The PDF Toolbox for ABAP — a cost-effective, open-source solution for integrating PDF forms into your BSP applications

by Cord Jastram

Cord Jastram
Software Engineer,
Computer Sciences
Corporation, Germany

Cord Jastram works for Computer Sciences Corporation (CSC) in Germany as a software developer. His main focus lies in software development using Java, ABAP, and C++. He holds a Ph.D. from Hamburg University where he did research in the field of numerical seismic modeling. Before joining CSC in 2000, he worked as a software developer for different companies. You may reach him at cjastram@csc.com.

The Business Server Page (BSP) technology introduced with the SAP Web Application Server (SAP Web AS) 6.10 is a mature technology to develop HTML-based Web applications. But there are some situations in which you may want to use a Portable Document Format (PDF) form instead of an HTML-based form:

- When you have a complex form and you want to offer high-quality printer output to the end user, it might be a good alternative to use a PDF form instead of an HTML-based form because PDF is designed for excellent printing quality.
- When you want to save the output of your Web application to a local disk for later reference, a PDF form is saved just as a single file, whereas HTML output might consist of several files.
- When you have a paper-based business process, it is much easier to create an electronic form that looks exactly like the paper-based form with PDF than with HTML.
- PDF forms work without any restrictions in an Internet browser for which JavaScript support is disabled for security reasons.

The Interactive Forms technology, which allows for the integration of PDF forms in your business process, was introduced with SAP NetWeaver '04 for the SAP NetWeaver Java stack and with SAP NetWeaver 2004s for the ABAP stack[1]. However, Interactive Forms is intended for use in combination with Web Dynpro technology either for Java or for ABAP, and unfortunately, Interactive Forms does not integrate seamlessly with BSP applications. Furthermore, you need an SAP Web AS 6.40 for a Java

[1] For information, 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" by Markus Meisl and Marc Chan (January/February 2006).

PDF Toolbox for ABAP

ABAP does not provide a library for manipulating PDF files, so I developed a solution — a toolset that would offer ABAP developers a way to manipulate PDF files. My PDF Toolbox for ABAP is an open-source solution. The purpose of this toolbox is to allow ABAP developers to programmatically modify existing PDF documents. The toolbox consists of two components: an RFC server and two ABAP Object classes.

The RFC server is implemented in Java and provides the functionality that you can use to make the actual modifications to the PDF files. The RFC server is a simple wrapper for an open source Java library called iText,* which you can use for modifying existing PDF files from Java. You can run the Java RFC server on any computer equipped with a Java 5 Runtime Environment** and with an SAP Java Connector (JCo).

The Java RFC server offers the following functions:

- Encrypting (password protecting) a PDF file and setting its usage permissions
- Adding an attachment to a PDF file
- Adding a comment to a PDF file
- Adding a background text to a PDF file
- Permanently filling in the fields of a PDF form
- Setting the status of a form field to read-only (that is, it can't be changed by the user)
- Concatenating two PDF files

Continues on next page

Note!

The solution shown in this article is based on the Adobe AcroForm technology, which was introduced by Adobe in its PDF 1.2. The other forms technology is the Adobe XML Forms Architecture (XFA), which was introduced with PDF 1.5 and which Interactive Forms uses. As a rule of thumb, you can create XFA-based forms using Adobe Designer; whereas, you create AcroForm-based forms using Adobe Acrobat Professional. However, several other commercial and open-source solutions exist for creating AcroForm-based PDF files (e.g., commercial solutions include Foxit PDF Editor from Foxit or FormMax and CutePDF Professional from CutePDF; whereas, open-source solutions include OpenOffice 2.0 and Scribus).

- Adding a toolbar to a PDF file
- Creating a ZIP file in which the newly created PDF file is stored, and adding additional files to the ZIP file

The two ABAP Object classes provide an application programming interface (API) for interacting with the RFC server. They provide an easy-to-use interface that allows ABAP developers to use the server without any knowledge of the Java programming language. Two ABAP classes, ZCL_FILE and ZCL_PDF_COMMAND_LIST, represent a file and a list of commands to be sent to the RFC server for processing.

The RFC server is the component of the solution that closes the gap between the Java and the ABAP worlds. It allows an ABAP developer to call code written in Java just like an ABAP function module. There are some pitfalls when you work with an RFC server, but the ABAP classes ZCL_FILE and ZCL_PDF_COMMAND_LIST simplify your work significantly. When you use these classes, you don't even notice that you are using an RFC server.

Your program creates one or more instances of the ZCL_FILE class to represent PDF files and an instance of ZCL_PDF_COMMAND_LIST to send one or more commands to the RFC server. You create a ZCL_FILE object by reading an existing file from the client or from the server. Another option is to create a ZCL_FILE object by converting the Output Text Format (OTF) of an SAP Smart Form to PDF and then to load the desired commands (with any associated parameters) into a ZCL_PDF_COMMAND_LIST object. Then you call the PROCESS method of the command list object that takes on the ZCL_FILE objects you want to work with as parameters. The PROCESS method calls the function module Z_PDF_PROCESS_COMMANDS on the RFC server, providing the command list and the file data as the input data. The RFC server, in turn, performs the modifications to the file and sends back a modified file. This is done behind the scenes, so your program is free to serve the PDF data to the user — for example, via the Web in a BSP application — who can save it to a file or further manipulate the data.

* iText offers additional functions (or services), such as the ability to add a watermark or digital signature.
** You don't need a J2EE system; you simply need a Java 5 Runtime Environment.

system, and you have to pay for a license for Adobe Document Services.

In this article, I introduce to you a solution that I have developed for integrating PDF forms into BSP applications. My solution uses an extended version of the PDF toolbox for ABAP, which I described in a recent *SAP Professional Journal* article, "Improve your business processes with quick and easy enhancements to PDF documents: A toolbox for modifying PDF files using ABAP" (May/June 2007).[2]

[2] For a brief overview of the PDF toolbox, see the sidebar that begins on the previous page.

You can use my PDF solution on SAP Web AS 6.10 and higher and it only needs an additional Remote Function Call (RFC) server. The solution is aimed to be a cost-effective approach to extend your already existing BSP applications with PDF forms. However, the solution should not be seen as a replacement or an alternative for Interactive Forms, which are better suited to handle highly variable forms than the solution I introduce.

This article provides an overview on the technical approach that I use to integrate a PDF form into a BSP application. The solution I propose here is based

