Improve your business processes with quick and easy enhancements to PDF documents:

A toolbox for modifying PDF files using ABAP

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.

Forms (such as invoices, pay slips, etc.) are the lifeline of modern business. If you're like most companies, you generate data-driven forms from your SAP system using SAPscript or SAP Smart Forms. The more adventurous among you may have experience converting SAPscript or SAP Smart Forms into PDF documents (e.g., to send them via email, post them on the Internet, archive them, or print them offline).

But what if you want to quickly modify a PDF document after it has been created without having to go through the cumbersome process of opening Adobe Acrobat and clicking through the various buttons and menu items? Consider the following scenarios:

- You have a PDF document, and now you want to password protect it using the encryption mechanisms available in the PDF format, or you want to add a Microsoft Excel file as an attachment to the PDF file.
- You want to assemble a personalized sales catalog PDF for a customer by combining a dynamically generated cover letter and price list (based on SAP data) with an existing, static product catalog PDF containing product descriptions and photos.
- You need to Web-enable an already existing PDF form in a Business Server Pages (BSP) application to provide the Web user a form with enhanced printing capabilities.

Because ABAP does not provide a library for manipulating PDF files, I knew I had to come up with a solution — a toolset that would offer an ABAP developer a comfortable way to manipulate PDF files.

This article introduces the PDF Toolbox, an open-source solution I developed. The purpose of this toolbox is to allow ABAP developers to programmatically modify existing PDF documents with ease. The toolbox consists of two components. The first is a remote function call (RFC)

server (implemented in Java[1]), which provides the functionality that you can use to make the actual modifications to the PDF files. As of this writing, the 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 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, a user can't change it)
- Concatenating two PDF files
- 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

> ***Note!***
>
> I'll discuss how you can extend the services offered by the server when I discuss the bar-code example later in the article.

The second component consists of two ABAP Objects classes that I wrote to provide an API for interacting with the RFC server. These classes provide an easy-to-use interface that allows ABAP developers to use the server without any knowledge of the Java programming language. You simply write an ABAP report using the two ABAP Objects classes, and you're on your way. 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.

[1] The RFC server I've written is a simple wrapper for an open source Java library called iText, which you can use for modifying existing PDF files from Java. iText offers additional functions (services) beyond those implemented in the toolbox, such as the ability to add a watermark or digital signature. Later in the article, I'll show you how to extend the RFC server to add such functions.

The article is divided into three parts. In the first part, I'll review the design of the PDF Toolbox solution — its components, ABAP Objects classes, and the commands you can send to the RFC server. In the second part, I'll step through an example that uses the PDF Toolbox to modify a PDF. Finally, in the third part, I'll review how to enhance the RFC server, and finally, how to deploy it. As you'll see, none of this is complex, and you'll be able to get the toolbox up and running quickly.

Programming with the PDF Toolbox

While ABAP does not provide a library for manipulating PDF files, one is available for Java — the open source library iText — so I wrote a simple RFC server wrapper to expose the functionality of the iText library to ABAP. You can run the Java RFC server on any computer equipped with a Java 5 Runtime Environment and an SAP Java Connector (JCo).

> ***Note!***
>
> You don't need a J2EE system; you simply need a Java 5 Runtime Environment.

Figure 1 shows the three key components of the solution's architecture:

- Your ABAP report (1), which leverages the two classes to interact with the server
- Two ABAP classes, ZCL_FILE (2a) and ZCL_PDF_COMMAND_LIST (2b), which represent a file and a list of commands to be sent to the RFC server for processing, respectively
- The RFC server (3), which calls the iText library to manipulate the PDF files

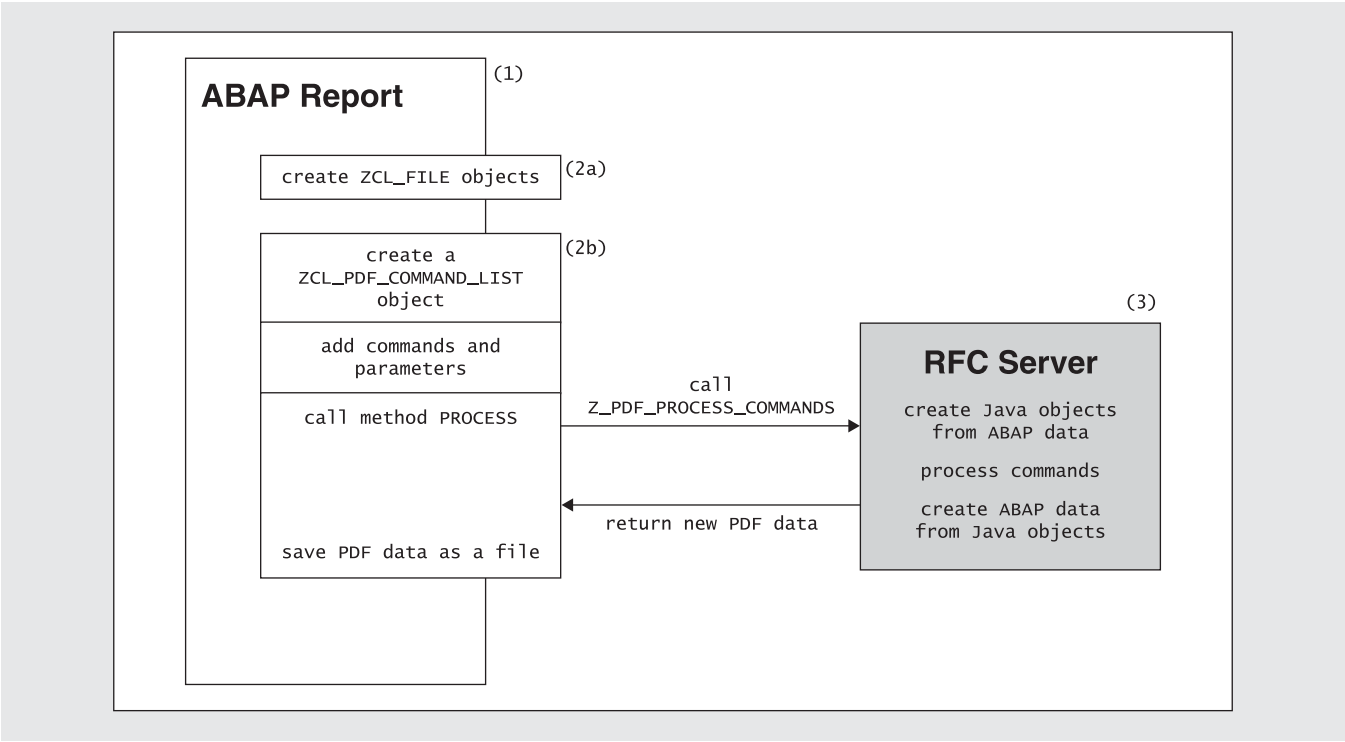

Figure 1 Schematic processing flow of the PDF Toolbox

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.

