Triana Command Line Application Tutorial

Ian Wang

13th December 2003

1 Introduction

In this tutorial we describe how workflows created within Triana can be packaged as standalone applications that can be executed from the command line, using a simple image processing workflow as an illustration. This tutorial assumes a full installation of Triana, which is available at `www.trianacode.org`.

2 Getting Started

Before a standalone command line application can be generated, the workflow that will be executed must first be created. As with any Triana workflow, this is done by dragging the required tools from the tool tree onto the workspace and then connecting cables between the tools as required. Note that only self-contained workflows (no group input/output nodes) can be turned into command line applications.

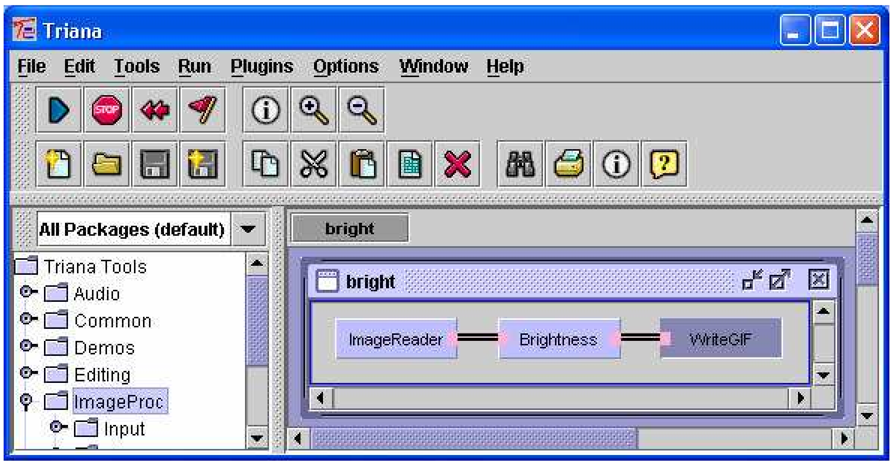


Figure 1: Example image processing workflow for altering the brightnesss on a GIF file.

In this tutorial we use a simple image processing workflow in our examples. This workflow, illustrated in Figure 1, involves connecting ImageReader, Brightness and WriteGIF tools; these tools can be found in the ImageProc.Input, ImageProc.Processing.Effects and ImageProc.Output toolboxes respectively. The fairly obvious purpose of this workflow is to alter the brightness of an image and save it as a GIF file

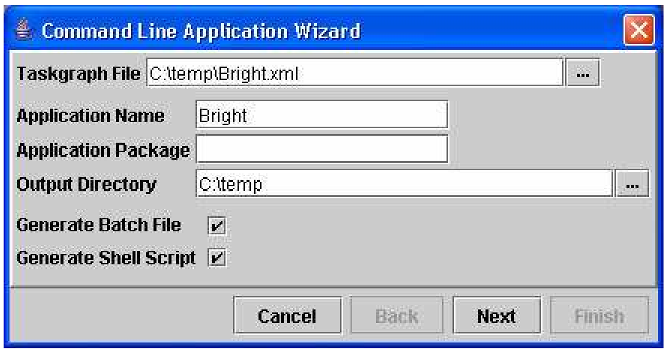
Once the workflow is created it should be saved (using the *File→Save* menu option). In our example we save the workflow in a file called Bright.xml (in the C:\temp directory).

3 Command Line Application Wizard

The command line application wizard can be started using the *Tools→Generate Command Line Application* menu option.

3.1 Taskgraph and Output Files

The initial panel in the command line application wizard (see Figure 2) allows the taskgraph (workflow) file that will be executed by the standalone application to be specified, and also the application name, package and output directory for the generated Java class.

