
Using SAP Interactive Forms in the ABAP Workbench to create and design PDF-based print forms that address a wide range of print needs

by Markus Meisl



Markus Meisl
SAP NetWeaver
Product Manager,
Forms Technology,
SAP AG

Since joining SAP's implementation methodology group in 1998, Markus Meisl has covered all roles in SAP Product Management, ranging from technical documentation and translation, through product definition and early training, to rollout and partner relations. After his move to SAP NetWeaver in 2002, his responsibilities have focused on the rollout of SAP forms technology, particularly SAP Interactive Forms by Adobe. You may reach Markus at markus.meisl@sap.com.

Over the last decade, PDF (Portable Document Format) has established itself as the de facto tool for creating and exchanging electronic documents, so much so that it is now a staple of modern business.¹ Recognizing the importance of PDF, SAP enhanced its two document output technologies — Smart Forms and SAPscript — with conversion routines giving them the ability to output to PDF. Internally, however, the design tools and underlying technologies for both Smart Forms and SAPscript were proprietary, and conversion was an imperfect process. Therefore, in 2002, SAP initiated a strategic development partnership with Adobe Systems in the area of electronic forms — the result of which was **SAP Interactive Forms by Adobe**.

SAP Interactive Forms — initially introduced in SAP NetWeaver '04 and enhanced in SAP NetWeaver 2004s — integrates Adobe PDF technology right into mySAP ERP. Unlike SAPscript and Smart Forms, both the design tools and underlying runtime components of SAP Interactive Forms are native Adobe components generating PDFs with Adobe libraries. The benefits are:

- You no longer need to convert documents from an internal format to PDF, yielding an efficient printing process and fewer formatting discrepancies.
- Customers gain Adobe PDF's sophisticated, best-of-breed printing and formatting capabilities, such as WYSIWIG form layout design through drag-and-drop operations.
- Enhancements that Adobe makes to its own platform reach SAP customers far more quickly.

¹ Relatively recent enhancements to Adobe Acrobat as well as Adobe Reader and the PDF format (e.g., the ability to embed JavaScript field validation logic in PDF forms) have made it a viable option for data entry in simple scenarios (e.g., time sheets, expense reports).

SAP Interactive Forms is fully integrated into native SAP tools to create and design forms for printing, such as purchase orders, invoices, pay slips, or labels. Since the first shipment of SAP Interactive Forms with SAP NetWeaver '04, SAP has progressively moved toward providing all application form templates based on the Adobe PDF technology.

This article explains what you need to know to get started with SAP Interactive Forms. It discusses the underlying architecture of SAP Interactive Forms,

Note!

SAP is committed to protecting the investment customers have made in existing solutions. Because both Smart Forms and SAPscript are integral parts of SAP NetWeaver, SAP will continue to support all your existing Smart Forms and SAPscript forms in addition to SAP Interactive Forms. In 2004, SAP began a comprehensive internal project targeted at converting all existing application form templates in mySAP ERP to the new Adobe-based solution to consolidate all forms in one technology and comply with legal requirements in the United States. In the future, application form templates shipped by SAP will be created using SAP Interactive Forms.

In mySAP ERP 2005, the existing Smart Forms or SAPscript form template constitutes the default setting in all mySAP ERP applications, but you can activate any of the approximately 650 PDF-based print forms delivered through application customizing. In general, you can continue to use the delivered Smart Forms and SAPscript templates if they meet your requirements. If you want to use new SAP-delivered forms or create your own custom forms, it makes sense to familiarize yourself with the PDF-based print forms solution.

which incorporates many of the known principles of Smart Forms technology. In the main part of the article, we'll explore how to create a simple form using the new Form Builder. The sample form that we'll create will include the essential components required in a print form process, which are:

- **Form interface:** The conduit used by the application in mySAP ERP to pass the current system data, such as the concrete items of an order, from your application to the form at run-time (i.e., when the business process requires the generation of such a document)
- **Form template:** The object that provides the structure of the content and layout (that is, the look and feel of your output documents)
- **Print program:** A program used in your application to trigger the print form generation

Note that the form template consists of two parts: the form context and the form layout. As you learn how to design the form layout, you will also learn about data binding, creating tables in a form, and scripting. This article concludes with some tips and tricks to help make your SAP Interactive Forms experience easier and more productive.

Let's begin by looking at the underlying structure of SAP Interactive Forms.

Note!

This article is for developers who are familiar with forms development, especially in the Smart Forms environment.² This article does not cover Java-based interactive forms, but rather focuses on the ABAP development environment.

² For more on Smart Forms, see the *SAP Professional Journal* article, "Create and Maintain Forms Without Programming? It's a Snap with SAP Smart Forms!" (May/June 2002).

Migration and translation

Two of the most frequently asked questions by SAP customers include how you migrate from Smart Forms to SAP Interactive Forms and how you translate PDF-based print forms.

- **Migration:** SAP offers a migration wizard to support the tools-based migration of Smart Forms to SAP Interactive Forms format. You can access the wizard on the entry screen of the Smart Forms transaction through the menu Utilities → Migration → PDF-based form. After you launch the wizard, you will see that SAP recommends certain settings for those Smart Forms elements that migrate well and usually require only minimal subsequent work in the PDF-based form. Due to fundamental architectural changes between Smart Forms and SAP Interactive Forms, certain elements do not migrate well, mainly the ones containing ABAP program logic.

SAP internal experience over the last three years has shown that, in most cases, it is less time consuming and requires less development effort to carefully analyze the Smart Form, determine which elements you need to keep, and then re-create the form in the Adobe-based solution.

One clear recommendation is to use one form template for each business form you create. This means that conditions, which were easily integrated into a Smart Form to drive the output of different business forms contained in the same Smart Form, need to be moved to your application (print) program to determine the use of the different templates.

- **Translation:** Just like all other ABAP-based form templates, you can translate PDF-based print forms created in the ABAP Workbench in the standard translation transaction SE63. You can find the split-level editor for doing this in the menu under Translation → Other ABAP objects → Forms and styles → PDF/A.

SAP Interactive Forms architecture

You create form objects using the Form Builder. You can access this tool in the ABAP Workbench Repository Browser (transaction SE80) or through the dedicated transaction SFP.

SFP was designed to resemble transaction SMARTFORMS as much as possible to facilitate moving to the new form solution (Form Builder is also the name of the Smart Forms development environment). SAP has endeavored to safeguard as many of the proven principles of Smart Forms as possible, such as the strict separation of data retrieval and form layout, despite integrating technology (that is, the Adobe PDF technology) from outside the ABAP stack. (For more information on migrating Smart Forms to the SAP Interactive Forms solution and

translating PDF-based print forms, see the sidebar on this page.)

The basic process of designing a form has not changed fundamentally. SAP Interactive Forms uses the runtime architecture known from Smart Forms. **Figure 1** on the next page depicts the runtime architecture of SAP Interactive Forms in back-end printing scenarios. As with Smart Forms, an application (that is, print) program triggers the data retrieval of the current system data from the application database. This data is passed to the generated ABAP function module that represents the form template. (The function module is generated upon activating the form object in the ABAP Workbench.) Both the form template and the application data are then passed to the runtime component (explained next), which generates the required output and returns the actual document, in this case, a PDF form.

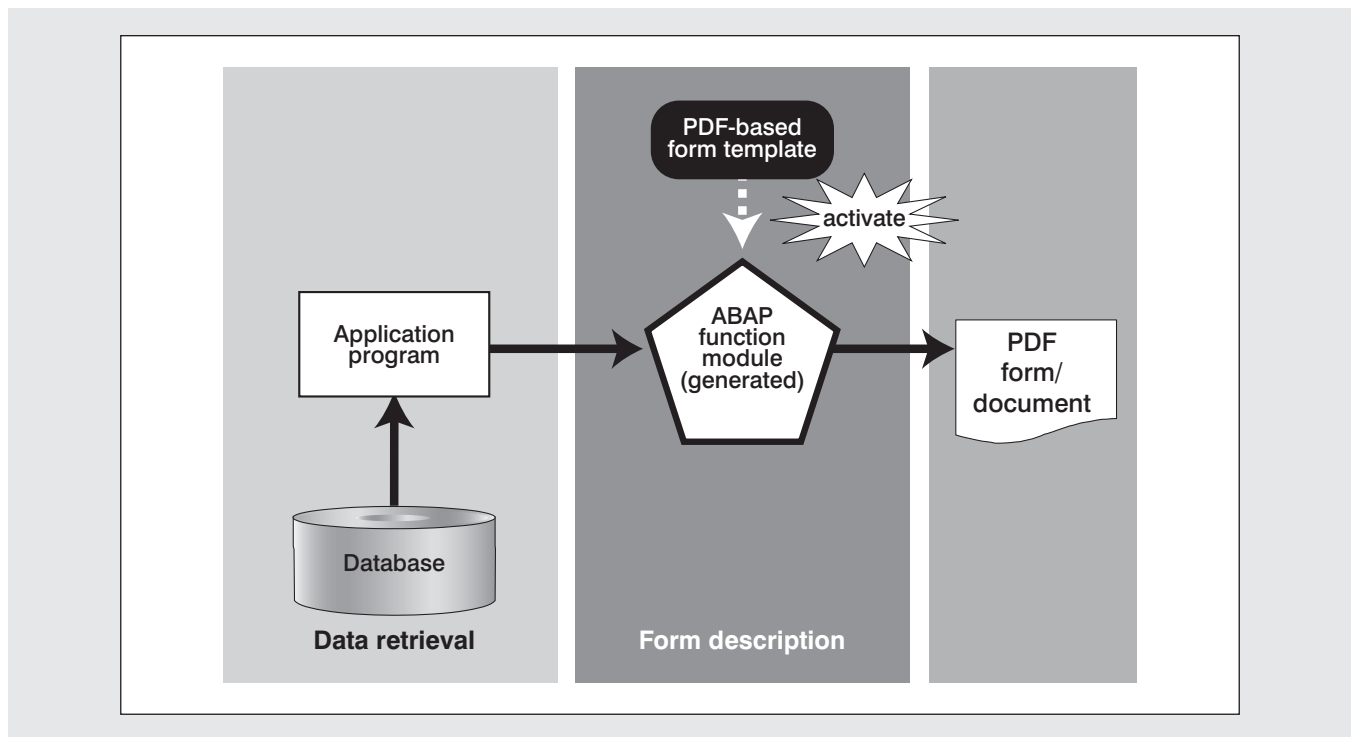


Figure 1 Runtime architecture of SAP Interactive Forms in back-end printing scenarios

The difference in SAP Interactive Forms is the use of Adobe LiveCycle Designer (or — for the purposes of this article — simply Designer) as the tool for creating the layout of the form and the use of Adobe document services (ADS) as the runtime component that generates the output formats, as illustrated in **Figure 2**.

The SAP Interactive Forms solution consists of technology provided by both SAP and Adobe.

SAP contributes:

- **SAP NetWeaver Application Server:** SAP NetWeaver Application Server (AS) includes the development environments for ABAP and Java (ABAP Workbench and SAP NetWeaver Developer Studio, respectively), as well as the necessary runtime environments.
- **SAP GUI:** The traditional UI technology used to access an SAP ABAP back end.

Adobe contributes:

- **Adobe LiveCycle Designer:** Designer is Adobe's

design-time forms layout tool that is integrated into the ABAP Workbench and SAP NetWeaver Developer Studio. It enables the creation of forms that combine high-fidelity presentation with XML data handling.³ The easy-to-use graphical interface of Designer enables users to quickly design forms, maintain form templates, define a form's business logic, make changes, and preview forms before they are deployed as PDF files.

- **Adobe document services (ADS):** ADS is a set of runtime services that is deployed on SAP NetWeaver AS Java, and provides a range of form and document creation and manipulation functions used by applications in business operations.

These services fulfill two basic tasks:

- Merge XML form templates (created using Designer) with current SAP application data in XML format, and convert the result to PDF.

³ This capability comes from the Adobe XML Forms Architecture (XFA). For more information on XFA, go to http://partners.adobe.com/public/developer/en/xml/xf_spec_2_4.pdf.

