Texts* option is set, even if the *Templates* option is deactivated.

Tables and Loops: If this option is selected, table or loop nodes of a Smart Form are displayed in the context as loop nodes, and in the layout as XFA <subform>. In the context, the work area of the table/loop from the Smart Form is copied.

Output Options: If this option is deactivated, the tables, rows, cells, and loop nodes from the Smart Form are ignored, but the subnodes are not affected (which means that a subnode is migrated if the corresponding migration option allows this). Texts of a loop, for example, are migrated if the *Texts* option is set, even if the *Tables and Loops* option is deactivated.

If this option is activated, the output options of every Smart Form node are migrated into the layout. The output options affect borders, shading, and page protection (apart from tables and templates).

Exception: Output options of *Window Nodes* are always migrated, regardless of whether they are set or not.

Header/Footer Events: If this option is selected, header and footer nodes of a Smart Form are displayed in the layout as XFA leader/trailer subforms.

If this option is deactivated, these nodes from the Smart Form are ignored, but the subnodes are not affected (which means that a subnode is migrated if the corresponding migration option allows this). Texts of a header/footer event are migrated, for example, if the *Texts* option is set, even if the *Header/Footer Events* option is deactivated.

Conditions: Smart Forms conditions can be defined on every node except window and event nodes.

Conditions can be split into two groups:

- *Layout-Driven:* conditions that are dependent on the current layout information (*Only on First Page*, for example)
- *Data-Driven:* conditions that are dependent on data values (CARRIER = 'LH', for example)

Data-driven conditions are analyzed on the ABAP side, whereas layout-driven conditions are analyzed by Adobe Document Services.

The migration process takes these aspects into consideration and creates a context condition for the data-driven condition, or a subform with the special field `subform_condition_` for the layout-driven condition.

Since the layout conditions are supported neither by Adobe Document Services nor by Adobe LiveCycle Designer, they have to be reproduced by using scripting in the layout.

In the default setting, the conditions are excluded from the migration. It is recommended that you check the form for conditions and evaluate them before migration by creating multiple forms.

Program Nodes: Since program nodes are permitted in migrated forms only, this option is deactivated by default. We recommend that you map the functions of the program nodes in other ways (by means of scripting in the layout, for example).

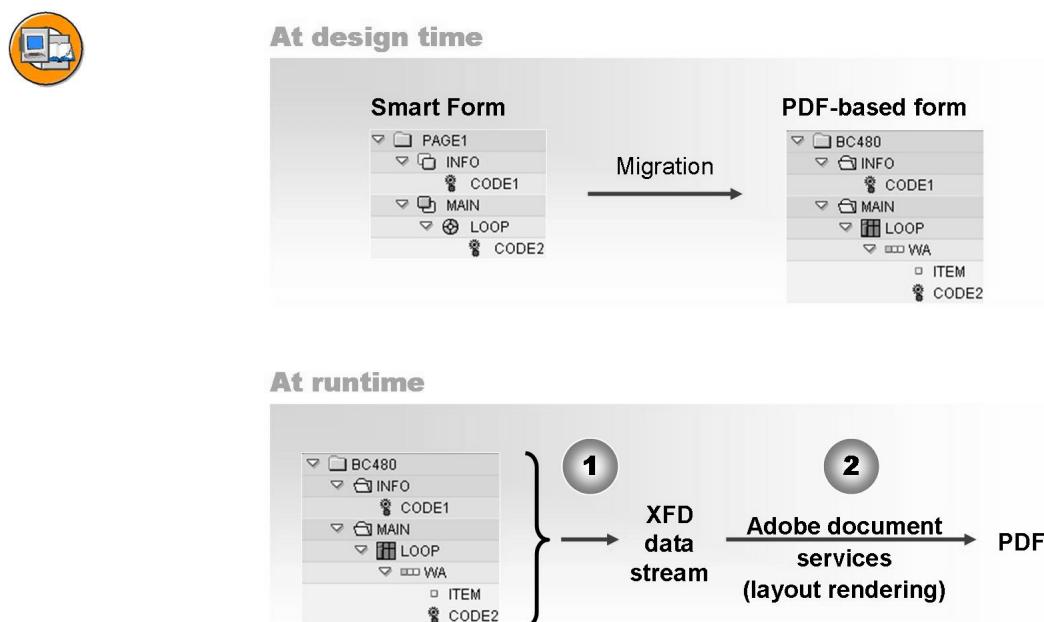


Figure 132: Program Lines in Migrated Forms

The migrated program line nodes will have the same coding as the original ones. It is important to know, however, that all migrated coding is executed when the data stream is created, that is, **before** the form is rendered by Adobe Document Services. This means you cannot rely on the coding being executed at certain layout-dependent times, such as at page break.

