



Saffron

Workgroup V1.0.1

Developers Guide

Copyright: Dynalivery 2003





Contents

DISCLAIMER..... 4

ACKNOWLEDGMENTS 4

CHAPTER 1 INTRODUCTION TO SAFFRON WORKGROUP 6

 BACKGROUND 7

 SAFFRON WORKGROUP COMPONENTS..... 7

 CONVENTIONS USED IN THIS MANUAL..... 8

 NOT INCLUDED IN THIS MANUAL..... 8

Information this manual does not contain..... 8

 GETTING HELP..... 9

 USING OUR WEB SITE 9

 EMAIL TECH SUPPORT 10

 IN YOUR EMAIL TO TECH SUPPORT, PLEASE INCLUDE 10

CHAPTER 2 INTRODUCTION TO SAFFRON SDK..... 11

MAIN CLASSES OF THE SAFFRON SDK..... 11

 SAFFRON CLIENT 11

SaffronEmbeddedClient 11

SaffronCORBAClient 11

 FORMATOPTIONS:..... 12

PDFFormatOptions: 12

RTFFormatOptions:..... 13

HTMLFormatOptions: 13

SVGFormatOptions: 13

RasterFormatOptions: 13

 DESTINATIONOPTIONS..... 14

ClientFileSystemDestinationOptions: 14

ServerFileSystemDestinationOptions: 14

StreamDestinationOptions: 14

 SAFFRONINPUTSOURCE: 15

SaffronLanguageInputSource: 15

SaffronPrintableInputSource: 15

 RUNNING SAFFRONCLIENTS 16

Requirements For Running Embedded Client..... 16

Requirements For Running a Remote Client..... 17

CHAPTER 3 INTRODUCTION TO SAFFRON PRINT SERVICE..... 18

CHAPTER 4 INTRODUCTION TO SAFFRON PRINTER (WINDOWS ONLY) 22

 CONFIGURING THE SAFFRON PRINTER 23

APPENDIX A FREQUENTLY ASKED QUESTIONS..... 24

 CAN SAFFRON EMBED TRUE TYPE FONTS IN PDF? 24

 WHAT IS THE ORB REQUIRED ON THE CLIENT SIDE?..... 24

 WHAT JRE IS REQUIRED FOR SAFFRON? 24

 WHAT POSTSCRIPT CAN SAFFRON ACCEPT? 24





WHY IS THE TEXT OVERLAPPING OR GETTING TRUNCATED IN SVG FILES? 24
WHY ARE THE FONTS GETTING SUBSTITUTED IN THE OUTPUT?..... 24
WHERE CAN WE FIND THE TRUETYPE FONT FILES ON THE MACHINE? 24





Disclaimer

Dynalivery Corporation provides sample source code, ScreenCam movies, and other documentation only as aids for learning specifically about Saffron Workgroup and Saffron Desktop while providing a general overview of web reporting.

These materials are subject to change without notice. We provide them on an "as is" basis without warranty.

Please do not treat them as production quality components to use within your own application without your own comprehensive testing.

Acknowledgments

"This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)"





The information in this manual refers to the Workgroup Edition of the Saffron Document Server by Dynalivery Corp.

Dynalivery Corp. endeavors to ensure that the information in this manual is correct, but does not accept liability for any error or omission.

The development of Dynalivery Corp. products and services is continuous and published information may not be up-to-date. Any particular issue of a product may contain only part of the facilities described in this manual, or may contain facilities not described here.

Specifications and statements relating to performance in this manual are Dynalivery Corp. estimates intended for general guidance. They may require adjustment in particular circumstances and are, therefore, not formal offers or undertakings.

Statements in this manual are not part of a contract or program product license unless they are incorporated into a contract or license by express reference. Issue of this manual does not entitle the recipient access to or use of the products described, and such access or use may be subject to separate contracts or licenses.

Copyright

This manual is copyrighted, with all rights reserved. Under copyright laws, this manual may not be copied, in whole or in part, without the written permission of Dynalivery Corp.

Copyright © 2003 Dynalivery Corp.

Trademarks

Parallel Crystal is a trademark of Dynalivery Corp.

TAO is a trademark of Washington University, St. Louis, Missouri, U.S.A. JacORB is a trademark of Freie Universität, Berlin, Germany. PostScript is a registered trademark of Adobe Systems, Inc. Other brand or product names are trademarks or registered trademarks of their respective holders.

Company Information

Dynalivery Corp.
320 West Port Plaza Drive, Suite 205,
St Louis, Missouri, 63146 USA