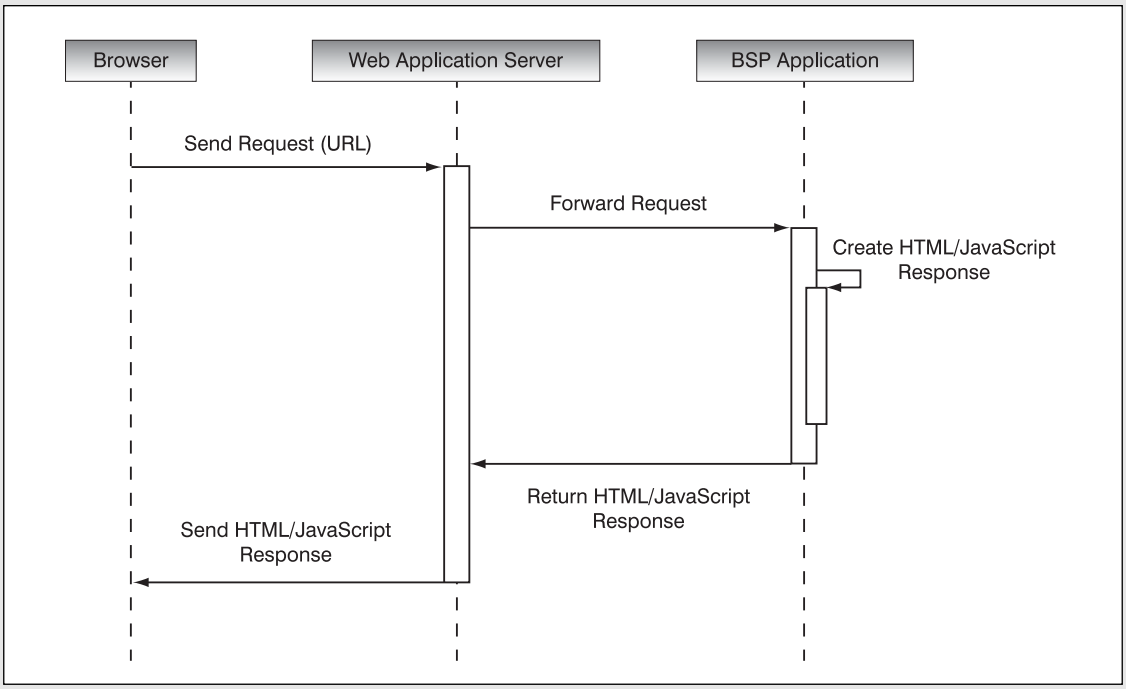

Figure 1 Overview of the processing flow of a BSP application

on a template, or a static PDF form. This form is modified at runtime by the BSP application, and the newly generated PDF form is sent back as the response. Included in this overview is a description of the example that showcases some of the features of the solution. Next, I explain how to integrate a PDF form into a BSP application using my PDF solution for ABAP, and how to create the BSP application so it takes advantage of my PDF toolbox solution. Finally, I explain the improvements I have made to my PDF Toolbox so it can be used more effectively with BSP applications and provide a closer look at the coding of the PDF-BSP sample application. Let's begin with an overview of the technical background of the solution.

Integrating a PDF form into a BSP application overview

A BSP application is typically called by an Internet browser, such as Mozilla Firefox or Microsoft Internet Explorer. As a first step, the browser sends a request to the SAP Web AS. Depending on the URL of the request, the server forwards the request to the corresponding BSP application. As a response to the request, the BSP application generates HTML code and sends it back to the server, which, in turn, sends the response back to the browser. Today, in most cases, there is also JavaScript code embedded in the HTML pages. The JavaScript code, for example, allows for client-side validation of form input data, which is not feasible with plain HTML code. **Figure 1** shows a schematic picture of the processing flow of a request.

The Web browser has a built-in HTML rendering engine that creates a graphical representation of the HTML code and a JavaScript interpreter that parses and executes the embedded JavaScript code. **Figure 2** shows a schematic view of the browser components.

A detailed description of HTML and JavaScript processing is beyond the scope of this article, but as an example, just have a look at the HTML/JavaScript code snippet shown in **Figure 3**. This snippet creates an HTML form with a single input field and uses

JavaScript to check that a number is entered in the input field. The first part is the form definition and the second part is the embedded JavaScript code.

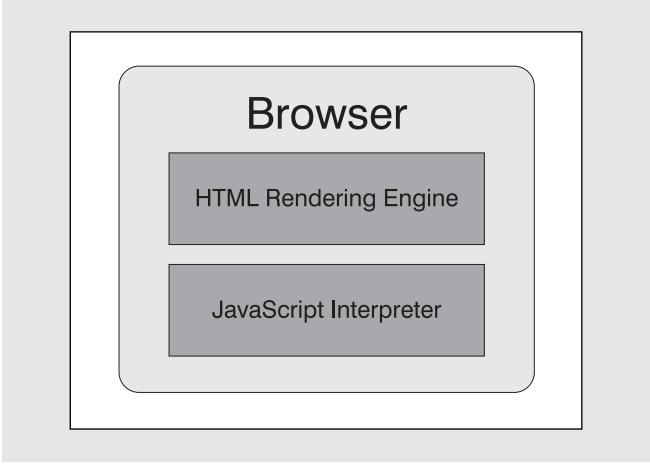

Figure 2 Web browser components for HTML and JavaScript processing

As shown in **Figure 4** on the next page, when you enter an invalid number and click on the Enter button, you get an error message (that is, "Please enter a number!"); whereas, when you enter a correct number, you get a confirmation of the number (that is, "You entered 3!"). The confirmation is created by JavaScript code.

In the PDF scenario, the server forwards the request to the BSP application. Then (through the help of my PDF Toolbox for ABAP solution, which has been loaded on the client) the application generates a PDF response (instead of HTML code), which is returned to the browser by the server. Just replace HTML with PDF in **Figure 1** and you have the diagram for the PDF scenario.

The main difference between the HTML scenario and the PDF scenario is the handling of the response by the browser. Because the browser's HTML rendering engine can't handle PDF data, the browser

```
<!-- the form definition-->
<form name="demo"  onsubmit="return checkForNumber(this.test);" >
  <p>Enter a number: <input type="text" id="test" name="test" /></p>
  <p><input type="submit" value="Enter" /></p>
</form>

<!-- the javascript code -->
<script type="text/javascript" language="javascript">

  // function definition
  function checkForNumber(number) {
    // validate input value
    if ( parseInt(number.value) != number.value ) {
      alert('Please enter a number!');
      return false;
    }
    // check the number and create the response
    document.write( "You entered " + number.value + " !");
    return true;
 }
</script>
```

Figure 3 An HTML and JavaScript code snippet

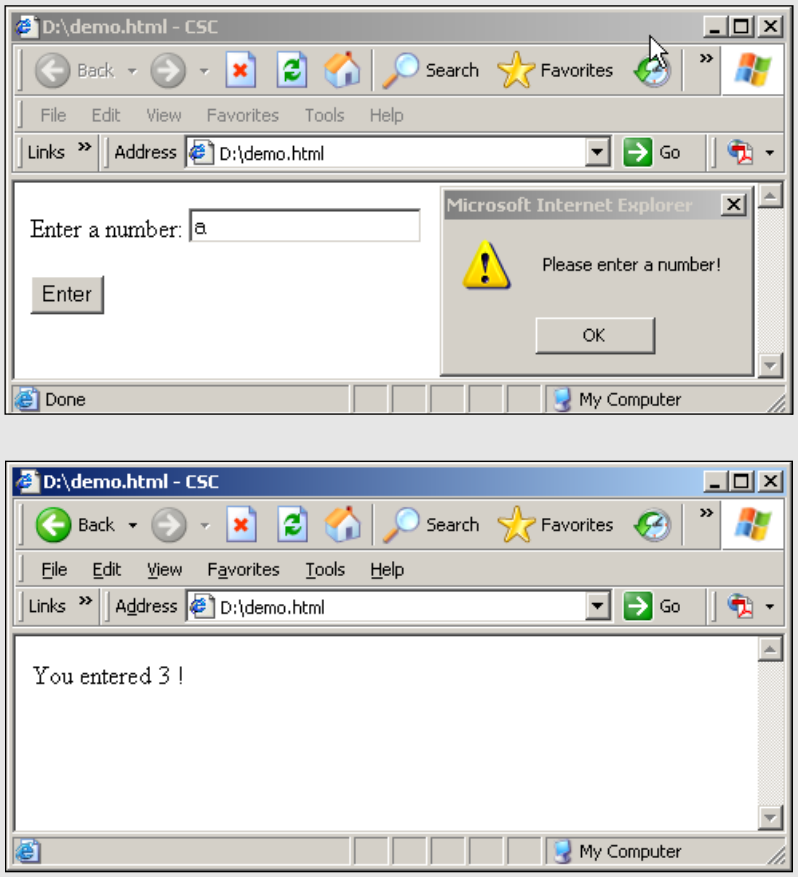

Figure 4 The HTML/JavaScript in a Web browser

loads the Adobe Acrobat Browser Plugin and then delegates the rendering of the response and the handling of the user interaction with the PDF form to the plugin. **Figure 5** shows the browser components involved in the PDF scenario. The plugin handles the rendering of the response; whereas, the toolbox handles the creation of the PDF response.

As you may know, you can improve an HTML page using JavaScript, but what you might not know is that there is a scripting language for PDF forms, JavaScript for Acrobat. See the sidebar on the next page for more information about JavaScript for Acrobat.

