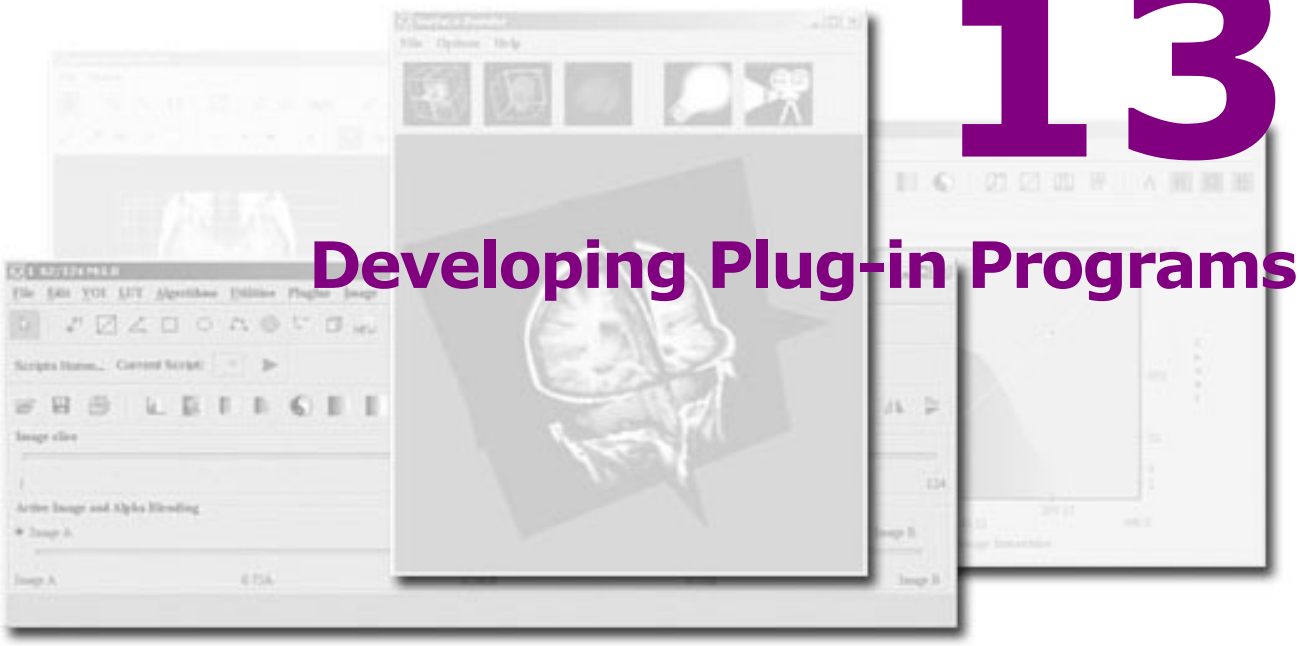
13

Developing Plug-in Programs



In this chapter. . .

"Understanding plug-in programs" on page 534
"Using the API documentation" on page 535
"Developing plug-in programs" on page 543
"Creating a self-contained plug-in frame" on page 557

Users who know how to program in Java can write a *plug-in program* that adds support for a new file format, creates a new view, or applies a new algorithm to an image. *This chapter does not intended to explain how to write a Java program; rather it presents information to help users who are writing plug-in programs to customize MIPAV*. You can find in this chapter how to:

- Gain access to and use the online MIPAV application programming interface (API) documentation
- Determine which version of Java to use
- Select one of the three plug-in types
- Include mandatory lines of code in plug-in programs so that they interface correctly with MIPAV
- Install plug-in programs

Understanding plug-in programs

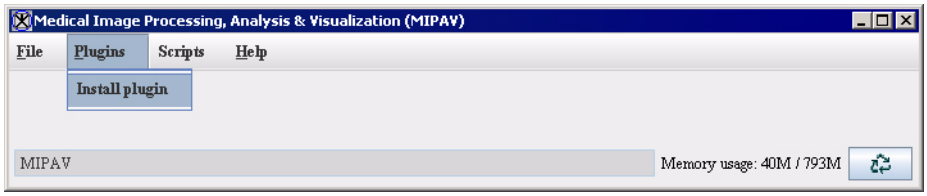
Plug-in programs, also known simply as *plug-ins,* are utilities or sets of instructions that add functionality to a program without changing the program. In MIPAV, you use Java to write and compile plug-in programs to perform specific functions, such as automatically removing all odd-numbered images from the image dataset or adding support for a new file format. There are three types of plug-in programs that you may write for MIPAV:

- **Algorithm**—An algorithm type of plug-in performs a function on an image. An example is a plug-in that applies a radial blur algorithm to an image. You can create plug-in algorithms through Java.
- **File**—A file type of plug-in allows MIPAV to support a new file format. An example is a plug-in that allows MIPAV to view Kodak Photo CD files (.pcd).
- **View**—A view type of plug-in introduces a new view, or the way in which the image is displayed. Examples include the lightbox, triplanar, and animate views.

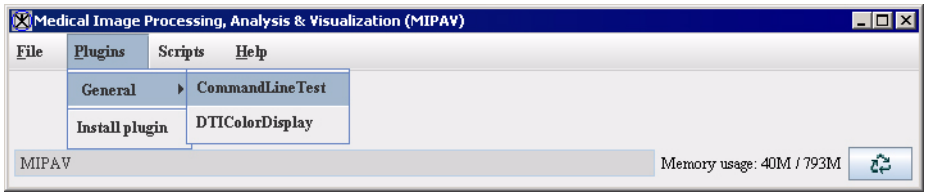
Note: Because MIPAV already supports a large number of file formats and views and its development team makes it a practice to extend its capabilities in these areas, it is generally unnecessary to add file or view types of plug-ins. Most plug-in programs, therefore, are algorithms.

After developing a plug-in program, you can then install the plug-in program into the MIPAV application and access it from the PlugIns menu in the MIPAV window. The MIPAV window labeled "(A)" in Figure 318 shows the PlugIns menu as it appears before any plug-in programs are installed. The picture labeled "(B)" in Figure 318 shows the PlugIns menu as it appears after two plug-in programs—in this case, the Fantasm plug-in program and the Talairach Transform plug-in program—are installed. Because the Fantasm and Talairach Transform plug-in programs are algorithms, they appear under the PlugIns > Algorithm menu.

Note: If a plug-in program is a file type of plug-in, it would appear under a PlugIns > File menu. If it is a view type, it would appear under a PlugIns > View menu.



a



b

Figure 318. Plug-ins menu in the MIPAV window: (a) Before a plug-in was installed and (b) after two plug-ins were installed

Using the API documentation

Documentation for the application programming interface (API) is located on the MIPAV web site <http://mipav.cit.nih.gov/>. You can use the documentation directly on the web site. However, if your internet access is limited or slow, you can download, install, and use either a zipped version of the documentation on a Windows workstation or a tar version on a UNIX workstation.

To access the API documentation via the internet

1. Go to the MIPAV web site:< http://mipav.cit.nih.gov/>.
2. Click Development in the links on the left side of the page. The Development page appears. See Figure 319.
3. Here, use the following links: MIPAV API and MIPAV XML based Formats.

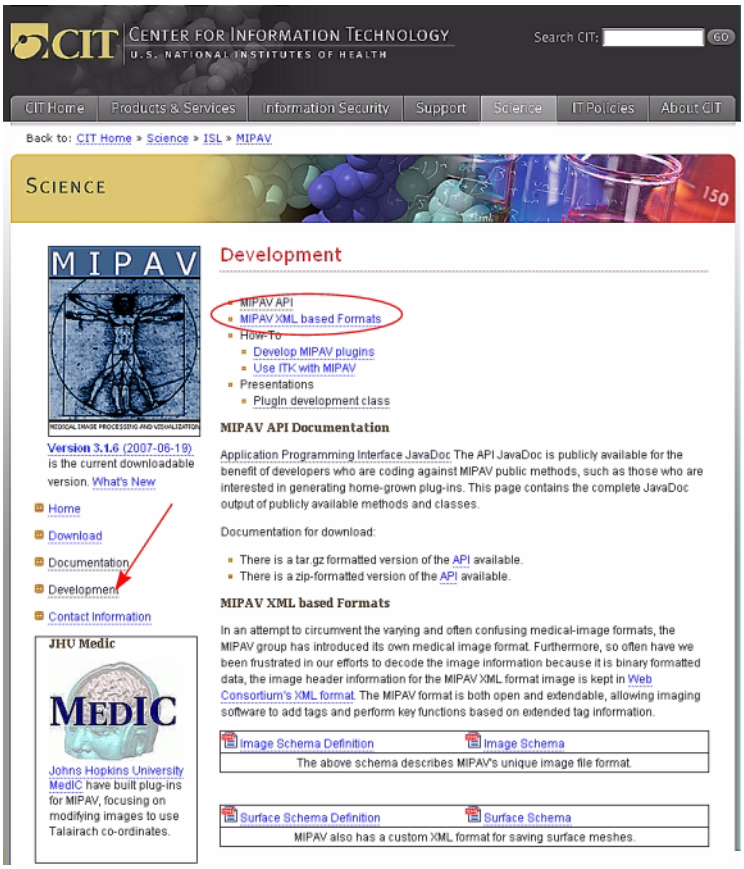


Figure 319. The Development page on the MIPAV web site offers a lot of helpful links

TO DOWNLOAD AND INSTALL THE API DOCUMENTATION ON A WINDOWS WORKSTATION,

1. Under **Documentation for download**, select a zip-formatted version. Save the file to a directory of your choice.
2. Go to the directory, double-click `api.zip`, and extract the files. Extraction creates a directory named "api" under the directory you chose to place the files.
3. Open the api directory, and double-click `index.html`. The API documentation appears in your browser.