Migration of Smart Forms: Error Log

First, run the conversion. Then analyze the conversion protocol by choosing *Utilities* → *Migration* → *Interactive Form* → *Display Log*.

→ **Note:** If there are no errors, no error log is displayed.

After you have migrated the Smart Form to the PDF-based form, finish the form in Designer. To access Designer, choose *Utilities* → *Migration* → *Interactive Form* → *Edit*.

Do not underestimate the time it might take to achieve the result you have in mind.

Then activate the new form and the interface.

Avoid using the following elements in Smart Forms because you must adapt them manually in the conversion process:

Pain Points in Smart Forms



- Using SAP characters (<123>)
- Program line nodes in flow logic
- Formatting in text nodes that diverge from the XHTML subset defined by Adobe (formatting that cannot be converted is lost during migration)
- Combination of data-specific output conditions (table control on conditions tab) and conditions with times (for example, only after the main window is closed)
- Obsolete SAP R/3 4.6C tables
- Template cells whose layout is interrupted by the layout of other cells (for example, text1 in cell 1-1, text2 in cell 1-2, text3 in cell 1-1)
- Table and export parameters:
 - The new interface optimized for processing with Adobe does not support table or export parameters.
 - Tables should be created as import parameters (TYPE <TABLETYPE>).
- Export parameters should not be created for application data; export parameters that refer to document output are generally provided.
- Combinations of output-related conditions (for example, *SFSY-PAGE* > 10) and data-related output conditions with the link OR to text modules include graphics, addresses, loops.
- Program lines that depend on a layout condition (current page = 1) or a layout-related event (page break). For example, changing a global variable that depends on the current number of pages.
- Using the output formats XDF or HTML.



Lesson Summary

You should now be able to:

- Describe the basic concepts of form processing with SAPscript and Smart Forms
- Name the steps required to reuse or migrate existing form objects from SAPscript and Smart Forms



Unit Summary

You should now be able to:

- Describe the formatting concept used for text modules
- Name relevant parts of Smart Styles
- Create and maintain styles
- Explain the mechanisms involved in catching and throwing object-oriented exceptions
- Catch exceptions that might be raised within a program that calls PDF-based forms
- Explain the principles of how Customizing works for PDF-based forms
- Customize one example transaction in mySAP ERP so that PDF-based forms are used for a printing scenario
- Set up an http connection for Adobe document services
- Name basic steps needed to install new fonts
- Use tables in Adobe LiveCycle Designer versions 6.0 and 7.0
- Describe the basic concepts of form processing with SAPscript and Smart Forms
- Name the steps required to reuse or migrate existing form objects from SAPscript and Smart Forms



Test Your Knowledge

1. In a form context, you can override the style of a text module.
Determine whether this statement is true or false.
 True
 False

2. Text in a text module can have any combination of the typical formats, such as italics, bold, or underlined.
Determine whether this statement is true or false.
 True
 False

3. All tabs in a style are left-aligned.
Determine whether this statement is true or false.
 True
 False

4. Which of the following statements on the class-based exception concept are true?
Choose the correct answer(s).
 A TRY. . . ENDTry. is sufficient to prevent a dump.
 B A programmer can always choose whether to use sy-subrc or exception objects when dealing with exceptions raised by a method.
 C Class-based exceptions can be raised in classes only.
 D Multiple CATCH clauses are possible to respond to different exceptions.

5. If a function module raises an exception of type cx_fp_api, how could you catch this exception?
Choose the correct answer(s).
 A CASE sy-subrc. WHEN 0. . . WHEN cx_fp_api. . . WHEN OTHERS. ENDCASE.
 B DATA gr_error TYPE REF TO cx_root. CATCH cx_root INTO gr_error.
 C DATA gr_error TYPE REF TO cx_fp_api. CATCH cx_fp_api INTO gr_error.
 D CATCH SYSTEM-EXCEPTIONS cx_fp_api.