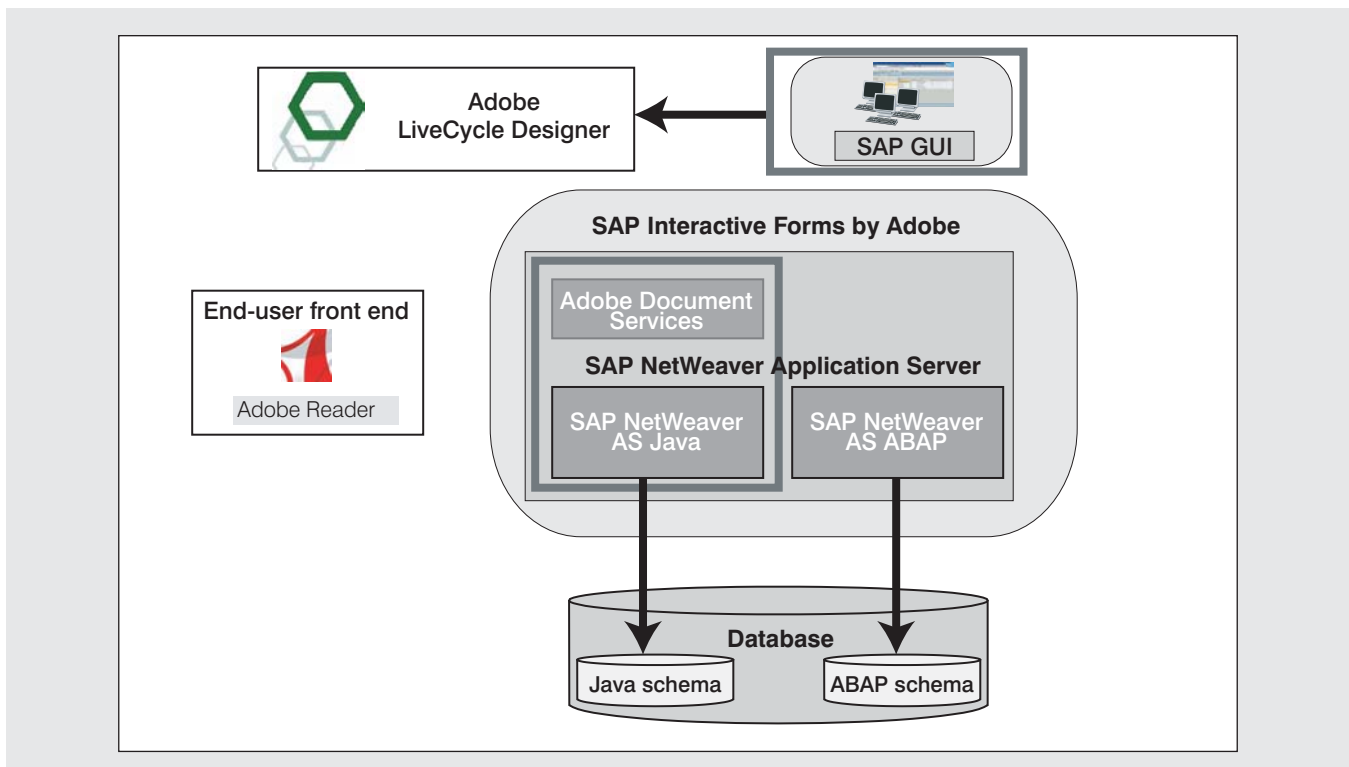


Figure 2 Underlying SAP Interactive Forms technology provided by SAP and Adobe

- In an interactive scenario, extract data entered by an end user from an interactive PDF form, and transfer the data back into SAP applications using XML. (This task is not covered in this article.⁴)
- **Adobe Reader:** Adobe Reader is the free viewer end users need for displaying PDF forms and editing interactive ones. Although Designer and ADS are shipped as an integral part of SAP NetWeaver, Adobe Reader is available for download from the Adobe Web site at www.adobe.com.

ADS acts as a server-side process for generating PDF forms. ADS was developed using the Enterprise JavaBeans (EJB) technology and runs only on the Java stack of SAP NetWeaver. This is important to remember especially when all you want to do is print

your documents from an ABAP back-end system! It means that whenever you want to use the SAP Interactive Forms technology, you *always* need the Java stack with ADS.

To make communication with ADS easy for both ABAP and Java applications, the services are exposed as a Web service, an open standard communication protocol that enables applications built from both ABAP and Java to talk to ADS easily using standard SOAP messages over HTTP.⁵ But be aware that the ADS Web service is currently used only internally in SAP NetWeaver AS and must not be used directly.⁶

To further ease the lives of SAP application developers, SAP has encapsulated all of the features and functions ADS provides in an SAP development

⁴ For more information on interactive scenarios, see the *SAP Professional Journal* article, "Streamline business processes and increase user productivity with SAP NetWeaver: Build forms-based Web Dynpro applications using Interactive Forms based on Adobe software" (January/February 2006), which I co-authored with Marc Chan.

⁵ See the article, "Extend the Internal and External Reach of Your Applications with ABAP-Based Web Services" (*SAP Professional Journal*, July/August 2005).

⁶ For technical prerequisites for enabling SAP Interactive Forms, see the sidebar on the next page.

Prerequisites

To create PDF-based print forms, you'll need the following components on the server and the front-end system:

On the server:

- SAP NetWeaver 2004s with usage type Application Server (preferably a double-stack installation — ABAP stack and Java stack installed on the same system)*
- Adobe document services (ADS) deployed and configured on the Java stack
- mySAP ERP 2005 (SAP ECC 6.0) on SAP NetWeaver 2004s

On the front-end system:

- SAP GUI 6.40
- Adobe LiveCycle Designer 7.1 (downloaded from the SAP Service Marketplace at <http://service.sap.com/swdc>)
- Adobe Reader 7.0.8 (available for free from www.adobe.com)

* You can also install a standalone Java stack with ADS and configure your ABAP back-end connection to point to this standalone server.

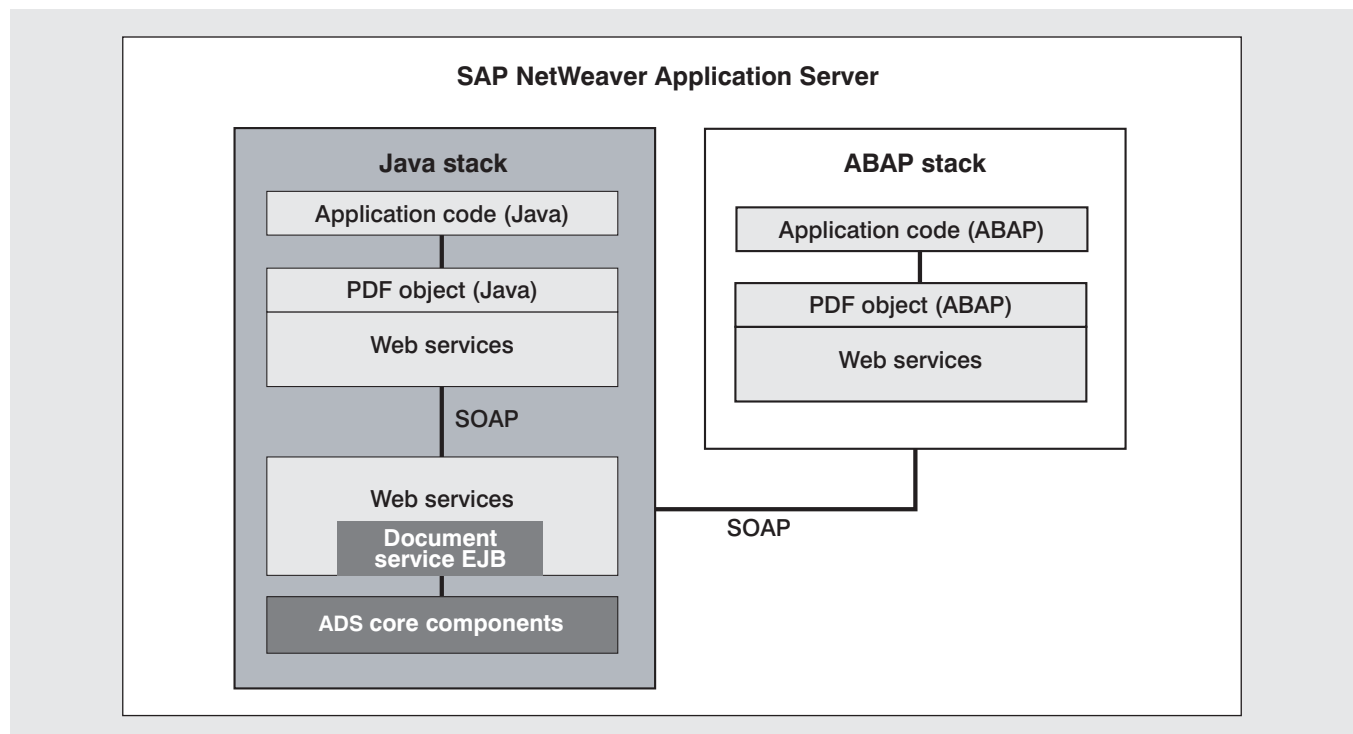


Figure 3 PDF Object is available on both the Java and ABAP stacks

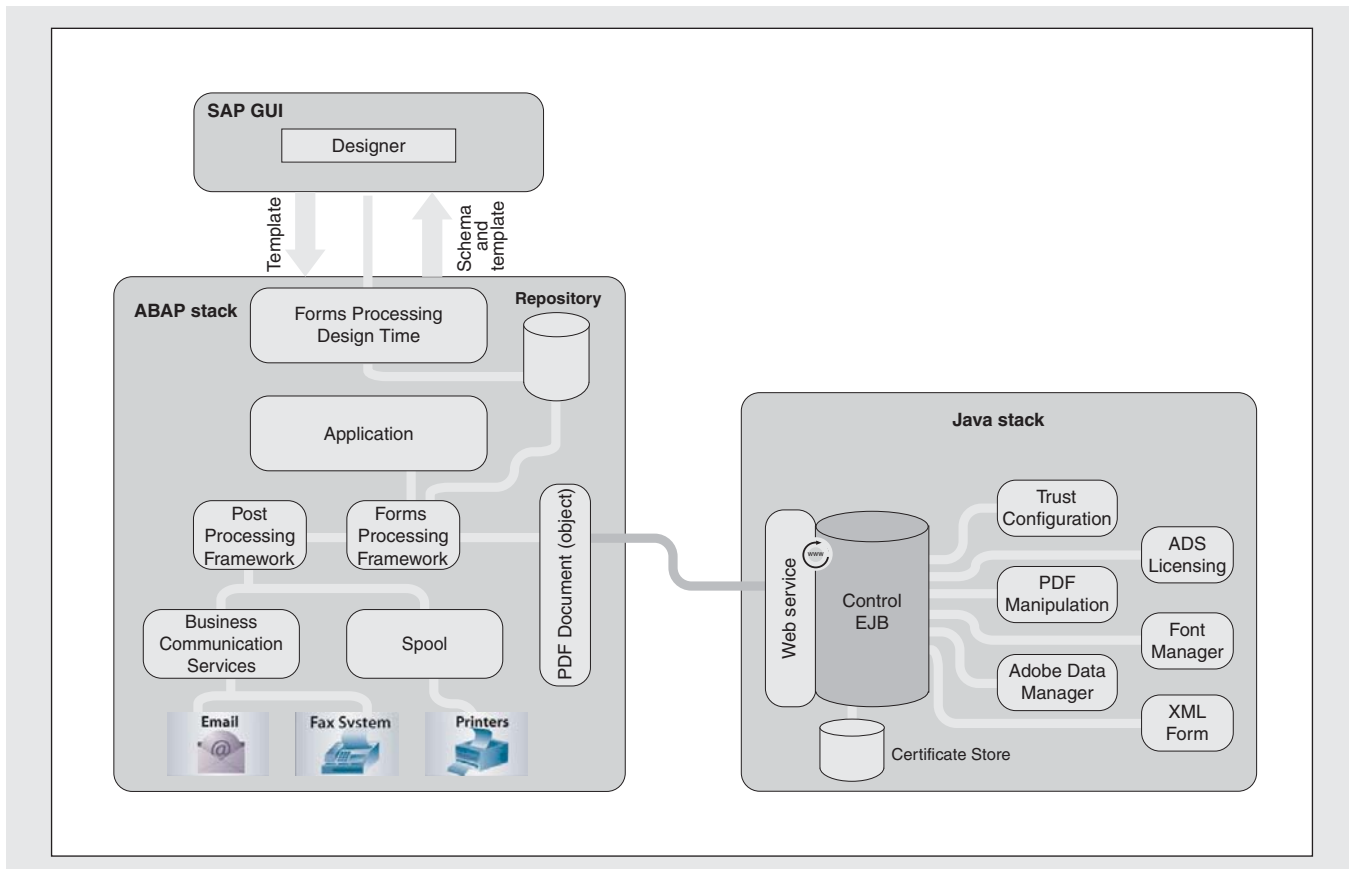


Figure 4 Complete overview of the SAP Interactive Forms architecture

object called the PDF Object. The PDF Object is available for ABAP and Java (see **Figure 3**). It is essentially a wrapper class (in ABAP, class CL_FP_PDF_OBJECT) that allows access to the ADS capabilities from both the ABAP and the Java development environments.

In general, the ABAP environment takes care of the communication with ADS via the PDF Object. This allows for data transfer from the back end into the form (and vice versa in an interactive scenario). Nevertheless, a developer can also call the PDF Object directly from the code to execute certain PDF-related functions on ADS.

By default, the PDF output is only used as the *preview* for a print form. The actual printing is done from one of the other output formats created by ADS, that is PostScript, PCL, or ZPL (for Zebra label printers). As with all legacy forms offerings, the

output files are placed in the SAP spool from where they are further processed in the standard SAP way.

Figure 4 provides a complete overview of the SAP Interactive Forms architecture.