To download and install the API documentation on a UNIX workstation,

1. Under **Documentation for download**, select a tar.gz-formatted version. Save the file to a directory of your choice.
2. Go to the directory, double-click `api.tar.gz`, and extract the files. Extraction creates a catalogue named "api" under the directory you chose to place the files.
3. Open the `api` directory, and double-click `index.html`. The API documentation appears in your browser.

Viewing MIPAV API documentation online

On the Development page, click the Application Programming Interface JavaDoc link <http://mipav.cit.nih.gov/documentation/api/index.html>. The API documentation page appears displaying the following three frames:

- **Top left frame**—Shows all of the Java packages for the MIPAV application. When you select the All Classes link at the top of this frame, all of the classes in MIPAV appear in alphabetical order in the bottom left frame. If you select a particular package, the bottom left frame displays only the classes that pertain to the selected package.
- **Bottom left frame**—Lists either all of the classes in the MIPAV application or all of the classes in a selected package.
- **Right frame**—Displays information based on the command that you select in the menu at the top of the frame:
- **Overview**—Lists all of the packages in the MIPAV application
- **Package**—Lists and summarizes all of the classes and interfaces in the package
- **Class or Interface**—Lists descriptions, summary tables, and detailed member descriptions
- **Tree**—Displays a hierarchy of the class or package
- **Deprecated**—Lists deprecated APIs

- **Index**—Provides an alphabetical list of all classes, interfaces, constructors, methods, and fields
- **Help**—Provides help for the API documentation

Several links appear beneath the menu.

- **Prev and Next**—These links take you to the next or previous class, interface, package, or related page.
- **Frames and No Frames links**—These links show and hide the HTML frames. All pages are available with or without frames. See Figure 320.

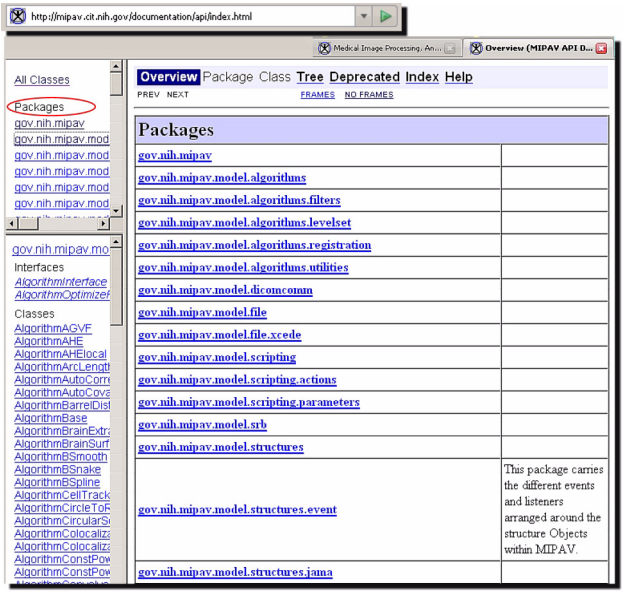


Figure 320. The Overview page

OVERVIEW PAGE

The Overview page is the page that initially appears when you gain access to the API documentation. This page displays a list of all of the packages in MIPAV. The Overview menu becomes available after you move to another page. To return to the Overview page from the any other page, click

Overview. The Overview page appears and displays a list of all of the packages in MIPAV.

PACKAGE PAGE

When you select one of the packages listed on the Overview page, the Package page appears. This page provides a summary of each interface (if any), class, and exception (if any) in the package. When you click an interface or class, the Interface page or the Class page appears. Clicking an exception displays the Exception page. See Figure 321.

INTERFACE OR CLASS PAGES

When you select an interface or class on the Package page, either the Interface page or the Class page appears. Each interface, nested interface, class, and nested class has its own separate page. Each of these pages has three sections consisting of an interface or class description, summary tables, and detailed member descriptions:

- Class inheritance diagram
- Direct known subclasses
- All known subinterfaces or subclasses
- All known implementing classes
- Interface or class declaration
- Interface or class description
- Nested class summary
- Field summary
- Constructor summary
- Method summary
- Field detail
- Constructor detail
- Method detail

Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer. See also Figure 321.

Note: Each serialized or externalized class has a description of its serialization fields and methods. This information is of interest to re-implementors, not to developers using the API. To access this information, go to any serialized class and clicking Serialized Form in the See also section of the class description.

EXCEPTION PAGE

The Exception page appears when an exception on the Package page is selected. This page includes a constructor summary and constructor detail.

TREE (CLASS HIERARCHY) PAGE

When you click Tree on the menu, a Tree, or class hierarchy, page appears. This page displays either the class hierarchy for a particular package, or, if you select All Packages, the class hierarchy for all packages. See Figure 321.

- If you were viewing the Overview page and then clicked Tree, the class hierarchy for all packages appears on the Tree page.
- If you were viewing a Package, Interface, Class, or Exception page and then clicked Tree, the hierarchy for only that package, which includes the class, interface, and exception hierarchies, appears on the Tree page.

Each hierarchy page contains a list of classes, interfaces, and exceptions (if any). The classes are organized by inheritance structure starting with `java.lang.Object`. The interfaces do not inherit from `java.lang.Object`.

DEPRECATED API PAGE

The Deprecated API page appears when you click Deprecated on the menu. This page lists all of the methods in the API that have been deprecated. A deprecated method is **not recommended** for use, generally due to improvements, and a replacement API is usually given.

Warning: Deprecated APIs may be removed in future implementations.

INDEX

The Index page provides an alphabetic list of all classes, interfaces, constructors, methods, and fields with definitions of each. Clicking an entry displays the usage in the product.

HELP PAGE

The Help page provides help for using the API documentation.

Figure 321. The Package, Case and Tree pages of MIPAV API

TO DISPLAY,

All interfaces, classes, and exceptions in a package

1. Go to <http://mipav.cit.nih.gov/documentation/api/>. The Overview page appears.
2. Click one of the packages listed in the:
   - **Frame on the right**—When you click one of the packages listed on this page, the Package page appears in the frame. The Package page displays a list of all interfaces, classes, and exceptions (if any) in the package.
   - **Top frame on the left**—The top frame on the left also lists all of the packages. When you select a package, the bottom frame on the left displays a list of interfaces, classes, and exceptions (if any) in the package.

The methods associated with an interface or with a class

1. Go to <http://mipav.cit.nih.gov/documentation/api/>. The Overview page appears.
2. Do either of the following:
   - Click one of the packages listed in the frame on the right or in the top frame on the left. The Package page appears in the right frame.
   - Click one of the packages in the top frame on the left. A list of interfaces, classes, and exceptions appear in the bottom frame on the left.
3. Do one of the following:
   - Click an interface. The Interface page appears in the right frame.
   - Click a class. The Class page appears in the right frame.
4. Scroll down the page, or click METHODS beneath the menu. The Method Summary table appears.
5. Click a method. The Method Detail section of the page, which lists a description of the method and its parameters, throws, and returns, appears.

Developing plug-in programs

MIPAV provides the following classes for developing plug-in programs:

- PlugInAlgorithm.class
- PlugInFile.class
- PlugInView.class

Plug-in programs are developed in the same way as the other Java programs are. The high-level steps of creating plug-ins follow.

1. **Determining the type of plug-in program**—Before you begin to write the code for the plug-in, determine the plug-in type: algorithm, file, or view. Refer to "Determining the type of plug-in program" .
2. **Determining which version of Java to use**—Detailed instructions appear in "Determining which version of Java to use" and Figure 322.

3 **Writing the source code**—Some lines of code must appear in the source code so that the plug-in program interfaces correctly with MIPAV. Refer to "Writing the source code" .

4 **Building and compiling plug-in programs**—You should keep back-up copies of the source and compiled files in case you need to update or change plug-in programs. See "Building and compiling plug-in programs" .

5 **Creating a self-contained plug-in frame**—A self-contained plug-in is a Java application that does not rely on the default MIPAV user-interface, but, instead, hides MIPAV and display its own image(s) with action/algorithm handling specific to its frame. See "Creating a self-contained plug-in frame" .

6 **Installing plug-in programs**—This section explains how to install plug-in programs. Refer to "Installing plug-in programs" .

7 **Sample plug-in programs**—This section provides a couple of examples of MIPAV plug-ins. Refer to "Examples of MIPAV plug-ins" .

Note: This section does not explain how to write a Java program; however, it explains what must be incorporated in the plug-in program so that it correctly interfaces with the MIPAV application.

Determining the type of plug-in program

The first step of creating a plug-in program is to determine the type you want to create, which depends on its purpose. As mentioned earlier, MIPAV plug-in programs can be of the algorithm, file, or view type. However, most users want MIPAV to perform very specific additional functions on images. Since these functions may not be currently available in MIPAV, users choose to add the functions by developing the algorithm type of plug-in program.

Determining which version of Java to use

To avoid compatibility problems when you create a plug-in program, use the same version of Java that was used to create MIPAV. To determine which version of Java the latest version of MIPAV uses, select Help > JVNM

Information in the MIPAV window. The About System dialog box opens. See Figure 322.

