JGimp User Manual
version 0.8

Michael Terry
scrumpty@sourceforge.net

15th May 2003

Contents

Chapter 1

Introduction

JGimp is an architecture that allows developers to create Java- and Jython-based plug-ins and extensions for the GIMP. Plug-ins allow developers to add functionality to the GIMP (through "filters" that operate on a given image or layer), while extensions are Java or Jython applications that use the GIMP as a "back-end" service providing image manipulation capabilities.

1.1 Feature summary

JGimp offers developers the following features:

- Plug-ins for the GIMP can be written in Java or Jython, allowing developers to use the rich set of UI techniques, widgets, libraries, and language features available for these two languages. The Desaturate plug-in provides one example of a filter with a dialog box written completely in Java, including a preview.
- Java/Jython code can use the GIMP as an image manipulation engine/service by implementing the "JGimpExtension" interface. This essentially gives Java/Jython developers the full power of the GIMP within their own application. Possible uses of this capability include:
  - Developing Java/Jython-based image manipulation server applications for dynamically-generated graphics
  - Creating new image manipulation applications in Java/Jython using the GIMP as the engine

1.2 System requirements

JGimp runs on the Linux or Windows platforms, and has the following requirements:

- JDK 1.4 or later
- GIMP 1.2.2 or later
- Optional: Jython 2.1 or later (http://www.jython.org)
- A healthy amount of RAM :)

To build it from source, you will also need:

- Ant 1.5.1 or later (http://ant.apache.org)

- GCC and GNU make for Linux, Microsoft Visual C++ 6.0 for Windows

The versions listed above represent the development versions we use; you might be able to compile JGimp using other versions of the software. For example, there is not much dependent on JDK 1.4 at this point, so a port to versions JDK 1.2 or 1.3 could probably be made with minimal effort (you would have to change how the JVM is loaded in the JNI code in jgimp.c, and change any 1.4-specific code in the Java code itself).

1.3 Quick start

Two sample Java plug-ins, ImageDividerPlugIn and DesaturatePlugIn, illustrate how to use this architecture to write Java-based plug-ins. The source for these plug-ins can be found in `jgimp/src/java/edu/gatech/cc/mterry/plugin`. Two Jython scripts, `imagedividernaive.py` and `imagedivider.py`, show the equivalent in Jython. They are located in `jgimp/src/jython`.

The included Java and Jython plug-ins demonstrate the basics of how to use this architecture: how to make PDB calls (the GIMP's procedural database, which keeps track of all functions and plug-ins), how to read and write pixels from images in the GIMP, how to install the plug-in in the GIMP, and so on. A README file in `jgimp/src/java/edu/gatech/cc/mterry/plugin` provides a brief overview of the process for Java plug-ins, including where to copy your plug-ins once you have made them. The jython script directory, `jgimp/src/jython` also has sample scripts and a README offering the type of information for Jython scripts. All examples are thoroughly commented and can easily serve as the basis for new plug-ins.

Aside from these examples, there are two other resources included with this distribution: the javadocs and this manual. Most of the Java API is documented. To generate the javadocs, go to `jgimp/src/java` and type `ant javadocs`. This will create the javadocs and place them in `jgimp/src/java/dist/javadoc`.

1.4 Project history

This project was originally started to support research into alternative interfaces for image manipulation. The idea was to prototype novel interfaces in Java, using the GIMP as the image manipulation engine. For more information about this research, see the section entitled "Side Views and Supporting Open-Ended Tasks" at http://www.cc.gatech.edu/~mterry/papers.

We are currently working on a version that integrates directly with the GIMP, rather than being loaded as a plug-in. By taking this tact, we can gain access to some internal GIMP information, such as the entire image rendering, which is not possible with the current plug-in architecture. Access to these data is essential to building full-fledged, standalone Java-based image manipulation applications that use the GIMP as their backend. This direct patch is still very alpha at this time.

1.5 Getting further help

If you have a question that is not answered by the docs or code, you can send a question to our discussion list: jgimp-developers@lists.sourceforge.net

Chapter 2

Installing JGimp

At a minimum, to run (not compile) JGimp, you need JRE 1.4 or later and GIMP 1.2.2 or later. Other versions of GIMP in the 1.2.X series may work as well. Platform-specific instructions follow for compiling and installing JGimp on Linux and Windows platforms.

TBD: NEED INSTRUCTIONS FOR JYTHON COMPILATION later in this document

2.1 Linux installation

To compile JGimp you need:

- Ant 1.5.1 or later (http://ant.apache.org)
- JDK 1.4 or later
- a recent version of GCC (we use 3.2.2 as of this writing)
- GNU make
- optionally, Jython 2.1 (http://www.jython.org) or later

Other versions of Ant, GCC, and Jython may work as well; these are the versions we use and thus know work.

