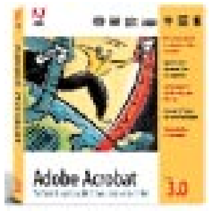
WebObjects PDF Layer

A Programmer’s View



This document describes the mechanisms through which WebObjects supports PDF and FDF.

1.0 The Basic Picture

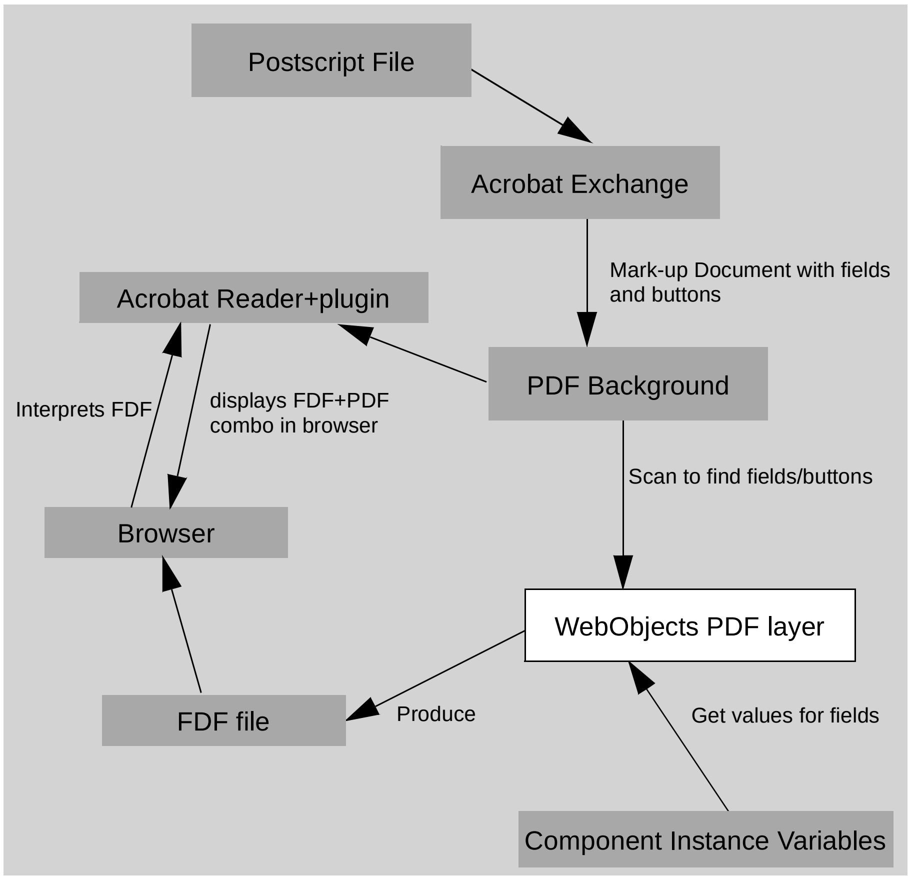
Figure 1 on page 2 shows how to create a PDF application with WebObjects. The major steps involved are:

1. Create a page background and put it into a PostScript file via the print to file option. You can use any word processor or presentation program that supports printing to PostScript.
2. Using Acrobat Exchange version 3.0, convert this PostScript file into a PDF file. (There are other ways to create PDF. This is just one possibility.)
3. In Acrobat Exchange, mark up this PDF document with active zones, such as text fields, lists, combo boxes, and buttons. There are two different types of mark-up fields that you have to consider:
   - Text Fields and all fields that take and return a value from a WebObject element. These include text fields, combo boxes, list boxes, and radio buttons. For those types, the name of the field is mapped to instance variables or methods (using the same name conventions as the standard WebObjects language mappings) of the corresponding component. For example, a field called “name” will be mapped to the methods **name**() and **setName**() or, if the component doesn’t implement these methods, to the component’s **name** instance variable. All values involved are of type `java.lang.String`.

     Read-only fields and accessor methods: If the field is marked read-only in Acrobat Exchange (i.e. the user will not able to modify it in the browser) then, it is not necessary to specify both **name**() and **setName**(). **name**() will suffice, as **name**() **setName**() will never be used.

     Dereferencing: As for regular HTML WebObjects components, it is possible to specify a key path instead of a single name. However, due to the special use of

Figure 1. The steps to PDF



the dot character by Adobe Acrobat, the slash character has to be used instead. For example, a field called “car/wheel/diameter” will specify the same value as car.wheel.diameter would specify for a regular WebObjects component.