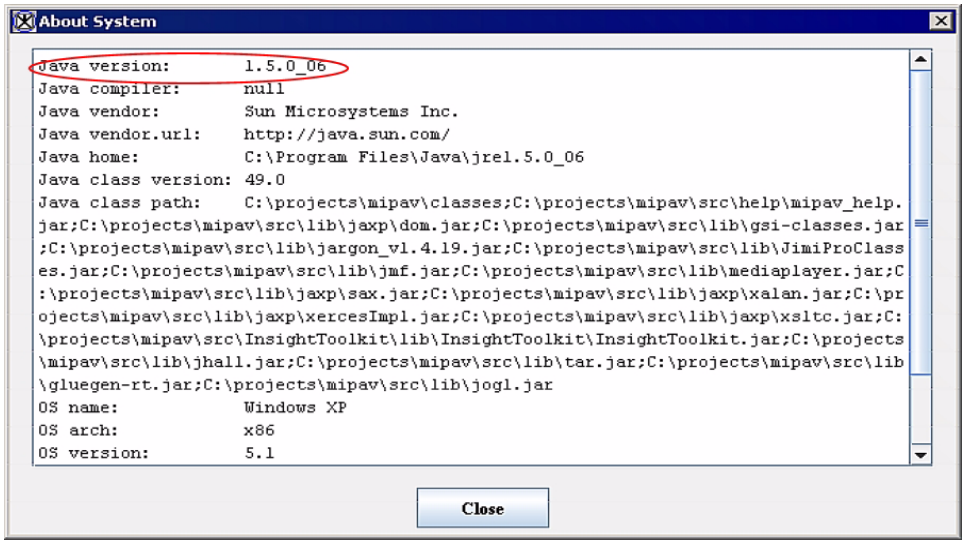


Figure 322. About System dialog box

The first line in the About System dialog box indicates the version of Java that was used to develop MIPAV. To obtain the correct version of Java, go to the following web site: <http://www.java.sun.com>

Writing the source code

Note: In this section, \$MIPAV is used to represent the MIPAV user directory, which is the directory where MIPAV is installed. The user directory is indicated in the About System dialog box. In the MIPAV main window, select Help > JVM Information to view the About System dialog box.

When you develop a plug-in for MIPAV, several lines must be present in the code so that it executes properly. Some mandatory code should be included in **all** plug-in files. Other code might change depending on the plug-in type.

Including mandatory code

The next three figures (Figure 323—Figure 325) show the mandatory source

code needed for creating a file type of plug-in, a view type of plug-in, and an algorithm type of plug-in. The plug-ins directory of MIPAV includes these three files (e.g. C:\[$MIPAV]\mipav\plugins):

- **PlugInFile.java**—Mandatory source code for a file type of plug-in. See Figure 323;
- **PlugInView.java**—Mandatory source code for a view type of plug-in. See Figure 324;
- **PlugInAlgorithm.java**—Mandatory source code for an algorithm type of plug-in. See Figure 325.

```
1     package gov.nih.mipav.plugins;
2
3     import gov.nih.mipav.view.*;
4
5     import java.awt.*;
6
7     public interface PlugInFile extends PlugIn {
8
9         /**
10        *   run
11        *   @param UI           MIPAV main user interface.
12        */
13        public void run(ViewUserInterface UI);
14    }
```

Figure 323. Mandatory code for a file type of plug-in (PlugInFile.java). For readability purposes, keywords in all code reproduced in this chapter appear in bold, and comments appear in green type

```
1     package gov.nih.mipav.plugins;
2
3     import gov.nih.mipav.model.structures.*;
4     import gov.nih.mipav.view.*;
5
6     import java.awt.*;
7
8     public interface PlugInView extends PlugIn {
9
10        /**
11        *   run
12        *   @param UI           MIPAV main user interface.
13        *   @param parentFrame frame that displays the MIPAV image.
14        *                       Can be used as a parent frame when building
15        *                       dialogs.
16        *   @param image        model of the MIPAV image.
17        *   @see   ModelImage
18        *   @see   ViewJFrameImage
19        *
20        */
21        public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image);
22    }
```

Figure 324. Mandatory code for a view type of plug-in (PlugInView.java). For readability purposes, keywords in all code reproduced in this chapter appear in bold, and comments appear in green type

```
1     package gov.nih.mipav.plugins;
2
3     import gov.nih.mipav.model.structures.*;
4     import gov.nih.mipav.view.*;
5
6     import java.awt.*;
7
8
9     public interface PlugInAlgorithm extends PlugIn {
10
11        /**
12        *   run
13        *   @param UI           MIPAV main user interface.
14        *   @param parentFrame frame that displays the MIPAV image.
15        *                       Can be used as a parent frame when building
16        *                       dialogs.
17        *   @param image        model of the MIPAV image.
18        *   @see   ModelImage
19        *   @see   ViewJFrameImage
20        *
21        */
22        public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image);
23
24
25    }
26
```

Figure 325. Mandatory code for an algorithm type of plug-in (PlugInAlgorithm.java). For readability purposes, keywords in all code reproduced in this chapter appear in bold, and comments appear in green type

REFERENCING FILES

To reference a class, you must specify it using the Import keyword. For example, line 2 in PlugInFile.java imports the view functions (Figure 326).

```
import gov.nih.mipav.view.*;
```

Figure 326. Importing the view functions in PlugInFile.java

Lines 3, 4, and 6 in the PlugInView.java and PlugInAlgorithm.java files import the model structures, view functions, and the basic Java package that has GUI functions (Figure 327).

```
import gov.nih.mipav.model.structures.*; // MIPAV package where main
     // MIPAV structures are located (e.g., model image)
import gov.nih.mipav.view.*;

import java.awt.*
```

Figure 327. Importing model structures, view functions, and [java.awt]

If you reference a class, you must include it in the plug-in package so that it can be called from the main file. After you write and compile, you must now install files in the user or home directory:

Windows

```
c:\Documents and Settings\<user ID>\mipav\plugins
```

UNIX

```
/user/<user ID>/mipav/plugins
```

An example of this appears in the first line of Figure 328.

```
package plugins; // added to plugins pkg. so PlugInSampleStub may
                 // call it.
```

Figure 328. Example of placing referenced files in the \$MIPAV\plugins directory

LINES OF CODE THAT ARE DEPENDENT ON PLUG-IN TYPE

Two lines of code depend on the type of plug-in program being developed:

- Declaration
- Parameters for the run method

Declaration

The declaration used in a plug-in depends on the type of plug-in being developed. For instance, in line 9 in PlugInAlgorithm.java (Figure 325), the combination of words "**public interface** *PlugInAlgorithm"* indicates that the plug-in in an Algorithm. For File or View types of plug-ins, simply replace *PlugInAlgorithm* with *PlugInFile* (line 7 in PlugInFile.java, see Figure 323) or *PlugInView* (line 8 in PlugInView.java, see Figure 324), respectively.

Table 4. Declarations dependent on type of plug-in

Type of plug-in	Declaration
File	`public interface PlugInFile extends PlugIn (`
View	`public interface PlugInView extends PlugIn (`
Algorithm	`public interface PlugInAlgorithm extends PlugIn (`

Parameters for the run method

The parameters for the run method also depend on the plug-in type. Compare the run methods used in PlugInFile.java (Figure 323), PlugInView.java (Figure 324), and PlugInAlgorithm.java (Figure 325).

Table 5. Parameters for run methods dependent on type of plug-in

Type of plug-in	Parameters for the run method
File	`public void run(ViewUserInterface UI);`
View	`public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image);`
Algorithm	`public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image);`

```
1     package gov.nih.mipav.plugins;
2
3     import gov.nih.mipav.model structures.*;
4     import gov.nih.mipav.view.*;
5
6     import java.awt *;
7
8     public interface PlugInAlgorithm extends PlugIn {
9
10    /**
11    * run
12    * @param UI           MIPAV main user interface.
13    * @param parentFrame Frame that displays the MIPAV image.
14    *                      Can be used as a parent frame when building dialogs.
15    * @param image        Model of the MIPAV image.
16    * @see ModelImage
17    * @see ViewJFrameImage
18    */
19    public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image;)
20
21    }
```

Figure 329. PlugInAlgorithm.java. For readability purposes, keywords in all code reproduced in this chapter appear in bold, and comments appear in green type

Building and compiling plug-in programs

To build a new plug-in program for MIPAV, you must first install a build environment, alter the path environment variable, and compile the plug-in files.

INSTALLING A BUILD ENVIRONMENT

1. Download and install Java SE Development Kit (JDK), version 1.6 (JDK 6u2) <**http://java.sun.com/javase/downloads/index.jsp**>.
2. Download and install Apache Ant 1.7.0 <http://ant.apache.org/>.

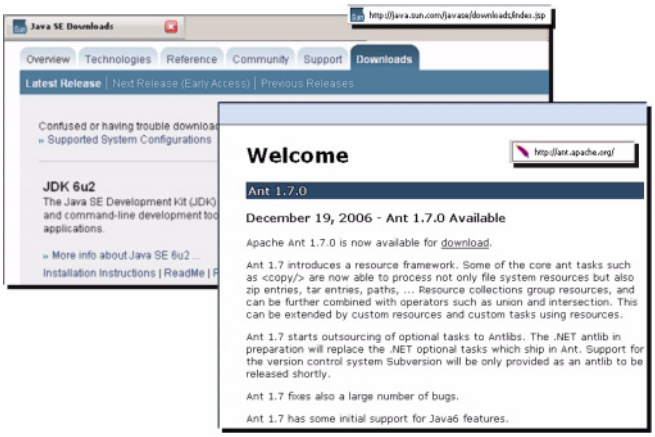


Figure 330. Download pages for Java SE Development Kit (JDK) and Apache Ant 1.7.0

CONFIGURING THE ENVIRONMENT

To configure your environment, you need to add two new variables—JAVA_HOME and ANT_HOME—and update the path variable in your system.

On Windows workstations

1. Click Start > Control Panel. The Control Panel window opens.
2. Double-click the System icon. The System Properties dialog box opens.
3. Click Advanced. The Advanced page of the System Properties dialog box appears.
4. Click Environment Variables. The Environment Variables dialog box opens.
5. Decide whether to add and edit variables in the User variables box or the System variables box based on which users should have access to the Java SDK and Ant.
6. Add the JAVA_HOME variable to your environment:
   a. Click New. The New User Variable dialog box or the New System Variable dialog box opens.
   b. Type `JAVA_HOME` in Variable name.

c Type the path for the Java SDK on your computer (e.g., `C:\Program Files\Java\jdk1.6.0_02`) in Variable value.

d Click OK. The JAVA_HOME variable appears in either the User variables box or System variables box as appropriate.