The flow chart is simple: Your program (the ABAP report) creates one or more instances of the ZCL_FILE class to represent PDF files, and creates an instance of ZCL_PDF_COMMAND_LIST to send 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 (we'll explore which parameters are required for each command shortly). 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 part is shown as a gray background in the figure to indicate that this is done "behind the scenes." Your program is then free to serve the PDF data to the user (e.g., via the Web in a BSP application), save it to a file, or further manipulate the data (e.g., issue another set of commands to combine it with other PDF data).

Let's now quickly review the specific methods of each class, and the available commands you can specify. We'll then put all of this to use through an example. Note that for reasons of clarity and simplicity, the example is intended only to demonstrate some of the methods offered by the toolbox; it is not intended to be a realistic example.

Using the ABAP classes

To use the functionality exposed by the RFC server in your ABAP report, you have to perform the following tasks, which I break down into more detailed steps later when we look at a programming example:

- Create an object of type ZCL_FILE for each file you want to process with the RFC server.
- Create an object of type ZCL_PDF_COMMAND_LIST for storing the commands and their parameters.
- Call the appropriate methods of the command list object that add commands to the command list.
- Call the PROCESS method of the command list object using the ZCL_FILE objects created earlier as the parameters.

Class ZCL_PDF_COMMAND_LIST is used to define the function calls that are executed on the RFC server. This class has been introduced to ease the use of the toolbox. By using this approach, the RFC server does not need to handle any state information — e.g., all data and all processing information are sent to the server using a single function call. Then the processing information works on the data and the resulting file is sent back to the SAP system.

Let's take a closer look at the two ABAP classes that drive the functionality of the toolbox, ZCL_FILE and ZCL_PDF_COMMAND_LIST.

ABAP class ZCL_FILE

Class ZCL_FILE is used to read files from the SAP server and from the client. The public attributes of the class are presented in **Figure 2**.

The size of the file is stored in the SIZE attribute, and the data is stored in the DATA attribute. The PATH attribute stores the path of the file, and the ZIP_ENTRY attribute defines the path and the file-name of the file inside a ZIP archive. Type ZTXROW of attribute DATA is the only user-defined type. It is a table type with the line type ZXROW, which is a user-defined data element with a length of 8192 bytes and the data type RAW.

Attribute	Description	Type
SIZE	The size of the file in bytes	INT4
DATA	The file data	ZTXROW
PATH	The file path	STRING
ZIP_ENTRY	The file path in a ZIP file	TEXT255

Figure 2 Public attributes of class ZCL_FILE

Class ZCL_FILE defines three public static methods, summarized in **Figure 3**, that create the ZCL_FILE instances:

- CREATE_FROM_CLIENT is used to create a ZCL_FILE instance from a client file.
- CREATE_FROM_SERVER is used to create a ZCL_FILE instance from a server file.
- CREATE_PDF_FROM_OTF is used to create a ZCL_FILE instance from OTF data from an SAP Smart Form.

There are also three public instance methods, summarized in **Figure 4**, that handle the file data:

- GET_STRING returns the file data as a string. You use this method for a BSP application to generate the response data that is sent back to the browser.
- SAVE_TO_CLIENT writes the file data to a local file on the client.
- SAVE_TO_SERVER writes the file data to the server.

ABAP class ZCL_PDF_COMMAND_LIST

Class ZCL_PDF_COMMAND_LIST is used to store the command names and the parameters associated with a command. The class has three private attributes, which are listed in **Figure 5**.

The attributes ITEMS and CURRENT_ITEM are used to store the commands and parameters added to a ZCL_PDF_COMMAND_LIST object. The

Method CREATE_FROM_CLIENT	
Importing parameter PATH	Type STRING
Returning parameter VALUE(FILE)	Type REF to ZCL_FILE
Method CREATE_FROM_SERVER	
Importing parameter PATH	Type STRING
Returning parameter VALUE(FILE)	Type REF to ZCL_FILE
Method CREATE_PDF_FROM_OTF	
Importing parameter OTF_DATA	Type TSFOTF
Returning parameter VALUE(PDF_FILE)	Type REF to ZCL_FILE

Figure 3 Public static methods of class ZCL_FILE

Method GET_STRING	
Returning parameter VALUE(RESULT)	Type STRING
Method SAVE_TO_CLIENT	
Importing parameter PATH	Type STRING
Returning parameter VALUE(RESULT)	Type SYSUBRC
Method SAVE_TO_SERVER	
Importing parameter PATH	Type STRING
Returning parameter VALUE(RESULT)	Type SYSUBRC

Figure 4 Public instance methods of class ZCL_FILE

Attribute	Description	Type
ITEMS	Command item table	ZT_COMMAND_ITEM
CURRENT_ITEM	The active command item	ZCOMMAND_ITEM
DESTINATION	The name of the RFC destination of the RFC server	RFCDEST

Figure 5 Private attributes of class ZCL_PDF_COMMAND_LIST

DESTINATION attribute is used for calling the RFC server. The default value for this attribute is PDF_SERVER, which the constructor can set. The attributes do not appear in the public part of the class, so you don't have to worry about them as long as you use the class as is.

The ZT_COMMAND_ITEM type of attribute ITEMS is a table type with the line type ZCOMMAND_ITEM, and the RFCDEST type of attribute DESTINATION is defined by the system. **Figure 6** on the next page shows the structure of the ZCOMMAND_ITEM type of attribute CURRENT_ITEM. Component types ZCOMMAND_NAME, ZPARAM_NAME, and ZPARAM_VALUE are of data type CHAR with lengths of 32, 32, and 255, respectively.

Component	Component type	Data type	Length
ITEM_NO	INT4	INT4	4
COMMAND	ZCOMMAND_NAME	CHAR	32
PARAMETER_NAME	ZPARAM_NAME	CHAR	32
PARAMETER_VALUE	ZPARAM_VALUE	CHAR	255

Figure 6 Structure of ZCOMMAND_ITEM

Method ADD_COMMAND	
Importing parameter COMMAND	Type ZCOMMAND_NAME
Method ADD_PARAMETER	
Importing parameter PARAMETER	Type ZPARAM_NAME
Importing parameter VALUE	Type ZPARAM_VALUE
Importing parameter INDEX	Type INT4 (default "0")
Method ADD_INT_PARAMETER	
Importing parameter PARAMETER	Type ZPARAM_NAME
Importing parameter VALUE	Type INT4
Method ADD_FLOAT_PARAMETER	
Importing parameter PARAMETER	Type ZPARAM_NAME
Importing parameter VALUE	Type ZFLOAT

Figure 7 Private instance methods of class ZCL_PDF_COMMAND_LIST

While there are no static methods, class ZCL_PDF_COMMAND_LIST defines four private instance methods, summarized in **Figure 7**, which are low-level methods used to implement the high-level public methods of the class:

- ADD_COMMAND adds a new command to the command list. After ADD_COMMAND has been called, the parameters of the command are added to the command list by calling either ADD_PARAMETER, ADD_INT_PARAMETER, or ADD_FLOAT_PARAMETER.
- ADD_PARAMETER adds a parameter to the command list with a character-based value type. The method ADD_PARAMETER offers the option to specify an index, which allows for adding several different values for a single parameter. When you specify 0 for the INDEX parameter (which is the default), you add a single value. When you want to add several values, you call ADD_PARAMETER with a value of anywhere from 1 to *n* for INDEX (where *n* is the number of values you want to add).
- ADD_INT_PARAMETER adds a parameter to the command list with an INT4 value type. As an example, think of a parameter describing the number of the page where you want to add some text.
- ADD_FLOAT_PARAMETER adds a parameter to the command list with a ZFLOAT value type. ZFLOAT is a data element of type FLTP.

