Image Exporter 1.0 Component Specification

1. Design

This component provides the ability to save the image of javax.swing.JComponent to an image file (the required specification of saving the image of java.awt.Component is dropped by the forum). It supports JPEG, GIF, TIFF, BMP, PNM, PNG formats.

The core of this design is JComponentImageExporter that permits to print the jComponent's image to a Graphics and export the image to a file or an output stream. A pluggable way for visibility is provided. The default way is DefaultVisibleMaker, but the way is also configurable in the config file. For exporting the jComponent's image is necessary to implement JcomponentImageExporter. By default ImageManipulationImageExporter is provided for all formats permitted by Image Manipulation Component. The encoders are also extensible through config, if Image Manipulation changes. ImageExporter is a facade class for all exporting operations. The exporters are configurable through config file, but by default ImageManipulationImageExporter is provided for all formats.

The extendibility of this component is obtained by Object Factory componet. The new exporters and new encoders for Image Manipulation can be defined in configuration file. Through Config Manager component, these properties and other properties are read and loaded in ImageExporter class and ImageManipulationImageExporter class.

1.1 Design Patterns

Façade – ImageExporter is a facade class of exporting jComponent's image.

Strategy – The pattern is applicate in ImageExporter class when an exporter is chosen using the exporters map. The pattern is also applicate in ImageManipulationImageExporter when the encoders are chosen through the encoders map.

1.2 Industry Standards

None

1.3 Required Algorithms

1.3.1 Export a Jcomponent's image using ImageManipulationImageExporter

This algorithm is the core of the class. The purpose is to write the Jcomponent's image to a Graphics context of a BufferedImage. It's necessary that the Jcomponent is visible and displayable.A pluggable way is used for set the visibility, but this way is used in ImageExporter and not in this algorithm.

This is the detailed algorithm:

```
//Retrieve the current state doubleBuffering
RepaintManager currentManager = RepaintManager.currentManager(jComponent);
boolean doubleBufferingOld=currentManager.isDoubleBufferingEnabled();
//Set the double buffering global to off (optimization)
currentManager.setDoubleBufferingEnabled(false);

//Create the bufferedImage using the input.
BufferedImage bufferedImage = new
BufferedImage(jComponent.getWidth(),jComponent.getHeight(),imageType);
```

```
        //Create graphics for printing the component to an Image
        Graphics2D graphics = bufferedImage.createGraphics();

        //Use the rendering hints (optimizations) if they exists
        graphics.setRenderingHints(renderingHints);

        //Print the image of the component to graphics of bufferedImage
        jComponent.print(graphics);

        //Capture only the rectangle area from the input. If the rectangle area covers entirely the component
then x=y=0
        //This variable is the field that will be used for the exporting.
       Image image=new
       MutableMemoryImage(bufferedImage.getSubimage(viewport.x,viewport.y,viewport.width,viewport.height))
       ;

        //Retrieve the encoder from encoders map and encode the image with the specified format
       BaseImageEncoder encoder=encodersMap.get(format);
        encoder.encode(image,outputStream,null);

       currentManager.setDoubleBufferingEnabled(doubleBufferingOld);
```

Catch every Exceptions derived from this algorithm, wrap it through a JComponentPrintImageException and throws this last.

1. Component Class Overview

com.topcoder.swing.imageexporter

ImageManipulationImageExporter

The purpose of this class is to export the jComponent's image through the encoders' package in Image Manipulation Component. For obtain this, the component print its image to a graphics of BufferedImage and this last is used in a MutableMemoryImage instance. Several optimizations are possible through RenderingHints object. Other encoders can be used by simply added them to encoders map: this mechanism permits to extend the encoders, if image manipulation changes, only using the config file of the component.

ImageExporter

This is a Façade class, a convenience utility for exporter JComponent(s) into an image file(s) or outputStream(s). When a jComponent is exported, the method controls if the format exists in exporters map. So the exportes map is a way for extend the component with new JComponentImageExporters. The main method for exporting is exportComponent(components:List<JComponent>, viewPort:Rectangle, outputStreams:List<OutputStream>, format:String) and is called by other methods.