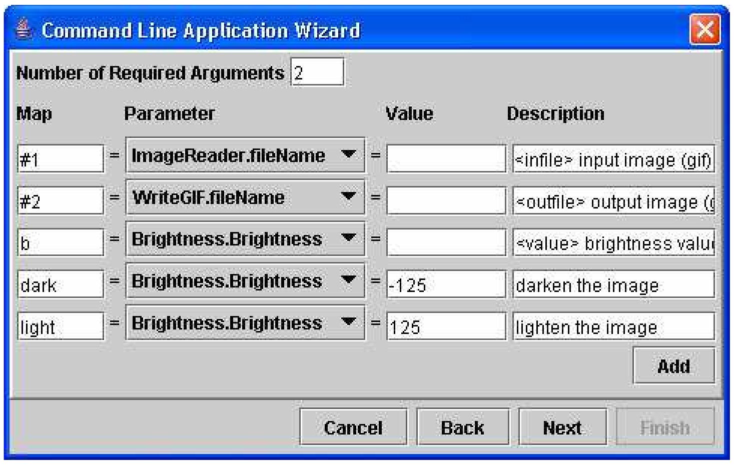

Command Line Application Wizard

Taskgraph File: C:\temp\Bright.xml
Application Name: Bright
Application Package:
Output Directory: C:\temp
Generate Batch File: ✓
Generate Shell Script: ✓

Figure 2: The taskgraph and output files panel on the command line application wizard.

When the taskgraph file is set, the command line application wizard suggests sensible defaults for the application name and output directory. In our example, the taskgraph file is set to C:\temp\Bright.xml, the suggested default for the application name is Bright and the suggested default for the output directory is C:\temp. If these defaults are accepted, as we shall assume, then the Java class generated will be C:\temp\Bright.java.

In addition to generating a Java class, the command line application wizard offers the option to generate a batch file for executing the application on Microsoft Windows based systems, and the option to generate a shell



Number of Required Arguments 2

Map		Parameter		Value	Description
#1	=	ImageReader.fileName	=		<infile> input image (gif)
#2	=	WriteGIF.fileName	=		<outfile> output image (
b	=	Brightness.Brightness	=		<value> brightness valu
dark	=	Brightness.Brightness	=	-125	darken the image
light	=	Brightness.Brightness	=	125	lighten the image

Figure 3: The parameter mapping panel on the command line application wizard.

script for executing the application on UNIX based systems. If these options are enabled, then the generated files inherit the application name and output directory; in our example these would be C:\temp\bright.bat (batch file) and C:\temp\bright (shell script). Application execution is discussed further in Section 5.

3.2 Parameter Mappings

The second panel in the command line application wizard allows mappings between command line arguments/options and internal Triana parameters to be specified (see Figure 3). For example, using our image processing workflow, if we wish our application user to be able change the brightness of an image using a `-b` option on the command line, then we map `b` to the parameter Brightness in the tool Brightness (*Brightness.Brightness*). From the command line the application user can then execute:

```
bright -b 125 in.gif out.gif
```

When this is executed the value of the Brightness parameter in the Brightness tool is set to 125. From the help documentation for Brightness, we see can that this means the images brightness will be increased by 125points (out of 255).

In the example above the user specifies a value for the `-b` option at the command line, an alternative is to associate a set value with an option. For example, if we want a `-dark` option to darken the input image by 125points then we can map `dark` to the parameter *Brightness.Brightness* and to the value -125. The application user can then simply execute:

```
bright -dark in.gif out.gif
```

When this is executed the value of the Brightness parameter in the Brightness tool is set to -125, the equivalent of specifying `-b -125`. Note that more than one parameter/value pair can be mapped to a single map string.

So far in this section we have concentrated on the option maps (`-b` and `-dark`); however the command line arguments (`in.gif` and `out.gif`) must also be mapped. To do this the number on required arguments is first set, in our example this is two (`in.gif` and `out.gif`). Once this is done, the map strings `#1` and `#2` can be mapped to the internal Triana parameters for the first and second arguments respectively. In our example, `#1` should be mapped to the input file name on the ImageReader tool (*ImageReader.fileName*), and `#2` mapped to the output file name on the writeGIF tool (*writeGIF.fileName*). Note that extra arguments in addition to the required arguments (`#3`, `#4` etc.) can also be mapped but are not *required* at runtime.