The sample form template that we are going to build will include a form interface, form template, and print program — the essential components required for creating a PDF-based print form. The data that is used in the example form is based on the SAP IDES flight data model. The goal of the form is to produce a PDF that lists flights booked with a particular customer sorted by airline companies. To create this form, we will adhere to the basic process of designing a form, which includes the following steps:

1. Create a form interface.
2. Create a form template (form context and form layout).

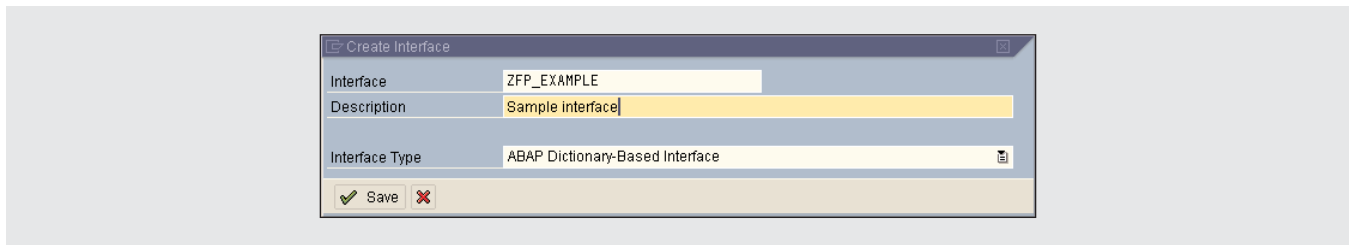


Figure 5 Create Interface dialog box in the Form Builder

3. Create a print program (transaction SE38).
4. Execute the print program and generate the PDF document.

You can create a form object (interface or form template) in the ABAP Workbench Repository Browser (transaction SE80) or the dedicated transaction SFP (a Basis transaction, with FP standing for Forms Processing), which provides direct access to the Form Builder, the environment used to build a print form.

In the ABAP Workbench Repository Browser, the two form objects have been added to the standard and context menus, so you can create them in the same way as any other object in the ABAP Workbench. For simplicity's sake, we will build the sample form in transaction SFP.

Step 1: Create a form interface

You define a form interface to send data from your application to the form. The form interface is a part of the interface form object. You define the form interface in the Form Builder. You insert fields as placeholders that define the appearance and location of certain data, such as data from the database tables of your application, data provided by the SAP system, or data you have calculated or implemented yourself. These fields are replaced by the current data at run-time. The form interface is structured in the same way as the interface of a function module in the system.

To create an interface form object in the Form Builder, you select the Interface radio button, and click on the Create button. The Create Interface

dialog box appears (see **Figure 5**). You enter a name (for example, ZFP_EXAMPLE). You can enter a description of the object you are creating (for example, Sample interface). Next, you need to select the appropriate interface type. The one you choose will depend on what type of form you are creating.

There are three types of interface from which to choose:

- **ABAP Dictionary-Based Interface:** You choose this type when you create a PDF-based print form. You define this interface in the next step in the Form Builder.
- **XML Schema-Based Interface:** You use this type when you create interactive PDF forms with Web Dynpro for ABAP. Instead of using the Form Builder to define the interface, the Web Dynpro framework generates the schema in the background based on the Web Dynpro context.
- **Smart Forms-Compatible Interface:** You use this type when you migrate a Smart Form to SAP Interactive Forms. A Smart Forms-Compatible Interface contains certain parameters that are considered obsolete (for example, the ones offered in the Tables option). You should try to avoid using this interface type for the new forms you want to create.

After you specify the name of the interface, enter a description, and select the type, you click on the Save button. The Change Interface screen for the sample form opens, shown in **Figure 6**. The Properties tab gives you the opportunity to change the type of interface you will use for the form.

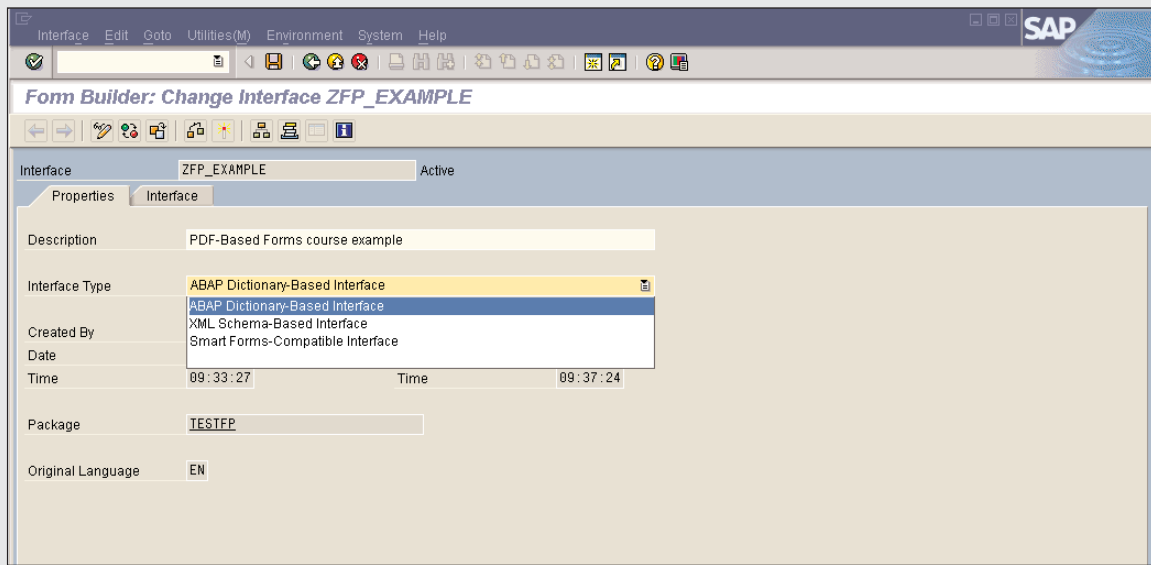


Figure 6 Properties tab in the Form Builder

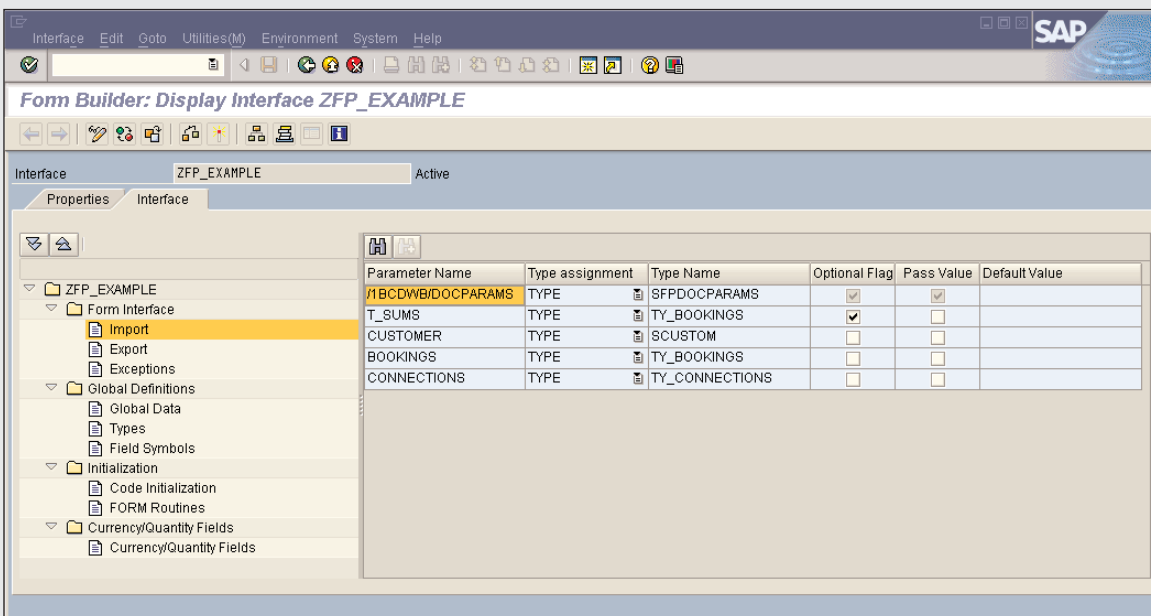


Figure 7 Interface tab in the Form Builder

Next you need to switch to the Interface tab (see **Figure 7**). The tab provides the general structure of a

form interface. Here is where you specify the parameters and values needed for your form.

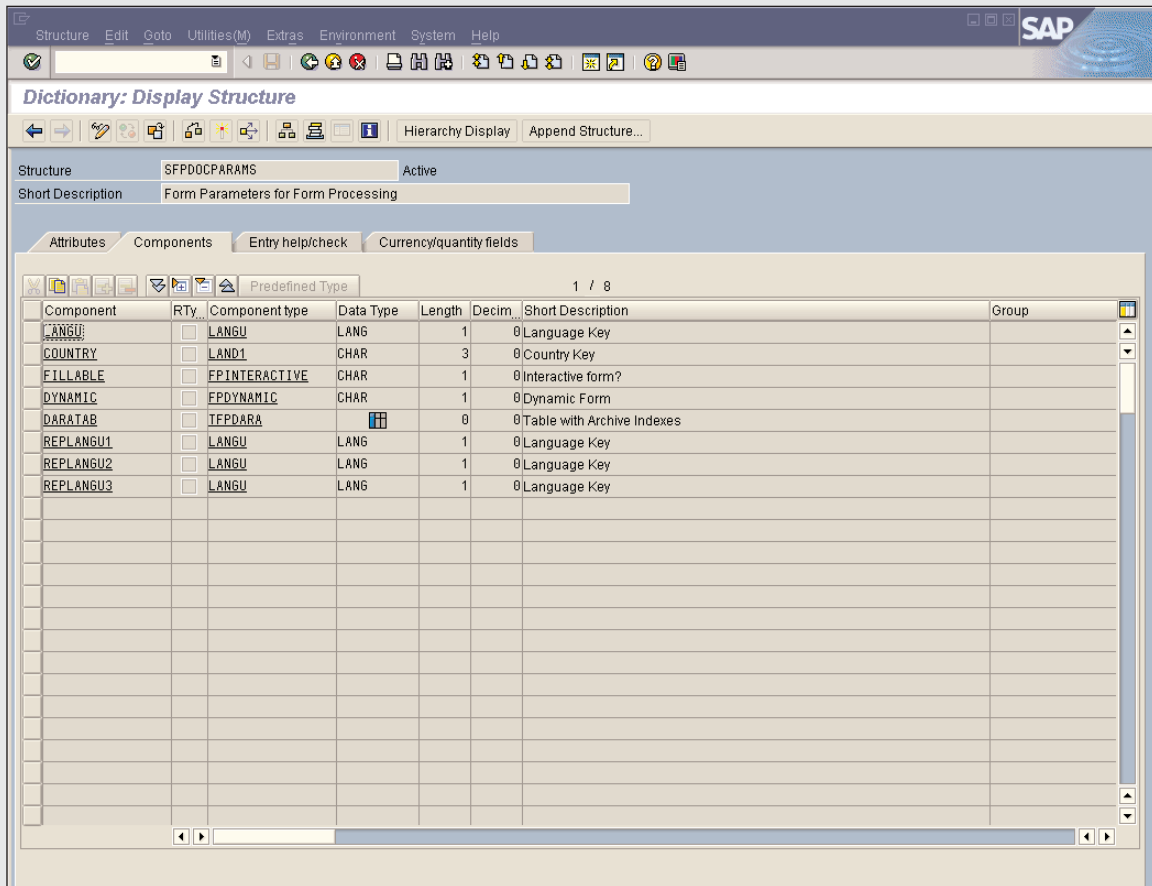


Figure 8 Specifying import parameters

Several standard parameters are part of any form interface. For example, you use import and export parameters to manage the input and output of the data via the form. Let's look at some of these parameters:

- Under Import, you find the standard parameter /1BCDWB/DOCPARAMS, which provides the foundation for the interface. You cannot change this parameter, but you can specify the values you want for this parameter in your application program (which in this case is a print program) — to set, for example, the value for the language settings of your form. You can display the structure of this import parameter in the ABAP Dictionary by double-clicking on SFPDOCPARAMS, which is the type name (see **Figure 8**).
- Under Export, you see the standard parameter /1BCDWB/FORMOUTPUT of the generated function module. One way to use this parameter in your application program is to make the generated form available as a PDF for further processing by sending a copy of an invoice by email or archiving the form in PDF format. To see the export parameters, as shown in **Figure 9**, double-click on FPFORMOUTPUT.
- Under Exceptions, you find the standard exceptions used by the generated function module: USAGE_ERROR, SYSTEM_ERROR, and