7 Add the **ANT_HOME** variable to your environment by doing the following:

a Click New under either the User variables box or the System variables box. The New User Variable dialog box or the New System Variables dialog box opens as appropriate.

b Type **ANT_HOME** in Variable name.

c Type the path for the Ant on your computer (e.g., `C:\Program Files\Ant\apache-ant-1.7.0`) in Variable value.

d Click OK. The ANT_HOME variable appears in either the User variables box or System variables box as appropriate.

8 Update either the PATH variable in the User variables box or the Path variable in the System variables box by doing the following:

a Select the PATH variable in the User variables box, or select the Path variable in the System variables box.

b Click Edit under the User variables box, or click Edit under the System variables box. Either the Edit User Variable dialog box or the Edit System Variable dialog box opens.

c Type `;%JAVA_HOME%\bin;%ANT_HOME%\bin` to the end of the PATH variable or to the end of the Path variable.

d Click OK. The edited variable appears either in the User variables box or the System variables box. See also Figure 331.

9 Open a new terminal for the change to take effect by doing the following:

a Click Start > Run. The Run dialog box opens.

b Type **cmd** in Open, and click OK. A terminal window opens.

10 Retrieve the sample Ant build file (build.xml) from the MIPAV web site and place it in the same directory as the plug-in `.java` files you want to compile.

11 Alter the *dir.mipav* and *dir.jdk* properties within the `build.xml` to point to the directory where MIPAV and the SDK are installed, respectively.

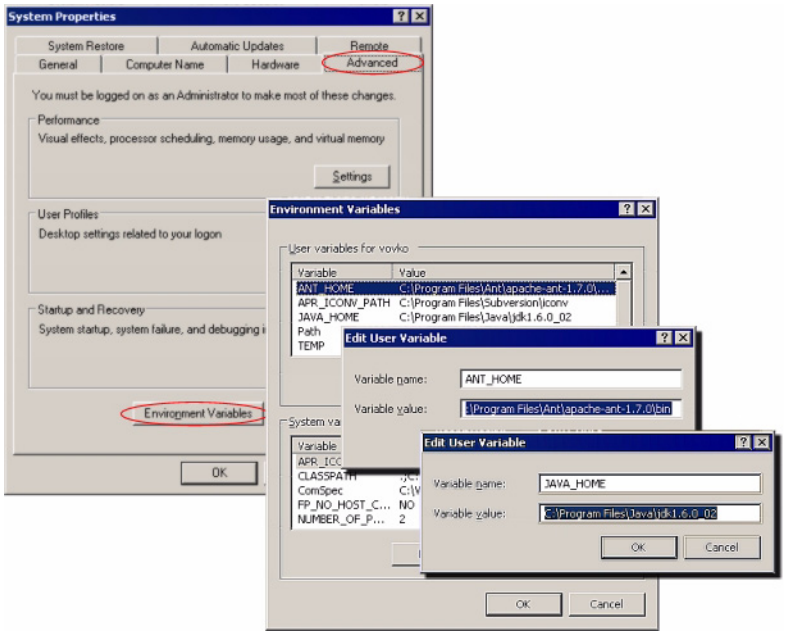


Figure 331. Configuring system variables for MS Windows

Note: Add and edit the variables in the User variables box if you want to limit the build environment to just yourself and no other users. Add and edit the variables in the Systems variables box to make the environment accessible to anyone who uses the workstation.

Recommendation: Although it is possible to update the path variable in either the User variables box or System variables box, you should add the statement to the same box in which you added the *JAVA_HOME* and *ANT_HOME* variables.

See also:

- "Installing Ant" on <http://ant.apache.org/manual/index.html>.
- "JavaTM SE 6 Release Notes—Microsoft Windows Installation (32-bit)" on <http://java.sun.com/javase/6/webnotes/install/jdk/install-windows.html>.

On Linux or UNIX workstations

Bash users should do the following:

1. Edit the file `$HOME/.bash_profile` and add lines similar to following:

```
ANT_HOME=/path/to/apache-ant-1.6.3
JAVA_HOME=/path/to/j2sdk1.4.2_08
PATH=$PATH:$JAVA_HOME/bin:$ANT_HOME/bin

export ANT_HOME
export JAVA_HOME
export PATH
```

   where ANT_HOME and JAVA_HOME are the paths where each application was installed.

2. Retrieve the sample Ant build file from the MIPAV web site, and place it in the same directory where the plug-in `.java` files you want to compile are located.

3. Alter the *dir.mipav* and *dir.jdk* properties within `build.xml` to point to the directory where MIPAV and the SDK are installed, respectively.

BUILD.XML

Figure 332 below displays the content of the **build.xml** file. build.xml is also available on the MIPAV web site <http://mipav.cit.nih.gov/documentation/presentations/plugins/build.xml>.

build.xml

```
<!-- build file for MIPAV plugin class -->
-
      <project basedir="." default="compile" name="mipav_plugin">
<property name="dir.mipav" value="c:\\Program Files\\mipav\\"/>
<property name="dir.jdk" value="c:\\Program Files\\Java\\jdk1.6.0_02"/>
-
      <target name="init">
<tstamp/>
-
      <path id="build.classpath">
<pathelement path="${dir.mipav}"/>
<pathelement location="${dir.mipav}/InsightToolkit/lib/InsightToolkit/InsightToolkit.jar"/>
-
      <fileset dir="${dir.mipav}">
<filename name="*.jar"/>
</fileset>
</path>
<property name="build.cp" refid="build.classpath"/>
</target>
-
      <target name="compile" depends="init">
<echo>classpath: ${build.cp}</echo>
-
      <javac debug="true" deprecation="true" description="Builds MIPAV" verbose="no"
listfiles="yes" nowarn="no" fork="true" memoryInitialSize="220M" memoryMaximumSize="1000M"
id="mipav build" source="1.4" target="1.4" destdir="." srcdir="." compiler="modern">
<classpath refid="build.classpath"/>
</javac>
</target>
-
      <target name="clean" depends="init">
-
      <delete>
-
      <fileset dir=".">
<include name="**/*.class"/>
</fileset>
</delete>
</target>
</project>
```

Figure 332. The contents of the build.xml file

COMPILING THE PLUG-IN FILES

Note: You should keep back-up copies of the source and compiled files in case you need to update or change the plug-in.

1. Type `ant compile` on your workstation (e.g., `cmd ant compile` on Windows or `xterm ant compile` on UNIX platforms). The BUILD SUCCESSFUL message should appear at the end of the Ant output.
2. Copy the `.class` files that Ant produced into MIPAV's plug-in directory.
   - On Windows platforms:

     `C:\Documents and Settings\username\mipav\plugins`
   - On UNIX platforms:

     `/home/username/mipav/plugins`

   where `username` is the name of your account on the system.
3. Install the plug-in file. Select PlugIns > Install Plugin in the main MIPAV window. In the Install PlugIn dialog box, use the Browse buton to navigate to the `\plugins` directory . Select the plug-in and Press OK.

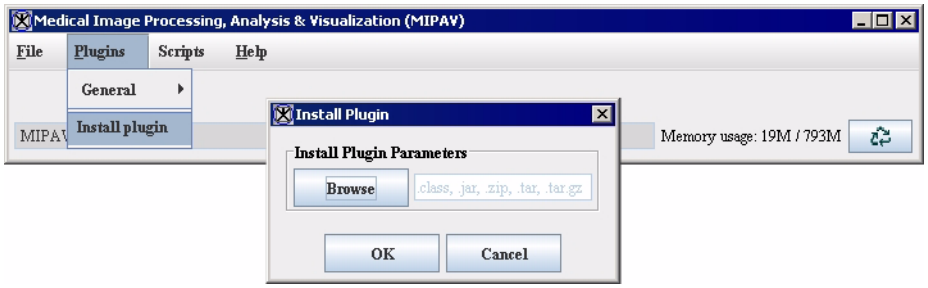


Figure 333. Installing a MIPAV plug-in.

Creating a self-contained plug-in frame

You can create a self-contained plug-in that does not rely on the default MIPAV user interface. When running, this type of plug-in hides MIPAV and displays its own image(s) with the action and algorithm handling specific to its frame.

TO CREATE A SELF-CONTAINED PLUG-IN:

1. Extend `ViewJFrameImage`, as this will allow the plug-in to use a wide-range of `ViewJFrameImage` and `ViewJFrameBase` specific functions for storing and displaying `ModelImages`. These functions include, for example, the image and on-screen buffers, menu and toolbar builders, etc.
2. Override the `ViewOpenFrameInterface openFrame(ModelImage)` function. This handles the creation of a new `PlugIn` frame based on whether a result image is created within the dialog of an algorithm. For example, when the user runs an algorithm and selects the destination New Image rather than Replace Image, a new frame will be created with the result of the algorithm. To set all algorithms to work in place and disallow creating of new frames, call `ViewUserInterface setForceInPlace(true)` function, which tells the dialogs that all algorithms must work in place.
3. Create an `init()` function, where the `PlugIn` frame is layout and components will be initialized.
4. In the `init()` function, several methods should be called:
   - Call `initLUT()` for the `ModelImage` look-up table,
   - Call `initResolutions()` for the `ModelImage` resolutions,
   - Call `initZoom()` for the frame's zoom factor,
   - `initComponentImage()` creates a displayable `ComponentImage`,
   - `initExtentsVariables()` initializes z-slice and time-slice positions.
5. To add toolbars and menus to your plug-in, within `init()`, create a `ViewControlsImage` object, and then

- Call `buildToolBar()` to create pre-defined toolbars for image, VOI, paint, and scripting controls;
- Or call `buildSimpleToolBar` and pass `Vector<CustomUIBuilder.UIParams>` using `addCustomToolBar()` for each of the Vectors.