Once these tools are installed, follow these steps:

1. From the root directory, run `configure`. You will probably need to pass in at least two flags: `--prefix` and `--with-jdk-prefix`. If you want Jython support, add `--with-jython` and make sure `jython.jar` (the main jar file distributed with Jython 2.1) is in your CLASSPATH (or it simply won't compile):

   (a) `--prefix` should point to the base directory of your GIMP installation

   (b) `--with-jdk-prefix` should point to the base directory of your Java installation

   For example:
   ```
   ./configure --prefix=/usr/local/gimp \
   --with-jdk-prefix=/usr/local/java
   ```

   (c) or, with Jython support:

   ```
   ./configure --prefix=/usr/local/gimp \
   --with-jdk-prefix=/usr/local/java --with-jython
   ```

2. Type `make`. If all goes well, both the Java code and C code will compile into their respective libraries.

3. Next, type `make install`. This will install JGimp within the GIMP's directory, so be sure you have permissions for that directory.

4. If you compiled with Jython support, make sure `jython.jar` is in your CLASSPATH, or simply copy it to the same location as the other JGimp jar files (`GIMP/lib/gimp/1.2/jgimp/javafiles`, where GIMP is the base directory of your GIMP installation).

2.2 Windows installation

This section provides instructions for installing JGimp using the installer, compiling the source, and installing the binaries by hand. In all cases, you must follow the step in the next section.

2.2.1 Putting jvm.dll's directory in your PATH

Whether you are using an installer or installing the binaries by hand, you *must* have the parent directory of `jvm.dll` in your PATH. This dll is needed by `jgimp.exe` to start up the Java virtual machine. `jvm.dll` is part of the Java package, and is typically found in `JAVA_BASE_DIR\jre\bin\client` (e.g., `c:\j2sdk1.4.1_01\jre\bin\client` for the SDK, `c:\Program Files\Java\j2re1.4.1_01\bin\client` for the JRE only). Locate this file and note the directory in which it resides. Then go to the Control Panel, open up the System Control Panel, go to the "Advanced" tab, click on "Environment Variables" and then add the path to `jvm.dll` to the System-wide PATH variable (making sure that it is separated by the other paths with a ";").

2.2.2 Using the installer

TDB

2.2.3 Compiling from source

To compile JGimp from source, you need the following:

- Microsoft Visual C++ 6.0
- JDK 1.4 or later
- Ant 1.5.1 or later (http://ant.apache.org/)
- GIMP, version 1.2.2 or later
- gimp-dev and glib-dev libraries (http://www.gimp.org/~tml/gimp/win32/downloads.html; at the time of this writing, these files are http://www.gimp.org/~tml/gimp/win32/gimp-dev-1.2.4-20030115.zip and http://www.gimp.org/~tml/gimp/win32/glib-dev-2.2.1)
- optionally, Jython 2.1 or later (http://www.jython.org)

Once you have everything, you will build the library in two steps: first, the Java portion, then the C-library. You must follow the steps in the order given, since the Java build process creates files the C-library depends on. To compile everything:

1. Create the directory `jgimp/Windows_port/support_files`.

2. Unzip the gimp-dev and glib-dev zip files into `jgimp/Windows_port/support_files/gimp_dev` and `jgimp/Windows_port/support_files/glib_dev`, respectively. You should end up with a set of directories under `gimp_dev` that include `bin`, `include`, `lib`, and `man`, with a similar set of directories under `glib_dev`.

3. From a command prompt, change into `jgimp/src/java` and if:

   (a) you *do not* want Jython support, just type `ant`.

   (b) you *do* want Jython support, first make sure `jython.jar` is in your CLASSPATH. jython.jar must be in your CLASSPATH at both compile and runtime, so you should add it to the environment variables for the entire system. Once it is in your CLASSPATH, type `ant all-jython`.

4. To build the javadocs, type `ant javadocs`.

5. With the Java portion built, you can now build the C-portion of the library. Open up the project workspace: `jgimp/Windows_port/Windows_port.dsw`.

6. You will need to set up some paths within the project to customize to your own particular system:

   (a) Open up the project settings (Project->Settings)

   (b) Select "All Configurations" in the "Settings For:" drop-down box

   (c) Go to the C/C++ tab and select "Preprocessor" from the Category drop-down

   (d) Make sure your "Additional include directories" include the following paths:

      i. the path to the gimp development include directory (should be something like `..\support_files\gimp_dev\include`)

      ii. the path to the glib development include directory (should be something like `..\support_files\glib_dev\include\glib-2.0`)

      iii. the path to the glib development lib include directory (should be something like `..\support_files\glib_dev\lib\glib-2.0\include`)

      iv. the path to the JNI include directory (something like `c:\j2sdk1.4.1_01\include`)

      v. the path to the JNI win32-specific directory (something like `c:\j2sdk1.4.1_01\include\win32`)

   (e) Once you have configured all those paths, the final set of include directories should look something like this:

```
..\support_files\gimp_dev\include,
..\support_files\glib_dev\include\glib-2.0,
..\support_files\glib_dev\lib\glib-2.0\include,
c:\j2sdk1.4.1_01\include,
c:\j2sdk1.4.1_01\include\win32
```

7. Make sure that JGIMP_COMPILE_AS_PLUGIN is defined by the preprocessor in the Preprocessor definitions text box (in the same tab as in the above step).