6. In any given mySAP ERP system, you cannot use Smart Forms **and** PDF-based forms together.
- Determine whether this statement is true or false.*
- True
 False
7. Adobe document services are located on the same server as the ABAP engine of the SAP NetWeaver Application Server.
- Determine whether this statement is true or false.*
- True
 False
8. PDF-based forms make use of the same fonts as SAPscript forms or Smart Forms.
- Determine whether this statement is true or false.*
- True
 False
9. Tables are represented by an outer subform, an inner subform, and the individual fields.
- Determine whether this statement is true or false.*
- True
 False
10. If rows in a table are to be separated by lines, the table's outer subform must have borders.
- Determine whether this statement is true or false.*
- True
 False
11. The captions of the table fields can be used as headers.
- Determine whether this statement is true or false.*
- True
 False

12. Once you have created a subform for the table header (called *HEADER*), you set it as the overflow header for the outer subform.

Determine whether this statement is true or false.

- True
- False

13. Smart Forms can be converted to PDF-based forms without loss.

Determine whether this statement is true or false.

- True
- False

14. Smart Forms text modules can be integrated directly into PDF-based forms.

Determine whether this statement is true or false.

- True
- False



Answers

1. In a form context, you can override the style of a text module.

Answer: True

If you leave the *Copy Style From Text Module* option unchecked, you can specify a style that differs from the one used in the text module.

2. Text in a text module can have any combination of the typical formats, such as italics, bold, or underlined.

Answer: False

Only those formats can be applied that are available in the style used for the text module.

3. All tabs in a style are left-aligned.

Answer: False

Those tabs that you define on the *Tabs* tab can be left-aligned, right-aligned, centered, or they can be decimal tabs.

4. Which of the following statements on the class-based exception concept are true?

Answer: D

`TRY. . . . ENDTRY.` is required to write a `CATCH` clause, but itself is not sufficient to prevent a dump.

The programmer of a method or a function module determines one of the two ways of raising exceptions. If you use this method or function module, you have no choice but to use the specific type of exception concept.

Class-based exceptions can be raised in all programming blocks (methods, subforms, function modules, ...).

5. If a function module raises an exception of type `cx_fp_api`, how could you catch this exception?

Answer: B, C

The object-oriented exception concept makes no use of `sy-subrc`. Hence, answers A and D are wrong.

With the coding of answer B, you could catch all possible class-based exceptions, as `cx_root` is the top-level error class from which all other error classes are derived. With the coding of answer B, you could catch only exceptions of type `cx_fp_api`.

6. In any given mySAP ERP system, you cannot use Smart Forms **and** PDF-based forms together.

Answer: False

All three form solutions (SAPscript, Smart Forms, PDF-based forms) can be used in one system. Check the Implementation Guide of your application to see which solution is available.

7. Adobe document services are located on the same server as the ABAP engine of the SAP NetWeaver Application Server.

Answer: False

Adobe document services must be deployed on the J2EE engine of SAP NetWeaver Application Server. This could be the same server that is used for the ABAP engine of the SAP Web AS, but it is typically not.

8. PDF-based forms make use of the same fonts as SAPscript forms or Smart Forms.

Answer: False

PDF-based forms make use of the fonts installed on the file system of Adobe document services.

9. Tables are represented by an outer subform, an inner subform, and the individual fields.

Answer: True

This structure is due to the XML representation of internal tables.

10. If rows in a table are to be separated by lines, the table's outer subform must have borders.

Answer: False

For row lines, the inner subform (called *DATA*) and/or the fields included in *DATA* must have borders.

11. The captions of the table fields can be used as headers.

Answer: False

Captions would be repeated for every item that is printed; hence you would typically remove captions completely.

12. Once you have created a subform for the table header (called *HEADER*), you set it as the overflow header for the outer subform.

Answer: False

Once you have created a subform for the table header, you must go to the *DATA* subform and specify that *HEADER* should be used as the overflow leader for *DATA*.

13. Smart Forms can be converted to PDF-based forms without loss.

Answer: False

Smart Forms can be migrated to PDF-based forms, but in particular aspects of the form logic need manual adjustments.

14. Smart Forms text modules can be integrated directly into PDF-based forms.

Answer: True

Though it is possible, you should check all text modules that they use only those field that have been defined in the interface/context of the PDF-based form.