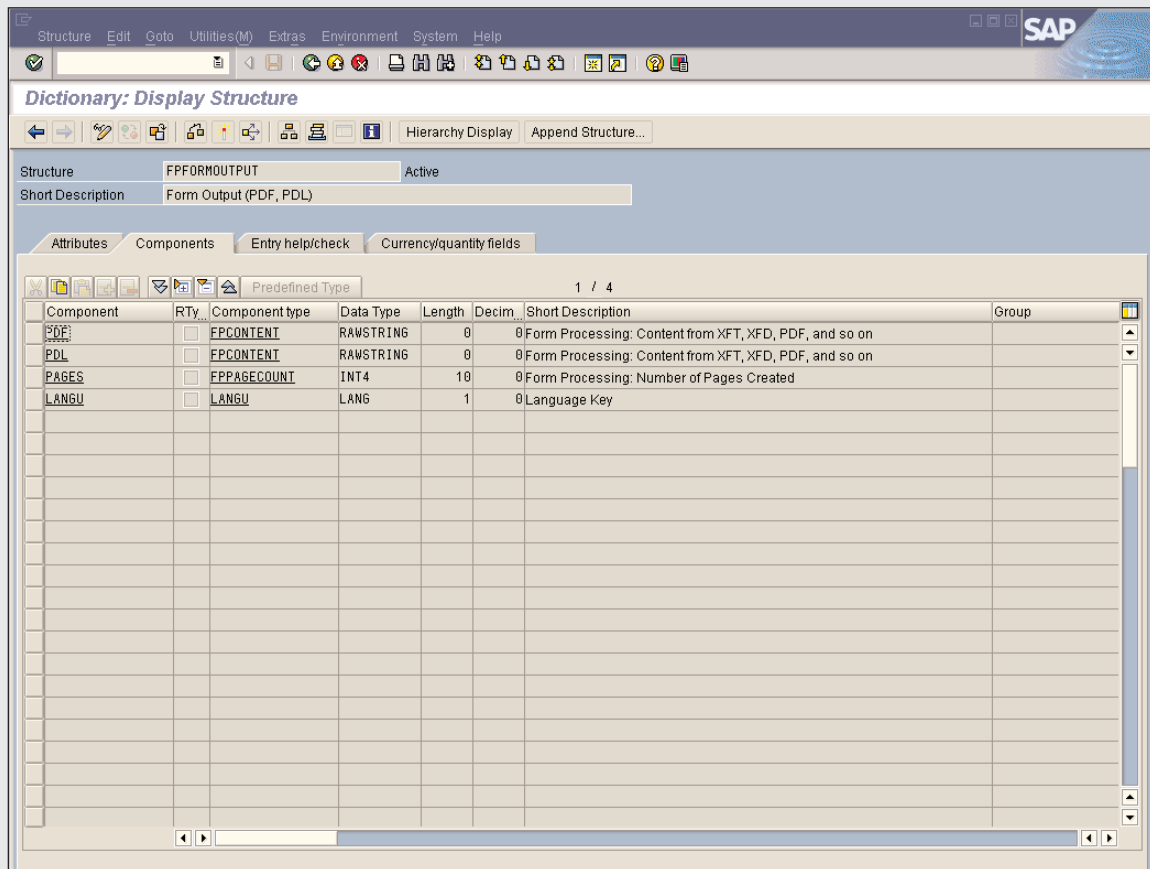


Figure 9 Specifying export parameters

Note!

The PDF form is generally used as a preview format, but you can send a PDF form via email or choose to archive it. The actual printing from the spool happens based on the other output formats generated by ADS — that is, PostScript or PCL. (This means that in most back-end printing scenarios you will not even generate a PDF file, because normally no one looks at the preview in such scenarios.)

If you want to archive a copy of the generated PDF form for legal or other purposes, you need to enable your application accordingly. As SAP and customer applications use a wide variety of document or content management systems, SAP NetWeaver cannot provide a standard persistence or archiving mechanism for the generated form. If you do not provide a way to store the PDF, it is discarded when you navigate away from the form.

Field Name	Reference Field	Data Type
BOOKINGS-FORCURAM	BOOKINGS-FORCURKEY	CURR
T_SUMS-FORCURAM	T_SUMS-FORCURKEY	CURR

Figure 10 Specifying export currency/quantity parameters

INTERNAL_ERROR. As with any exception, you can add or append specific criteria to raise an exception.

As a developer, you know that you can create your own import and export parameters (using the Insert Row or Append Row icons and filling the corresponding fields). In the Form Builder, however, you can also work with:

- **Global Definitions:** To define your own fields to be used anywhere in the form. You can use global data, types, and field symbols.
- **Initialization:** To execute ABAP program code before a form is processed. You can initialize the global data, for example, to convert selected application data. You can also select additional data without having to make changes in the application program.
- **Currency/Quantity Fields:** To specify currency or quantity references in your interface. In the example, we want to use the two references shown in **Figure 10**.

Figure 7 (on page 14) shows the import parameters needed for the sample form. (In addition to the standard import parameter, /1BCDWB/DOCPARAMS, we

need to enter four other database tables, which the print program needs to access the data from the back end.) The default export parameter is acceptable in this case. We don't need any additional parameters in the other categories of the form interface.

Now that we've created the form interface and specified the necessary parameters, we can go on to the next step.

Step 2: Create a form template

To create a form (template) object in the Form Builder, you select the Form radio button in transaction SFP, and click on the Create button. The Create Form dialog box appears (see **Figure 11**). You enter a name (for example, ZFP_EXAMPLE). You can enter a description of the object you are creating (for example, Sample form template). Next, you need to assign the form to an existing interface. Place the cursor in the Interface field, press F4, and select the relevant interface (in the example, the ZFP_EXAMPLE interface we created in the previous step). To complete the process of creating the form template, click on the Save button.

As mentioned earlier, the form template consists of two parts: the form context and the form layout. First, we will look at the form context.

Creating the form context

The form context is the link that binds the form interface to the layout. In essence, the form context is the data model for your form. After you create the form template and choose Change, you are automatically sent to the Context tab of the Form Builder.

Note!

The Form Builder editor for creating the parameters does not support all features of the ABAP Workbench; for example, it does not support forward navigation. This environment is designed for simple tasks and therefore does not handle a highly complex one.

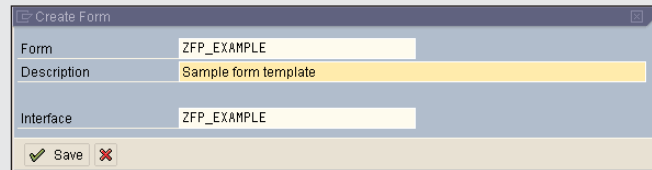


Figure 11 Create Form dialog box in the Form Builder

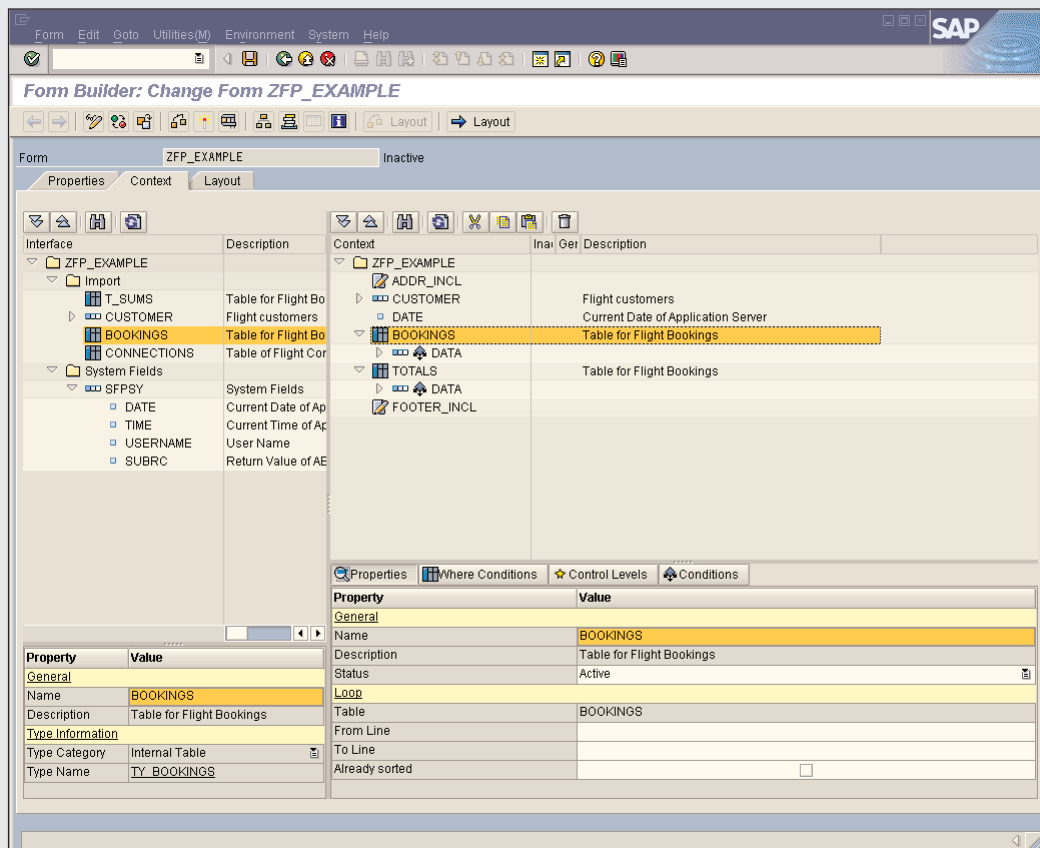


Figure 12 Creating a form context

Let's take a moment to review the different sections of the Context tab, as shown in **Figure 12**.

In the top left section of the Context tab, you see the interface we just created. It is an exact reflection of what we completed on the Interface tab. This section is the main foundation for specifying which

data is copied from the interface to the form. You include this data as a node in the hierarchy structure on the right. On the Context tab, the Interface information is visible in a read-only format — that is, if you want to make changes to the interface, you need to do so in the Interface tab of the Form Builder (refer back to Step 1). After you make your change there and

switch back to the Context tab, your changes are reflected in the Interface section on the left.

When you select a node in the Interface section by double-clicking on it, its related information, which comes from the ABAP Dictionary, appears in the bottom left section of the Context tab (for example, double-clicking on BOOKINGS under the Import node displays details on this database table). As with the interface information, you cannot make any changes from here.

The more important part of the Context tab is the context itself in the top right section. As previously mentioned, the form context constitutes the data model of your form. While the form interface is considered the maximum amount of data available for use in a form, the context is supposed to slim down that data amount to the data actually needed in the business form you want to print. Defining more narrowly which data you need in the form speeds up the time required for processing the form at runtime and therefore enhances performance. In the hierarchy of the Context section, you decide on the form logic by specifying conditions for processing the nodes. As in Smart Forms, a node or all subnodes are only processed if the particular conditions are met. When you first create a form, the context contains only a top-level folder representing the context.

In general, you have two options for adding elements (i.e., data) to the form context: dragging and dropping existing elements from the Interface section to the Context section or creating what you need using a context menu in the Context section.

Our first task is to identify the nodes that will provide the data for the sample form. We will need data from the CUSTOMER structure and the BOOKINGS table, as well as the Date field from the System Fields structure. (Remember, the goal of the form is to produce a PDF that lists flights booked with a particular customer sorted by airline companies.)

To create the form context for the sample form, we will use the Context tab to complete the following steps using both options for adding elements:

1. Drag the CUSTOMER node, the BOOKINGS

node, and the Date field from under the System Fields node from the Interface section onto the top-level node in the Context section. In the case of the table nodes, this action generates a copy of the node with all the subordinates nodes (representing the fields of the table), which are flagged with a lock graphic. This graphic shows that you cannot delete, copy, or move generated nodes. You can set any node that you do not need in your form as inactive, which is recommended from a performance point of view because the content of inactive nodes is not passed to the form at runtime.

2. Next, we need additional data regarding the flights that have been booked. We need to make sure that the data from the CONNECTIONS table will pass through the BOOKINGS table, as each booking is related to a specific flight. Drag the CONNECTIONS table onto the BOOKINGS node in the Context section to nest it. This move automatically creates a generated node CONNECTIONS at a lower hierarchy level in the structure.
3. Because we want to provide the total cost of all bookings for a specific customer on the form as well, we need data from the T_SUMS table. Drag the T_SUMS node from the Interface section onto the top-level node in the Context section. Double-click on this node to display the properties for this node in the bottom right section of the Context tab. This works for every node in the context. In this section, you can change some of the node properties (e.g., the name). For most nodes, you can also assign conditions — for tables you can insert WHERE conditions or control levels.
4. To make it clear what we use in the context, we want to give this table a different name in the context. Under General, change the name of the field to “TOTALS.” This name now also appears in the context hierarchy.
5. In order to reduce the time it takes to generate a form at runtime, it makes sense to structure your context according to the flow of the form (this again is a known Smart Forms principle). Putting nodes in the order in which they should be

Field Name	Status
Client	Enabled
Airline Code	Enabled
Flight Connection Number	Enabled
Flight date	Enabled
Booking number	Disabled
Customer Number	Disabled
Customer type	Disabled
Smoker	Disabled
Weight of Luggage	Disabled
Weight Unit	Disabled
Invoice flag	Disabled
Flight Class	Disabled
Booking price in foreign currency (dependent on loca...	Disabled
Payment currency	Disabled
Price of booking in local currency of airline	Disabled
Local currency of airline	Disabled
Booking Date	Disabled
Number of sales office	Disabled
Travel Agency Number	Disabled
Cancellation flag	Disabled
Reservation flag	Disabled
Name of the Passenger	Disabled
Form of address	Disabled
Date of Birth of a Passenger	Disabled
Table of Flight Connections	Enabled
Table for Flight Bookings	Enabled
Table for Flight Bookings	Enabled
Table for Flight Bookings	Enabled

Figure 13 Specifying the fields for the form context

Note!

For greater clarity, you can rename all nodes passed from the interface to the form context in the context itself. This can facilitate working on the form layout. When you switch from defining the context to the Layout tab to design the look and feel of the form template, the system replicates the field names you use in the context to the XML structure of Adobe LiveCycle Designer. These names are by default the system field names such as CARRID. To better identify in Designer the fields you can use from the back end, you may prefer to call this field “Airline Code” (the short text from the system) in the context, which would then be the name shown in Designer. (In the sample form, we simply use the system field names.)

processed at runtime facilitates this processing and has a positive effect on performance. You also want to disable nodes that are not needed — again to facilitate processing.