Method PROCESS	
Importing parameter FILE_1	Type ZCL_FILE
Importing parameter FILE_2	Type ZCL_FILE
Importing parameter FILE_3	Type ZCL_FILE
Importing parameter FILE_4	Type ZCL_FILE
Importing parameter FILE_5	Type ZCL_FILE
Exporting parameter RETURN	Type BAPIRETURN
Exporting parameter RESULT_FILE	Type ZCL_FILE
Method ADD_ATTACHMENT	
Importing parameter DESCRIPTION	Type ZPARAM_VALUE
Importing parameter FILENAME	Type ZPARAM_VALUE
Importing parameter FILEINDEX	Type INT4
Method CONCAT	
Importing parameter FILEINDEX	Type INT4
Method ENCRYPT	
Importing parameter USERPASSWORD	ZPARAM_VALUE
Importing parameter OWNERPASSWORD	ZPARAM_VALUE
Importing parameter PERMISSION	Type INT4 (default "0")
Method ADD_BACKGROUND_TEXT	
Importing parameter TEXT	Type ZPARAM_VALUE
Importing parameter FONT	Type ZPARAM_VALUE (default "Helvetica")
Importing parameter FONTSIZE	Type ZFLOAT
Importing parameter XPOS	Type ZFLOAT
Importing parameter YPOS	Type ZFLOAT
Importing parameter GRAY	Type ZFLOAT
Importing parameter ROTATION	Type ZFLOAT (default "0.0")
Importing parameter PAGE	Type INT4 (default "1")

Figure 8 Public instance methods of class ZCL_PDF_COMMAND_LIST

Now let's look at some of the public instance methods of class ZCL_PDF_COMMAND_LIST, which are summarized in **Figure 8**:

- PROCESS is used to send the command list and the file parameters to the RFC server. The parameters of the PROCESS method are easy to understand. You can work on up to five ZCL_FILE objects, and the newly generated ZCL_FILE object is returned by the exporting parameter RESULT_FILE. The parameter RETURN is used to return any error that might have appeared during the processing of the files.

Note!

By default, the PROCESS method works on the file associated with FILE_1; the additional ZCL_FILE parameters FILE_2 to FILE_5 are optional parameters used for files that might, for example, be attached to FILE_1.

- ADD_ATTACHMENT adds a file as an attachment to the FILE_1 object specified in PROCESS. You can supply a description and a filename, which are later displayed in Acrobat Reader when you open the PDF returned in RESULT_FILE. The FILEINDEX parameter informs the ADD_ATTACHMENT method which file should be used as an attachment. FILEINDEX must be set to "2" to attach FILE_2, "3" to attach FILE_3, and so forth.
- CONCAT is used to concatenate two PDF files. It has just one importing parameter with the name FILEINDEX and type INT4. When you want to concatenate FILE_1 and FILE_3, for example, simply call CONCAT with "3" for the variable FILEINDEX, and FILE_3 will be appended to FILE_1 during the call of the PROCESS method. That's all.
- ENCRYPT is used for the encryption of a PDF file. You have to supply two different passwords — a user password and an owner password — and an INT4 value describing the permissions. The details of the encryption will be discussed later in this article. The default 0 for the PERMISSION means that no permissions are given to the file and therefore the strongest restrictions are applied.
- ADD_BACKGROUND_TEXT adds a text to an already existing PDF file. This method allows you to specify text that is actually added, as well as its font and its font size. You have to specify the X and Y coordinates of the insertion point, the rotation of the text, and the page on which the text is added. The parameter GRAY defines the shade of gray to be used for the text. It can vary from 0 (black) to 1 (white), with anything in between (e.g., "0.5") indicating a shade of gray. When the text should be inserted on the last page, set PAGE to "0."

Now that you have a solid understanding of the key components of the toolbox, including the ZCL_FILE and ZCL_PDF_COMMAND_LIST ABAP classes that execute its functionality, let's take a look at how the toolbox works in practice by walking through an example.

Programming example: Password-protecting a PDF document

To show you the toolbox in action, I have written an example ABAP report called Z_PDF_ENCRYPT. In this report, we'll modify an existing PDF document that resides locally on a client PC by combining it with a PDF generated from an SAP Smart Form (the name of the SAP Smart Form is hard coded into the example ABAP report) and attaching a simple TXT file to the resulting combined PDF. As mentioned earlier, this isn't intended to be a very realistic example; rather, it is a simplified example intended to clearly show some of the toolbox's methods in action.

Note!

To follow along with this example, and the barcode example later, you must have downloaded and installed the PDF Toolbox RFC server, as well as the set of ABAP classes in your SAP system. The code and instructions on how to do this are provided at www.SAPpro.com. The example ABAP report, PDF document, SAP Smart Form, and TXT file are also included in the download.

Figure 9 shows the dialog that appears when you run the ABAP report Z_PDF_ENCRYPT.