8. Make sure you have put `jvm.dll`'s parent directory in your `PATH`. See Section 2.2.1.

9. Set the active configuration to "jgimp - Win32 Release" (in Build -> Set Active Configuration...).

10. Build all. There will be some warnings, but everything should compile if you have all the paths set up correctly.

2.2.4 Installing the compiled binaries by hand

To manually install the binaries you have built, you need to copy the files into the GIMP program directory. GIMP will be installed in some place like `c:\Program Files\GIMP`. We'll abbreviate that to `GIMP` in the following instructions.

1. It's assumed you have built (or have the binaries for) `jgimp.exe`, `jgimp.jar`, `desaturate.jar`, and `image_divider.jar`. If you compiled all this by hand (as described above), then the jar files will be in the `jgimp/src/java/dist` directory. `jgimp.exe` will be in the `jgimp/Windows_port/jgimp/Release` directory.
2. Copy `jgimp.exe` to the plug-ins directory of the GIMP (e.g., `GIMP\lib\gimp\1.2\plug-ins`).
3. Make a directory under `GIMP\lib\gimp\1.2` called `jgimp`.
4. Under `jgimp`, make a directory named `javafiles`.
5. If you want Jython support (and you compiled it in), create another directory off of `jgimp` called `jython_scripts`.
6. It is important to get the name and punctuation correct when creating these directories, as JGimp will look here to load its various support files.
7. Copy `jgimp.jar`, `desaturate.jar`, and `image_divider.jar` to `GIMP\lib\gimp\1.2\jgimp\javafiles`. Optionally, you can also copy `jython.jar` to the same directory, if you don't want to put it in your CLASSPATH.
8. For Jython support, copy the scripts from `jgimp\src\jython_scripts` to `GIMP\lib\gimp\1.2\jgimp\jython_scripts`.
9. Make sure jvm.dll's directory is in your PATH. See Section 2.2.1.

2.3 Testing the installation

1. Try starting up the GIMP and cross your fingers :) If everything went well, then you should have some new Java-based filters installed. Open up an image, then right-click on it. Check to see if Filters->Digital Cameras->Nikon Image Divider is there, and check if Filters->Colors->Desaturation Tools->Desaturation Dialog... is there. If those two items are there, then you've installed everything correctly! If you've installed Jython support, then there will also be Jython versions of the Nikon Image Divider plug-in.
2. Now you can install your own Java-based plug-ins and extensions in the `GIMP/lib/gimp/1.2/jgimp/javafiles` directory. You can also put Jython scripts in the `GIMP/lib/gimp/1.2/jgimp/jython_scripts` directory.

Chapter 3

Making Your Own Plug-Ins and Extensions

More information will be available here, time permitting... In the meantime:

- To create a plug-in, implement the `org.gimp.jgimp.plugin.JGimpPlugIn` interface and use the sample Java plug-ins in `jgimp/src/java/edu/gatech/cc/mterry/plugin` as a guide on how to construct your own Java-based plug-ins.
- To write a Jython script, check out the `README` in `jgimp/src/jython` and use the sample Jython scripts in that directory as a guide.
- Be sure to check out the javadocs for the library. You can build these by `cd`'ing into `jgimp/src/java` and typing `ant javadocs`. They will be placed in `jgimp/src/java/dist/javadoc`.

Some random notes until better documentation comes along:

- You can place any support jar/class files needed by your plug-in in the same directory as your plug-ins (`GIMP/lib/gimp/1.2/jgimp/javafiles`). JGimp will automatically load them.
- JGimp will add the environment's CLASSPATH to its own CLASSPATH when searching for classes.
- JGimp sets two keys in the `gimprc` file: `jgimp_javafiles_dir` and `jgimp_jython_script_dir`. You can customize these to point to other directories, if you wish.
- The javadocs for JGimp use an extra javadoc tag, @pdb, which is used to indicate which GIMP PDB functions are directly called by that method. This information is listed in the generated javadocs.
- Jython automatically converts return parameter lists having a single element into an atomic element (rather than as a 1-element list).

Some notes on the interactive Jython interpreter:

- It does not allow you to install plug-ins at this time.
- It always appears when booting up if you have Jython support installed (no menu entry yet).

- It automatically sets the variable `gimpApp` to point to a org.gimp.jgimp.GimpApp object. From this object you can do everything else. For example:
  - `myImage = gimpApp.createRGBImage(100, 100)` creates a new RGB image of 100x100 pixels
- You can type `importPlugInNames()` at the prompt and it will import all PDB names into the namespace. Then you can do things like:
  - `myImage = gimp_image_new(100, 100, 0) # don't need to go through gimpApp object, can call PDB functions directly`