To achieve the best structure for the sample form, we need the ID, NAME, FORM, STREET, POSTBOX, POSTCODE, CITY, COUNTRY, REGION, and TELEPHONE fields from the CUSTOMER structure. From the CONNECTIONS table, we need the FLTIME, DEPTIME, and ARRTIME fields. In the TOTALS table, we need to disable all nodes, except FORCURAM and FORCURKEY.

Expand the tables and structures in the context, and then disable the nodes not needed. As shown in **Figure 13**, all the fields in the BOOKINGS table, except CARRID, CONNID, FLDATE, FORCURAM, and FORCURKEY, have been deactivated. (Use the context menu on each node to deactivate it. To make multiple selections, press Ctrl+Shift as you click to select the fields.)

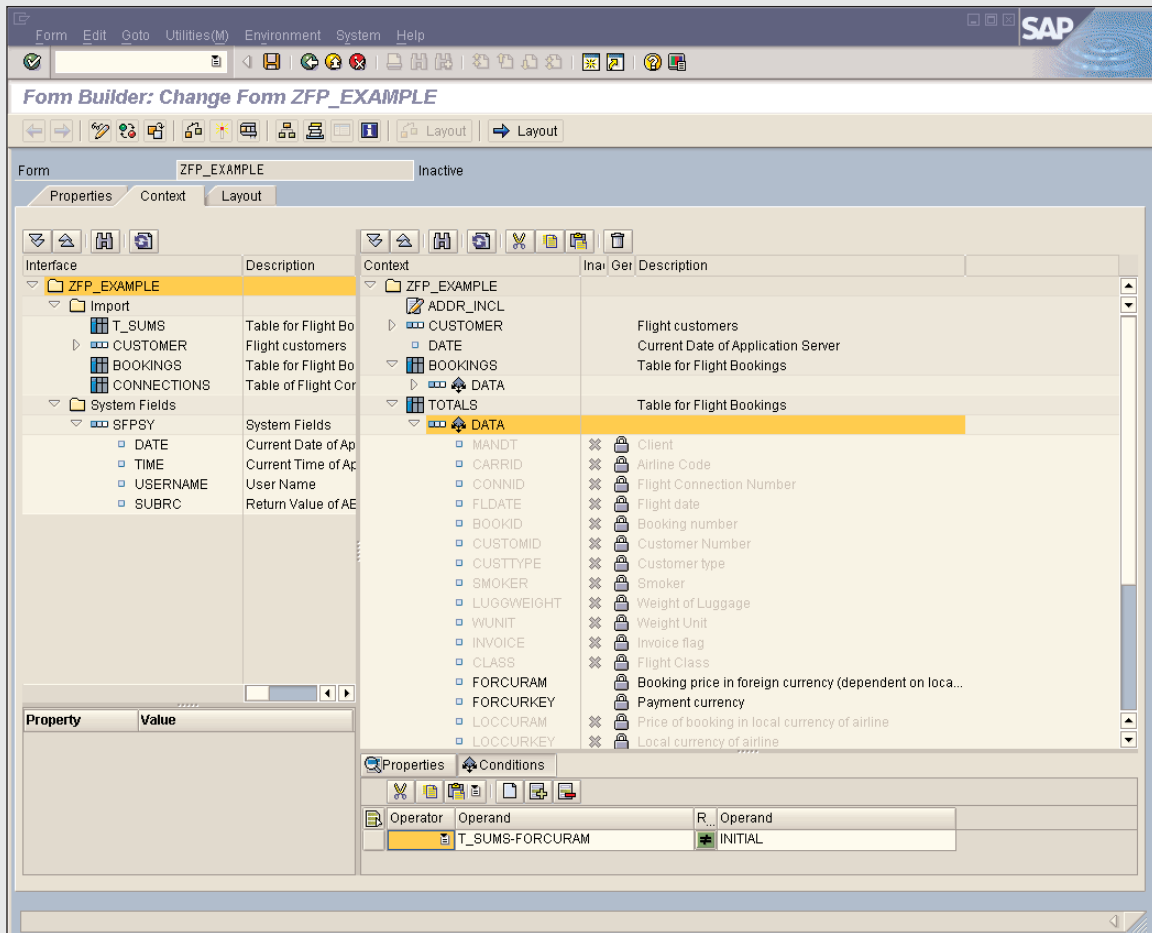


Figure 14 Specifying the condition for the BOOKINGS table

6. Next, we need to create two conditions and one WHERE condition on nodes we have already added to the context. These conditions drive some of the processing of the form at runtime. We need to set up the DATA node of the BOOKINGS node so that the price of each booking has a value. The same condition applies for the DATA node of the TOTALS node, except that in this case it relates to the T_SUMS table instead of the BOOKINGS table.
 - To create this condition for the BOOKINGS table, double-click on the DATA node under BOOKINGS. In the Properties section on the bottom right, click on the Conditions button, and enter “BOOKINGS-FORCARAM” under

Operand, select the “not equal to” operator under Relational operator, and enter “INITIAL” in the second Operand field. Proceed likewise for the DATA node under TOTALS, except enter “T_SUMS-FORCARAM” to replace BOOKINGS-FORCARAM in the first Operand field. In **Figure 14**, you can see the condition attached to the TOTALS node that is based on the T_SUMS table in the interface.

- The processing of the CONNECTIONS node, which we nested under the BOOKINGS node in the Step 2, is subject to a WHERE condition at runtime. The form should only list flights for which the airline code and the flight connection



Operator	Operand	R... Operand
AND	CARRID	BOOKINGS-CARRID
AND	CONNID	BOOKINGS-CONNID

Figure 15 Specifying the WHERE condition for the CONNECTIONS node at runtime

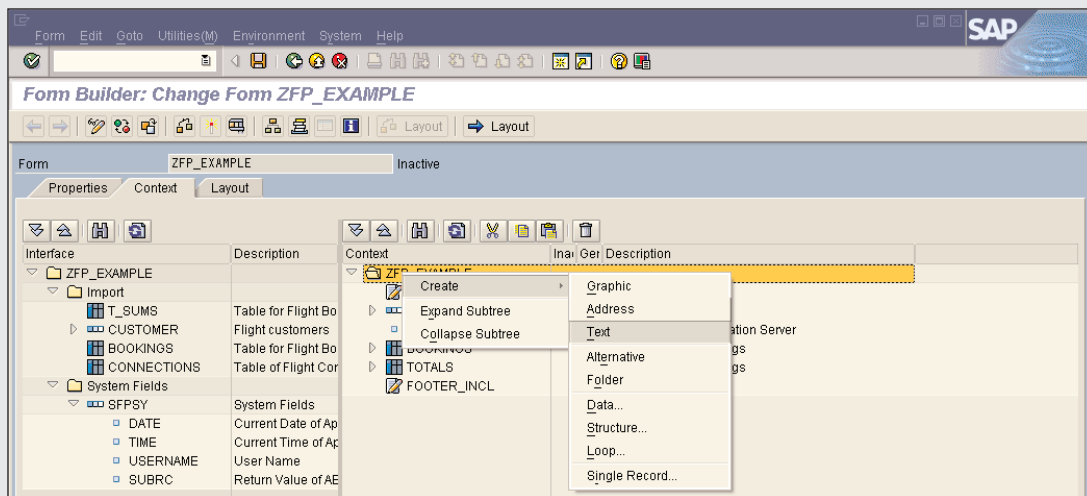


Figure 16 Creating a Text node

number of this table corresponds to the airline code and flight connection number of the BOOKINGS table. To create the WHERE condition, double-click on the CONNECTIONS node, and choose the Where Conditions button in section below the context. Here you can change the properties of the context nodes and specify conditions for processing the nodes. Depending on the type of node you want to edit, the options provided in this area vary, as shown in **Figure 15**.

7. We also need to provide address information on the form. In the example, the information we need is stored in Smart Forms text modules in the system. As this information is not passed to the form through the form interface, we need to create the required nodes in the hierarchy structure by manually adding the nodes using the context menu.

You can create new nodes from scratch by using the context menus on any node in the context hierarchy. The nodes you can create are essentially the same ones as in Smart Forms. For the example, we will create an address area and a footer, which will appear on every page of the printout.

- Create two Text nodes in the context by right-clicking on the context node under which you want to insert the text nodes: the top-level node ZFP_EXAMPLE for the address node, and TOTALS for the footer node. From the context menu, choose Create and then choose Text, which inserts a Text node (see **Figure 16**).
- In the Properties section for each node, call the nodes ADDR_INCL and FOOTER_INCL, respectively. To assign the Smart Forms text modules SF_ADRS_SENDER and

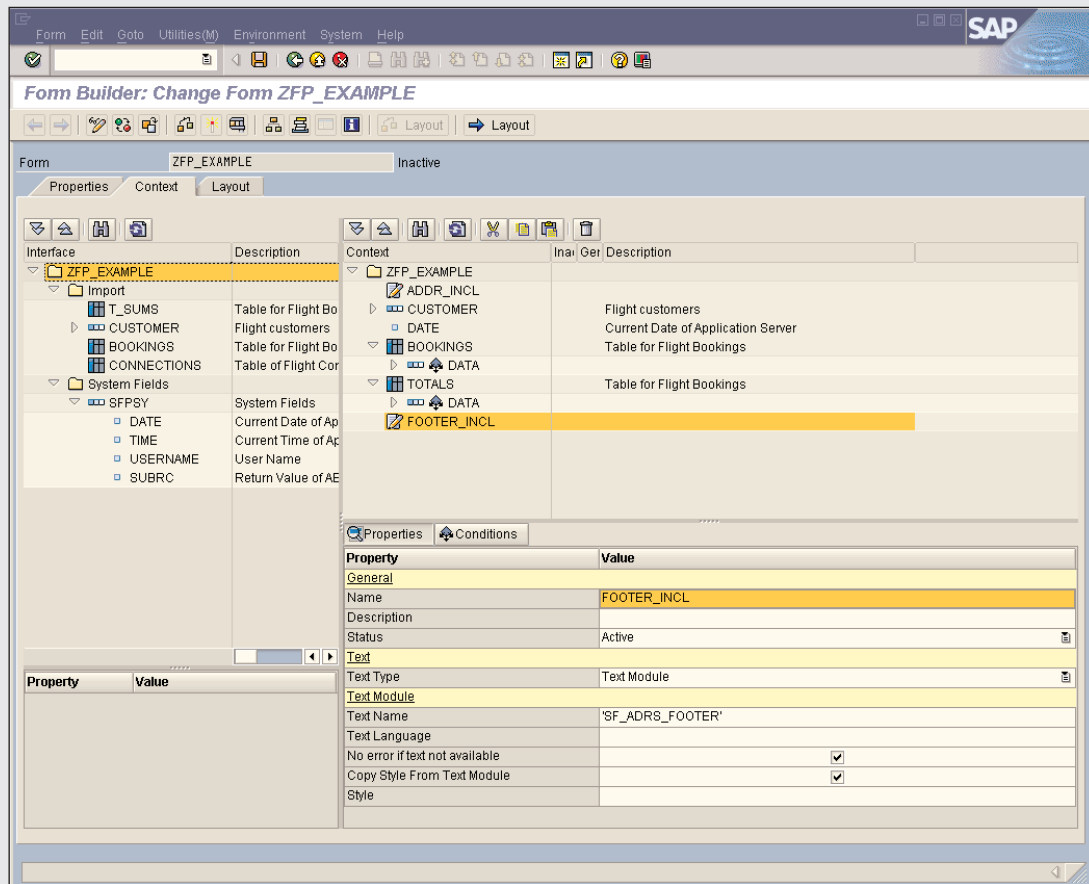


Figure 17 Creating a footer that uses a text module

SF_ADRS_FOOTER, respectively, to the nodes, select “Text Module” for the Text Type, and then select the appropriate module in the Text Name field. (You can include Smart Forms text modules or SAPscript INCLUDE texts in the context to reuse units of text stored in the back end. This greatly reduces the maintenance efforts for your forms.) See the footer example in **Figure 17**.

Our next task is laying out the form, which we will do in Adobe LiveCycle Designer.

Designing the form layout

Be aware that there is more to laying out a form than

Note!