Pre-defined button and menu parameters are located in `CustomUIBuilder`. Pre-defined as well as the user-defined `UIParams` can be added and used in both toolbars and menus.

6 Create `ViewMenuBar`. This allows you to add either pre-defined or `UIParam` menus. The Vector from above (used on the custom toolbar) can be passed into the `ViewMenuBar makeCustomMenu()` function. `ViewMenuBar` also has pre-defined menus for a file, help, image, look-up tables, etc.

7 Finally, the `init()` function should handle the container for the `ViewJComponentEditImage` created from `initComponentImage()`. The component image should be added to a `JScrollPane` to accommodate the variable size of the display.

8 Override the `actionPerformed()` method to catch (handle) `ActionEvents`. If a custom toolbar and (or) menu bar was created using a Vector of `UIParams`, the `UIParam` contains the action event for each button and (or) menu item.

9 Override the `componentResized()` method to properly handle (or ignore) the resizing of the plug-in frame. Using the `ViewJFrameImage componentResized` function would likely create unwanted behavior as the layout of the plug-in is different from MIPAV's standard `ViewJFrameImage`.

10 Create a basic `PlugInGeneric` class that will be called as a command-line argument. This class should have the ability to choose/open a `ModelImage` using the `FileIO.readImage()` method. The self-contained plug-in frame should be instantiated within this class by passing in the `ModelImage`.

11 When running MIPAV, pass in the arguments `-hide -p [YourGenericPlugin]`. The `-hide` flag tells MIPAV not to bring up the User Interface and `MessageFrame`, while the `-p` flag tells which plug-in to run. See Figure 334.

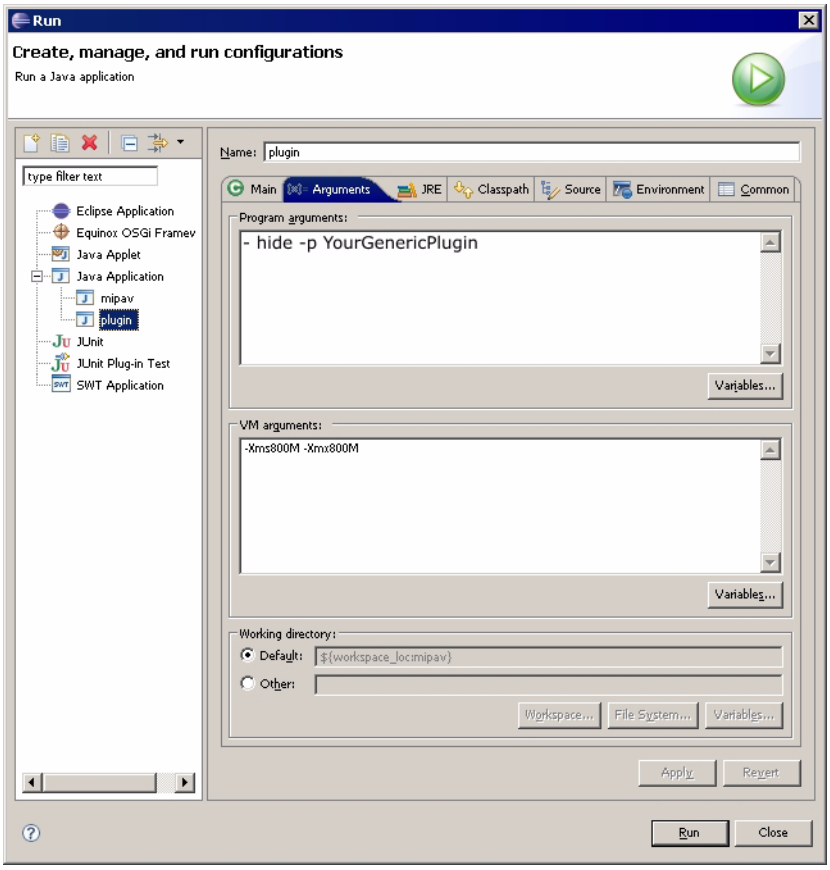


Figure 334. The arguments for running the self-contained plug-in frame.

Optional:

The `ViewJFrameMessage` Data and Debug tabs (as well as others) can be added to the plug-in frame by retrieving the `JTabbedPane` through `ViewUserInterface.getReference().getMessageFrame().getTabbedPane()`. This enables the Data and Debug message output to be displayed outside of the separate message frame that accompanies MIPAV.

See also: Figure 335 and Figure 336.

PlugInDialogImageVOIDisplay.java

```
1     import java.awt.*;
2     import java.awt.event.*;
3     import java.util.Vector;
4
5     import javax.swing.*;
6     import gov.nih.mipav.model.file.FileInfoBase;
7     import gov.nih.mipav.model.structures.ModelImage;
8     import gov.nih.mipav.model.structures.ModelLUT;
9     import gov.nih.mipav.model.structures.ModelRGB;
10    import gov.nih.mipav.model.structures.VOI;
11    import gov.nih.mipav.view.*;
12    import gov.nih.mipav.view.dialogs.*;
13
14
15    /**
16     * Plugin example class for creating a simple, self-contained frame that extends ViewJFrame
      Image
17     * Contains a subset of the VOI functions, as well as the message frame contained within the
      frame itself
18     * @author linkb
19     *
20     */
21    public class PlugInDialogImageVOIDisplay extends ViewJFrameImage implements MouseListener,
      AdjustmentListener {
22
23
24
25      //~ Constructors -----------------------------------------------------------------------
      ------------------------
26
27       /**
28        * Default constructor
29        */
30        public PlugInDialogImageVOIDisplay(ModelImage image) {
31          super(image, null, null, false, false);
32            init();
33        }
34
35
36        /**
37         * ViewOpenFrameInterface function for opening a model image (result) into a new frame
38         */
39        public PlugInDialogImageVOIDisplay openFrame(ModelImage image) {
40          return new PlugInDialogImageVOIDisplay(image);
41        }
```

Figure 335. A part of the code for PlugInDialogImageVOIDisplay.java. The full code can be found in "Examples of MIPAV plug-ins" , Figure 345.

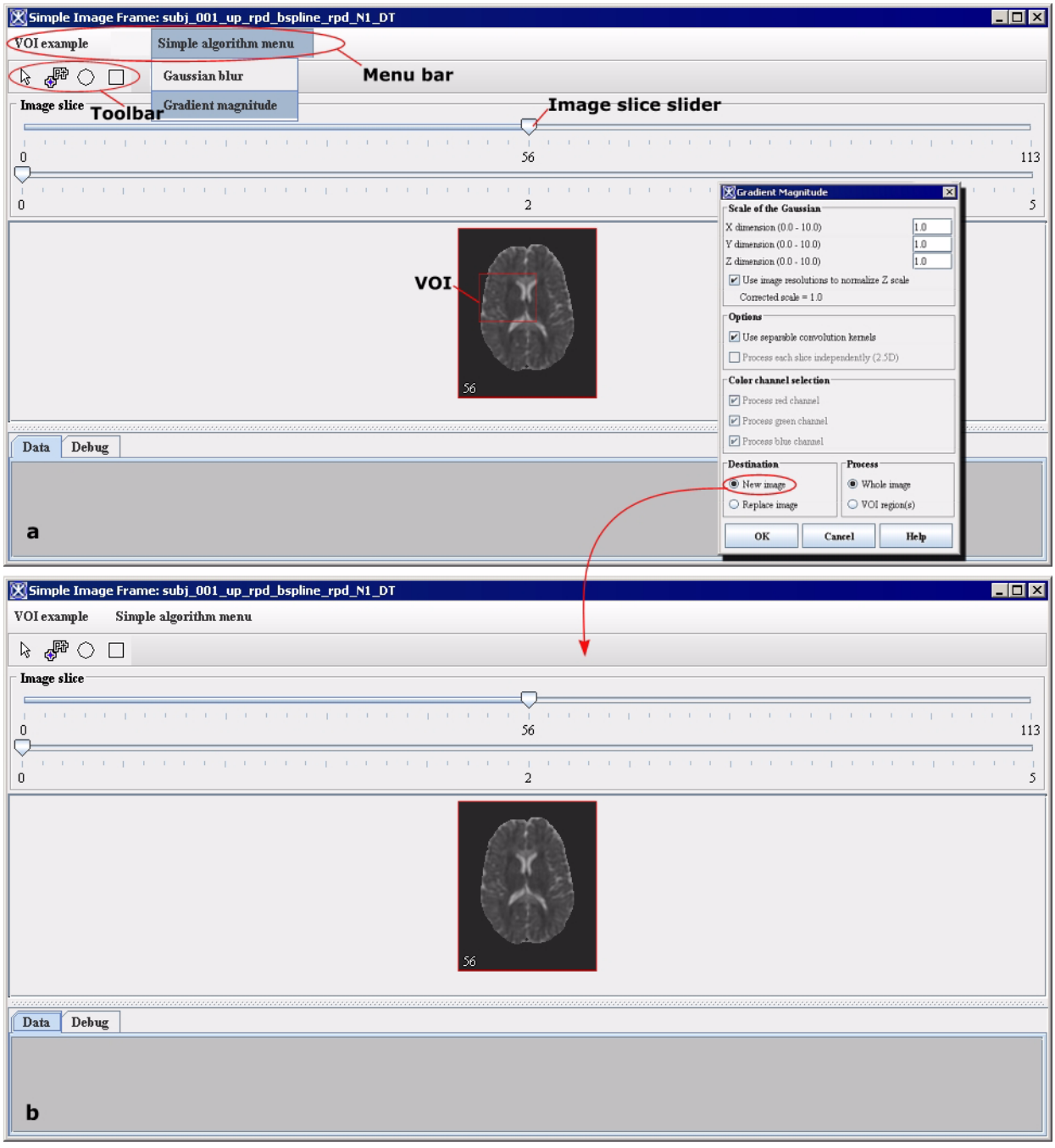


Figure 336. The Simple Image Frame plug-in opens the MIPAV independent image frame for a selected image. (a) The image frame contains the following elements of the interface – the VOI toolbar, image slice slider and menu bar. The Gradient Magnitude algorithm was called from the menu and applied to the image. (b) The result image opened in a new image frame.