PDF forms can have the same input fields as HTML forms, and they can have user interface (UI) elements, such as a submit button. After the PDF form has been loaded into the Acrobat Browser Plugin, the

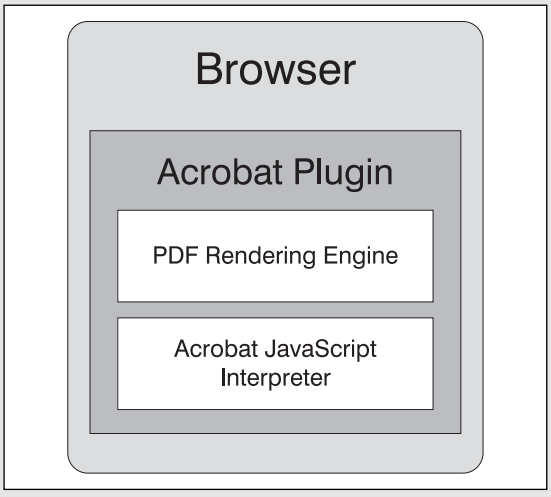

Figure 5 Web browser components for PDF processing

JavaScript for Acrobat

Acrobat JavaScript is an object-based scripting language based on JavaScript 1.5, which was originally developed by Netscape Communications. In addition to the JavaScript standard objects, such as Math, String, Date, Array, and RegExp, Adobe implemented numerous Acrobat-specific objects and methods that allow you to read form field data and to enhance the Acrobat user interface. You can even add dialogs and a toolbar to a PDF document. For more information, go to www.adobe.com and search for "JavaScript for Acrobat."

Note!

The Acrobat Browser Plugin is automatically installed on your computer when you install Adobe Acrobat Reader. I have tested the plugin successfully on Windows XP-based computers using Microsoft Internet Explorer and on Linux using Mozilla Firefox 1.5.

Many more advanced features of JavaScript were introduced with Adobe Reader 7, which is the version I used to create my PDF solution. To use the PDF solution described here, you need at least version 7 installed on your computer.

plugin controls the user interaction until the user clicks on the submit button on the PDF form. When the user clicks on the submit button, the plugin creates a response that is exactly the same as if the user clicks on the submit button on an HTML form. From a technical point of view, this is very helpful because this means that you can replace an HTML form with a PDF form without changing the server-side application, which handles the response of the form and that you can use the same technology (in our case BSP) to handle the browser response from a PDF form.

Creating the PDF response

The solution introduced in this article creates the PDF response using a template-driven approach. This means that the PDF response is created at runtime by merging an existing PDF form (that is the template) with dynamic data. This modified PDF form is sent as the response to the user. Using this approach, the integration of a PDF form in a BSP application consists of two parts: The first part is the design of the PDF template and the second part is the development of the business logic that modifies the PDF template.

You can create the layout of the PDF template without any programming knowledge using a program such as Microsoft Word. You convert the Word document into a PDF file using the Acrobat Professional. Then, using the form tools provided by Acrobat Professional, you add the form fields, which act as placeholders for dynamic text and as input fields, within the PDF.

The business logic that modifies the PDF form is typically implemented in the application class of the BSP application in which the PDF form is integrated. The actual modification of the PDF forms is done using the PDF Toolbox for ABAP. With that said, let's look at an example in which I integrate a PDF form into a BSP application.

Integrating a PDF form as the front end of a BSP application

The example we'll look at shows a simple hotel reservation PDF form as the front end of a BSP application. The starting point of the application is a simple HTML page with a link to the hotel reservation form, as shown in **Figure 6** on the next page. This HTML page is the starting page of my BSP application.

Figure 6 HTML page with a link to a PDF form

When a user clicks on the link for the hotel reservation form, a PDF form appears inside the browser, as shown in **Figure 7**. I created the PDF form using Word and Acrobat Professional.[3]

In the upper part of the form, the user enters personal data and the arrival and departure dates. The next part of the form has six radio buttons that the user clicks on to select the price category for either a single or a double room. The actual price information is not hard-coded in the form. It is inserted before the form is sent to the browser, so you can change the price ranges without changing the PDF form.

Below the price information is a list box from which to select a smoking or non-smoking room. The options available in this list box are also filled at runtime, so you can easily change them without changing your PDF form. The reservation date and reservation number fields, which are filled by the hotel after the reservation application has been completed, are at the bottom of the form.

If you want the user to interact with a PDF form inside a browser, you need to supply a UI element. A simple solution is to add the UI element directly on the form. For example, you might add a submit button to the PDF form so the user can submit the data entered in the form by simply clicking on this button. However, there is a disadvantage to this approach. Because the user may scroll up and down or zoom in and out on a PDF form, the submit button

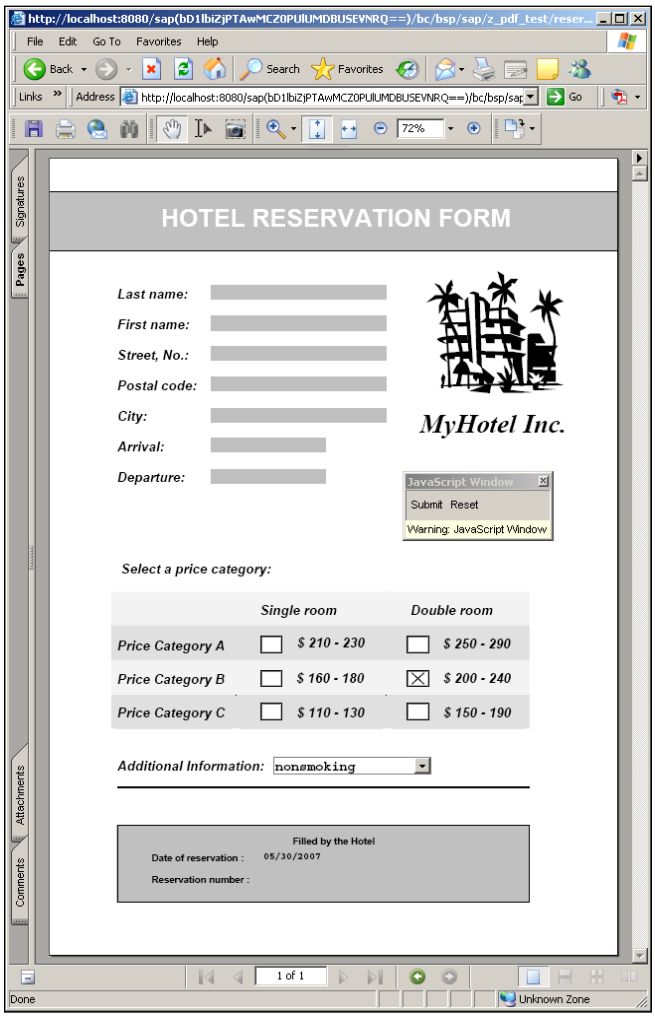

Figure 7 A PDF form for a hotel reservation inside a browser

[3] I give some tips on creating a PDF form later in the article.

may not always be visible. To address this problem, I suggest adding a toolbar for submitting the data from your PDF form. Every PDF file can have a single toolbar created by JavaScript for Acrobat.

You create this toolbar using my PDF Toolbox for ABAP. The toolbox adds the Acrobat JavaScript, which is needed to create a toolbar in the PDF form. When the PDF form is opened on the client side, the Acrobat Browser Plugin executes the Acrobat JavaScript code and adds the toolbar on the fly.

There are two buttons[4] on this toolbar. Because the toolbar is not a standard UI element, there is always a warning on it that it was created by JavaScript for Acrobat. The first button submits the form data to the server. The second button resets the default values in the form.

After the user clicks on the Submit button on the toolbar, the form data (the form field names and the form field values) is sent to the server. The BSP application fills in the reservation date, adds the reservation number, and adds the text "Accepted" to the form, as shown in **Figure 8**. The user can then save the form locally for future reference by clicking on the save icon of the Acrobat Browser Plugin. The user clicks on the second button to reset the form fields to the user's default values.

Creating a PDF template

In this part of the article I step you through the process of creating the hotel reservation form. This part is not intended to be an in-depth discussion of this topic, but it should give you a feeling of how you create a PDF form. The tools I used for this purpose are Word and Adobe Acrobat Professional. However, there are also excellent open-source tools, such as OpenOffice 2.0 and Scribus 1.3.3.x, that allow you to create PDF forms without Acrobat Professional.

Creating a PDF form is a four-step process:

1. Create the layout using Word or other word processing software.

[4] You can add other buttons to suit your needs.