Covering every detail and element of the layout of the sample form goes beyond the scope of this article. This article focuses on the general principles of form design in SAP Interactive Forms and provides concrete examples to help illustrate those principles.

determining where labels and fields physically appear on the form. You need to create the *data binding* — that is, you need to identify which data from which

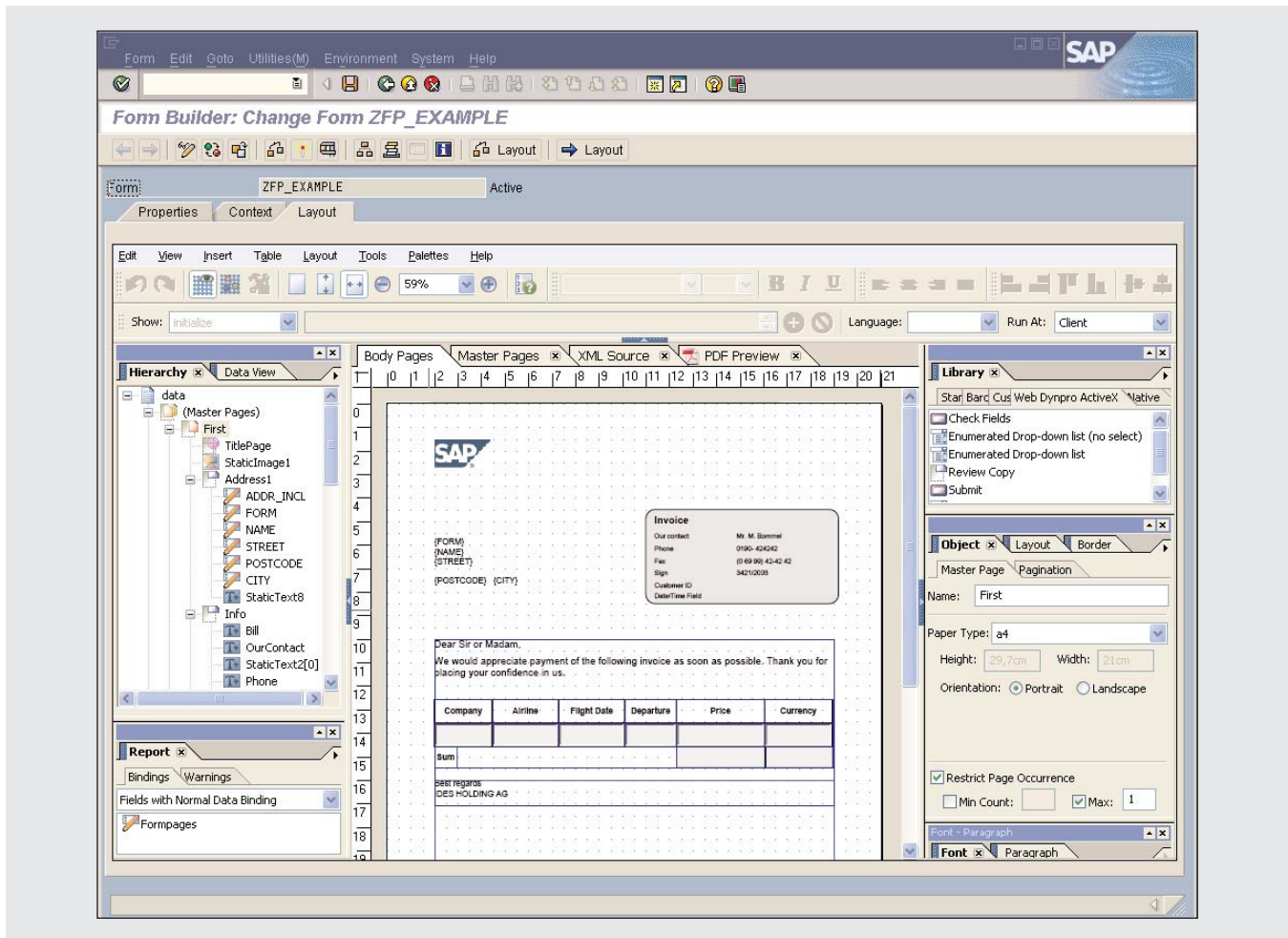


Figure 18 Specifying the form layout

back-end field is connected to which field on the form. Through this activity, ADS is then able to determine where to merge the system data passed from the application (remember the architecture) into the form.

Because we'll have to display a long list of flights on each invoice, we will also create a table that displays the relevant information for each flight.

To work on the form layout, you switch to the Layout tab in the Form Builder. This action automatically launches Adobe LiveCycle Designer inside the SAP GUI, as shown in **Figure 18**.⁷

⁷ Figure 18 shows the form template that is the goal of this article. When you create a new form template, the work area is empty.

First, let's look at the environment in which we will work now.

Designer is completely framed by SAP GUI, so that you continue to work in your SAP development environment and can take advantage of the seamless integration of Designer into the ABAP Workbench. As you can see, Designer has its own menu,⁸ a central work area where you design the form layout, and palettes with different features and functions on either side of the work area.

Before we design the layout, let's first examine the four main palettes provided by the Designer.

⁸ Note that the standard File menu is deactivated in the SAP environment, because the form template is stored inside the SAP database only.

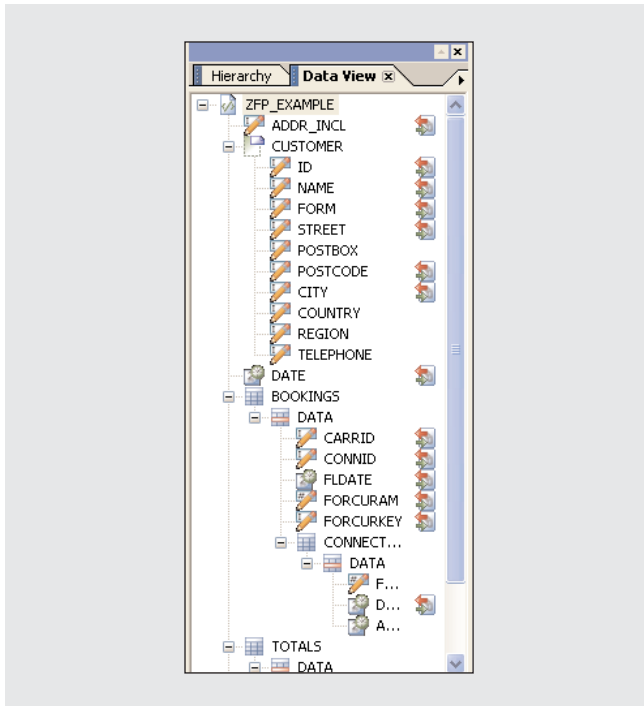


Figure 19 Data View palette in Adobe LiveCycle Designer

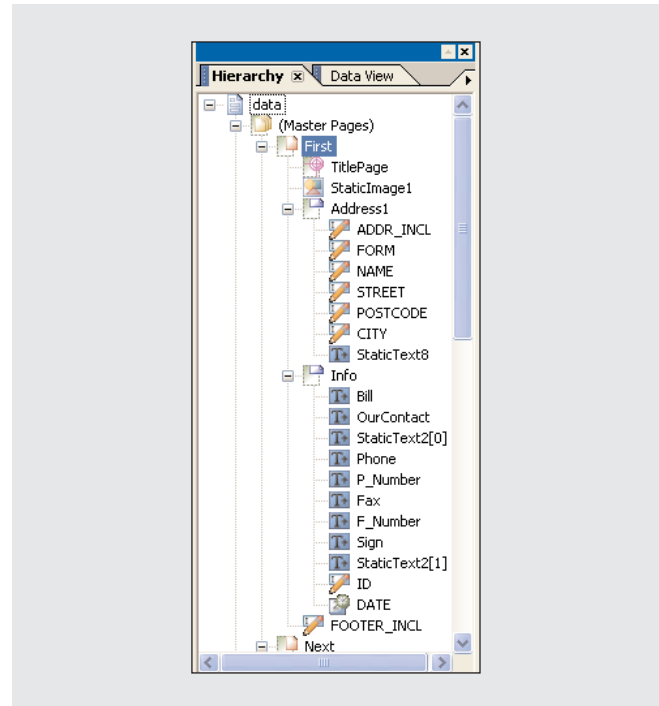


Figure 20 Hierarchy palette in Adobe LiveCycle Designer

Designer palettes

Figure 19 shows the Data View palette, which displays the data you compiled in the form context for use in your form. This display is based on an XML schema generated automatically by the SAP back end when you clicked on the Layout tab. The data appears exactly as in the hierarchy on the Context tab. If you make changes to your form context, they will be reflected here because the schema is generated each time you switch to Designer.

Note!

Remember the Note on page 17 that describes changing the names of the nodes in the context? **Figure 19** shows the system field names we used.

To design the form layout, you can drag and drop any number of fields from the Data View onto the

work area. Designer “understands” the type of back-end fields used and creates a form object (form field) in the work area that corresponds to the back-end field type, for example, a Date/Time object for a date field, or a Text object for the output of a long-text field from the back end. For the sample form, we use the drag-and-drop method only to create the Customer ID field in the Invoice section of the form.

In contrast to the use of the Data View, the Hierarchy palette displays all of the form elements (or objects) that you have used in your form layout (see **Figure 20**).⁹ Every object you place in the work area is automatically integrated into the hierarchical structure of the Hierarchy palette. Because it provides a complete overview of your form, the Hierarchy palette is very useful when you have a complex form with many form objects and need to select individual objects on the form.

⁹ Figure 20 displays the fields in the top part of the sample form, that is, the SAP logo (StaticImage1), the sender's address section (the group of elements entitled “Address1”), and the section entitled “Invoice” (the group of elements entitled “Info”).

The Library palette groups all available form objects in different categories. The Standard, Barcodes, and Custom tabs are part of Adobe's stand-alone Designer delivery. In its own delivery, SAP has added special objects, in particular for the integration of PDF forms into Web Dynpro applications. (These objects are not relevant to our exercise.)

On the Standard palette, you find the most commonly used form objects, including (static) Text, Text Field, and Date/Time Field to create fields you need on your form. You use a Text Field object for fields that can be filled at runtime by, for example, long texts from the back end. They are basically placeholders for texts. You use a (static) Text object for texts that reside only in the form itself (for example, a label "Invoice"). Static texts have no connection to a back-end field; the information they contain is solely available to the form itself. You use a Date/Time Field object for fields that contain dates and/or times fed by the back end (for example, the date of a flight). Tables are another example of form objects, which we'll cover in more detail shortly.

In the sample form, we use, among others:

- An Image object for the SAP logo
- A static Text object containing floating fields (i.e., variables) for the sender's address
- Several static Text objects for the labels in the Invoice section
- Static Text objects for addressing the recipient of the form ("Dear Sir or Madam..." and "Best regards...")
- A Table object containing static Text objects in the header row and Text and Date/Time Fields in the body row

The Object palette is the most important one for the form layout (see **Figure 21** on the next page). With the Object palette (as well as with the Border and Layout palettes), you configure each of the form objects you place on the work area. Although you use

the Border and Layout palettes to set the properties for the object's look and feel and its location on the form, you use the Object palette to:

- Determine the data from the back-end system, if any, that will appear in the form as part of the particular object using the Binding tab. (This is the data passed to the form through the interface and the context.)
- Specify the field type, caption, and other appearance-related properties for the particular object you are configuring using the Cell tab.
- Determine the character of the object (interactive or read-only, mandatory or optional, etc.) using the Value tab.

You can also see that in the Data View we selected the FLDATE node, which corresponds to the Flight Date field of the table in the work area. To the right of the work area, the Binding tab of the Object palette is displayed. You can see that the FLDATE node of the Data View structure is bound to the FLDATE field in the back end (under Default Binding).

Designing the layout of a form is more than just positioning fields on the work area — it also has to do with making a "connection" between the fields on the form and the back-end sources of the data needed for processing. This connection is called *data binding*. Data binding tells the key component of the SAP Interactive Forms architecture (that is, ADS) where the data comes from and to where the processed data needs to be written.

Data binding

There are essentially two ways to bind form objects against the database fields. If you use the drag-and-drop feature in the Data View, the action automatically creates a connection between the field in the back end and the field on the form template — therefore automatically binding the data. You can also use the Library palette to drag objects onto the work area. Although this approach provides a real layout of the

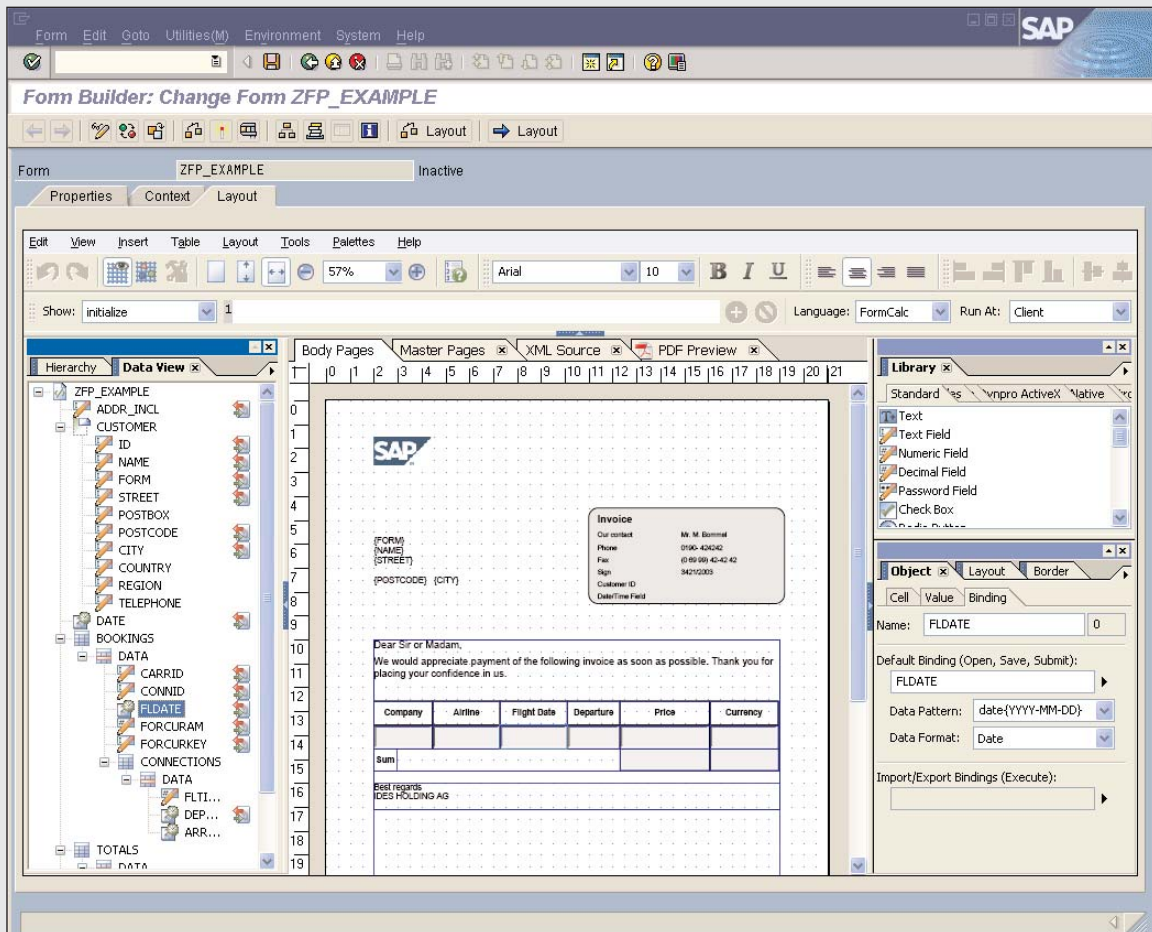


Figure 21 Object palette in Adobe LiveCycle Designer

form, it does not create the connection needed to bind the data. If you use the Library palette to lay out form objects, you need a second step to manually establish the required connection between the form and the back end.