Web <http://www.dynalivery.com/>

Tel. (314) 205-8995

Fax. (314) 205-8622

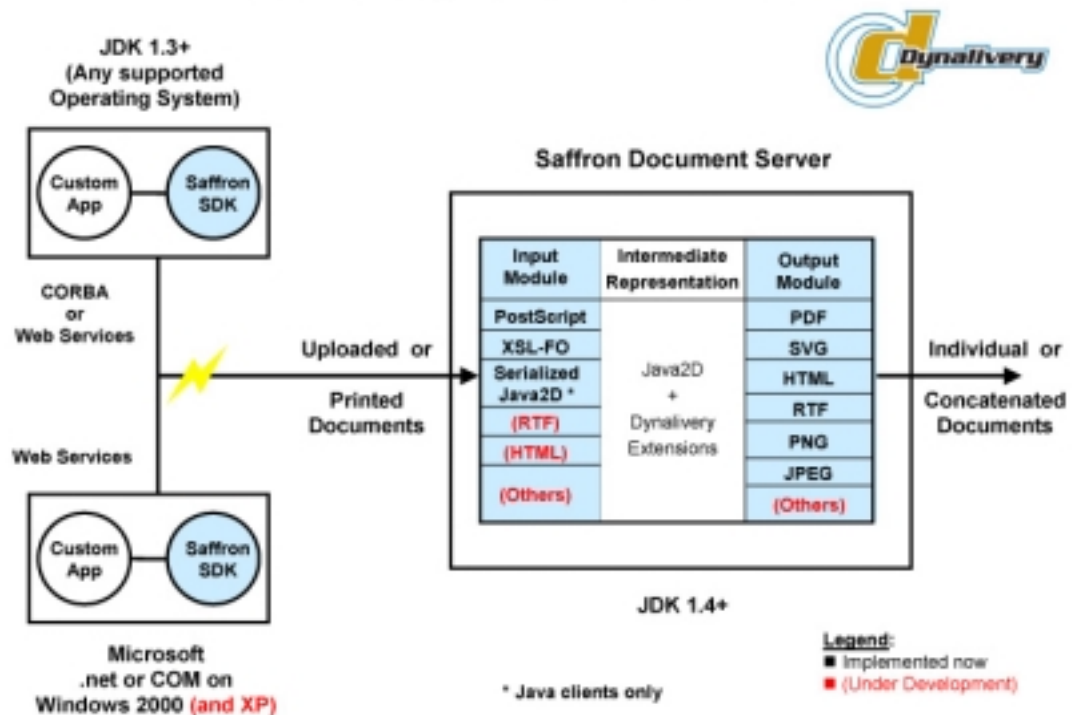


Chapter 1

Introduction to Saffron Workgroup

Saffron is virtual printer software. Applications typically connect to Saffron over intranets or the Internet, and "draw" successive text and graphics objects (using Sun's [Java2D](#) specification) that comprise the pages of documents. Saffron then translates these pages to a variety of popular document formats and transmits them immediately to users waiting at web browsers. Saffron can "archive" documents on paper or in databases for "later" use. Saffron also optionally acts as a "local virtual printer," executing on the same computer as the invoking application. Saffron includes an all-Java PostScript® language compatible interpreter and a PostScript print driver for Microsoft Windows NT/2000.

Saffron Workgroup and SDK Architecture



Background

A core capability of Saffron is to generate Portable Document Format ([PDF](#)) output, similar to Adobe Acrobat [Distiller](#). However, Saffron is a high concurrency Java server application, not a single-user desktop product. The PDF format originated in the graphic arts industry and has become a de-facto standard for precisely formatted [business documents](#) such as bank statements and insurance forms that users transmit over the Internet. Unlike [HTML](#), PDF documents retain their formatting across many different computers, operating systems, Internet browsers, and printers. Users may read PDF documents with a free browser plug-in called Acrobat Reader that is downloadable from the web sites of Adobe and thousands of its customers.

Dynamalivery's first product, Parallel Crystal, has generated PDF documents from one application, [Crystal Reports](#), since 1999. It uses a [thread-safe](#) C++ library developed by Dynamalivery. It processes complex report templates with sub-reports, charts, lines, and bitmaps. Parallel Crystal faithfully renders them with a compact file size and page breaks in the right locations, matching the Crystal Reports Designer [Print Preview window](#). To see for yourself, download the sample Box Office Movie Report, [Box.pdf](#), generated with Parallel Crystal v2.6 and Crystal Reports 7. (If you have Crystal Reports Designer 7 please feel free to download and review our [box.rpt](#) sample report template.) Dynamalivery's enterprise customers have achieved scalability in the range of 10,000 to 100,000 PDF pages per day of highly personalized output with Parallel Crystal.

Dynamalivery developed Saffron in response to customer demand to break out the popular C++ language PDF technology in Parallel Crystal into a separate server product that would work with Crystal Reports, competitors to Crystal Reports, and even non-reporting server applications. By implementing in [Java](#) we have met an additional customer requirement for portability across operating systems and computing devices such as appliances, printers, PCs, workstations and high-end servers. Saffron will have a number of enhancements over Acrobat Distiller including a variety of output formats beyond PDF (HTML, [RTE](#), JPEG, PNG and [SVG](#)), document [concatenation](#), and document storage in an Enterprise Java Bean ([EJB](#)) repository. Note that Adobe [publishes](#) the structure of its PDF file format to [assist developers](#) like Dynamalivery in the creation of software products that complement Acrobat. Dynamalivery currently relies upon public specifications of the other standard file format types that it targets with Saffron output modules.

Saffron Workgroup Components

Saffron SDK

Saffron Print Service

Saffron Printer (Windows only)

Saffron Desktop GUI (see separate User Guide)

The above-described components get installed on a machine as a part of the Saffron Workgroup based on the underlying OS. The next couple of Chapters give an introduction about using each of the above components. The "*Dynamalivery/Saffron/samples*" folder contains java samples for using the SDK and the Print Service. During the course of next couple of chapters, they will be explained in detail.

Conventions used in this Manual

All class names will appear in the font Arial Narrow Bold like this: **ClassName**

All method names will appear in the font AvantGarde Bold Italic like this: *MethodName*

All package names and parameters will appear in italics like this: *package.name*

All code snippets will appear in Courier New like this:

```
CodeSnippet CS = new CodeSnippet ();
```

Not Included in this Manual

Information this manual does not contain

- **Installation instructions**
Installation instructions for Saffron Desktop can be found in the [Saffron Desktop Users Guide](#)
- **System Administration information and trouble-shooting**
For System Administration and trouble-shooting, contact Tech Support.
- **Networking information and trouble-shooting**
For Networking information and trouble-shooting please contact Tech Support.
- **Detailed Frequently Asked Questions (FAQs)**
The Saffron FAQ page can be found on our website at http://www.dynamalivery.com/customersupport/Saffron_FAQs/Saffron_FAQs.html

Saffron Document Server Customization APIs

If you are looking for detailed descriptions of the Saffron Customization APIs, consult our web site for the online API. The API Reference contains detailed descriptions of all the API classes and methods that comprise the various Saffron client packages. Whenever you get an API programming error, go directly to the Reference Documentation for help. The *Saffron API Reference Documentation* is automatically installed with Saffron Workgroup and is available from the Start menu.



Getting Help

Dynamalivery Corp. provides a variety of sources for problem solving. Visit our website, or email us with questions, comments, suggestions, and problems.

Using our Web Site

Our web site contains links to the latest online versions of this documentation and many other helpful resources.