Internal Use SAP Partner Only

Internal Use SAP Partner Only



Course Summary

You should now be able to:

- Use transaction SFP
- Create and model form contexts
- Create and model form interfaces
- Use Adobe LiveCycle Designer
- Create and design complex forms for printing with the tool “Interactive Forms”
- Integrate forms into ABAP programs

Index

Numerics/Symbols

*¹, 57, 254
*(asterisk), 52

A

ABAP, 206, 219
ABAP Dictionary, 19, 152, 159
Acceptance request, 269
Accessibility, 240
Accessors, 198
Activating a form, 4, 205
Active Component Framework, 37, 80
ADDRESS INTO PRINT-FORM, 47–48, 58
Addresses, 48, 58
Adobe document services, 6, 130, 132, 222, 272–273
Adobe Reader, 2, 80
Alignment, 152
Alternative, 44
Archive, 223
Authorization, 98, 205

B

Background color, 90
Barcodes, 260
Bitmaps, 46, 130
Blanks, 154
Boilerplate objects, 109, 129, 133
Borders, 90
Boxes, 278
BTE, 268
Business Address Services, 47
Business Communication Services, 15, 224
Business Transaction Event, 268

C

calculate, 197
Captions, 39, 89, 145, 243
CATCH, 263
Central Address Management, 47
CH, 254
Circles, 133
CLOSE_FORM, 220
Column width, 279
Commands, 198, 285
Conditional break, 175
Conditions, 43, 45
Content areas, 109
Control levels, 42, 177
Country, 158
Courier, 132
Currencies, 19, 152, 159
Customizing, 268
CX_FP_EXCEPTION, 265
CX_ROOT, 263

D

Data binding, 147
Data pattern, 155
Data pattern symbols, 156
Data schema, 85
Data View, 87, 144, 149
Date fields, 152
Decimal fields, 151–152
Decimals, 152
Default format, 52
Default values, 146
Designer, 45
Toolbars, 83
Workspace, 82
Dictionary, 14, 145
Display pattern, 154
Downloading form objects, 94

- Downloading Form Objects, 234
 Downloading PDF file, 235
 Dynamic elements, 143
 Dynamic properties, 149
 Dynamic texts, 56
- E**
 Edit pattern, 155
 ENDTRY, 263
 Events, 197
 Exceptions, 16, 220, 262
 EXIF, 46, 130
 Exists, 204
- F**
 Fax, 224
 Field symbols, 18
 Floating fields, 153
 Folders, 44, 290
 Font, 131
 Font size, 255
 Fonts, embedding, 274
 Fonts, installing new, 273
 Fonts, replacing, 274
 Form object, 4
 Form Properties, 96
 Form routines, 19
 Form template, 4
 form:ready, 197
 Formats
 Text modules, 52
 Formatting options, 53, 154
 FormCalc, 96, 195
 FP_FIELD_LIST, 219
 FP_FUNCTION_MOD-
 ULE_NAME, 219–220,
 224, 265
 FP_JOB_CLOSE, 219–220
 FP_JOB_OPEN, 96, 219–220,
 274
 FPFONTREPL, 274
 Function Builder, 95
 Function module, 4
- G**
 Geometrical objects, 133
- GIF, 46, 130
 Global fields, 18
 Graphic content, 151
 Graphic reference, 151
 Graphics, 45
 Grid, 86
- H**
 HasValue, 204
 Header, 280
 Headers, 174
 Hiding elements, 206
 Hiding fields, 43
 Hierarchy palette, 85, 87
 HTML, 46
 http, 6
 http connections, 222, 272
- I**
 IF THEN ELSE, 149, 198
 Images
 Size, 130
 Implementation Guide, 268
 Importing PDF Documents, 236
 Include texts, 49
 Indents, 257
 Info palette, 85
 Initialization, 18
 initialize, 197
 Interactive forms, 6, 96, 223
 Interface type, 220
 Interfaces, 4, 13
 Internal tables, 41
 ITF, 56
- J**
 J2EE, 6
 Java, 6
 JavaScript, 96, 195, 203
 JPG, 46, 130
- L**
 Language, 55, 158, 223
 Layout Palette, 89
 layout:ready, 197
 Library, 85
 Library objects, 93