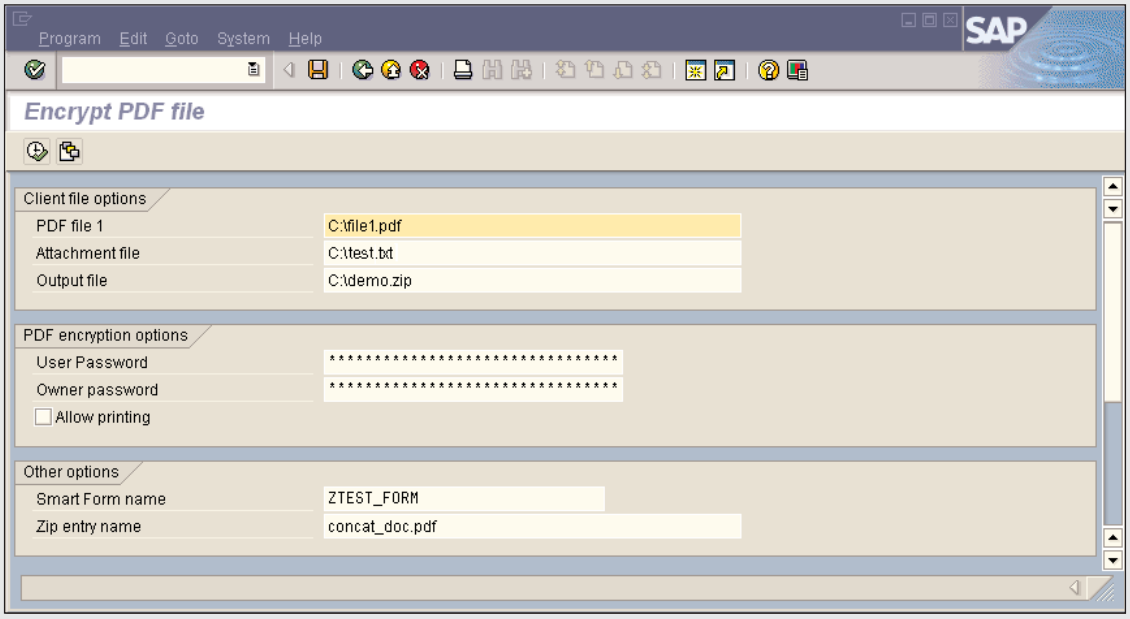

Figure 9 ABAP dialog for concatenating and encrypting PDF files

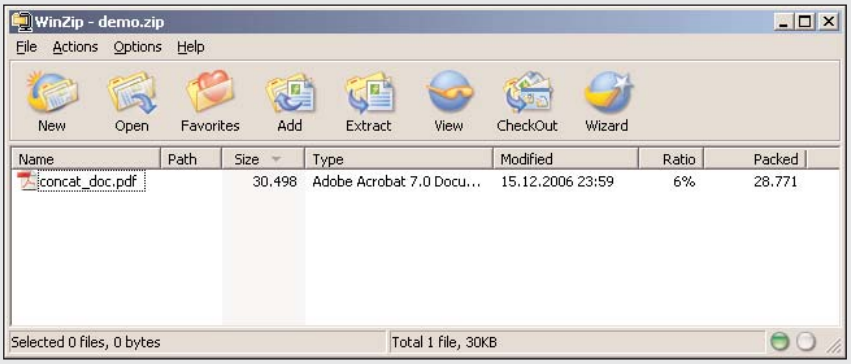

Figure 10 Opening a ZIP archive with WinZip

In the Client file options section, you can enter the path and name of the PDF file that will be read from the client computer, the path and name for the attachment, and the path and name for the output file. In the PDF encryption options section, you enter the user password and the owner password that will be used to encrypt the resulting PDF file. If you select the Allow printing checkbox, you will be able to print the resulting PDF file from Adobe Reader. In the Other options section, you enter the name of the ZIP archive file (including the correct path of the file) and the name of the Smart Form. For this example, I have created a simple Smart Form named ZTEST_FORM, which consists of a single page.

After you run the report with the input data shown in **Figure 9**, you open the resulting ZIP archive with WinZip, as shown in **Figure 10**.

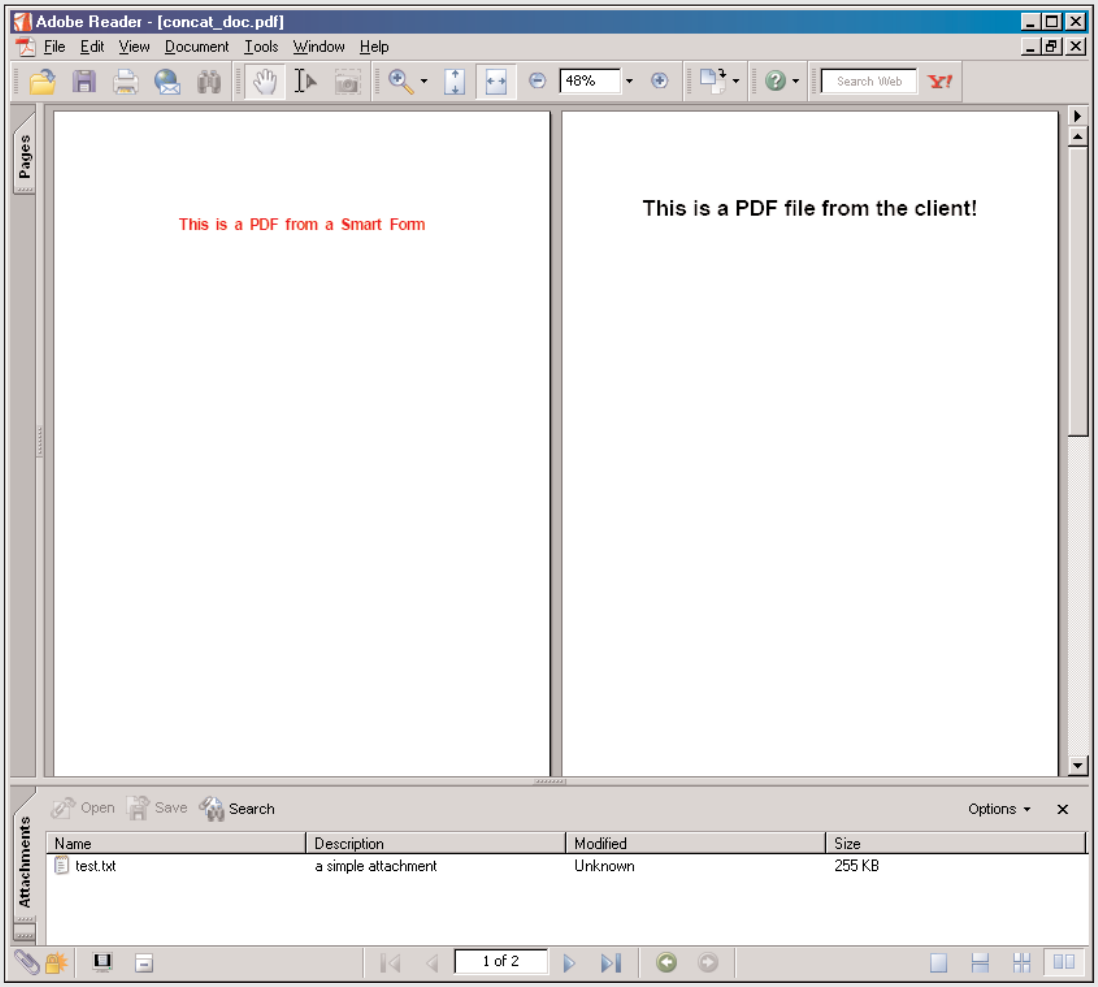

Figure 11 The two-page PDF form with its attachment opened in Acrobat Reader

Next, double-click on the concat_doc.pdf file to open it. When you open it with Acrobat Reader 7.0 (or higher), you are prompted for a password. Enter one of the passwords you specified in **Figure 9**. After Adobe Reader opens the document, you see a document consisting of two pages and an attachment, as shown in **Figure 11**.