- **Home Page:**
<http://www.dynamalivery.com>
- **Saffron FAQ Page:**
http://www.dynamalivery.com/customersupport/Saffron_FAQs/Saffron_FAQs.html
- **Saffron FAQ Search Page:**
<http://www.dynamalivery.com/customersupport/searchsaffron.html>
- **Demos/Samples:**
To provide a better understanding of Saffron, sample code is available on our website at:
http://www.dynamalivery.com/customersupport/samples/Saffron_Samplecode.html

These samples are included in the Saffron Workgroup installation CD. However, in some cases, the samples on the website are more current because we have made revisions that have not yet been included in a release of Saffron.





Email Tech Support

Technical Support for products of Dynamivery Corp. starts with an email to:
TechSupport@dynamivery.com.

In your email to Tech Support, please include

- A detailed text description of the nature of the problem
- Screen shots of errors, if any
- Specifications of the system environment
- The version of Saffron installed
- Saffron Log
- Client Code
- Input Files



Chapter 2

Introduction to Saffron SDK

Saffron SDK is a set of Java classes, which lets you connect to a Saffron Server and convert PostScript, and FO (Formatting Objects) documents into one of the many supported output formats. The SDK can be used to convert a single document or concatenate multiple documents of the same type or different types into a single output document. The client package “*com.dynamalivery.saffron.client*” defines all the necessary Java classes for using the Saffron Server. The Client is classified into two types:

Embedded Client **Remote Client**

If the Saffron Client and the Server are running in the same JVM, then the embedded client is used. If the Server is running on a different machine or different JVM then the remote client is used. In the case of remote client, CORBA (Component Object Request Broker Architecture) is used for communication between the client and the server. The examples folder has sample Java apps, which show how to use the SDK for converting a single document or concatenating multiple documents to a single output file.

Main Classes of the Saffron SDK

SaffronClient
FormatOptions
DestinationOptions
SaffronInputSource

Saffron Client

SaffronClient is an abstract class, which lets you connect to the Saffron Server. There are two implementations of this class: SaffronCORBAClient and SaffronEmbeddedClient. Depending on the type of client, as described in the above section, a SaffronClient object is instantiated.

SaffronEmbeddedClient

SaffronClient has a static helper method called *createSaffronEmbeddedClient* for creating a SaffronEmbeddedClient. This method takes a *java.util.Properties* object as a parameter. A “*null*” can be passed for properties.

```
SaffronEmbeddedClient client = SaffronClient.createSaffronEmbeddedClient (null);
```

SaffronCORBAClient

SaffronClient has a static helper method *createSaffronCORBAClient* for creating a SaffronCORBAClient. This method also takes a *java.util.Properties* object as a parameter. The Properties object must contain two properties a) *serverName* b) *serverPort*. SaffronCORBAClient tries to connect to

the Saffron Server on the machine specified by “serverName”. By default Saffron Server starts on port 14030. This port information is configurable. If the Server is configured to start on a different port, that information needs to be provided at the time of creating the client. If no port information is provided it will default to use port 14030.

```
Properties props = new Properties();  
props.put (SaffronClient.PROPERTY_SERVER_NAME, "myserver");  
props.put (SaffronClient.PROPERTY_SERVER_PORT, "14030");  
SaffronCORBAClient client = SaffronClient.createSaffronCORBAClient (props);
```

SaffronClient has 3 main methods in it: *connect*, *export* and *disconnect*. After instantiating the **SaffronClient**, the first method called should be *connect* which connects the client to the server and makes it ready to accept any **SaffronInputSource** for converting into one of the available output formats. The last method called on any **SaffronClient** must be the *disconnect* method.

The *export* method takes three parameters: **FormatOptions**, array of **SaffronInputSource**'s and **DestinationOptions**. Once the client connects to the Saffron Server, *export* is called, at which point, Saffron Server concatenates the **SaffronInputSource**'s into a single output document whose type and options are specified by the **FormatOptions** provided and delivers the output document to the destination specified by the **DestinationOptions**. We will learn more about these parameters in the coming sections.

FormatOptions:

FormatOptions is an abstract class. Each output format (PDF, RTF, HTML, ...) has a corresponding **FormatOptions** Class. There are 5 implementations of this class:

- 1) **PDFFormatOptions**
- 2) **RTFFormatOptions**
- 3) **HTMLFormatOptions**
- 4) **SVGFormatOptions**
- 5) **RasterFormatOptions**.

Based on the type of **FormatOptions** object provided into the *export* method, the output type is selected accordingly.

PDFFormatOptions:

When **PDFFormatOptions** are provided to the *export* method a PDF file is produced. **PDFFormatter** has two options:

linearized
fontEmbedding

When the **linearized** option is set to true, a linearized PDF is generated. When **fontEmbedding** is set to true, all “TrueType” fonts used by this document will be embedded into the PDF File.

RTFFormatOptions:

When RTFFormatOptions are provided to the *export* method, a RTF file is produced. RTFFormatter has no specific options at this time.

HTMLFormatOptions:

When HTMLFormatOptions are used, a HTML document is generated. HTMLFormatter requires 3 options:

useFrames
imageType
baseFileName

The HTML output will be a set of HTML files. There is a HTML file for each page in the document, with the baseFileName forming the basis for naming all these files. When useFrames option is set, a HTML file with the baseFileName will be generated. This file when viewed in the browser displays two frames with all the HTML files listed in the left frame, and output displayed in the right frame. When the useFrames option is not set, the HTML file corresponding to the first page is named according to the baseFileName. Due to the inability of the HTML files to render line art all the graphics are converted into images and the imageType option specifies the type of images that need to be generated. There are two supported image types: JPEG and PNG. The imageType constants are defined as static constants in the "ImageType" utility class.