JComponentVisibleMaker

This interface defines the contract for render visible a jComponent. The methods are called by ImageExporterClass before the print method. makeVisibile(...) is the firts method called, after the print mechanism occurs and then restoreVisibility(...) is called

DefaultVisibleMaker

This is the default way for render visibility of a JComponent. The default way is to do nothing if the component is visibile and if it isn't set the visible just a moment. This is for maintaining the correct layout of the jComponent's container, without removing the jComponent.

JComponentImageExporter

This interface defines the contract for exporting a jComponent's image. For extendibility using Object Factory, it's expected that the classes that implement this interface have an empty constructor.

1.5 Component Exception Definitions

ImageExporterException

This is the general exception of this component. It is thrown when some error occurs during the exporter process.

JComponentVisibleMakerException

This exception is thrown by JComponentVisibleMaker when some error occurs.

ImageExporterConfigurationException

This exception is thrown when a constructor read from config file and some error occurs: for example the config file is malformed.

1.6 Thread Safety

The normal scenario of this component is used in UML Application in a single thread for exporting the images of jComponents, so the thread safe is not required. But all the properties of the components are loaded through the constructor and all method of the facade are synchronized. So the normal use, through the facade, guarantees the thread safe.

2. Environment Requirements

2.1 Environment

Java 1.5 is required for compilation and executing test cases.

2.2 TopCoder Software Components

Base Exception 1.0 : for the custom exceptions.

Configuration Manager 2.1.5 : it's used for load all parameters from configuration file in ImageExporter and ImageManipulationImageExporter classes.

Object Factory 2.0.1: for provide the extendibility of the component : create new exporters and new encoders.

Image Manipulation 1.0: for perform the encode of the JComponent's image in several formats

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

- **Java Advanced Imaging (JAI)**

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.topcoder.swing.imageexporter

3. Configuration Parameters

These are the parameter for Image Exporter namespace

Parameter	**Description**	**Values**
exportersMap	A multivalue property that define the map of new exporters for ImageExporter.	several String that represents object_factory_key_1 object_factory_key is the key for instantiate the exporter by object factory *optional*
jComponentVisibleMaker	The visibility way for set he Jcomponent visibile.	a String that represents the key for Object Factory *optional*
objectFactoryNameSpace	The namespace for Object Factory component.	a String that represents the namespace for Object Factory *optional* or *required* if the **exportersMap** or **jComponentVisibleMaker** exists.

These are the parameters for “ImageManipulationImageExporterProperties” namespace.

encodersMap	A multivalue property that define tha map of new encoders of Image Manipulation Component.	<image_format>,<object_Factory_key>; image_format is a String that represents the image format object_Factory_key is a String that represent the key for instantiate the encoder by Object Factory *optional*
imageType	The image type used for construct the BufferedImage.	String: the name of a valid constant from BufferedImage class *optional*
renderingHints	A multivalue property that define rendering hints for render the Jcomponent's image for ImageManipulationImageExporter	<rendering_hints_key>,<rendering_hints_value> rendering_hints is a valid name RenderingHints.Key from constants of RenderingHints class

		rendering_value is a valid value name for RenderingHints.Key from constants of RenderingHints class *optional*
objectFactoryNameSpace	The namespace for Object Factory component.	a String that represents the namespace for Object Factory *optional* or *required* if the encodersMap exists.

3.3 Dependencies Configuration

Image Manipulation needs to be configured properly.

Object Factory needs to be configured properly.

4. Usage Notes

4.1 Required steps to test the component

Extract the component distribution.

Follow .

Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Follow configuration instructions.