- Buttons with a submit action. In this case, the name of the field has no meaning for WebObjects. You use the URL in the submit action to indicate the action to be taken. The URLs to be used follow the same syntax as the binding for a WOHyperlink:

  - `action="open"`, will trigger the **open**() method on the receiving component (notice the double quotes),

  - `pagename="NextPage"`, will return the page named NextPage as the target of the hyperlink.

4. When you save the PDF files make sure you save optimized PDF, as it is what the PDF parser included in WebObjects expects. One way of doing that is to use the ‘Save as’ command in Exchange.
5. At runtime, when the PDF component receives a request, the marked-up PDF file that you generated is parsed to retrieve the mark-up fields. An FDF form is then generated. This FDF forms contains:

- values taken from instance variables/methods to fill the different mark-up fields
- dynamic targets for submit form buttons
- a pointer to the PDF background.

6. The FDF is then sent to your WWW browser, which passes it on to the Acrobat plug-in. The Acrobat plug-in downloads the PDF background and fills the fields with the values included in the FDF form.

2.0 The Classes

The PDF layer is purely a Java add-on to the Web Objects Framework; the core of the framework has not been modified. The source code of the Java classes is available with this example.

“PDFness” is decided on a page-by-page basis. The application can mix and match HTML and PDF as desired with the restriction that a given page is either all HTML or all PDF.

2.1 PdfComponent, PdfStaticComponent

For a given page to be based on PDF, the component has to subclass `pdf.PdfComponent`. When a PdfComponent is instantiated, it looks for a PDF file to find which elements need to be instantiated. The PDF scanner looks for mark-up fields or buttons and maps them to dynamic element types.

Figure 2. Mapping between mark-up fields and dynamic elements

Acrobat Exchange Mark-up field type	WebObjects Dynamic Element type
text, list box, combo box	PDFTextField
check box, radio button	PDFCheckbox
buttons with submit action	PDFHyperlink

In essence, the parser generates on the fly the same information from the PDF file that exists in the HTML/WOD files in a regular WebObjects component. If caching is enabled for that component, the PDF file will only be parsed once and the resulting data re-used.

You can override the default type mapping provided by the PDF component. If an HTML and a WOD file are found in the resource path for this component, they override the PDF scanning process. The dynamic elements found in the HTML/WOD combination are instantiated.

If you want to display a static PDF file (one containing no fields) inside a WebObjects application (such as the file you are reading now) use `pdf.PdfStaticComponent` as a superclass instead of `pdf.PdfComponent`.

2.1.1 Dealing with exceptions/Displaying alert messages

PDF allows alerts to be displayed when an FDF form is received by the plug-in. By default, if an exception is thrown during processing of a form, an alert displaying this exception will be displayed on the client side. You can have finer control over the message displayed by using the following API:

- `public String status()`
- `public void newStatus(String newStatus)`
- `public void addToStatus(String addToStatus)`

The resulting status, if not empty, will be displayed as an alert to the user.

As an alternative, the component can have a error field, invisible to the user as long as it is empty, in which error messages can be displayed in a highly visible font.

2.2 PDFTextField, PDFCheckbox

PDFTextField and PDFCheckbox are very similar. Both will take a `java.lang.String` value and put its value into the appropriate slot in the generated FDF form. When a form is submitted, they will grab the value from the incoming form and put it in the appropriate instance variable (or call the **set***VarName*() method)

2.2.1 Default Values

There are two options to handle default values with the PDF layer of WebObjects: use the ‘reset form’ action of Acrobat Exchange in a button, or create a button that calls a ‘reset()’ method on the server side. The first option has the advantage of not requiring a round-trip to the server but gives you less control. The second option allows any kind of treatment, but requires a round-trip to the server to reset the fields.

Reset Local to the Client: You must

- Specify default values in Acrobat Exchange for the fields you create.
- Create a button with an Acrobat ‘reset form’ action.
- For each field on the component side, you must give it the special value `pdf.PDFTextField.defaultValueTag`. This tag tells the dynamic element to leave the default value found in the PDF file unchanged (the normal behavior is to override this value with the current content of the field)

reset() Method: You must

- Create a button that submits a form to `action="resetForm"`
- In the component, create a **resetForm**() method that modifies the values of the fields to their original values.

2.3 PDFHyperlink

PDFHyperlink supports the following bindings:

- `action="myMethod"` invokes the **myMethod**() method when the button is pressed.

- `pagename="aPage"` returns the page aPage when the button is pressed.

3.0 New In 4.0

- All the PDF elements are now non-synchronizing components (they were dynamic elements in 3.5) which makes their implementation a lot easier to understand
- Because they are now component, the PDF ‘elements’ have moved from the pdf package to the top-level package.