> ***Tip!***
>
> There is a tryout version of Acrobat Professional available that works for a 30-day trial period, so you don't have to buy anything for your initial testing of your PDF forms. The source code of the RFC server, a compiled version of the RFC server, and the source code of the ABAP classes are available for download at www.SAPpro.com.

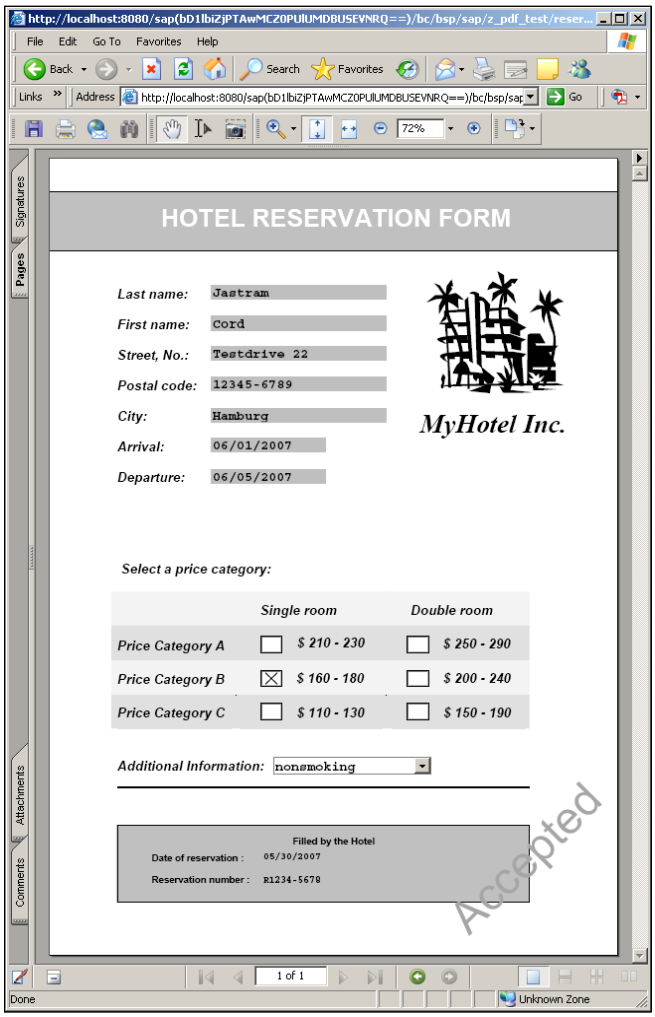

Figure 8 The PDF form after it has been successfully submitted to the BSP application

2. Convert the layout document to a PDF form.
3. Add form fields using Acrobat Professional.
4. Set up the format properties of the fields and implement client-side validations.

Step 1: Create the layout using Word or other word processing software

To create a PDF form, start by laying it out in Word. For the example, I created the layout using standard Word formatting tools. As I expect most developers are familiar with those tools, I won't go into the details of creating the form layout. Remember, however, that for a good user experience, you must provide a well-organized, easily comprehensible layout.

Step 2: Convert the layout document to a PDF form

During the installation of Acrobat Professional, PDF-related menu items are added to the Word menu bar

Tip!

When you intend to use a PDF form on the Internet, pay attention to the size of the form. A PDF form displayed over the Internet should be minimally sized to reduce the time it takes to load in a user's browser, especially when you can't be sure that all users have a high-speed connection.

To this end, I recommend restricting the number of fonts you use in your forms, especially fonts that have to be embedded, because embedded fonts increase the form's size. Acrobat Professional comes with 14 standard fonts. Restricting the number of fonts in the form and using only the standard fonts keeps your form's size to a minimum and results in a more professional-looking form.

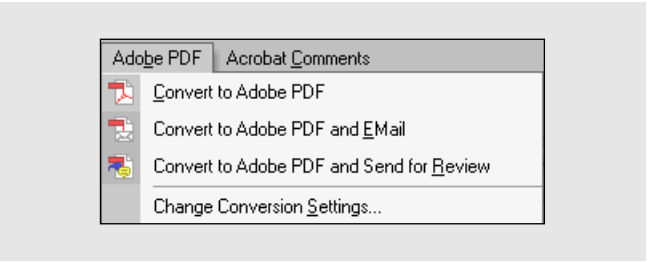

Figure 9 Acrobat PDF-related menu and commands added to the Word menu bar

(see **Figure 9**). These menu items allow for an easy conversion of the Word document to a PDF document. After you have laid out your form in Word, select the Convert to Adobe PDF on the Adobe PDF menu. After a few seconds, the new PDF file automatically opens in Acrobat Professional. You can then save the generated PDF file at a convenient location.

Step 3: Add form fields using Acrobat Professional

Before you start to add form fields to a PDF file, you should decide on a concise naming convention for the names of the form fields, which is a best practice whenever you are developing forms to maintain consistency. The simple naming convention I use can be summarized as follows:

- Use uppercase letters for all field names (e.g., CITY, FIRST_NAME, etc.). Form field names are case sensitive, and therefore using only uppercase letters minimizes potential errors. Note that you can use special characters such as an underscore in a field name.
- Use a prefix in a field name to indicate to which group the field belongs. Separate the prefix from the field name using a period (e.g., USER.FAMILY_NAME). In some cases, you'll need more than one prefix (e.g., TEXT.PRICE_CATEGORY.DRA). I'll explain the reason shortly.

For the hotel reservation form, I used the prefix USER for fields that are filled by the user, HOTEL for fields that are filled by the hotel personnel, and TEXT for fields that are just used as placeholders for

Field name	Type
USER.LAST_NAME	Text field
USER.FIRST_NAME	Text field
USER.STREET_NO	Text field
USER.POSTAL_CODE	Text field
USER.CITY	Text field
USER.ARRIVAL_DATE	Text field
USER.DEPARTURE_DATE	Text field
USER.PRICE_CATEGORY	Radio button
USER.ADDITIONAL_INFO	Combo box
HOTEL.RESERVATION_DATE	Text field
HOTEL.RESERVATION_NUMBER	Text field

Figure 10 PDF fields used for capturing input data from the user and from the hotel personnel

Field name	Type
TEXT.PRICE_CATEGORY.DRA	Text field
TEXT.PRICE_CATEGORY.DRB	Text field
TEXT.PRICE_CATEGORY.DRC	Text field
TEXT.PRICE_CATEGORY.SRA	Text field
TEXT.PRICE_CATEGORY.SRB	Text field
TEXT.PRICE_CATEGORY.SRC	Text field

Figure 11 Field names of placeholder fields

dynamic text. An example of a TEXT field is the six price ranges of the room categories. They should be filled at runtime because the price ranges may vary, but they should not be editable once they have been filled. To achieve this static state, these fields are converted to text during the server-side processing of the PDF form. This process is called "form field flattening." See **Figure 10** for the USER and the HOTEL fields of the form.

For the TEXT group shown in **Figure 11**, I used a a second prefix PRICE_CATEGORY and the actual field name consists of three characters. The first two characters indicate the room size (DR for double room and SR for single room) and a third character (A, B, or C) for the price category (i.e., the higher the price, the more amenities available).

Now it is time to actually add the form fields. Let's start with a text field for USER.LAST_NAME. From the Acrobat Tools menu, select Advanced Editing → Forms → Text Field Tool, as shown in **Figure 12**.

Using your pointer, define the text field area on the PDF form page, positioning and aligning the fields as best you can. After you define the area, a properties dialog box appears. On the General tab, enter the name of the field (for example, USER.LAST_NAME)

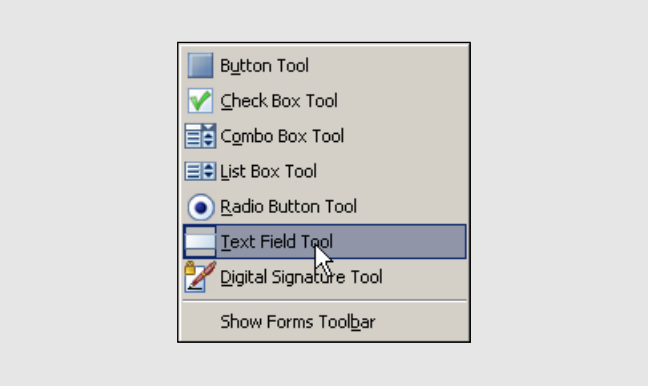

Figure 12 Selecting the Text Field Tool from Acrobat

and the tooltip text (e.g., "Enter your last name!"). Mark this field as required. When you mark a field as required, an error message appears if the field is empty when you click on the submit button. See **Figure 13**.