SVGFormatOptions:

When SVGFormatOptions are used, a SVG document is generated. SVGFormatter requires 4 options:

useCSS
embeddedFontsOn
embeddInHTML
baseFileName

Like HTML, the SVG output will be a set of files. There is a SVG file for each page in the document, with the baseFileName forming the basis for naming these SVG files. When the embeddedFontsOn option is set to true, SVGFormatter subsets the Font and embeds the font in the SVG file. When embeddInHTML option is set to true it puts each of the SVG files in a HTML file, which allows for easy navigation through the SVG files in the browser. When useCSS option is set to true, CSS (Cascading Style Sheets) is used for proper rendering of the SVG File.

RasterFormatOptions:

RasterFormatOptions are used for generating raster image documents like JPEG and PNG. RasterFormatter requires two options:

imageType
baseFileName

Each page in the document is rendered as a separate image file, with the baseFileName forming the basis for naming these files. The output image type is based on the imageType option provided to the RasterFormatter.

DestinationOptions

DestinationOptions is an abstract class. Each destination type has a corresponding **DestinationOptions** class.

There are two different types of Destinations: a) File System and b) Streams.

There are two types of file system destination options:

ClientFileSystemDestinationOptions
ServerFileSystemDestinationOptions

Different **FileSystemDestinationOptions** are used based on the type of client (Embedded or Remote) and also based on whether the output file should be left on the server file system or whether the file should be brought back to the client machine and placed on the client file system. There is a difference between these two destination options only in the context of using the **SaffronCORBAClient**. With **SaffronEmbeddedClient**, both the destination options behave the same way, by putting the output file on the same disk.

ClientFileSystemDestinationOptions:

These options are used along with the Remote Client if the output file needs to be brought back to the client machine. CORBA is used for transferring the file, back from the server machine to the client machine. The **fullPathName** option gives the path and name of the output file on the client machine.

ServerFileSystemDestinationOptions:

These options are used along with the Remote Client if the output file needs to be left on the Server machine, instead of bringing it back to the client machine. The **fullPathName** option gives the path and name of the output file to be placed on the server machine (the machine on which the Saffron Server is running).

The **fullPathName** must be a valid path to a file on any drive of the machine, therefore please be certain that the directory structure indeed exists. **Saffron WILL NOT create any missing directories in the path.**

There is only One Stream Destination Options, in which the generated output file is obtained as **java.io.InputStream**.

StreamDestinationOptions:

This option should be used with **Embedded Client (ONLY)** to retrieve the output as a **java.io.InputStream** instead of as a file on the file system. Once the job is completed, the **getOutputAsStream** method of this class needs to be called to retrieve the **InputStream**.

The following code snippet illustrates how you could retrieve the **InputStream**:

```
StreamDestinationOptions destOptions = new StreamDestinationOptions ();  
client.export (formatOptions, slis, destOptions);  
java.io.InputStream is = destOptions.getOutputStream ();
```

To ensure proper cleanup of resources, it is very important that you close the `InputStream` once the job is complete.

SaffronInputSource:

Saffron Server can take multiple types of inputs. There are two main types of input a) a completely formatted, self contained document described using one of many document layout languages like (PostScript, PDF and FO) b) a Java object which is capable of rendering a document, i.e. a Java object which implements the `java.awt.print.Printable` Interface. A single client can concatenate many different types of above-mentioned inputs into a single output file.

SaffronInputSource is an abstract class and there are two implementations of this class:

SaffronLanguageInputSource
SaffronPrintableInputSource.

SaffronLanguageInputSource:

This object represents a **SaffronInputSource**, which has already been formatted using the document layout languages like PostScript, PDF and Formatting Objects. The current implementation of Saffron can interpret PostScript and FO. Support for RTF and PDF inputs is still under development. All the supported language types are defined as constants in the utility class called **LanguageType**. If we have a PS file, which needs to be used as a **SaffronInputSource**, then create a **SaffronLanguageInputSource** for that PS file as shown below.

```
SaffronLanguageInputSource slis = new SaffronLanguageInputSource  
(LanguageType.POSTSCRIPT, new FileInputStream (psFileName));
```

Where `psFileName` is the name of the PS (PostScript) file, which needs to be passed.

Similarly, a **SaffronLanguageInputSource** must be created for an FO input File.

SaffronPrintableInputSource:

This object represents a **SaffronInputSource** that is capable of laying out an entire document. Any Java object that implements the `java.awt.print.Printable` interface is a valid **SaffronPrintableInputSource**. The constructor of this object takes 3 parameters:

PageFormat
isPageFormatAvailable
Printable Object.

If no **PageFormat** is provided, the object will be rendered in page of size 612 x 792, with no borders and in portrait orientation. If you want a different page size, borders or different orientation for the document, a

PageFormat with the desired page size and orientation should be created and passed to the constructor. If we have an object JavaPrintEngine which implements the java.awt.print.Printable interface, we can use it as follows.

```
SaffronPrintableInputSource spis = new SaffronPrintableInputSource (new  
    JavaPrintEngine ());
```

Or

```
SaffronPrintableInputSource spis = new SaffronPrintableInputSource (null, false,  
    new JavaPrintEngine ());
```

The above line creates a SaffronPrintableInputSource with the printable interface provided, which will be rendered on a page of size 612x792 with no border in a portrait orientation.

If you want the document to be printed on a page of size 612x792 in landscape orientation, you can do that as shown below:

```
PageFormat pf = new PageFormat ();  
Paper paper = new Paper ();  
  
//removing the margins by making the imageable  
//area equal to the paper size.  
paper.setImageableArea(0, 0, 612, 792);  
  
pf.setPaper (paper);  
pf.setOrientation (PageFormat.LANDSCAPE);  
SaffronPrintableInputSource spis = new SaffronPrintableInputSource (pf, true,  
    new JavaPrintEngine ());
```

By default, a PageFormat represents a letter-sized paper with 1-inch margins in Portrait orientation. If you want to change the margins or the paper size, you can do that by creating a Paper object and by setting the size and ImageableArea on it as shown above. The above set of lines show how to create a SaffronPrintableInputSource to be printed on a letter-sized paper with no margins in landscape orientation.

Running SaffronClients

Requirements For Running Embedded Client.