To create a manual data binding for a form field in the sample form, you switch to the Binding tab on the Object palette, and click on the arrow next to the Default Binding field (see **Figure 22**). As the structure opens, you can navigate through the nodes (which reflect the structure of the Data View and therefore the form context) to the node you want to bind this object against. The binding information is then inserted into the Default Binding field.

One important element of many business forms, such as order lists or invoices, is tables. Tables are useful for displaying data in a structured way so that the reader of the information can get a good overview of the data shown on the form. In the sample form, we want to add a table to our form layout that has six columns, displaying information such as the airline, the flight date, and the price of a flight.

Designing tables

In SAP NetWeaver 2004s (and SAP NetWeaver '04 SPS 18 or higher), you can create tables in Designer using a table wizard.

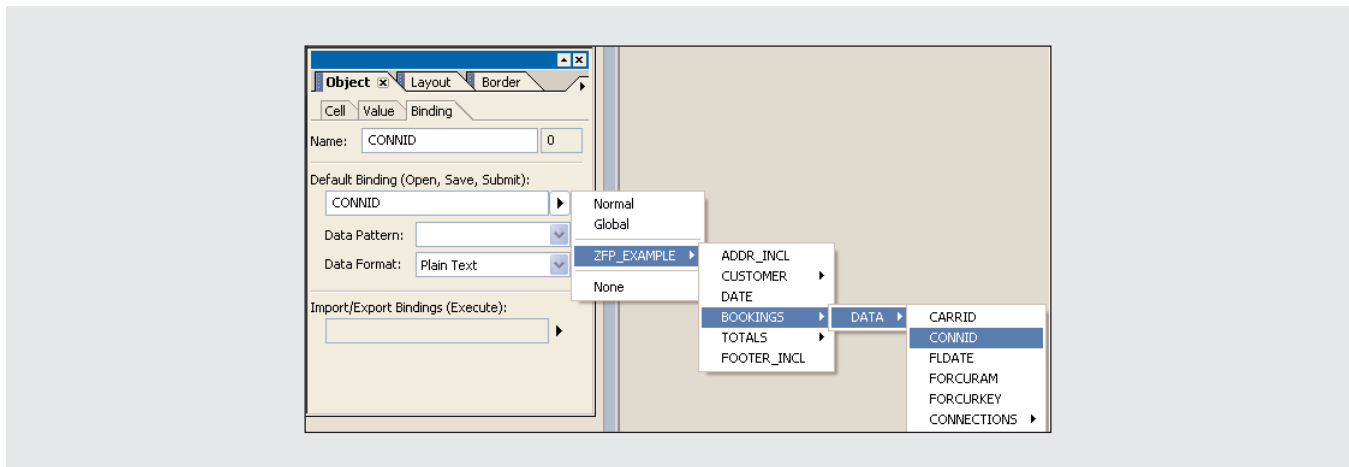


Figure 22 Manual data binding

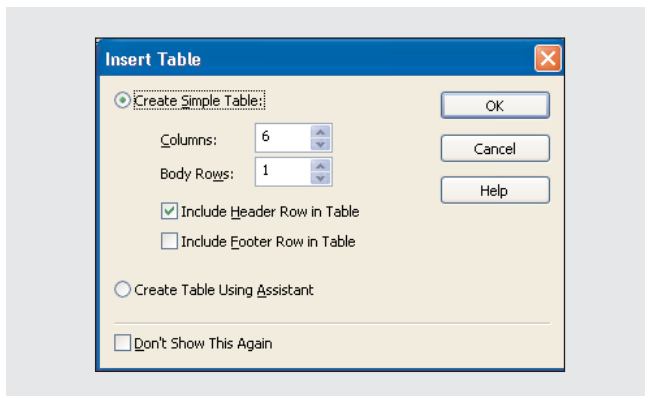


Figure 23 Creating a table object on a form

For the sample form, drag the Table object from the Standard tab of the Library palette onto the work area. Enter the values in the Insert Table dialog, as shown in **Figure 23**. As you can see, there are other functions you can use to set up more complex tables, which do not apply to the sample form.

After you specify the values needed for the table object, the table wizard creates the table. You can see the structure of the table in the Hierarchy palette, as shown in **Figure 24** on the next page.

By default, all table fields are created as static texts. Fields that are static texts are acceptable when all you want to do is change the header titles for the different columns. However, for the sample form, we

Note!

We have only specified one body row although presumably a form with dozens of bookings will require more than that. For the purposes of this article, this is the key information you need. Setting up a table to accommodate a large amount of back-end data at runtime is a simple configuration step in the form template, which we'll not cover here.

Note!

You need to manually rename all nodes in the Hierarchy View if you use the table wizard to create your table. There are other ways to create tables, but this one is the most convenient if you look at the overall process of working with tables in Designer.

need to change the field types of the body row of the table to Text Fields (for regular data fields) and

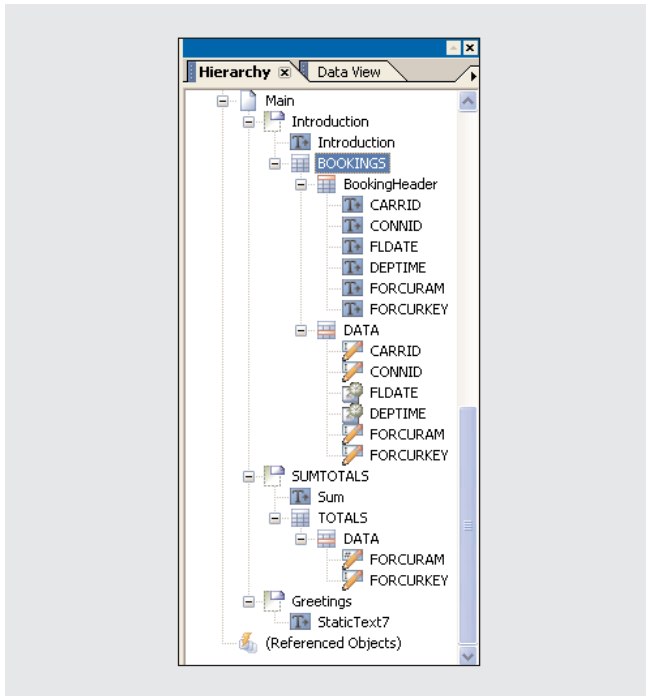


Figure 24 BOOKINGS table hierarchy

Date/Time Fields for the flight date and departure time fields (i.e., FLDATE and DEPTIME). To do so, we need to select the cell (that is, the field) we want to change in the work area or on the Hierarchy tab, switch to the Object palette, and change the type using the drop-down menu (see **Figure 25**).

For each of the data fields, we also need to create the data binding by navigating the data structure on the Binding tab. By way of an example, here's what you need to do for the Flight Date column (which you would have to do for all columns of the table):

1. Select the third field of the header row, and double-click on the default text inserted by Designer. Replace this text with "Flight Date." (Remember that the header row fields can remain static texts.)
2. Select the third field of the body row right under the Flight Date field, switch to the Cell tab of the Object palette, and change the field type from Text to Date/Time.

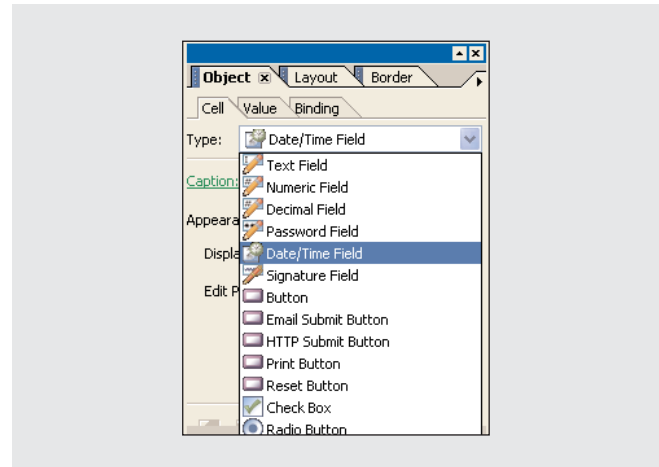


Figure 25 Changing the field type on the Object palette

3. Switch to the Binding tab of the field. Click on the arrow next to the Default Binding field, and navigate through ZFP_EXAMPLE → BOOKINGS → DATA → FLDATE. After you select this entry in the menu, it appears in the Default Binding field.
4. For more clarity in the Hierarchy palette, change the labels of the two fields (header and body row) to FLDATE.

Work area tabs

As you may have already noticed, the work area has several other tabs: Body Pages, Master Pages, XML Source, and PDF Preview.

Body pages represent the pages of a form. The back-end data you display in a form is placed on the body pages. Each body page derives its page size and orientation from a master page. A master page is essentially the background of a form. If you have items you want to place on every page of the generated form, you need to put them on the master page. A form can have several master pages, for example, one for the first page of a document and another for all subsequent pages.

In the sample form, the logo, the sender's address, and the Invoice section are placed on the first master

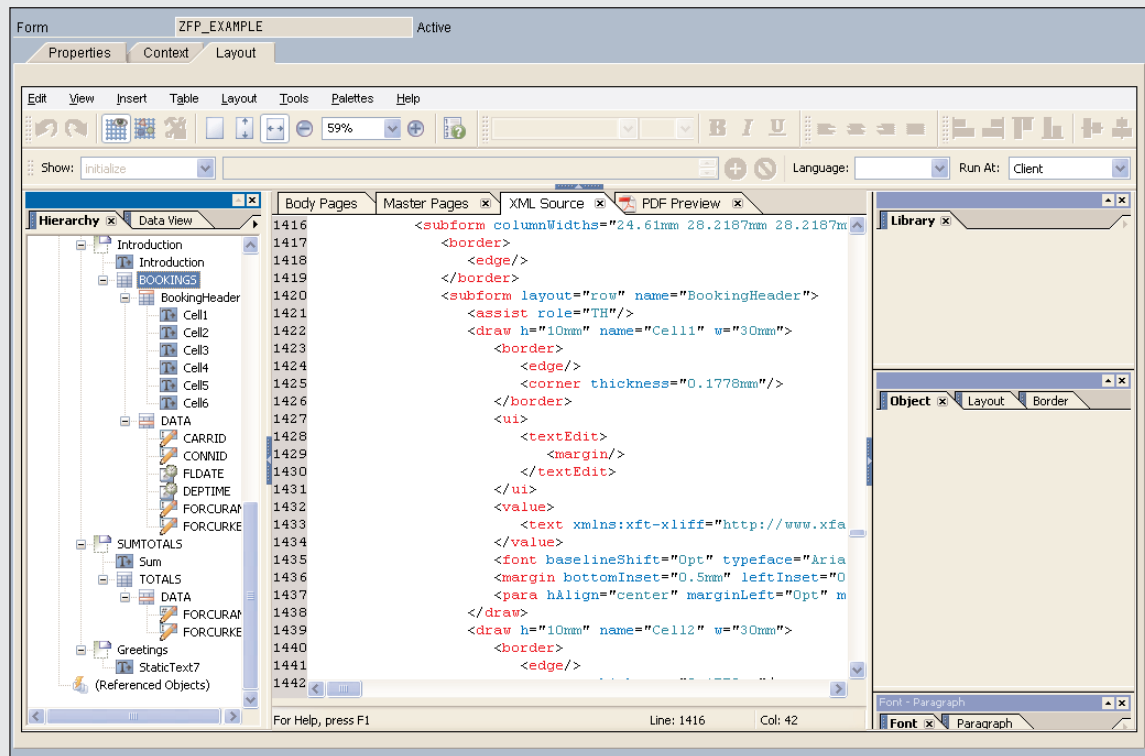


Figure 26 Displaying the XML code of the BOOKINGS table

page (i.e., these elements appear only on the first page of the generated form). As it is quite likely that the kind of form we want to generate contains dozens if not hundreds of flight bookings covering many pages, and we do not want to see the sender's address on each (but instead, for example, a page counter such as "Page X of Y" on every subsequent page), we have created a second master page, which ADS uses as the template for every page after the first page of the generated form is completed. This second master page contains the logo, the page counter, and the footer we inserted in the context. The footer will appear on every page, because it is also part of the first master page. The body page of our example contains the salutation and introductory text, the table, and the final greeting.