Installing plug-in programs

Installing simple plug-in programs merely copies files into the user's home directory.

Windows

```
c:\Documents and Settings\<user ID>\mipav\plugins
```

UNIX

```
/user/<user ID>/mipav/plugins
```

You can choose one of two methods for copying the files:

- Use MIPAV's plug-in installation tool, e.g. in the MIPAV window, select PlugIns > Install PlugIns.
- Use the operating system's tool for copying the files. This method requires the user to restart MIPAV so that the new plug-in appears in the PlugIns menu. When MIPAV starts, it parses the user's home directory and builds the PlugIns menu.

Warning: The MIPAV installation tool does *not* work for more complex plug-ins that consist of more complicated package class hierarchy, such as the Medic Talairach plug-in program. To learn more about Medic Talairach plug-in program, refer to MIPAV Technical Guide 1.

Examples of MIPAV plug-ins

To build plug-in programs, three files are typically required:

- **PlugIn*Foo*.java**—Provides an interface to MIPAV and the plugin.
- **PluginDialog*Foo*.java**—Invokes the dialog to get user-supplied parameters; it can be hidden when no parameters are required.
- **PluginAlgorithm*Foo*.java**—Provides the actual algorithm to be implemented. It can be a mixture of calls to MIPAV's API, C programs, Perl, ITK, etc.

Where *Foo* is the name that you supply for the program. The following sample plug-in program(s) are included in MIPAV documentation:

- PlugInSample—a sample plug-in, see "Sample plug-in program" below.

- PlugInCT_MD—a typical plug-in. (Refer to the MIPAV Users Guide, PDF version.)
- PlugInAlgorithm.Median—a very complicated plug-in. Refer to MIPAV Volume 1 Users Guide, Appendix D.
- PlugInDialogImageVOIDisplay.java – a self contained plug-in.

SAMPLE PLUG-IN PROGRAM

The source code for the plug-in program, PlugInSample.java is an example of a simple algorithm type of plug-in. See Figure 337.

PlugInSample.java

```
1     import gov.nih.mipav.plugins.*;  // needed to load PluginAlgorithm / PluginView /
2                                      // PlugInFile interface
3     import gov.nih.mipav.view.*;
4     import gov.nih.mipav.model.structures.*;
5     import java.awt.*;
6
7     /***  This is a simple plugin to display a image in a new frame @see PlugInAlgorithm */
8
9     /** This is an Algorithm type of PlugIn and therefore must implement PlugInAlgorithm
10    ** Implementing the PlugInAlgorithm requires this class to implement the run method
11    ** with the correct parameters */
12
13    public class PlugInSample implements PlugInAlgorithm {
14          /**
15           * Defines body of run method, which was declared in the interface.
16           * @param UI User Interface
17           * @param parentFrame ParentFrame
18           * @param image Current ModelImage--this is an image already loaded into
19           * MIPAV. Can be null.
20           */
21
22          public void run (ViewUserInterface UI, Frame parentFrame, ModelImage image){
23                if (parentFrame instanceof ViewJFrameImage) {
24                 new PlugInDialogSample(parentFrame,image);
25                } else {
26                 MipavUtil.displayError("PlugInSample only runs on an image frame.");
27              }
28          }
29    }
```

Figure 337. PlugInSample.java

PlugInSample.java opens an image in a new image frame using its own dialog box. It requires three files:

- **PlugInSample.java**—Provides an interface to MIPAV and the plug-in program. See Figure 337 on page 563.
- **PlugInDialogSample.java**—Invokes the dialog to get user-supplied parameters. Refer to Figure 338 on page 564.
- **PlugInAlgorithmSample.java**—Implements the algorithm. See Figure 339.

PlugInDialogSample.java

```
1     import gov.nih.mipav.view.*;
2     import gov.nih.mipav.view.dialogs.*;
3     import gov.nih.mipav.model.structures.*;
4     import gov.nih.mipav.model.algorithms.*;
5
6     import java.awt.event.*;
7     import java.awt.*;
8     import java.util.*;
9     import javax.swing.*;
10
11
12    public class PlugInDialogSample extends JDialogBase implements AlgorithmInterface {
13
14    /** Source image reference. */
15        private ModelImage image; // source image
16             private ViewUserInterface userInterface;
17
18    /** Sample algorithm reference. */
19          private PlugInAlgorithmSample sampleAlgo = null;
20
21          public PlugInDialogSample(Frame theParentFrame, ModelImage im) {
22                 super(theParentFrame, false);
23
24                  if ((im.getType() == ModelImage.BOOLEAN) || im.isColorImage()) {
25                      MipavUtil.displayError("Source Image must NOT be Boolean or Color");
26                      dispose();
27
28                      return;
29                  }
30
31                  image = im;
32                  userInterface = ViewUserInterface.getReference();
33                  init();
34          }
35
```

Figure 338. PlugInDialogSample.java

```
36          // ************************************************************************
37          // ************************** Event Processing ****************************
38          // ************************************************************************
39
40          /**
41           * Closes dialog box when the OK button is pressed and calls the algorithm.
42           * @param  event  Event that triggers function.
43           */
44
45         public void actionPerformed(ActionEvent event) {
46              String command = event.getActionCommand();
47
48              if (command.equals("OK")) {
49              callAlgorithm();
50              } else if (command.equals("Cancel")) {
51                  dispose();
52              }
53          }
54
55          /**
56           * Sets up the GUI (panels, buttons, etc) and displays it on the screen.
57          */
58          private void init() {
59
60              // Build the Panel that holds the OK and CANCEL Buttons
61              JPanel OKCancelPanel = new JPanel();
62
63              JLabel questionLabel = new JLabel("Display Images?");
64
65              // size and place the OK button
66              buildOKButton();
67              OKCancelPanel.add(OKButton, BorderLayout.WEST);
68
69              // size and place the CANCEL button
70              buildCancelButton();
71              OKCancelPanel.add(cancelButton, BorderLayout.EAST);
72              getContentPane().add(questionLabel, BorderLayout.NORTH);
73              getContentPane().add(OKCancelPanel, BorderLayout.SOUTH);
74
75              pack();
76              setVisible(true);
77              setResizable(false);
78              System.gc();
79          }
```

Figure 338. PlugInDialogSample.java (continued)

```
80    /*** This method is required if the AlgorithmPerformed interface is implemented. It is called by
      the algorithm when it has completed or failed to to complete, so that the dialog can be display
      the result image and/or clean up. */
81
82    /** @param  algorithm  Algorithm that caused the event. */
83
84        public void algorithmPerformed(AlgorithmBase algorithm) {
85         if (algorithm instanceof PlugInAlgorithmCT_MD) {
86           if ( sampleAlgo.isCompleted() ) {
87                 dispose();
88           }
89           }
90        }
91
92
93    /*** Once all the necessary variables are set, call the Gaussian Blur algorithm based on what
      type of image this is and whether or not there is a separate destination image. */
94
95        protected void callAlgorithm() {
96            sampleAlgo = new PlugInAlgorithmSample(null, image);
97            sampleAlgo.addListener(this);
98            setVisible(false); // Hide dialog
99
100           if (isRunInSeparateThread()) {
101
102   //*** Start the thread as a low priority because we wish to still have user interface work
      fast.*/
103               if (sampleAlgo.startMethod(Thread.MIN_PRIORITY) == false) {
104                   MipavUtil.displayError("A thread is already running on this object");
105               }
106           } else {
107           sampleAlgo.run();
108           }
109       }
110
111   }
```

Figure 338. PlugInDialogSample.java (continued)

PlugInAlgorithmSample.java

```
1     import gov.nih.mipav.model.algorithms.AlgorithmBase;
2     import gov.nih.mipav.model.structures.*;
3
4     import gov.nih.mipav.view.*;
5
6
7     public class PlugInAlgorithmSample extends AlgorithmBase {
8
9           private ViewJFrameImage frame;
10
11    /*** Constructor for 3D images in which changes are placed in a predetermined destination
      image.
12    */
13
14    /**
15    * @param  destImg  Image model where result image is to stored.
16    * @param  srcImg   Source image model.
17    */
18        public PlugInAlgorithmSample(ModelImage destImg, ModelImage srcImg) {
19            super(destImg, srcImg);
20        }
21
22        //~ Methods ------------------------------------------------------------------------------/
23
24    /**
25    * Prepares this class for destruction.
26    */
27        public void finalize() {
28            destImage = null;
29            srcImage = null;
30            super.finalize();
31        }
32
33
34    /**
35    * Starts the algorithm.
36    */
37        public void runAlgorithm() {
38          frame = new ViewJFrameImage((ModelImage)srcImage.clone());
39          setCompleted(true);
40        }
41
42    }
```

Figure 339. PlugInAlgorithmSample.java

PLUGINCT_MD, A TYPICAL PLUGIN PROGRAM

PlugInCT_MD is a typical example of a plug-in program. It consists of three files:

- **PlugInCT_MD.java**—Provides an interface to MIPAV and the plug-in program.
- **PlugInDialogCT_MD.java**—Invokes the dialog to get user-supplied parameters.
- **PlugInAlgorithmCT_MD.java**—Implements the algorithm.

PlugInCT_MD.java

The file in Figure 340 provides an interface between MIPAV and PlugInCT_MD.