- Line breaks, 51, 131
 Lines, 133
 LN, 254
 Locales, 96, 155, 158
 Loops, 41
- M**
- Mail, 224
 Margins, 89
 master pages, 109
 Migration, 16, 224, 284
 MIME Repository, 130, 284
 MS Word, 238
 MSAA, 244
 Multiple lines, 146
 mySAP ERP, 3, 269
- N**
- Nested objects, 88
 Nested tables, 42, 179
 Numbering, automatic, 258
 Numeric fields, 151
 Numeric pattern symbols, 159
- O**
- Offset, 154
 OPEN_FORM, 220
 Output Control, 6
- P**
- Package, 37, 50
 Page breaks, 109, 111, 114
 Page numbers, 154
 Pages
 Orientation, 110
 Size, 110
 pages (design view), 109, 111
 Palette windows, 82
 Palettes, 82, 84
 Paragraph, 131
 Payment request, 269
 PCL, 222
 PDL, 222, 224
 Performance, 147, 206
 Picture clause, 155
 PNG, 46, 130
 Post Processing Framework, 5
 PostScript, 222, 273
- Print preview, 95–96, 226
 Printer Definition Language, 222
 Program, 219
 Public sector, 269
- Q**
- Quantities, 19
- R**
- Radix alignment, 152
 RDI, 222
 Reader rights, 223
 \$record, 148, 198
 Rectangles, 133
 Report palette, 85
 Request printouts, 269
 RFC destination, 222
 Rotating objects, 89
 RSTXLDMC, 284
 RTF, 238
 RVORDER01, 285
- S**
- SAP GUI, 2, 6, 80
 SAP NetWeaver Application Server, 5, 273
 SAPscript, 109, 154, 220, 268–269, 282
 SAPscript commands, 51
 SAPscript texts, 49, 55
 Script Editor, 82, 195
 SE78, 284
 Server scripting, 195
 SET COUNTRY, 158, 223
 SFP, 36
 SFPOUTPUTPARAMS, 240
 SFPSY, 39, 43, 53
 SM59, 272
 Smart Forms, 14, 16, 49, 109, 154, 219, 224, 268, 283
 Smart Styles, 55, 252
 SOAP, 6
 Sorting tables, 178
 SP01, 2, 219, 226
 Spacing, 132
 Spool job, 15, 222

- SSF_FUNCTION_MODULE_NAME, 224
SSL, 272
 Standard paragraph, 254
 Static text, 131
 STXFPCUST, 225
 Style Builder, 254
 Styles, 50, 52, 54–55, 252
 Subforms, 110, 113
 Substrings, 204
 SY-..., 19
 sy-subrc, 262
 Syntax check, 204
 SYST, 53
- T**
 T006, 19
 Table cells, 278
 Table headers, 174
 Table row, 278
 Tables, 171–172, 276
 Tables, nested, 179
 Tabs, 257
 Tagged PDF, 240
 TCURX, 19
 Text
 Alignment, 132
 Format, 131
 Static, 131
 Text modules, 49, 52
 Text variables, 97
 TIF, 46
 Time fields, 152
 Time pattern symbols, 156
 Times, 132
 TLINE, 56
 Trace, 205
 Transport, 234
 Transport Organizer, 49
 True Type, 273
- TRY, 263
 TTF, 273
U
 Underlined, 255
 Units, 86
 upper- and lowercase, 155
 URL, 130
- V**
 Validation pattern, 155
 Variables, 97, 201
- W**
 Web Dynpro, 6
 Web Dynpros, 6
 WHERE clause, 42
 Where conditions, 179
 WRITE_FORM, 282
 WYSIWYG, 131
- X**
 XDF, 15
 XFA, 94
 XFD, 41
 XFO, 93
 XFP, 222
 XHTML, 147
 XML, 4
 XML Data Package, 4
 XML data stream, 41, 44
 XML Schema, 14, 20
 XSD, 20, 41
 XSF, 15
- Z**
 Zebra printer, 222
 Zebra stripes, 174
 Zero Client Installation, 37
 Zeros (omitting), 154

Feedback

SAP AG has made every effort in the preparation of this course to ensure the accuracy and completeness of the materials. If you have any corrections or suggestions for improvement, please record them in the appropriate place in the course evaluation.