The first page is a PDF form generated from an SAP Smart Form; the second page is the PDF file uploaded from the client. The attachment is the simple TXT file we specified in **Figure 9**.

Just to show you that the document permissions setting works as you might expect, open the Document Properties dialog by pressing Ctrl+D and clicking on the Security tab. As you can see in **Figure 12**, the only permission allowed is printing, as specified in the PDF encryption options in **Figure 9**.

A look inside the example program

Let's take a look inside the example ABAP report Z_PDF_ENCRYPT to provide you with a foundational understanding of how to create your own ABAP report for leveraging the toolbox. In the next sections, I walk you through the following steps:

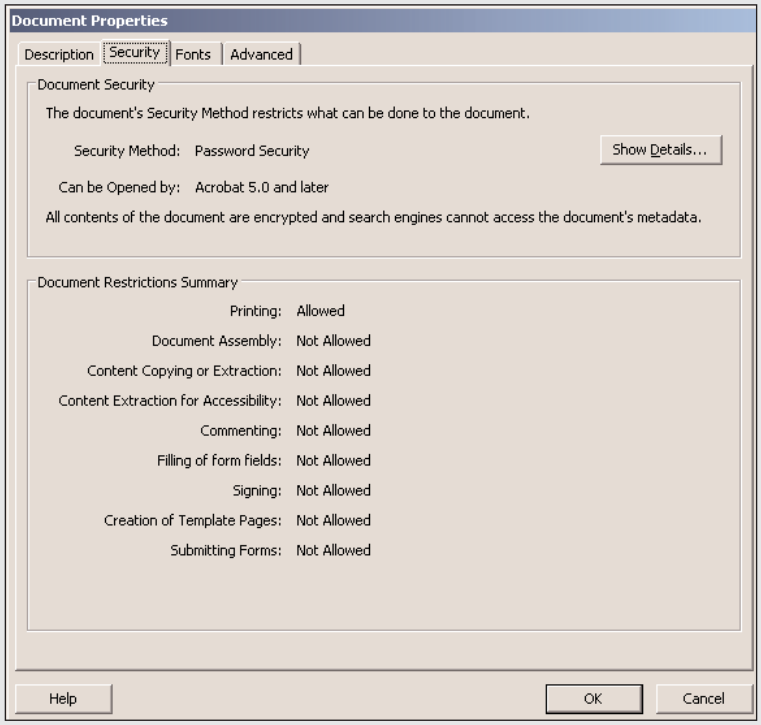

Figure 12 Security tab in the Document Properties dialog

1. Create a PDF file from an SAP Smart Form.
2. Create a PDF file from the client computer.
3. Concatenate the files from step 1 and step 2 into a single PDF file.
4. Add an attachment to the single PDF file created in step 3.
5. Password-protect the single PDF file and set the access permissions to it.
6. Add the ZIP option and perform the actual processing.
7. Save the newly created file on the client computer.

> ***Note!***
>
> I do not cover all details of the source code; this discussion is intended to simply give you an overview of how to use the toolbox.

Step 1: Create a PDF file from an SAP Smart Form

First, I created a PDF file from an SAP Smart Form. As an example, I have created the SAP Smart Form ZTEST_FORM, which consists of a single page with the text *This is a PDF from a Smart Form*.[2] Using the function module SSF_FUNCTION_MODULE_NAME, I retrieved the name of the function module to print the Smart Form. After retrieving the name of the function module, I set up the printing options to

[2] For more on creating SAP Smart Forms, see the *SAP Professional Journal* articles "Create and Maintain Forms Without Programming? It's a Snap with SAP Smart Forms!" (March/April 2001) and "Render Online, Interactive 'Web Forms' in Your Next Web Application — It's Easy with SAP Smart Forms!" (May/June 2002).

```
DATA : fm_name TYPE rs38l_fnam,
    params TYPE ssfctrlop,
    data TYPE ssfcrescl,
    options TYPE ssfcompop,.

 CALL FUNCTION 'SSF_FUNCTION_MODULE_NAME'
  EXPORTING
   formname      = 'ZTEST_FORM'
  IMPORTING
   fm_name      = fm_name
  EXCEPTIONS
   no_form      = 1
   no_function_module = 2
   OTHERS       = 3.

 params-no_dialog = 'X'.
 params-preview = ' '.
 params-getotf = 'X'.

 options-tddest = 'LP01'.
 options-tdnoprev = 'X'.

 CALL FUNCTION fm_name
  EXPORTING
   control_parameters = params
   output_options   = options
   user_settings   = ' '
  IMPORTING
   job_output_info  = data
  EXCEPTIONS
   formatting_error  = 1
   internal_error   = 2
   send_error     = 3
   user_canceled   = 4
   OTHERS       = 5.
```

Figure 13 Code for creating a PDF from an SAP Smart Form

```
DATA txt_file3 TYPE REF TO zcl_file.
txt_file3 = zcl_file=>create_from_client( file_name3 ).
  command_list->add_attachment( fileindex = 3
                                filename  = 'test.txt'
                                description = 'a simple attachment' ).
```

Figure 14 Code for adding an attachment

print without a dialog and to create OTF data. After calling the function module for printing the Smart Form, I have the OTF data as a result. The code for this task is shown in **Figure 13**.

Next, I transform the OTF output into a PDF file and load the second PDF file and the attachment file from the client. This is done using two static methods of the class ZCL_FILE, which is part of the ABAP portion of the solution, as shown here:

```
DATA pdf_file1 TYPE REF TO
     zcl_file.
pdf_file1 = zcl_file=>create_
     pdf_from_otf( data-otfdata ).
```

Step 2: Create a PDF file from the client computer

Creating a ZCL_FILE object from the client computer is done with just two lines of code:

```
DATA pdf_file2 TYPE REF TO
     zcl_file.

pdf_file2 = zcl_file=>create_
     from_client( file_name1 ).
```

Step 3: Concatenate the files from step 1 and step 2 into a single PDF file

Now, I set up the commands that describe the processing steps for the files. First, I create an instance of the ABAP class ZCL_PDF_COMMAND_LIST, and then I call the method CONCAT, as shown here:

```
DATA command_list TYPE REF TO
     zcl_pdf_command_list.
CREATE OBJECT command_list.

command_list->concat( fileindex
     = 2 ).
```

Step 4: Add an attachment to the single PDF file created in step 3

To add an attachment, I load the file that should be attached from the client, as shown in **Figure 14**.

Step 5: Password-protect the single PDF file and set the access permissions to it

To encrypt the PDF file, I simply call ENCRYPT supplying two passwords and the permissions.

```
   command_list->encrypt(
     userpassword  = upwd
     ownerpassword = opwd
     permissions   = permissions ).
```

The permissions are defined as integer constants, as shown in **Figure 15**.