4.3 Demo

1. Use the ImageExporter utility class for exporting jcomponent(s)'s image(s).

Exports images to files:

```
        //exports a JComponent image to a file
        exporter.exportComponent(jComponent, OUTPUT_DIR,
"test_export_to_file.jpg", ImageExporter.JPEG_FORMAT);

        //exports a JComponent image to a file, capturing only a rectangle
area of jComponent's image
        exporter.exportComponent(jComponent, viewport, OUTPUT_DIR,
"test_export_to_file_viewport.gif",
            ImageExporter.JPEG_FORMAT);

        //exports a list of JComponents to files
        List<JComponent> jComponentsList = new ArrayList<JComponent>();
        jComponentsList.add(jComponent);

        List<String> fileNamesList = new ArrayList<String>();
        fileNamesList.add("test_export_to_file_list_1.jpg");

        exporter.exportComponent(jComponentsList, OUTPUT_DIR,
fileNamesList, ImageExporter.JPEG_FORMAT);

        //exports a list of JComponents to files, capturing only a rectangle
area of jComponents' images
        fileNamesList = new ArrayList<String>();
        fileNamesList.add("test_export_to_file_viewport_list_1.jpg");
        exporter.exportComponent(jComponentsList, viewport, OUTPUT_DIR,
fileNamesList, ImageExporter.JPEG_FORMAT);
```

Export images to output streams:

```
        //uses FileOutputStream to export the component
        os = new FileOutputStream(OUTPUT_DIR +
"test_export_to_outputStream.bmp");
        //exports a JComponent to the outputStream
        exporter.exportComponent(jComponent, os,
ImageExporter.BMP_FORMAT);

        //uses FileOutputStream to export the component
        os = new FileOutputStream(OUTPUT_DIR +
"test_export_to_outputStream_viewport.bmp");
        //exports a JComponent to an outputStream with viewport restricted
        exporter.exportComponent(jComponent, viewport, os,
ImageExporter.BMP_FORMAT);

        //exports to a list of output streams
        List<OutputStream> outputStreamsList = new
ArrayList<OutputStream>();
        outputStreamsList.add(new ByteArrayOutputStream());

        List<JComponent> jComponentsList = new ArrayList<JComponent>();
        jComponentsList.add(jComponent);
        exporter.exportComponent(jComponentsList, outputStreamsList,
ImageExporter.BMP_FORMAT);

        //exports to a list of output streams with viewport restricted
        outputStreamsList = new ArrayList<OutputStream>();
        outputStreamsList.add(new ByteArrayOutputStream());

        jComponentsList = new ArrayList<JComponent>();
        jComponentsList.add(jComponent);
        exporter.exportComponent(jComponentsList, viewport,
outputStreamsList, ImageExporter.BMP_FORMAT);
```

2. Use ImageExporter with custom namespace

```
ImageExporter imageExporter=new ImageExporter("topcodernamespace");
```

4. Example of configuration file

```
<?xml version="1.0" ?>
<CMConfig>
        <Config name="com.topcoder.swing.imageexporter">


                <Property name="exportersMap">
                        <Value>myExporter1</Value>
                        <Value>myExporter2</Value>
                </Property>

                <Property name="jComponentVisibleMaker">
                        <Value>myVisibleMaker</Value>
                </Property>

                <Property name="objectFactoryNamespace">
                        <Value>objectFactoryNamespaceImageExporter</Value>
                </Property>
        </Config>


        <Config name="ImageManipulationImageExporterProperties">

                <Property name="imageType">
                        <Value>TYPE_INT_RGB</Value>
                </Property>

                <Property name="renderingHints">
                        <Value>
                                KEY_FRACTIONALMETRICS,VALUE_FRACTIONALMETRICS_ON
                        </Value>
                        <Value>KEY_ANTIALIASING,VALUE_ANTIALIAS_ON</Value>
                </Property>

                <Property name="encodersMap">
                        <Value>NEW_FORMAT_A,myEncoderA</Value>
                        <Value>NEW_FORMAT_B,myEncoderB</Value>
                </Property>

                <Property name="objectFactoryNamespace">
                        <Value>
                                objectFactoryNamespaceImageManipulationImageExporter
                        </Value>
                </Property>

        </Config>

        <Config name="objectFactoryNamespaceImageExporter">
                <Property name="myExporter1">...</Property>

                <Property name="myExporter2">...</Property>

                <Property name="myVisibleMaker">...</Property>

        </Config>

        <Config
                name="objectFactoryNamespaceImageManipulationImageExporter">

                <Property name="myEncoderA">...</Property>

                <Property name="myEncoderB">...</Property>

        </Config>
</CMConfig>
```

5. Future Enhancements

It's possbile to add other formats, example: SVG format or PDF.