SaffronEmbeddedClient requires **JRE 1.4** or higher. To run a SaffronEmbeddedClient the application requires the following jars in the CLASSPATH.

Saffron.jar
SaffronClient.jar
SaffronStubs.jar
DynaFop.jar
Svg.jar
Jai_codec.jar



Jai_core.jar
FOPSupport.jar

In addition to the above mentioned jar files, it also needs the license.key in the CLASSPATH. In the examples folder you will see a “SaffronEmbeddedClientApp”, which is a simple case of simple client app connecting and running Saffron in the same JVM. You can run it as shown below.

```
SAFF_HOME= C:\Program Files\Dynalivery\Saffron
```

```
%SAFF_HOME%\jre\bin\java -cp %SAFF_HOME%\lib\Saffron.jar; %SAFF_HOME%\lib\SaffronStubs.jar;  
%SAFF_HOME%\lib\DynaFop.jar; %SAFF_HOME%\lib\FOPSupport.jar;%SAFF_HOME%\lib\svg.jar;  
%SAFF_HOME%\lib\jai_codec.jar; %SAFF_HOME%\lib\jai_core.jar; %SAFF_HOME% -  
Dcom.dynalivery.saffronProperties= %SAFF_HOME%\saffron.properties SaffronEmbeddedClientApp
```

Where SAFF_HOME is the folder in which the Saffron is installed on the machine.

Requirements For Running a Remote Client

SaffronCORBAclient requires JRE1.3 or higher. Before running a remote client, make sure that the Saffron Server has been started. You can start the Saffron Server by double clicking on the “startSaffron.bat” in the “bin” folder of Saffron. Saffron Server by default uses JacORB as the ORB. In the examples folder you will see a “SaffronCorbaClient”, which is a simple case of simple client app connecting and running Saffron. To run a Remote Client you need JacORB jar in the bootclasspath before the rt.jar and also the following jars in the CLASSPATH.

SaffronClient.jar and SaffronStubs.jar.

In addition to the jars you need to set two more options for the JVM to use JacORB as shown below.

```
Java -Xbootclasspath/p:jacorb.jar -cp SaffronStubs.jar;SaffronClient.jar -Dorg.omg.CORBA.ORBClass=org.jacorb.orb.ORB -  
Dorg.omg.CORBA.ORBSingleton=org.jacorb.orb.ORBSingleton SaffronCorbaClient.
```



Chapter 3

Introduction to Saffron Print Service

Saffron Print Service is based on the JDK 1.4 printing framework. Saffron sets up a Print Service called “Saffron Print Service”. This is a factory object for creating print jobs. Saffron Print Service implements the *javax.print.MultiDocPrintService*. This print service is capable of handling both a *SingleDocPrintJob* and a *MultiDocPrintJob*. A Doc is a single piece of print data. A *MultiDocPrintJob* is capable of accepting a set of docs and the Saffron Print Service concatenates all of the docs into a single output file.

There are different types of print data and these different types are called DocFlavors. All these DocFlavors are represented as static constants in the **DocFlavor** class. The Saffron Print Service is capable of accepting two different types of DocFlavors they are:

DocFlavor.INPUT_STREAM.POSTSCRIPT
DocFlavor.SERVICE_FORMATTED.PRINTABLE.

Where the Doc Flavor **INPUT_STREAM.POSTSCRIPT** means the print data is formatted by the client using PostScript language and will be provided as an Input Stream to the print service. The latter **DocFlavor SERVICE_FORMATTED.PRINTABLE** means the print data will be provided as a Java object, which implements the *java.awt.print.Printable* interface and the document has to be formatted by the Print Service.

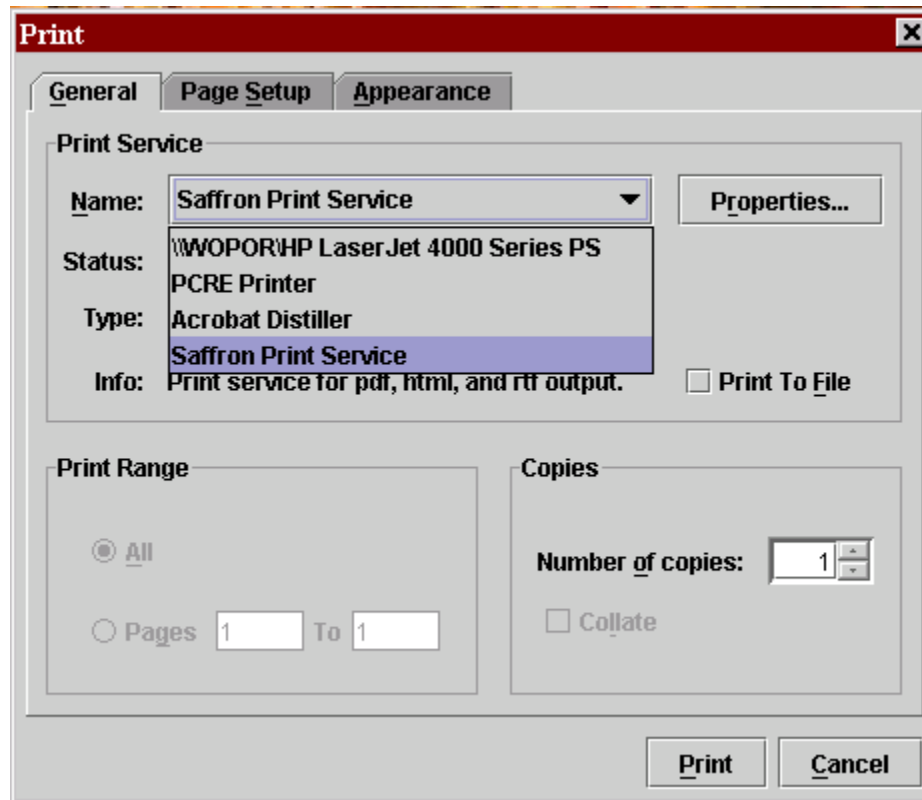
The *javax.print.PrintServiceLookupFactory* can be used to get an instance of the Saffron Print Service. This can be done in couple of ways. The *LookupFactory* can be asked to list all the print services available for the JVM or the *LookupFactory* can be asked for a list of *PrintServices*, which are capable of handling certain types of DocFlavors. The Saffron Print Service should show up in the list of *PrintServices* either way, provided that “saffron-client.jar” is in the CLASSPATH. The Saffron Print Service can be used either in user mode where a Dialog box pops up showing the available print services, the user selects the Saffron Print Service and sets the appropriate properties for the Saffron job, or the Saffron Print Service can be used in silent mode where the user sets all the properties and uses the Print Service without displaying a dialog box. The samples folder has samples illustrating both options.

```
PrintService[] services = PrintServiceLookup.lookupPrintServices(  
    DocFlavor.INPUT_STREAM.POSTSCRIPT, null);
```