Switch to the Appearance tab, and then select gray as the fill color, a font size of 14 pt, and Courier Bold as the font style.

You continue to use the Text Field Tool until you have added all fields. Use the General tab and Appearance tab to set the properties for the fields. When you have added all fields to the form, it should look like the one shown in **Figure 14**.

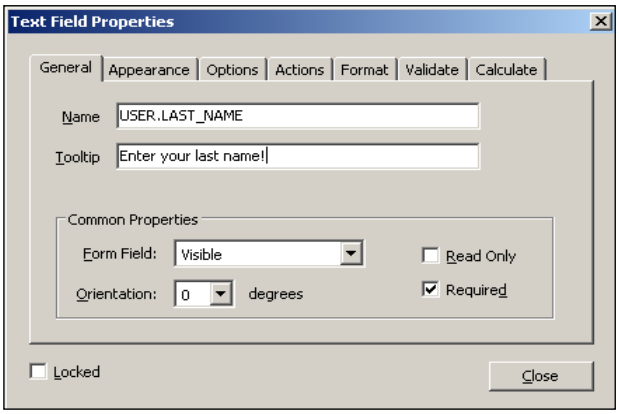

Figure 13 Specifying the name, tooltip text, and common properties of the field

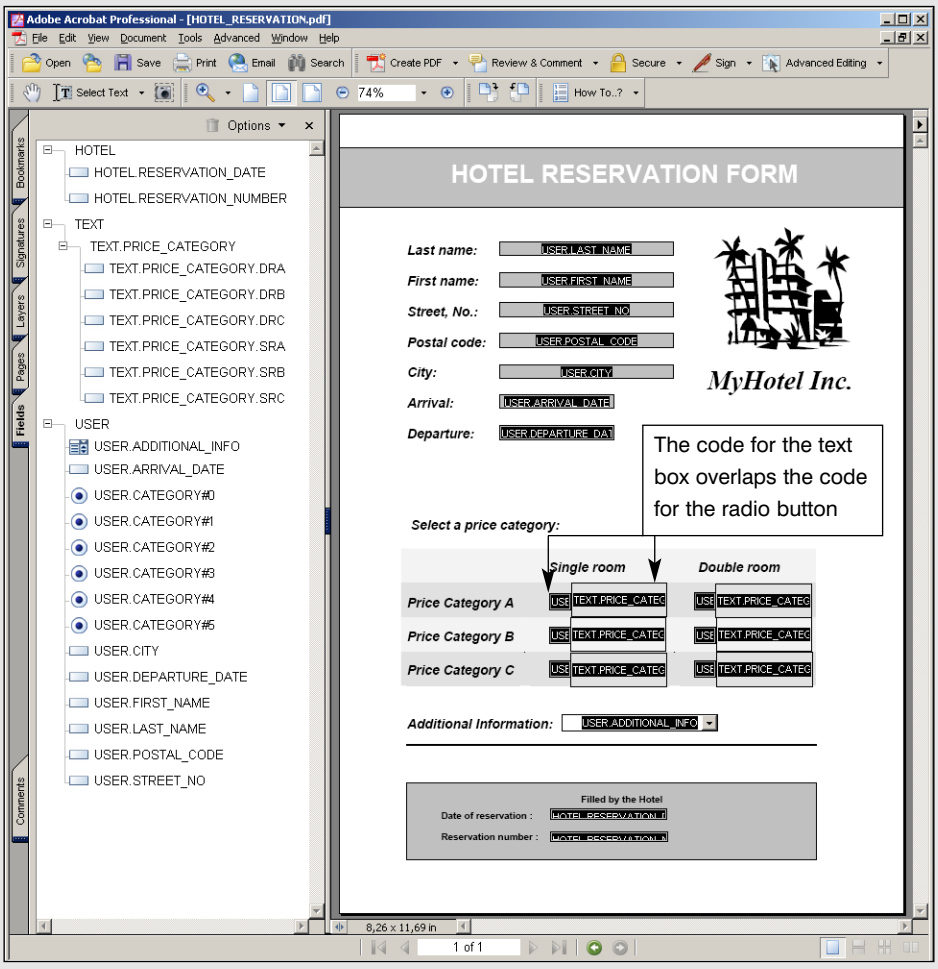

Figure 14 The PDF form in Acrobat Professional

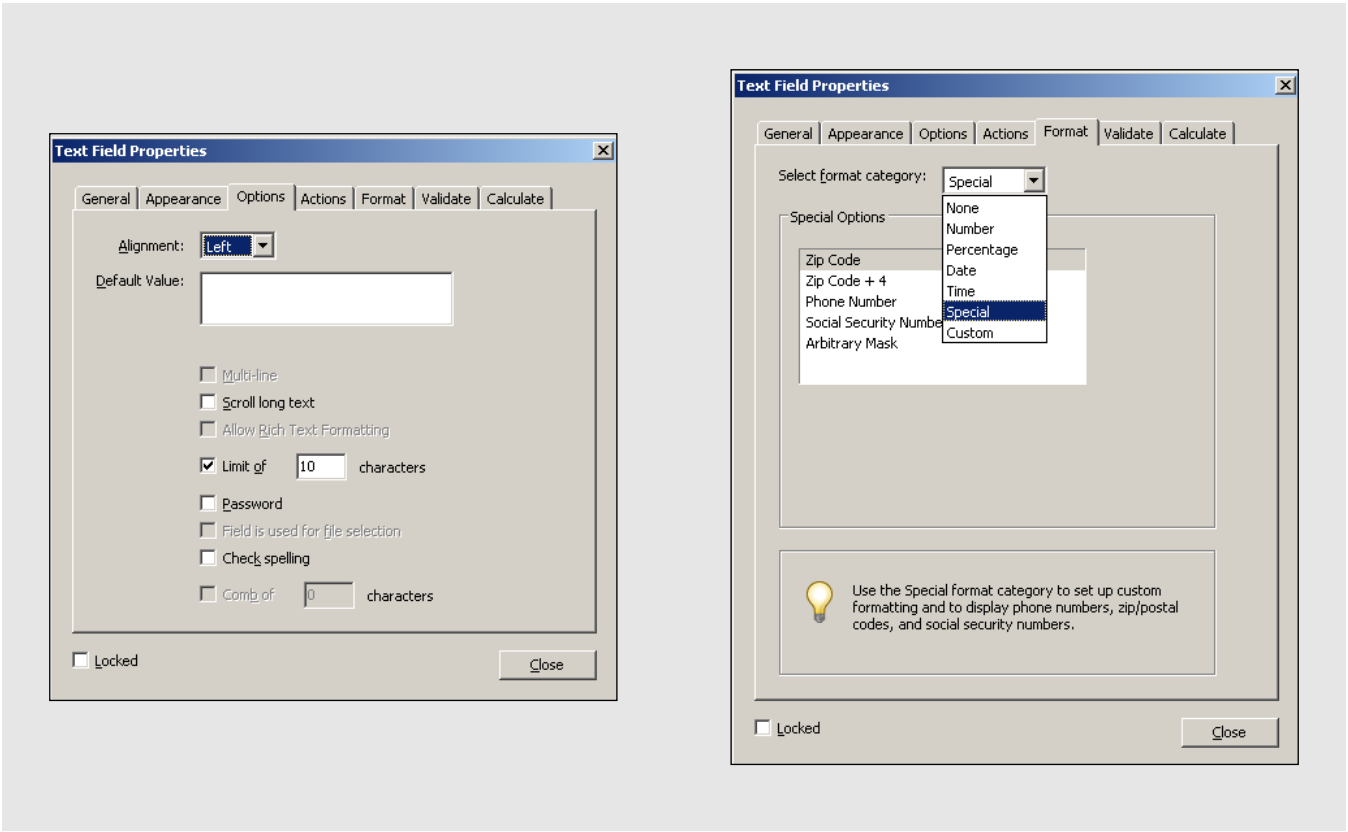

Figure 15 The Options and Format tabs of the Text Field Properties dialog

Step 4: Set up the format properties of the fields and implement client-side validations

One of the assets of PDF forms is that you can easily set up client-side validations, which are executed by the Acrobat Browser Plugin, without sending a request to the server. You find the corresponding options on the Option and Format tabs of the Text Field Properties dialog. It is always a good idea to limit the number of characters that can be entered into a form field. You use the Options tab to specify the number of characters that a field allows (e.g., a state field might allow just two characters). The Format tab offers several format categories that you can use to further control the input data (e.g., telephone numbers and Social Security numbers). See **Figure 15**. For a detailed discussion of the options, I recommend the online help of Acrobat Professional.