PlugInDialogCT_MD.java

The PlugInDialogCT_MD.java file invokes a dialog box to obtain user-supplied data. Refer to Figure 341 on page 570.

PlugInAlgorithmCT_MD.java

Figure 343 on page 580 shows the content of PlugInAlgorithmCT_MD.java.

PlugInDialogImageVOIDisplay.java

Figure 345 on page 586 shows a sample code for a self-contained frame plug-in.

Note: For readability purposes, keywords in all code reproduced in this chapter appear in bold, and comments appear in green type

PlugInCT_MD.java

```
1     import plugins.PlugInDialogCT_MT;     //associated class file
2     import gov.nih.mipav.plugins.*;       //needed to load PlugInAlgorithm / PlugInView /
3                                           //PlugInFile interface
4     import gov.nih.mipav.view.*;
5     import gov.nih.mipav.model.structures.*;
6
7     import java.awt.*;
8
9     /**
10    *  This is a simple plugin for the University of Maryland to simple segment an
11    *  imagebased on CT Hounsfield units.
12    *
13    * @see PlugInAlgorithm
14    */
15
16    //This is an Algorithm type of PlugIn, and therefore must implement PlugInAlgorithm
17    //Implementing the PlugInAlgorithm requires this class to implement the run method
18    //with the correct parameters
19    public class PlugInCT_MD implements PlugInAlgorithm {
20
21       /**
22         * Defines body of run method, which was declared in the interface.
23         * @param UI            User Interface
24         * @param parentFrame   ParentFrame
25         * @param image         Current ModelImage--this is an image already loaded into
26         *                      MIPAV. Can be null.
27         */
28        public void run (ViewUserInterface UI, Frame parentFrame, ModelImage image){
29
30            if (parentFrame instanceof ViewJFrameImage)
31              new PlugInDialogCT_MD (parentFrame,image);
32
33            else
34              MipavUtil.displayError ("PlugIn CT_MD only runs on an image frame.");
35            }
36        }
37    }
```

Figure 340. PlugInCT_MD.java

PlugInDialogCT_MD.java

```
1     import gov.nih.mipav.view.*;
2     import gov.nih.mipav.view.dialogs.*;
3     import gov.nih.mipav.model.structures.*;
4     import gov.nih.mipav.model.algorithms.*;
5
6     import java.awt.event.*;
7     import java.awt.*;
8     import java.util.*;
9
10    import javax.swing.*;
11
12
13    /**
14    *
15    *   JDialogBase class.
16    *
17    *   Note:
18    *
19    *   @version    July 12, 2002
20    *   @author
21    *   @see        JDialogBase
22    *   @see        JDialogMedian
23    *   @see        AlgorithmInterface
24    *
25    *   $Logfile: /mipav/src/plugins/PlugInDialogCT_MD.java $
26    *   $Revision: 6 $
27    *   $Date: 8/05/04 5:44p $
28    *
29    */
30    public class PlugInDialogCT_MD extends JDialogBase implements AlgorithmInterface {
31
32        private     PlugInAlgorithmCT_MD ctSegAlgo = null;
33        private     ModelImage  image;                // source image
34        private     ModelImage  resultImage = null;   // result image
35         private ViewUserInterface userInterface;
36
37        private     String       titles[];
38
39        private     float        correctionVal;
40        private     JTextField   fatLValTF;
41        private     JTextField   fatHValTF;
42        private     JTextField   ldmLValTF;
43        private     JTextField   ldmHValTF;
44        private     JTextField   hdmLValTF;
45        private     JTextField   hdmHValTF;
46
47        private     int         fatLVal;
48        private     int         fatHVal;
49        private     int         ldmLVal;
50        private     int         ldmHVal;
51        private     int         hdmLVal;
52        private     int         hdmHVal;
53
```

Figure 341. PlugInDialogCT_MD.java

```
54        /**
55         *  Creates new dialog for Median filtering using a plugin.
56         *  @param parent          Parent frame.
57         *  @param im              Source image.
58         */
59
60           public PlugInDialogCT_MD(Frame theParentFrame, ModelImage im) {
61             super(theParentFrame, true);
62             if (im.getType() == ModelImage.BOOLEAN || im.isColorImage()) {
63                 MipavUtil.displayError("Source Image must NOT be Boolean or Color");
64                 dispose();
65                 return;
66             }
67             image = im;
68             userInterface = ((ViewJFrameBase)(parentFrame)).getUserInterface();
69             init();
70           }
71
72         /**
73         * Used primarily for the script to store variables and run the algorithm.  No
74         * actual dialog will appear but the set up info and result image will be stored
75         *  here.
76         * @param UI   The user interface, needed to create the image frame.
77         * @param imSource image.
78         */
79         public PlugInDialogCT_MD(ViewUserInterface UI, ModelImage im) {
80             super();
81             userInterface = UI;
82             if (im.getType() == ModelImage.BOOLEAN || im.isColorImage()) {
83                 MipavUtil.displayError("Source Image must NOT be Boolean or Color");
84                 dispose();
85                 return;
86             }
87
88             image = im;
89         }
90
91         /**
92         * Sets up the GUI (panels, buttons, etc) and displays it on the screen.
93         */
94           private void init(){
95
96       setForeground(Color.black);
97             setTitle("CT_segmentation");
98
99                  JPanel inputPanel = new JPanel(new GridLayout(3, 3));
100                 inputPanel.setForeground(Color.black);
101                 inputPanel.setBorder(buildTitledBorder("Input parameters"));
102
103                JLabel labelFat = new JLabel("Fat thresholds: ");
104                labelFat.setForeground(Color.black);
105                 labelFat.setFont(serif12);
106                 inputPanel.add(labelFat);
107
```

Figure 341. PlugInDialogCT_MD.java (continued)

```
108                fatLValTF = new JTextField();
109                fatLValTF.setText("-190");
110                fatLValTF.setFont(serif12);
111                inputPanel.add(fatLValTF);
112
113                fatHValTF = new JTextField();
114                fatHValTF.setText("-30");
115                fatHValTF.setFont(serif12);
116                inputPanel.add(fatHValTF);
117
118                JLabel labelLDM = new JLabel("Low density muscle thresholds: ");
119               labelLDM.setForeground(Color.black);
120                labelLDM.setFont(serif12);
121                inputPanel.add(labelLDM);
122
123                ldmLValTF = new JTextField();
124                ldmLValTF.setText("0");
125                ldmLValTF.setFont(serif12);
126                inputPanel.add(ldmLValTF);
127
128                ldmHValTF = new JTextField();
129                ldmHValTF.setText("30");
130                ldmHValTF.setFont(serif12);
131                inputPanel.add(ldmHValTF);
132
133                JLabel labelHDM = new JLabel("High density muscle thresholds: ");
134                  labelHDM.setForeground(Color.black);
135                labelHDM.setFont(serif12);
136                inputPanel.add(labelHDM);
137
138                hdmLValTF = new JTextField();
139                hdmLValTF.setText("31");
140                hdmLValTF.setFont(serif12);
141                inputPanel.add(hdmLValTF);
142
143                hdmHValTF = new JTextField();
144                hdmHValTF.setText("100");
145                hdmHValTF.setFont(serif12);
146                inputPanel.add(hdmHValTF);
147
148           getContentPane().add(inputPanel, BorderLayout.CENTER);
149
150        // Build the Panel that holds the OK and CANCEL Buttons
151           JPanel OKCancelPanel = new JPanel();
152
153           // size and place the OK button
154           buildOKButton();
155             OKCancelPanel.add(OKButton, BorderLayout.WEST);
156           // size and place the CANCEL button
157           buildCancelButton();
158             OKCancelPanel.add(cancelButton, BorderLayout.EAST);
159             getContentPane().add(OKCancelPanel, BorderLayout.SOUTH);
```

Figure 341. PlugInDialogCT_MD.java (continued)

```
160                 pack();
161                 setVisible(true);
162                 setResizable(false);
163           System.gc();
164
165           } // end init()
166
167          /**
168         *  Accessor that returns the image.
169         *  @return          The result image.
170         */
171        public ModelImage   getResultImage(){return resultImage;}
172
173
174
175           /**
176           *     Accessor that sets the correction value
177           *     @param num  Value to set iterations to (should be between 1 and 20).
178           */
179           public void  setCorrectionValue(float num){correctionVal = num;}
180
181        //************************************************************************
182        //************************** Event Processing ****************************
183        //************************************************************************
184
185           /**
186           *  Closes dialog box when the OK button is pressed and calls the algorithm.
187           *  @param event       Event that triggers function.
188           */
189           public void actionPerformed(ActionEvent event) {
190                 String command = event.getActionCommand();
191
192                 if (command.equals("OK")) {
193                       if (setVariables()) {
194                             callAlgorithm();
195                       }
196                 }
197                 else if (command.equals("Script")) {
198                       callAlgorithm();
199                 }
200                 else if (command.equals("Cancel")) {
201                       dispose();
202                 }
203       }
204
205        //************************************************************************
206        //************************** Algorithm Events ****************************
207        //************************************************************************
208
209        /**
210        * This method is required if the AlgorithmPerformed interface is implemented.
211        *   It is called by the algorithm when it has completed or failed to to complete,
212        *   so that the dialog can be display the result image and/or clean up.
213        *   @param algorithm   Algorithm that caused the event.
214        */
215        public void algorithmPerformed(AlgorithmBase algorithm) {
```