To set more than one permission, I simply set the permission to the sum of their integer constants.

```
CONSTANTS :
    c_allowprinting TYPE i VALUE 2052,
    c_allowmodifycontents TYPE i VALUE 8,
    c_allowcopy TYPE i VALUE 16,
    c_allowmodifyannotations TYPE i VALUE 32,
    c_allowfillin TYPE i VALUE 256,
    c_allowscreenreaders TYPE i VALUE 512,
    c_allowassembly TYPE i VALUE 1025,
    c_allowdegradedprinting TYPE i VALUE 4.
```

Figure 15 Code for defining integer constants

Password-protecting a PDF file

Adobe Acrobat's default security handler for PDFs allows access permissions and up to two passwords to be specified for a document — an owner password and a user password:

- Opening the document with the owner password allows full access to the document, including the ability to change the document's passwords and access permissions.
- Opening the document with the user password (or opening a document that does not have a user password) gives a more restricted access because the passwords can't be changed. Furthermore, the access permissions of the document might disallow some operations like printing or changing the document.

However, the access permissions are only enforced by the viewer application (Adobe Reader), and therefore the protection offered by the permissions is weak.

For example, if you want to allow printing and fill-in, set the permission to 2052 + 256 = 2308. Using this approach, you can describe all combinations of the permissions in one INT4 variable. For more information on password protection, see the sidebar above.

Step 6: Add the ZIP option and perform the actual processing

Because I want the PDF file to be returned inside a ZIP archive, I need to set the ZIP_ENTRY attribute of the file to the corresponding value of the input file in the report Z_PDF_ENCRYPT. Then I call the PROCESS method for the command list object. **Figure 16** shows the code required for these two tasks.

Step 7: Save the newly created file on the client computer

In the last step, I check whether the call to the PROCESS method was successful, and then I save the generated file to a local path, as shown in **Figure 17**. As a result, I have a PDF file with an attachment inside a ZIP archive.

You've now seen the toolbox in action, and had a peek under the hood to see how the ABAP report uses the ABAP classes to leverage the functionality provided by the RFC server. While the RFC server comes with several commonly used modification capabilities, you can also add additional capabilities. I'll show you how to do this next.

Adding a new command to the RFC server

The RFC server provides only some common functionality for working with PDF files. You can also take advantage of some of the more advanced functionality offered by the iText library by extending the functionality of the RFC server.[3] Using the previous example, I'll show you how you might extend the RFC server by placing a two-dimensional (2D) barcode on the generated PDF file (see the sidebar at right for an overview of the 2D barcode format). First, I will extend the RFC server and the ABAP class ZCL_PDF_COMMAND_LIST to provide the additional functionality. Then, I will

[3] For an overview of the additional options offered by the iText library, take a look at the exhaustive tutorial available at www.lowagie.com/iText. Bruno Lowagie, one of the authors of the iText library, has also written an excellent book on iText (*iText in Action: Creating and Manipulating PDF*. Manning Publications Co., 2006).


```
DATA : lreturn TYPE bapireturn.

  result_file->zip_entry = p_zipent.

  CALL METHOD command_list->process
    exporting
      file_1 = pdf_file1
      file_2 = pdf_file2
      file_3 = txt_file3
    IMPORTING
      return = lreturn
      result_file = result_file.
```

Figure 16 Code for adding a ZIP option and processing the command list object

```
  CASE lreturn-type.
    WHEN 'S'.
      WRITE 'Success'.
      file_name1 = p_filout.
      IF result_file->save_to_client( file_name1 ) <> 0.
        WRITE : 'Cannot save file ', file_name1.
      ENDIF.
    WHEN 'E'.
      WRITE : 'Error:', lreturn-message.
  ENDCASE.
```

Figure 17 Code for saving the file to the client computer

The 2D barcode format

PDF417 (Portable Data Format, not to be confused with Portable Document Format) is a two-dimensional (2D) barcode format that was developed by Ynjium Wang at Symbol Technologies in 1991. 2D barcodes have much higher data capacity than one-dimensional barcodes. PDF417 uses Reed Solomon error correction and allows for nine different levels of error detection and correction. At level 0, you can encode up to 1850 characters, and errors can be detected but not corrected. At level 8, you can encode only 850 characters, because a lot of redundant information must be encoded in the barcode at this level; however, you have a much better error-correcting functionality.

```
public void addPDF417Barcode(CommandParameterMap map)
         throws DocumentException {

      int page = getPageNumber(map);
      String text = map.getString(Parameter.TEXT);
      float xPos = map.getFloat(Parameter.XPOS);
      float yPos = map.getFloat(Parameter.YPOS);
      float scale = map.getFloat(Parameter.SCALE);

      BarcodePDF417 pdf417 = new BarcodePDF417();
      pdf417.setText(text);

      Image image = pdf417.getImage();
      image.scalePercent(scale);
      image.setAbsolutePosition(xPos, yPos);
      stamper.getOverContent(page).addImage(image);
   }
```

Figure 18 Adding a PDF417 barcode to a PDF file

modify the report Z_PDF_ENCRYPT so that an additional barcode is added to the first page of the resulting PDF file.

The manipulation of the PDF files on the Java side is done by the class PdfHandler, which is located in the package com.cjastram.pdf.handler. As a first step, I add an empty method to this class that takes a CommandParameterMap object as its argument. The class CommandParameterMap is located in the package com.cjastram.pdf.command, and it is used as a container for the parameters of the new command. The empty method looks like the following:

```
public void addPDF417Barcode
         (CommandParameterMap map)
{

}
```

Next, I add a new public instance method with the name ADD_PDF417_BARCODE to the class ZCL_PDF_COMMAND_LIST using the ABAP Class Builder. In this method, I add the command addPDF417Barcode to the command list:

```
METHOD add_pdf417_barcode .
  me->add_command
     ('addPDF417Barcode').
ENDMETHOD.
```

It is important that the name of the command in the ABAP method (the parameter of the method ADD_COMMAND) and the name of the new method in the class PdfHandler on the Java side are identical. The first part of adding the new functionality is now accomplished. When I first call the add_pdf417_barcode method for a ZCL_PDF_COMMAND_LIST and later on the PROCESS method for the list, the command addPDF417Barcode will be executed on the RFC server. The matching of the command name and the method name on the Java side is done at runtime using the Java Reflection API. The Java Reflection API is used to fetch information about the names and parameters of methods in the class PdfHandler.

Next, I implement the new functionality by adding a PDF417 barcode. **Figure 18** shows the Java coding for adding a PDF417 barcode.