The last section in this article gives you a glimpse at the code I have developed for implementing the BSP application with the PDF form. But before looking at the code, I want to introduce to you some new capabilities of the PDF Toolbox that you can use to manipulate the PDF forms you create.

Implementing the BSP application

I developed the PDF Toolbox to help ABAP developers to programmatically modify existing PDF documents. But I have added Java-based features to the toolbox that you can use in your ABAP coding as additional methods in the ABAP class ZCL_PDF_COMMAND_LIST. **Figure 16** (on the next page) shows the methods and the parameters I developed. I give some examples of how you use these methods later.

Method ADD_SUBMIT_BUTTON PROCESS	
Importing parameter LABEL	Type ZPARAM_VALUE
Importing parameter TOOLTIP	Type ZPARAM_VALUE
Importing parameter URL_PARAMETER	Type ZPARAM_VALUE
Method ADD_RESET_BUTTON.ATTACHMENT	
Importing parameter LABEL	Type ZPARAM_VALUE
Importing parameter TOOLTIP	Type ZPARAM_VALUE
Method SET_FIELD_VALUE	
Importing parameter FIELD	Type ZPARAM_VALUE
Importing parameter VALUE	Type ZPARAM_VALUE
Method SET_FIELD_READONLY	
Importing parameter FIELD	ZPARAM_VALUE
Importing parameter STATUS	C default "X"
Method SET_LIST_OPTION	
Importing parameter FIELD	Type ZPARAM_VALUE
Importing parameter DISPLAY_VALUES	Type ZPARAM_VALUE
Importing parameter EXPORT_VALUES	Type ZPARAM_VALUE
Method PARTIAL_FLATTEN	
Importing parameter FIELD	Type ZPARAM_VALUE
Method ADD_FIELD_INFO	
Importing parameter FIELD	Type ZPARAM_VALUE
Importing parameter SHORT_INFO	Type ZPARAM_VALUE
Importing parameter INFO	Type ZPARAM_VALUE
Importing parameter MESSAGE	Type ZPARAM_VALUE
Method SIGN	
No parameter	

Figure 16 Methods of the ZCL_PDF_COMMAND_LIST class

You use the methods ADD_SUBMIT_BUTTON and ADD_RESET_BUTTON to add a submit button or a reset button to the JavaScript toolbar of a PDF document. Both methods take the LABEL and the TOOLTIP text as a parameter. For the ADD_SUBMIT_BUTTON method you can add a text to the URL of the button. You use the method SET_FIELD_VALUE to set the value of a PDF form field. When you want to prevent a user from editing a form field, you use the method SET_FIELD_READONLY to which you supply the name of the field you want to set to read-only. If you want to set a read-only field to editable, you leave the optional parameter STATUS blank or empty. You use this feature for simple workflows. For example, users should not be able to fill fields that require approval. When the person who is authorized to approve values opens the form, the approval fields should be set to

editable, but the fields entered earlier by the users should be read-only.

Using the method SET_LIST_OPTION, you provide a list of options for a combo or list box field from which you can select a single value. The parameter DISPLAY_VALUES is a table of type ZPARAM_VALUE and holds the text elements shown in the list or combo box. The parameter EXPORT_VALUES is a table with the corresponding keys of the entries.

As mentioned earlier, you can use form fields as a placeholder for dynamic text. Using the method PARTIAL_FLATTEN, you convert a form field into simple text (for example, when you convert the price range for the different room categories into plain text).

You use the method ADD_FIELD_INFO to add additional information to a form field. This kind of information is to be used for server-side field validations. I will show an example of ADD_FIELD_INFO later in this article. You can use the method SIGN if you want an electronic signature added to the PDF form. The signature can be used to check whether the PDF form has been changed after it has been created. However, a discussion of the handling of electronic signatures is out of the scope of this article.

Creating a BSP application

Now let's take a quick look at the BSP application, named Z_PDF_TEST, I wrote for the purpose of this article. **Figure 17** shows the application in the Web Application Builder using transaction SE80.

As you can see, the application consists of two BSP pages — default.htm and reservation.htm. Z_PDF_TEST is a stateless BSP application with the application class ZCL_PDF_APPLICATION. (We'll take a closer look at this class later in the article.)

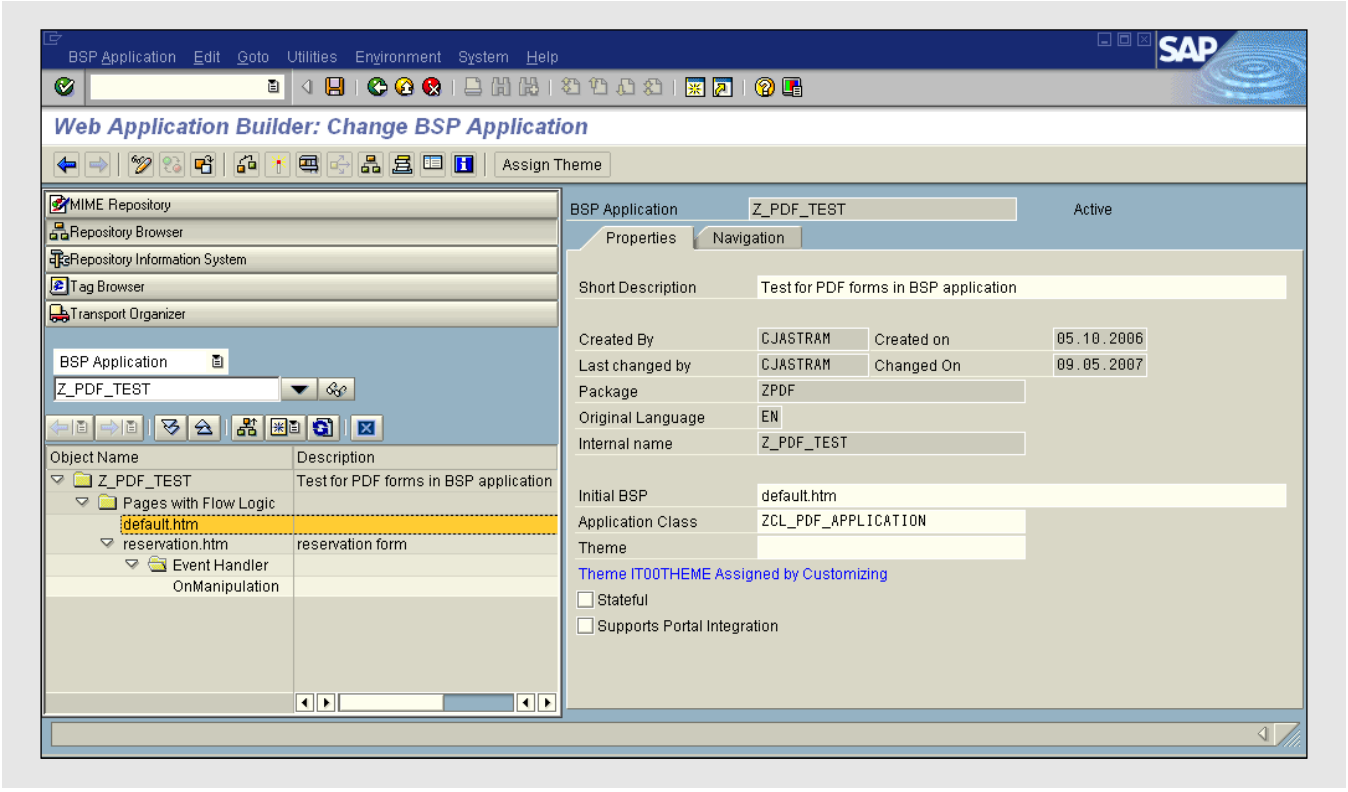

Figure 17 The BSP Application Z_PDF_TEST shown by transaction SE80

```
<%@page language="abap" %>
<%@extension name="htmlb" prefix="htmlb" %>
<htmlb:content design="design2003" >
  <htmlb:page>
    <br>
    <htmlb:gridLayout id          = "myGridLayout1"
                      width       = "60%"
                      cellSpacing = "40"
                      columnSize  = "1"
                      rowSize     = "1" >
      <htmlb:gridLayoutCell rowIndex            = "1"
                            columnIndex         = "1"
                            style               = "ALTERNATING"
                            horizontalAlignment = "center" >
        <htmlb:link id        = "pdf"
                    reference = "reservation.htm" >
          Open the HOTEL RESERVATION form
        </htmlb:link>
      </htmlb:gridLayoutCell>
    </htmlb:gridLayout>
  </htmlb:page>
</htmlb:content>
```