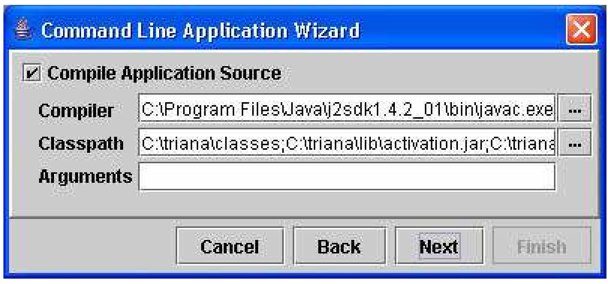
The command line wizard allows a description for each option/argument to be specified. This description is shown when the command line application is run with a `-?` option. For arguments/options where a value is required, this description should start with a marker for the type of value required enclosed with < and >. For `#1` in our example an appropriate description would be:

```
<infile> input image file (gif)
```

and for the `-b` option an appropriate description would be:

```
<value> brightness value (min -255, max 255)
```

The `-dark` option would not require a value type (*infile*, *value* etc.) to be specified as it does not expect a value.


Command Line Application Wizard

☑ Compile Application Source

Compiler: C:\Program Files\Java\j2sdk1.4.2_01\bin\javac.exe

Classpath: C:\triana\classes;C:\triana\lib\activation.jar;C:\triana

Arguments:

Figure 4: The compilation panel on the command line application wizard.

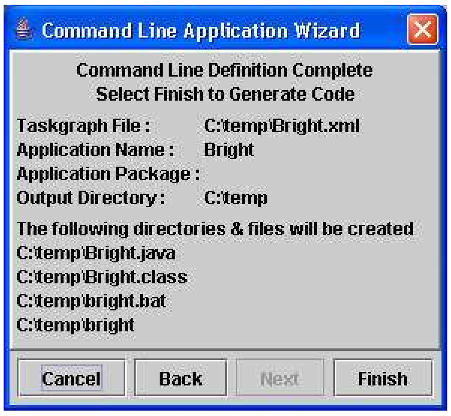
4 Compiler and Classpath

The third panel in the command line application wizard is used to specify whether the Java class generated by the wizard is automatically compiled (see Figure 4). If automatic compilation is chosen then the location of the Java compiler (javac), the compiler classpath and the compiler arguments are required, however these should automatically be detected by the wizard.

If automatic compilation is disabled then the application Java class must be compiled manually from either the command line or using an IDE. In either case the classpath used when compiling must include the main Triana class directory and all the jar files contained in Trianas lib directory.

4.1 Code Generation

The final panel on the command line application wizard provides a summary of the input and output file details (see Figure 5), including which new files will be generated and which existing files will be overwritten. It is only after finish is selected from this panel that any code is generated.



Command Line Definition Complete
Select Finish to Generate Code

Taskgraph File :	C:\temp\Bright.xml
Application Name :	Bright
Application Package :	
Output Directory :	C:\temp

The following directories & files will be created
C:\temp\Bright.java
C:\temp\Bright.class
C:\temp\bright.bat
C:\temp\bright

Figure 5: The code generation panel on the command line application wizard.

5 Execution

There are several ways that a generated and compiled application can be executed. The most generic way is to use the `java` execution command directly, for example (from C:\temp directory):

```
java Bright -b 125 in.gif out.gif
```

Note that for this method to work the java classpath must include the main Triana class directory and all the jar files contained in Trianas lib directory.

On Microsoft Windows based systems the generated batch file can be used to execute the application, and on UNIX based machines the generated shell script can be used (assuming batch file/shell script generation was enabled on the wizard). The command for application execution using the batch file and shell script is identical, for example (from C:\temp directory):

```
bright -b 125 in.gif out.gif
```

The batch file/shell script should automatically set the correct execution classpath; if this fails then ensure that the TRIANA environment variable is correctly set to the base Triana directory.