As mentioned at the beginning of this article, SAP Interactive Forms is based on Adobe's XML Forms Architecture (XFA) — that is, the XML format used to

transfer data between the SAP system, Designer, and ADS. Anything you do during form design is automatically converted by Designer to this XML format and displayed on the XML Source tab. **Figure 26** shows some of the XML code for the completed bookings table. Although you can edit the code on this tab, SAP recommends that you do not use this feature in the SAP integration, because the SAP environment adds important generated tags to the XML code, which could corrupt the form template if they are accidentally deleted.

The PDF preview function allows you to check during form development what your template will look like when generated as a PDF document. Obviously, at design time the form contains no data (so a table would not appear at all), but it allows you to check the general look and feel of your form. To display a PDF, Designer loads the Adobe Reader plug-in used in Microsoft Internet Explorer.

Script editor

The script editor does not apply to our example form, but given the significance of certain business logic in a form, it is important for you to know about.

To create limited business logic inside the form, which can only be executed at runtime when the back-end data has been merged with the template (for example, calculations such as additions of values in certain fields), you can use the script editor in Designer, which is located right above the work area. The Adobe PDF technology supports scripting in JavaScript and FormCalc, an Adobe proprietary language that allows you to execute spreadsheet-type functions.

Although scripts can be quite extensive, it is recommended that you keep them short because scripting increases the size of the PDF and the time required to generate the output formats.

After we have finished the form template (form context and form layout), we can activate it. Similar to those in Smart Forms, system checks run on the different parts of the form (i.e., the interface and the context). Other checks are possible by executing a print program that calls the form and then provides a preview of the form with data.

Step 3: Create a print program

As with Smart Forms and SAPscript, an application needs to provide a print program that calls the generated function module representing the form.

A typical print program contains five parts:


- A SELECT statement to retrieve the correct data from the database
- A function call `FP_FUNCTION_MODULE_NAME` to retrieve the name of the generated function module
- A function call `FP_JOB_OPEN`, which opens a spool job and starts the form processing
- A function call of the generated function module as well as a loop over the selected data

- A function call `FP_JOB_CLOSE` to close the spool job

Although the details of the development of the print program needed for the sample form are beyond the scope of this article, you can find the required code for our print program for download at www.SAPpro.com.

With the form and print program complete, we are ready to execute the program and see the form in action.

Step 4: Execute the print program and generate the PDF document

After you have completed your application (or print) program, you should run it to test the data retrieval and form generation capabilities. In transaction SE38, you execute the corresponding report. Testing the sample form takes you to the screen shown in **Figure 27**. Here, enter the name of the form (i.e., `ZFP_EXAMPLE`), and click on Execute (.

Next, on the Text Function Module screen and then on the Print Preview screen, click on Execute and Print Preview, respectively. See **Figure 28** for a preview of the generated form in PDF format in the SAP GUI.

As you can see, using SAP Interactive Forms provides a flexible alternative to and complements the proven functionality of Smart Forms and SAPscript.

Tips & Tricks

Here are some tips and tricks that will help to make your SAP Interactive Forms experience easier:

- If you work with a team of form developers and want to reuse certain form objects in Adobe LiveCycle Designer, you can create, save, and share such elements. For example, you may need to use the same header layout (which consists of

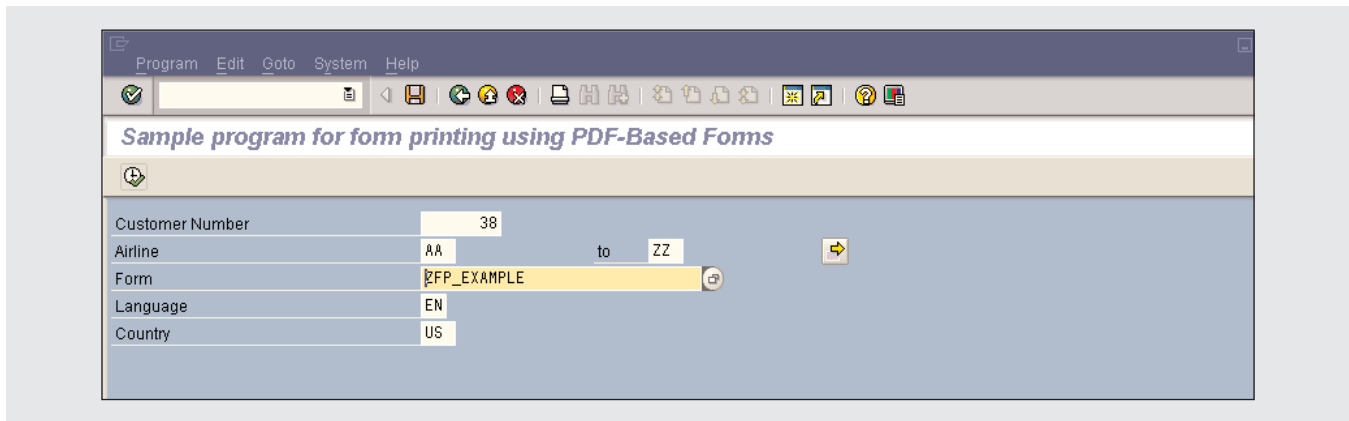


Figure 27 Sample program for forms generation

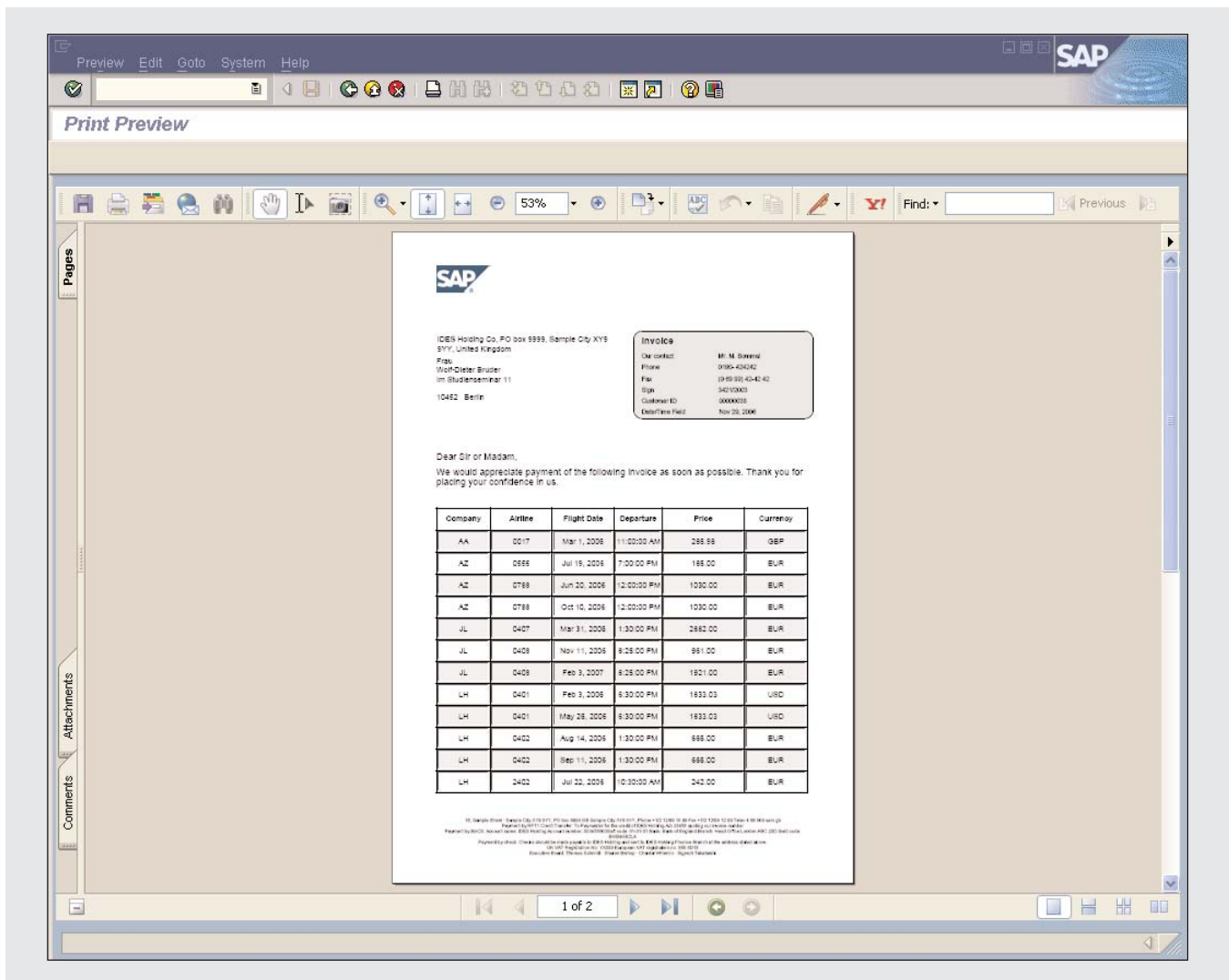


Figure 28 Previewing the sample form

graphic elements such as lines, specific colors, the title of the form, and the company logo) for all your forms. To make such a header reusable, arrange these elements in the work area, group them (press and hold the Ctrl key while you click on each element), and then drag the group onto the Library palette. In the dialog that appears, enter a name and a description for your custom object, and then assign it to a Library group tab. You can also create your own tab for your custom objects.

Because this only saves the object locally on your computer, you need to place it on a shared server to make it available to other form developers, who simply point their corresponding Library tab to the server by editing the Library group properties.

- To meet its enterprise readiness and supportability goals, SAP provides extensive logging and tracing capabilities for SAP Interactive Forms, especially its runtime. If you go to the Settings menu on the SFP entry screen, you can select a Very Detailed Trace and save all relevant runtime information (including the form template and the runtime data file) in a PDF generated by ADS (see **Figure 29**).
- Using XML as the basis of communication (in our case for all processes between the SAP environment and ADS) requires, by definition, a certain administrative overhead compared to mere ABAP processing, which may result in slower performance for certain forms. In mySAP ERP 2005, SAP recommends that you use a double-stack installation of SAP NetWeaver (that is, the ABAP and the Java stack on the same physical system) to be able to take advantage of all performance-related enhancements shipped with this release.¹⁰
- SAP Interactive Forms is currently subject to some release restrictions in both SAP NetWeaver '04 and 2004s. Before starting a project with PDF-based print forms, familiarize yourself with these restrictions, which are documented in SAP Notes 863893 and 894389 for '04 and 2004s, respectively.

¹⁰ SAP is aware that in certain situations a double stack is not feasible and is currently working to provide another high-performance alternative, the so-called Remote ADS. See SAP Note 993612.

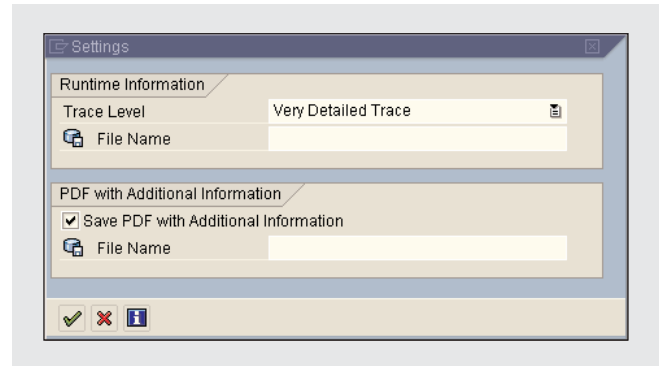


Figure 29 Specifying runtime settings for your print form

You can get more tips and tricks and contact other developers through a dedicated SAP Interactive Forms site at <http://sdn.sap.com/irj/sdn/interactiveforms>.

Conclusion

If you are familiar with Smart Forms, you've seen throughout this article that SAP and Adobe have upheld many of the Smart Forms principles in transaction SFP. This should make it easy to transition your forms processes to SAP's strategic forms technology. Be assured that existing Smart Forms and SAPscript forms will continue to work as before, which should help you choose the right time for switching to the new forms technology.

Although all the necessary foundations were available with mySAP ERP 2004 (based on SAP NetWeaver '04), mySAP ERP 2005 is the release that offers hundreds of form templates based on the PDF technology in practically all applications.¹¹

Although SAP ships extensive SAP Library documentation for the Form Builder, and the integrated Adobe LiveCycle Designer also comes with detailed documentation, you may want to first attend a training course to familiarize yourself with the new technology. SAP Education offers the BC480 course on a regular basis in many training centers around the world.

¹¹ For a list of all PDF-based forms shipped and three documents on SAP Interactive Forms, go to <http://service.sap.com/erp> and then follow the menu path Media Library → mySAP ERP Overview.