Figure 18 Code for the BSP default.htm page

> ***Note!***
>
> The ABAP coding shown in this article has only been written for demonstration purposes. To improve the readability of the source code, I have left out most of the error handling code and used hard-coded text values.

The initial BSP default.htm file

The BSP default.htm file is used to create the starting HTML page (**Figure 4**). From the page layout shown in **Figure 18**, you see that it simply consists of a link to the reservation.htm page.

The BSP reservation.htm file

The BSP that displays the PDF form in the browser is named reservation.htm. The layout of this page is shown in **Figure 19**.

Note this page just shows the error message, "AN INTERNAL ERROR OCCURRED!" The reason for this will become evident in the next part.

Generating the PDF response

Each BSP has several associated event handlers that are called at different stages of the processing of the page. For the PDF integration, I use the event handler OnManipulation, which allows the manipulation of the HTTP data stream that is sent to the browser because a response has been created. The request is called after the page response has been created. Therefore, in the


```
<%@page language="abap" %>
<%@extension name="htmlb" prefix="htmlb" %>
<htmlb:content design="CLASSIC" >
  <htmlb:page title="BSP ERROR" >
    <htmlb:textView design="HEADER1" >
      <br> AN INTERNAL ERROR OCCURED!
    </htmlb:textView>
  </htmlb:page>
</htmlb:content>
```

Figure 19 Code for the BSP reservation.htm page

Note!

This part is intended for people who are familiar with the development of a BSP application, so I do not show all details. For a good introduction to BSP application development, see the following SAP Professional Journal articles:

- "Developing Custom Applications for SAP Enterprise Portal — Starting with the 'Right' Options in Light of SAP NetWeaver" by Patrick Dixon (March/April 2005)
- "Develop More Extensible and Maintainable Web Applications with the Model-View-Controller (MVC) Design Pattern" by Ken Huang and Markus Wieser (January/February 2004)
- "Build More Powerful Web Applications in Less Time with BSP Extensions and the MVC Model" by Karl Kessler (March/April 2003)
- "A Developer's Guide to Creating Powerful and Flexible Web Applications with the New Web Application Builder" by Karl Kessler (January/February 2002)

OnManipulation event handler, I can create a response of my own — the PDF form. When the creation of the PDF form fails, I don't modify the response and the error message created by the BSP reservation.htm is shown. If the PDF form has been successfully created, I use the PDF form as the response.

Now let's have a look at the coding of the OnManipulation event handler coding, as shown in **Figure 20** on the next page.

At first I check whether the request, which calls the page, is a GET or a POST request (#1). The GET request type corresponds to the initial loading of the PDF form, whereas the POST request type corresponds to submitting the form data (that is, the user has clicked on the submit button on the PDF Forms toolbar). Depending on the request type, I call different methods (#2 and #3) of the application class. For the GET request I call handle_get_reservation, and for the POST I call handle_post_reservation. These methods return either the PDF form shown in **Figure 7** (GET) or **Figure 9** (POST).

If an error occurs during the generation of the PDF

```
* event handler to manipulate dynamically the HTTP stream

DATA : request_type TYPE string.

*#1 get the request type
request_type = request->get_header_field(
      name = '~request_method' ).

*#2
IF request_type = 'GET'.
  application->handle_get_reservation( ).
ENDIF.

*#3
IF request_type = 'POST'.
  application->handle_post_reservation( request ).
ENDIF.

IF application->error_flag = 'X'.
*#4- do nothing
ELSE.
*#5
  response->set_header_field(
    name  = if_http_header_fields=>content_type
    value = 'application/pdf' ).

  response->server_cache_expire_rel( expires_rel = -1 ).

  response->set_header_field(
   name  = 'content-disposition'
   value = 'inline').

*#6
  response->set_cdata( application->response ).
ENDIF.
```

Figure 20 ABAP coding of the OnManipulation event handler of the BSP page reservation.htm

form, both methods set the attribute ERROR_FLAG of the application class ZCL_PDF_APPLICATION to "X." I just have to check this flag (#5). If an error occurred, I simply do nothing (#4) and the message defined in the layout part of the BSP page is sent to the browser. Otherwise, I modify some of the header fields of the response to change the response content type to PDF (#5). This step is very important because from the content type the browser decides whether the Acrobat Browser Plugin (mentioned in the introduc-

tion of the article) should be loaded or not. As a last step, the response of the request is set to the PDF data created earlier (#6).

The coding of the OnManipulation event handler will be similar for all BSP pages that generate PDF output, so you can use it as a template for your own coding.

Implementing the application class

Although it is not mandatory, each BSP application should have an associated application class. Then in each event handler you can automatically access an object with the name "application" with the type of the associated application class.

For the Z_PDF_DEMO application, I have written a class ZCL_PDF_APPLICATION with the superclass ZCL_BSP_APPLICATION. This class has three private attributes shown in **Figure 21**.

This class has two public methods named HANDLE_GET_RESERVATION and HANDLE_POST_RESERVATION and a set of private methods that you use to implement the public methods. The listing of HANDLE_GET_RESERVATION is shown in **Figure 22**.

Attribute	Description	Type
PDF_FILE	The PDF template file data	ZCL_FILE
COMMAND_LIST	The command list to modify the PDF template data	Z_PDF_COMMAND_LIST
ERROR_FLAG	A flag indicating that an error has occurred during the processing of the command list	C

Figure 21 Private attributes of class ZCL_PDF_APPLICATION

```
METHOD handle_get_reservation .

  DATA : result_file TYPE REF TO zcl_file,
         lreturn TYPE bapireturn.
*#1
  me->load_pdf_template( ).

*#2
  me->command_list->add_submit_button( ).
  me->command_list->add_reset_button( ).

*#3
  me->fill_hotel_data( ).

*#4
  me->flatten_fields( ).

*#5
```

Continues on next page

Figure 22 Code for the HANDLE_GET_RESERVATION method

Figure 22 (continued)

```
  me->fill_combo_box( ).

*#6
  CREATE OBJECT result_file.

  CALL METHOD command_list->process
    EXPORTING
      file_1      = me->pdf_file
    IMPORTING
      return      = lreturn
      result_file = result_file.

*#7
  IF lreturn-type = 'S'.
    response_string = result_file->get_string( ).
  ENDIF.

ENDMETHOD.
```

```
METHOD check_dates .

  data : arrival type string,
         departure type string,
         temp type zparam_value.
```

Continues on next page

Figure 23 Coding for the HANDLE_POST_RESERVATION method

At first (#1) the PDF template is loaded. Then I add the submit and the reset toolbar buttons (#2). Next, I modify the form fields of the PDF form: I fill the hotel form fields (#3), handle the placeholder form fields for the price ranges (#4), and then fill the option list of the combo box for the additional information (#5). For the combo box, I set the field values (#5) and then convert them to simple text (#6). Then I set the list options of the field USER.ADDITIONAL_INFO to smoking and nonsmoking (#7). At the end the PDF template (#6) is processed, and depending on the success of this operation, I return the created PDF form or an empty string (#7). As a result, you see a PDF form in your browser (see **Figure 7**).