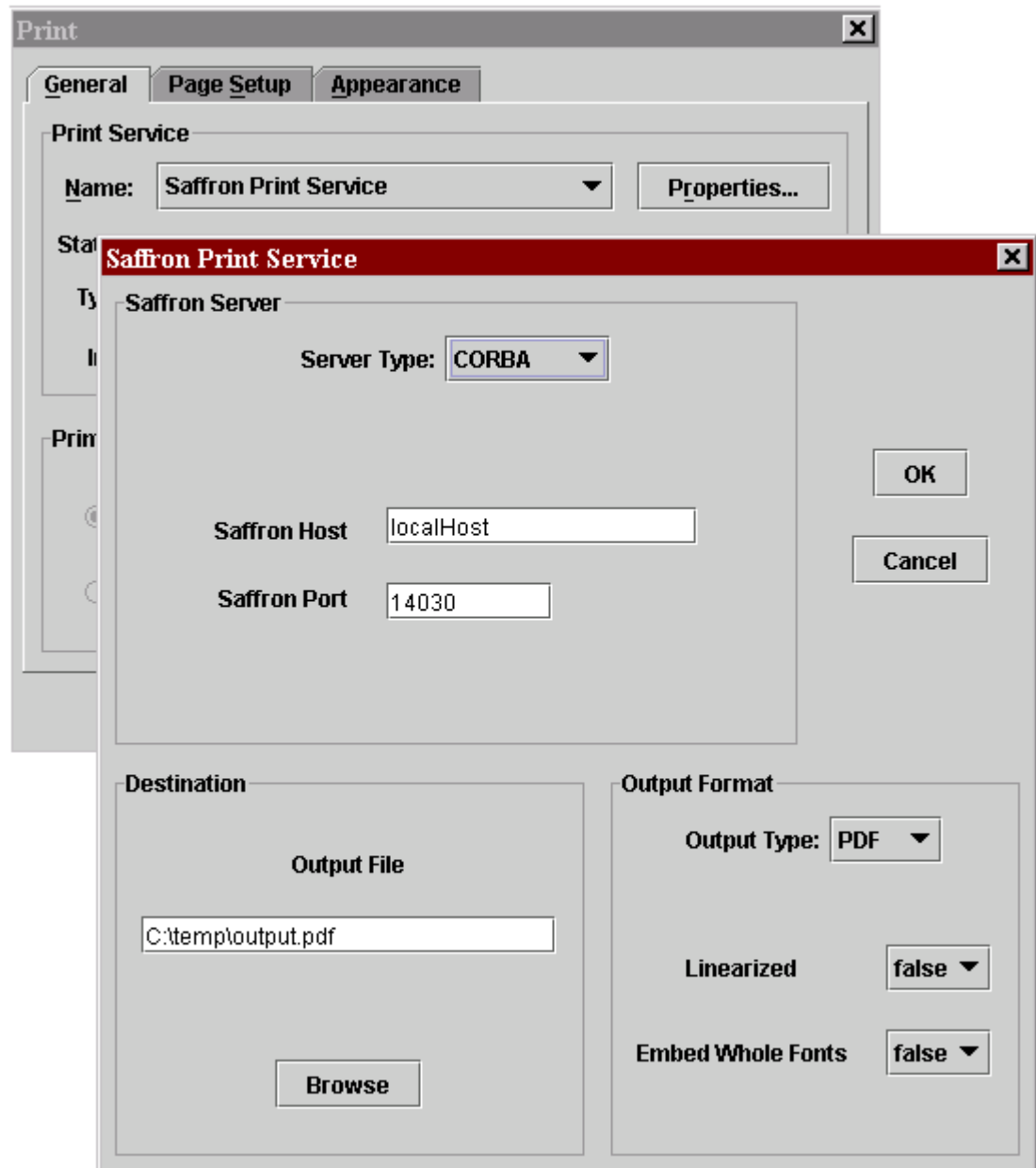
The above line of code shows how to get all the print services available to JVM , which are capable of accepting print data in PostScript language as an input stream. Saffron Print Service should be there in the array of *PrintServices* returned as long as the *saffron-client.jar* is there in the CLASSPATH. Saffron Print Service needs some more information for it to be able to connect to the Saffron Server and process the print job. This information is provided in two ways.

```
PrintRequestAttributeSet attributes = new HashPrintRequestAttributeSet();  
PrintService service = ServiceUI.printDialog(null, 50, 50, services, null,  
    null, attributes);
```

The execution of the above set of lines cause a dialog box to be displayed as shown below. Here you can see the Saffron Print Service in the list of Print Services. Information to the Saffron Print Service can be provided by clicking on the “Properties” button, which is next to the list of Print services in the picture below.



When the properties button is clicked, another dialog box is displayed as shown in the next picture where the necessary selections are made. The type of Server is selected, either “Embedded” or “Remote”. The difference between the two is best described as the JVM in which the server is running as explained in the previous chapter. The desired output type and the necessary format options, if any, for that output type, are selected. The name of the output file is to be provided. Once you click on “OK”, the PrintRequestAttributeSet object provided to the ServiceUI.printDialog method gets populated with the properties and then passed on to the Saffron Print Service.