First, I retrieve the parameters of the command from the CommandParametermap. The parameters

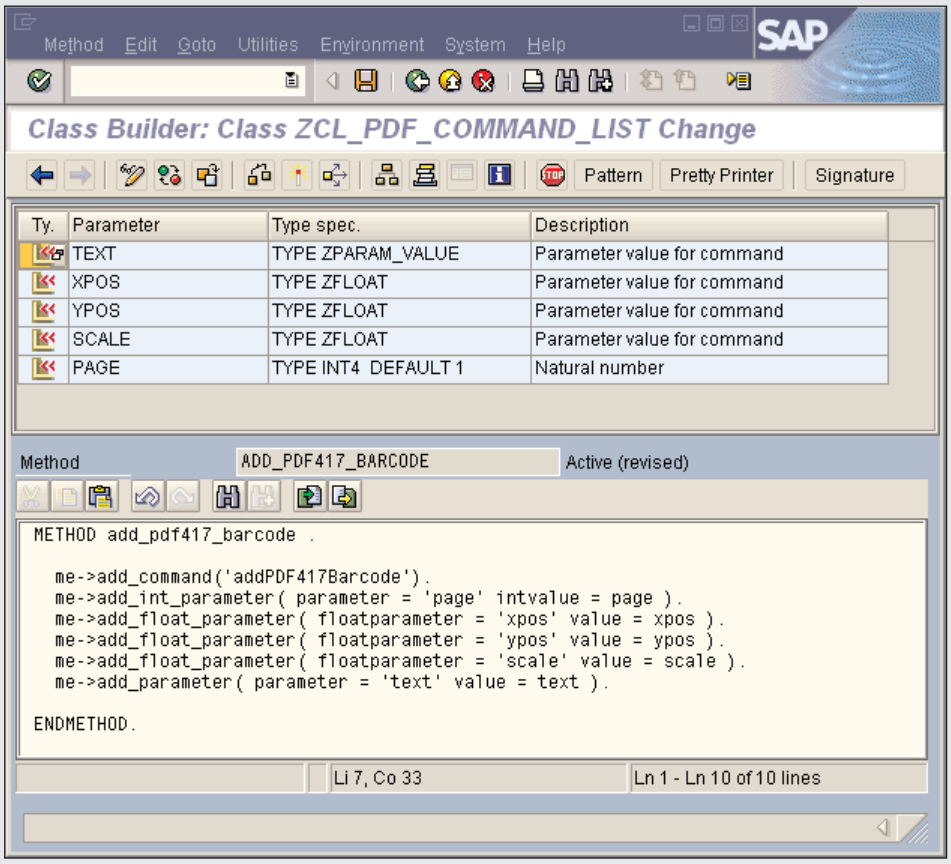

Figure 19 Parameters and coding for method ADD_PDF417_BARCODE of class ZCL_PDF_COMMAND_LIST

used are the number of the page where the barcode should be placed, the text that is encoded in the barcode, the X and Y coordinates of the insertion point of the barcode, and an overall scaling factor for the barcode image.

With the Java coding now complete, I next need to adapt the ABAP coding. For each of the five parameters used in the Java coding, I have to add a parameter to the signature of my ABAP method and a call to ADD_INT_PARAMETER, ADD_FLOAT_PARAMETER, and ADD_PARAMETER.

Now I only have to add these parameters to the method ADD_PDF417_BARCODE on the ABAP side. **Figure 19** shows the parameters and the new coding. As a last step, I add a call of the method ADD_PDF417_BARCODE to the report Z_PDF_ENCRYPT just before the PROCESS method is called. See **Figure 20** on the next page.

Finally, I run the report using the same input parameter as in the previous example. When I open the generated PDF in Acrobat Reader, a big PDF417 barcode is placed on the first page of the document, as shown in **Figure 21** (on the next page).

You now have all of the tools you need to create the ABAP report that will leverage the functionality in the RFC server, and to add any additional capabilities you need to the RFC server. Once your ABAP report is ready and you've made any necessary modifications


```
command_list->add_pdf417_barcode( page = 1
                                  text = 'Hello this is a test'
                                  ypos = '210.0'
                                  xpos = '110.0'
                                  scale = '500.0' ).
```

Figure 20 Coding for calling the method ADD_PDF417_BARCODE in the report Z_PDF_ENCRYP