Now let's have a look at some snippets of the coding of the HANDLE_POST_RESERVATION method. The most interesting part of this method is the implementation of a server-side validation in the method CHECK_DATES. As an example, I check whether the departure date is before the arrival date. To do this, I get the values entered in the form from the request (#1). Then I check both dates and, if necessary, I add field information to both fields and set the field VALIDATION_FAILED_FLAG to "X" (#2), as shown in **Figure 23**.

There are two ADD_FIELD_INFO methods — one for the arrival field and one for the departure

Figure 23 (continued)

```
*#1
  arrival = request->get_form_field_cs(
      name = 'USER.ARRIVAL_DATE' ).

  departure = request->get_form_field_cs(
     name = 'USER.DEPARTURE_DATE' ).

  temp(4) = arrival+6(4).
  temp+4(2) = arrival(2).
  temp+6(2) = arrival+3(2).

  arrival = temp.

  temp(4) = departure+6(4).
  temp+4(2) = departure(2).
  temp+6(2) = departure+3(2).

  departure = temp.

  IF departure < arrival.

    DATA : shortinfo TYPE zparam_value,
           info TYPE zparam_value,
           message type zparam_value.

    shortinfo = 'Arrival date is after departure date!'.
    info = 'Please change the arrival or the departure date!'.
    message =
      'Some form fields have invalid entries.\nPlease check them!'.

    me->command_list->add_field_info( field = 'USER.ARRIVAL_DATE'
                                  shortinfo  = shortinfo
                                  info = info
                                  message = message ).

    me->command_list->add_field_info( field = 'USER.DEPARTURE_DATE'
                                  shortinfo  = shortinfo
                                  info = info
                                  message = message ).
*#2
    me->validation_failed_flag = 'X'.
  ENDIF.

ENDMETHOD.
```

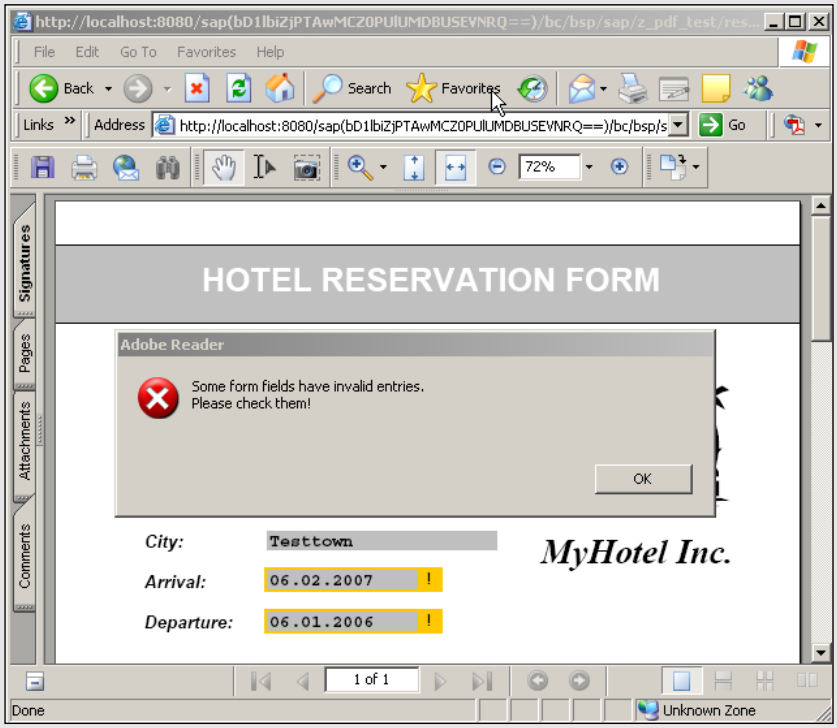

Figure 24 Initial error message for server-side validations

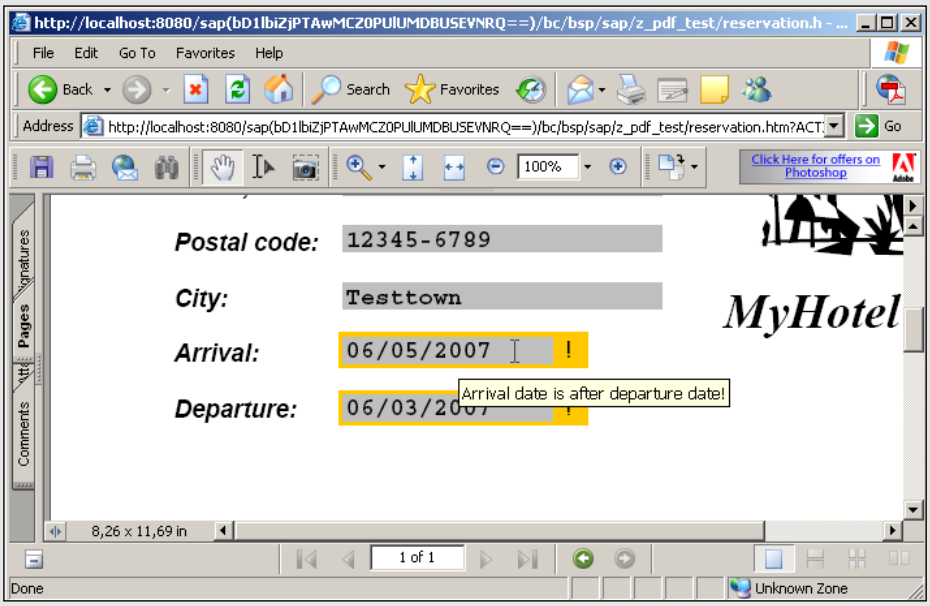

Figure 25 Tooltip indicates the date entered in the field is invalid

field. If an invalid value is entered in either of these fields, the method displays a dialog, as shown in **Figure 24**. The method ensures that the dialog displays only once even if it is called several times. The parameter MESSAGE of the method provides the dialog text. As shown in **Figure 25**, the method adds a yellow frame around the field and provides a tooltip that describes the field (when the pointer is positioned over the field). It also adds a button on the right side of the field. Clicking on this button displays detailed field-related information, as shown in **Figure 26**.

At the end of the method HANDLE_POST_REQUEST, I provide appropriate information or instructions depending on the VALIDATION_FAILED_FLAG (see **Figure 27**).

If no error occurs, I set the reservation number, add the Accepted text on the form, and sign it electronically. Otherwise, I add a submit button and a reset button. The user can then change the arrival and/or the departure date and submit the form again.[5]

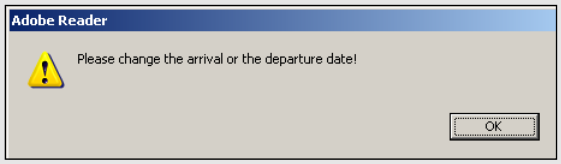

Figure 26 Popup with instructions for correcting the error in the field

[5] For a description of the deployment of the solution, see the section "Deploying the RFC server" in my *SAP Professional Journal* May/June 2007 article and the installation instructions included in the download at www.sappro.com.

```
  IF validation_failed_flag = space.

*#1
    command_list->set_field_value( field = 'HOTEL.RESERVATION_NUMBER'
                                   value = 'R 1235-5678' ).

    command_list->set_field_readonly( field = 'HOTEL.RESERVATION_NUMBER' ).


    command_list->add_background_text( text = 'Accepted'
                                       fontsize = '48'
                                       gray = '0.65'
                                       ypos = '20'
                                       xpos = '440'
                                       rotation = '45' ).

    command_list->sign( ).
  ELSE.
    command_list->add_submit_button( ).
    command_list->add_reset_button( ).
ENDIF.
```

Figure 27 Code for addressing invalid information entered in fields

Conclusion

In this article, I've shown how you can integrate PDF forms in a BSP application. The solution generates the PDF forms using a template-based approach. A static PDF template is modified at runtime by the BSP application and the newly generated PDF form is sent back as the response. The solution works with any SAP Web AS ABAP 6.10 and higher, and needs only an additional RFC server to modify the PDF files.

For highly variable PDF forms, Interactive Forms are better suited, but when you want to Web-enable only a small number of PDF forms, you should give the solution a try.