The other way of providing the properties to a Print Service is to populate the Attribute set without having the dialog box displayed and then passing on the Attribute Set to the Print service as shown below.

```
PrintRequestAttributeSet attributeSet = new HashPrintRequestAttributeSet();  
attributeSet.add(SaffronOutputType.SAFFRON_OUTPUT_TYPE_HTML);  
attributeSet.add(HTMLImageType.SAFFRON_HTML_IMAGE_TYPE_PNG);  
attributeSet.add(HTMLUseFrames.SAFFRON_WITH_HTML_FRAMES);  
attributeSet.add(new SaffronServerType(ServerType.CORBA));
```

```
attributeSet.add(new SaffronHost(server));
File file = new File("multidoc.html");
String filename = file.getAbsolutePath();
filename = filename.replace('\\', '/');
Destination destination = new Destination(new URI("file:/// " + filename));
attributeSet.add(destination);
DocFlavor[] requiredDocFlavors = { DocFlavor.INPUT_STREAM.POSTSCRIPT,
    DocFlavor.SERVICE_FORMATTED.PRINTABLE };
printServices = PrintServiceLookup.lookupMultiDocPrintServices(
    requiredDocFlavors, attributeSet);
MultiDocPrintJob printJob = printService.createMultiDocPrintJob();
    • • •
printJob.print(multiDocContainer, attributeSet);
```

The above lines of code show how to set the properties to the Saffron Print Service without having a dialog box displayed. The above example uses a remote SaffronServer. Whenever a remote Saffron Server is used, the “SaffronHost” attribute needs to be provided. If it is not provided it defaults to the local host. By default, the Saffron Print Service will always try to connect to Saffron Server on port “14030” unless a “SaffronPort” property is provided. If you decide to use “Embedded” SaffronClient, you do not need to provide the SaffronHost and SaffronPort properties.

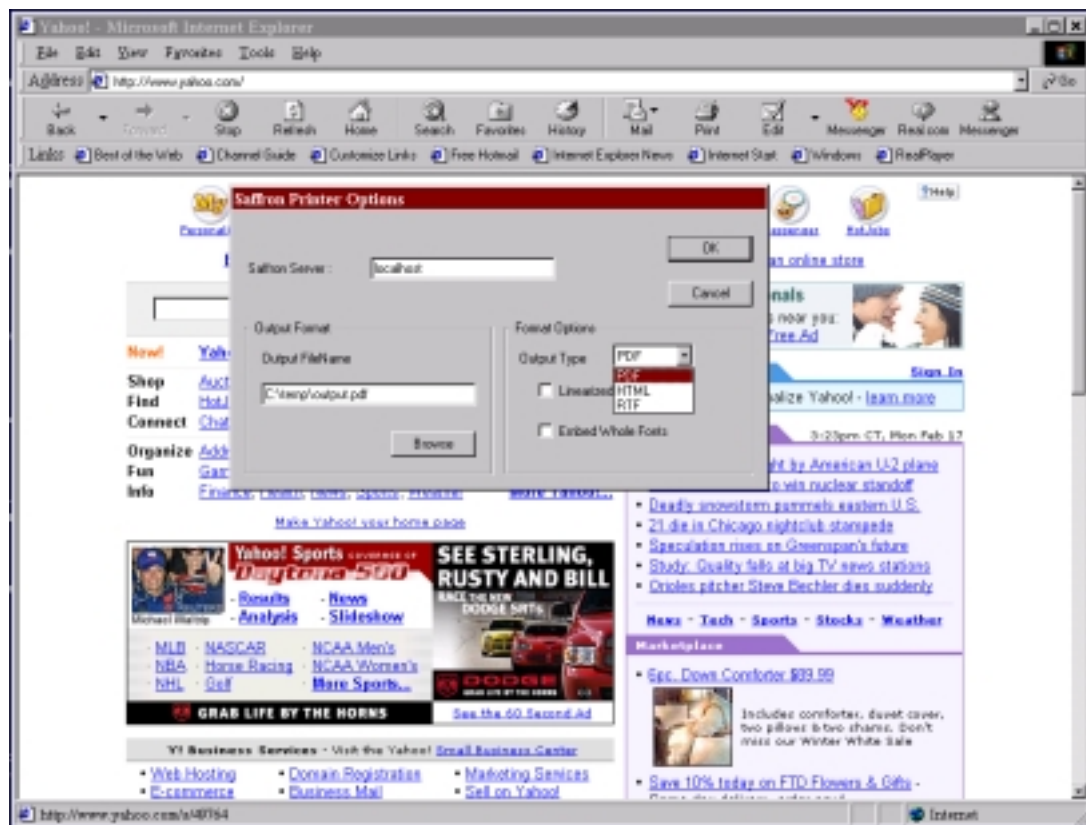
In the above example, Saffron is being asked to generate “HTML” files as output with two options. Different attributes need to be set based on the type of output desired. If no options are provided for the formatter default values are assumed. Documentation about the Saffron Print Service attributes can be found in the Saffron SDK JavaDocs.

A Java application using the Saffron Print Service is run in the same way as an Embedded Client or a Remote Client as explained in the previous chapter.

Chapter 4

Introduction to Saffron Printer (Windows Only)

The Saffron Workgroup installation installs a Saffron Printer on Windows machines. Any Windows based application (IE, MS Word, MS Excel etc...) can print to this printer to generate documents in PDF, RTF or HTML. When an application prints to this printer, a dialog box is displayed asking for the “*Saffron Server*”, the machine on which the Saffron Server is running, “*OutputFileName*” the name of the final document to be generated, the required output format type and the format options if any.