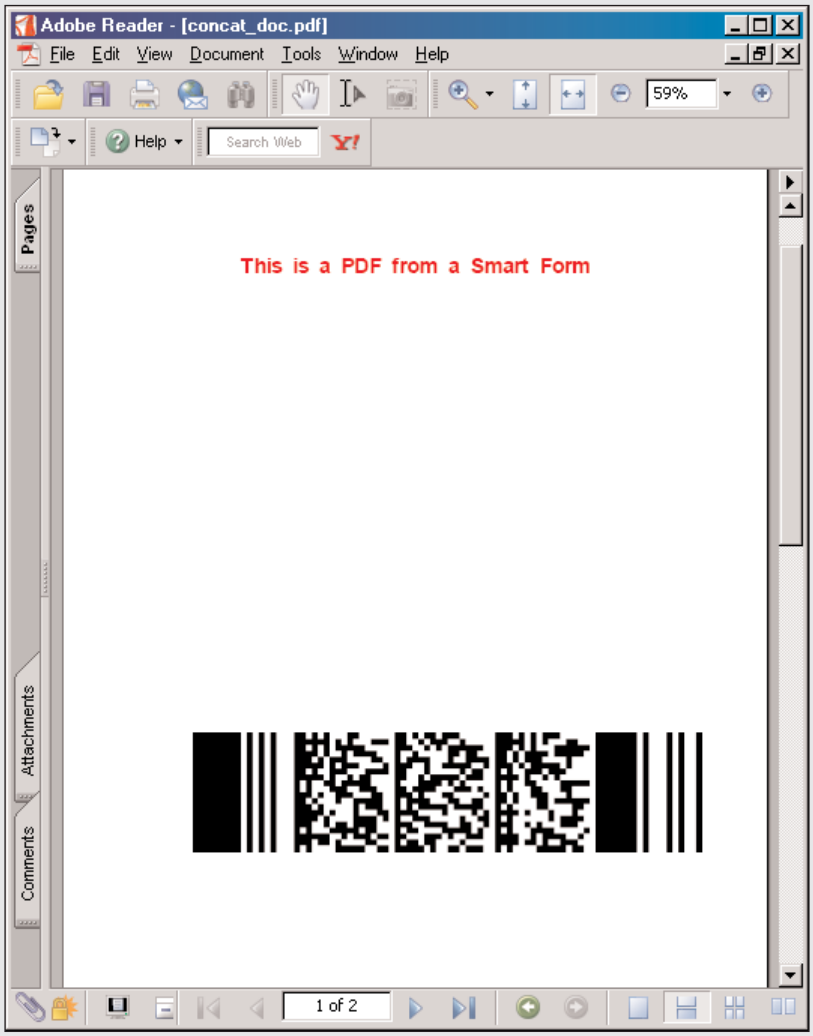

Figure 21 A PDF417 2D barcode placed on the first page of a PDF

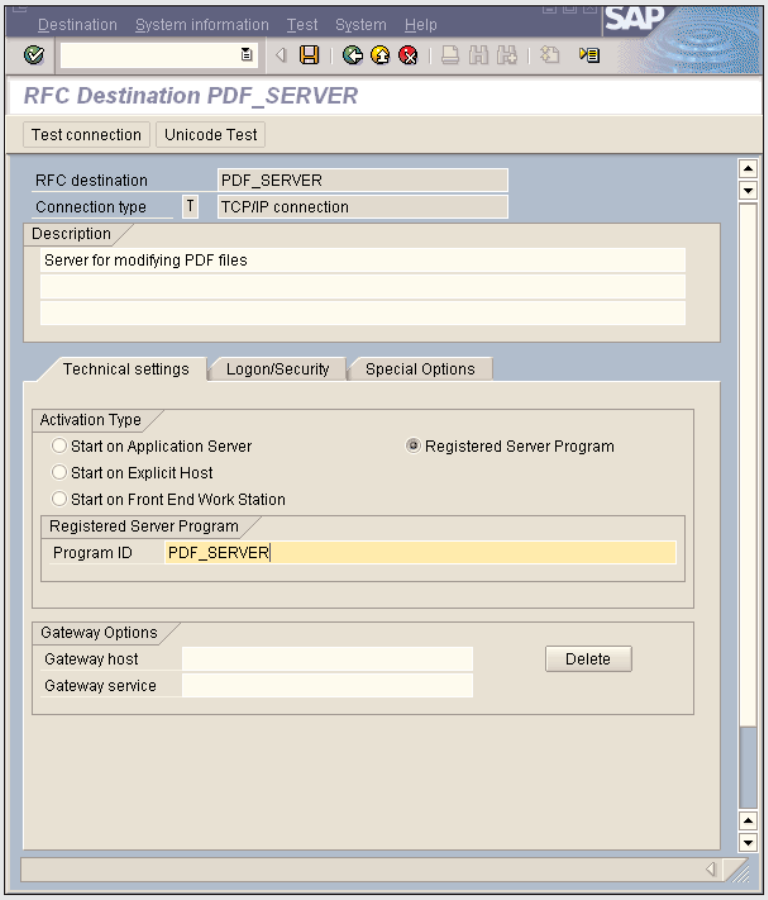

Figure 22 Creating an RFC destination using transaction SM59

to the RFC server, you are ready to deploy the RFC server and start taking advantage of the PDF Toolbox.

Deploying the RFC server

To deploy the solution, you need the following components:

- Java 5 Runtime Environment. I use the Java Runtime Environment delivered with SAP NetWeaver Developer Studio EE 5 Edition (Build 1.5.0_08-b03).
- SAP (JCo), which is available from the SAP Service Marketplace (www.sap.com/services/index.epx). I have used version 2.0.12; however, all versions from 2.x and higher should work.
- A compiled version of the iText Java library (version 2.0.1) and a compiled version of the RFC server, both of which are available for download from www.SAPpro.com.

For the installation, you have to complete the following three steps:

1. **Create an RFC destination in your SAP system.** The RFC destination is created using transaction SM59. For this purpose, create a new RFC destination PDF_SERVER with the connection type TCP/IP using transaction SM59. As shown in **Figure 22**, select Registered Server Program as

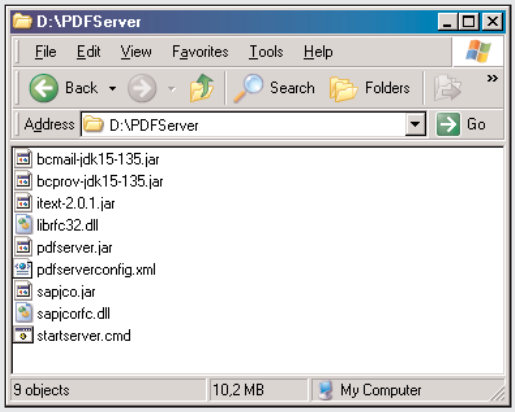

Figure 23 Installation directory of the RFC server

the activation type and use PDF_SERVER as the program ID.

2. **Assemble the server files.** First, create a new empty directory, and then copy the files librfc32.dll, sapjcorfc.dll, and sapjco.jar, which are shipped with SAP JCo, into the new directory. You should always use an empty directory to avoid any problems with already existing files. Then, extract the files iText-2.0.1.jar, bcmail-jdk15-135.jar, bcprov-jdk15-135.jar, startserver.cmd, pdfserver.jar, and pdfserverconfig.xml from the downloaded ZIP file.[4] Your directory should now look like the one shown in **Figure 23**.

3. **Edit the configuration file.** The configuration data of the RFC server is stored in an XML file with the name pdfserverconfig.xml. There are just three configuration parameters: gwhost and gwserv are the parameters for connecting to the SAP system,[5] and the third parameter is loglevel. For a production server, set loglevel to error; if you are interested in detailed information (e.g., the name of the methods that are executed by the RFC server and the names and values of the parameters that are passed to these methods), set it to info.

Now you can start the server by double-clicking on the file startserver.cmd. A command shell similar to the one shown in **Figure 24** should appear. (If the command shell doesn't appear, go back and check your settings.)

Conclusion

As you have learned in this article, the PDF Toolbox enables an ABAP developer to use some of the more advanced features of the PDF technology, such as encrypting a PDF file, concatenating two PDF files, and adding attachments to a PDF file, without having to go through the tedious process of making the modifications in Adobe Acrobat itself — instead, the changes can be made programmatically, on the fly. PDF files are actually manipulated by an RFC server that has been implemented in Java, which uses the open source library iText to perform the modification operations on the PDF files. To ease the use of the RFC server, I have written two ABAP classes that provide a simple interface to the methods exposed by the RFC server. Furthermore, it is easy to extend the RFC server to include some of the more advanced functionalities of the iText library, but for this you have to have some Java skills and some knowledge of the iText library, which is used to actually modify the PDF files.

[4] Since version 2.0, iText uses the Bouncy Castle Crypto API for Java (available from www.bouncycastle.org). for this version you must extract the files bcmail-jdk15-135.jar and bcprov-jdk15-135.jar.

[5] I don't go into the details of these parameters. Ask you SAP administrator for the correct values for your SAP system.

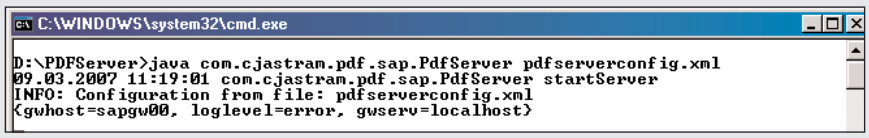

Figure 24 The RFC server after startup