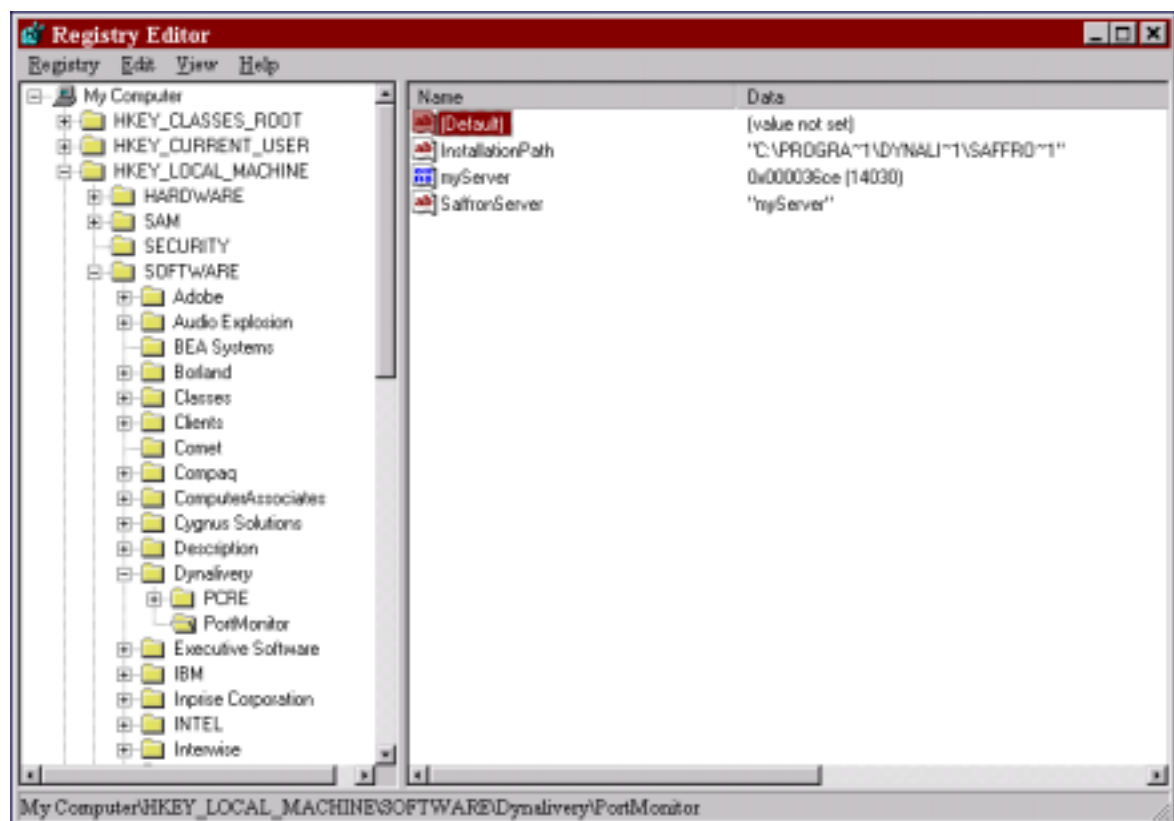


The above picture shows an example of Internet Explorer on a Windows NT machine printing to the Saffron Printer. Saffron Printer displays a dialog box as shown in the diagram above, where the necessary values are provided. Before clicking OK, make sure that the Saffron Server is running on the machine specified in the Saffron Server input box. Clicking on “Start/Programs/DynaLive/Saffron” can start Saffron Server. If for some reason a Saffron Server cannot be reached a message box is displayed with the right message.

Configuring the Saffron Printer

The Saffron Printer can be configured to use a default Saffron Server at all times, by making a registry setting. By default it will be *localhost*. If you want the Saffron printer to use *myServer* as the Saffron Server, then set the value of the *SaffronServer* key, which is of type string, to *myServer* under the *HKEY_LOCAL_MACHINE/Software/Dynamalivery/PortMonitor* registry key.

By default, the Saffron Server starts on port 14030. If the Saffron Server on machine *myServer* has been configured to start on a different port number, for example 14035 a registry key needs to be added so that the Saffron Printer will try to connect to the server on port 14035. Add a DWORD registry subkey *myServer* whose value is 14035(decimal) under the *HKEY_LOCAL_MACHINE/Software/Dynamalivery/PortMonitor* key.



Appendix A Frequently Asked Questions

Can Saffron embed TrueType fonts in PDF?

Saffron by default embeds all symbolic TrueType fonts used in the document but it doesn't embed normal TrueType fonts by default in the PDF File. Saffron at runtime takes an option which when set it embeds all the TrueType fonts used by the document, provided the font is available on the system on which Saffron is running.

What is the ORB required on the client side?

Saffron is tested and works with JacORB v1.4.1, which is distributed along with Saffron Workgroup. The Server is configured to use JacORB v1.4.1.

What JRE is required for Saffron?

Saffron Server runs in Sun's JRE 1.4. We recommend Sun's JRE 1.4 or higher on the client side, but the Remote Client can also run in Sun's JRE 1.3.

What PostScript can Saffron accept?

Saffron only accepts the PostScript generated from Dynamalivery's PostScript driver. Saffron Workgroup sets up a printer called "*SaffronDesktop Printer*" on Windows machines. When an application prints to this printer it displays a dialog box asking for the PostScript file name, in which the resulting PostScript is stored.

Over time, Dynamalivery will remove these limitations and Saffron will accept arbitrary PostScript.

Why is the Text overlapping or getting truncated in SVG Files?

Saffron takes an option called "useCSS" for SVG, which when set should show the text properly.

Why are the fonts getting substituted in the Output?

Saffron requires access to the TrueType Font files (.ttf extension files) for proper working of the Post Script interpreter and the PDF output. **saffron.properties** file has a property named "**com.dynamalivery.fonts**", which should point to the list of folders (separated by the platform dependent path separator) containing the ttf files. If the document were using a font for which Saffron is not able to locate the corresponding ttf file, the font would be substituted with **Courier** font by default.

Where can we find the TrueType Font files on the machine?

On a Windows machine, TrueType Font files are located in the "fonts" folder under Windows directory. Solaris machines do not have TrueType Font files by default. Some applications like XWindows and Open Office when installed on Solaris, install their own set of TrueType Font files. These application folders contain the ttf files and these folders can be added to the saffron.properties file for Saffron's use.

Please understand that Dynamalivery would not be providing the TrueType Font files.