Figure 341. PlugInDialogCT_MD.java (continued)

```
216     ViewJFrameImage imageFrame = null;
217           if ( algorithm instanceof PlugInAlgorithmCT_MD) {
218               image.clearMask();
219               if(ctSegAlgo.isCompleted() == true && resultImage != null) {
220                   //The algorithm has completed and produced a new image to be displayed.
221
222                   updateFileInfo(image, resultImage);
223                   resultImage.clearMask();
224                   try {
225                       //resultImage.setImageName("Median: "+image.getImageName());
226
227                       int dimExtentsLUT[] = new int[2];
228                       dimExtentsLUT[0]    = 4;
229                       dimExtentsLUT[1]    = 256;
230                       ModelLUT LUTa   = new ModelLUT(ModelLUT.COOLHOT, 256, dimExtentsLUT);
231                      imageFrame = new ViewJFrameImage(resultImage, LUTa, new Dimension(610,200),
232                                   userInterface);
233                   }
234                   catch (OutOfMemoryError error){
235                       System.gc();
236                       MipavUtil.displayError("Out of memory: unable to open new frame");
237                   }
238               }
239               else if (resultImage == null) {
240                   // These next lines set the titles in all frames where the source image
241                // is displayed to image name so as to indicate that the image is now
242                // unlocked! The image frames are enabled and then registered to the
243                // userinterface.
244                   Vector imageFrames = image.getImageFrameVector();
245                   for (int i = 0; i < imageFrames.size(); i++) {
246                          ((Frame)(imageFrames.elementAt(i))).setTitle(titles[i]);
247                          ((Frame)(imageFrames.elementAt(i))).setEnabled(true);
248                          if  ( ((Frame)(imageFrames.elementAt(i))) != parentFrame) {
249                             userInterface.registerFrame((Frame)(imageFrames.elementAt(i)));
250                          }
251                   }
252                   if (parentFrame != null) userInterface.registerFrame(parentFrame);
253                   image.notifyImageDisplayListeners(null, true);
254               }
255               else if (resultImage != null){
256                       //algorithm failed but result image still has garbage
257                       resultImage.disposeLocal();  // clean up memory
258                       resultImage = null;
259                       System.gc();
260               }
261           }
262           if (ctSegAlgo.isCompleted() == true) {
263                  if (userInterface.isScriptRecording()) {
264                     userInterface.getScriptDialog().append("Flow " +
265                     userInterface.getScriptDialog().getVar(image.getImageName()) + " "
266                     + correctionVal + "\n");
267           }
268           }
269           dispose();
270
```

Figure 341. PlugInDialogCT_MD.java (continued)

```
271     } // end AlgorithmPerformed()
272
273
274        /**
275         * Use the GUI results to set up the variables needed to run the algorithm.
276         * @return      <code>true</code> if parameters set successfully, <code>false
277        *  </code> otherwise.
278         */
279        private boolean setVariables() {
280           String tmpStr;
281
282
283            // verify iteration is within bounds
284            tmpStr = fatLValTF.getText();
285             if ( testParameter(tmpStr, -4000, 4000) ){
286                 fatLVal = Integer.valueOf(tmpStr).intValue();
287             }
288             else{
289                 fatLValTF.requestFocus();
290                 fatLValTF.selectAll();
291                 return false;
292             }
293
294             tmpStr = fatHValTF.getText();
295             if ( testParameter(tmpStr, -4000, 4000) ){
296                 fatHVal = Integer.valueOf(tmpStr).intValue();
297             }
298             else{
299                 fatHValTF.requestFocus();
300                 fatHValTF.selectAll();
301                 return false;
302             }
303
304             tmpStr = ldmLValTF.getText();
305             if ( testParameter(tmpStr, -4000, 4000) ){
306                 ldmLVal = Integer.valueOf(tmpStr).intValue();
307             }
308             else{
309                 ldmLValTF.requestFocus();
310                 ldmLValTF.selectAll();
311                 return false;
312             }
313
314             tmpStr = ldmHValTF.getText();
315             if ( testParameter(tmpStr, -4000, 4000) ){
316                 ldmHVal = Integer.valueOf(tmpStr).intValue();
317             }
318                 else{
319                 ldmHValTF.requestFocus();
320                 ldmHValTF.selectAll();
321                 return false;
322             }
323
324
```

Figure 341. PlugInDialogCT_MD.java (continued)

```
    tmpStr = hdmLValTF.getText();
           if ( testParameter(tmpStr, -4000, 4000) ){
               hdmLVal = Integer.valueOf(tmpStr).intValue();
           }
           else{
               hdmLValTF.requestFocus();
               hdmLValTF.selectAll();
               return false;
           }

           tmpStr = hdmHValTF.getText();
           if ( testParameter(tmpStr, -4000, 4000) ){
               hdmHVal = Integer.valueOf(tmpStr).intValue();
           }
           else{
               hdmHValTF.requestFocus();
               hdmHValTF.selectAll();
               return false;
           }

         return true;
       }   // end setVariables()

         /**
         *      Once all the necessary variables are set, call the Gaussian Blur
         *      algorithm based on what type of image this is and whether or not there
         *      is a separate destination image.
         */
         private void callAlgorithm() {
               String name = makeImageName(image.getImageName(), "_CTseg");

           // stuff to do when working on 2-D images.
           if (image.getNDims() == 2 ) {                   // source image is 2D
               int destExtents[] = new int[2];
               destExtents[0] = image.getExtents()[0];     // X dim
               destExtents[1] = image.getExtents()[1];     // Y dim

               try{
                   // Make result image of Ubyte type
                   resultImage     = new ModelImage(ModelStorageBase.UBYTE, destExtents, name,
                                       userInterface);

                   // Make algorithm
                   boolean entireFlag = true;

     //ctSegAlgo = new PlugInAlgorithmFlowWrapFix(resultImage, image, iters,
    // kernelSize, kernelShape, stdDev, regionFlag);
                   ctSegAlgo = new PlugInAlgorithmCT_MD(resultImage, image);

                   System.out.println("Dialog fatL = " + fatLVal + " fatH = " + fatHVal);
                   ctSegAlgo.fatL = fatLVal;
                   ctSegAlgo.fatH = fatHVal;
                   ctSegAlgo.ldmL = ldmLVal;
                   ctSegAlgo.ldmH = ldmHVal;
```

Figure 341. PlugInDialogCT_MD.java (continued)

```
                ctSegAlgo.hdmL = hdmLVal;
                ctSegAlgo.hdmH = hdmHVal;



                // This is very important. Adding this object as a listener allows the
              // algorithm to notify this object when it has completed or failed. See
                // algorithm performed event.
                // This is made possible by implementing AlgorithmedPerformed interface
                ctSegAlgo.addListener(this);
                setVisible(false);  // Hide dialog

                if (runInSeparateThread) {
                    // Start the thread as a low priority because we wish to still have
                // user interface work fast.
                    if (ctSegAlgo.startMethod(Thread.MIN_PRIORITY) == false){
                        MipavUtil.displayError("A thread is already running on this object");
                    }
                }
                else {
                    ctSegAlgo.run();
                }
            }
            catch (OutOfMemoryError x){
                MipavUtil.displayError("Dialog median: unable to allocate enough memory");
                if (resultImage != null){
                    resultImage.disposeLocal();  // Clean up memory of result image
                    resultImage = null;
                }
                return;
            }
        }
        else if (image.getNDims() == 3 ) {
            int destExtents[] = new int[3];
            destExtents[0] = image.getExtents()[0];
            destExtents[1] = image.getExtents()[1];
            destExtents[2] = image.getExtents()[2];

            try{
                // Make result image of float type
                resultImage     = new ModelImage(ModelStorageBase.UBYTE, destExtents, name,
                                  userInterface);
                boolean entireFlag = true;

                ctSegAlgo = new PlugInAlgorithmCT_MD(resultImage, image);
                ctSegAlgo.fatL = fatLVal;
                ctSegAlgo.fatH = fatHVal;
                ctSegAlgo.ldmL = ldmLVal;
                ctSegAlgo.ldmH = ldmHVal;
                ctSegAlgo.hdmL = hdmLVal;
                ctSegAlgo.hdmH = hdmHVal;
```

Figure 341. PlugInDialogCT_MD.java (continued)

```
431                     // This is very important. Adding this object as a listener allows the
432                     // algorithm to notify this object when it has completed or failed.
433                 // See algorithm performed event. This is made possible by implementing
434                     //  AlgorithmedPerformed interface
435                     ctSegAlgo.addListener(this);
436                     setVisible(false);        // Hide dialog
437
438                     if (runInSeparateThread) {
439                         // Start the thread as a low priority because we wish to still have
440                     // user interface work fast.
441                         if (ctSegAlgo.startMethod(Thread.MIN_PRIORITY) == false){
442                             MipavUtil.displayError("A thread is already running on this object");
443                         }
444                     }
445                     else {
446                         ctSegAlgo.run();
447                     }
448                 }
449                 catch (OutOfMemoryError x){
450                     MipavUtil.displayError("Dialog median: unable to allocate enough memory");
451                     if (resultImage != null){
452                         resultImage.disposeLocal();      // Clean up image memory
453                         resultImage = null;
454                     }
455                     return;
456                 }
457             }
458         } // end callAlgorithm()
459
460     }
```

Figure 341. PlugInDialogCT_MD.java (continued)

PlugInAlgorithmCT_MD.java

```
1     import gov.nih.mipav.model.algorithms.*;
2     import gov.nih.mipav.model.structures.*;
3     import gov.nih.mipav.view.*;
4
5     import java.io.*;
6     import java.util.*;
7
8
9     /**
10    *
11    *   This shows how to extend the AlgorithmBase class.
12    *
13    *     Supports the segmentation
14    *     CT scans:
```

Figure 342. PlugInAlgorithmCT_MD.java

```
15    *     Fat:                  -190 to -30
16    *     Low density muscle:       0 to  30
17    *     High density muscle:     31 to 100
18    *     If you have any questions, please drop me a line.
19    * =====
20    * Matthew J. Delmonico, MS, MPH
21    * Graduate Research Assistant, Exercise Physiology
22    * 2132 HHP Building
23    * University of Maryland
24    * College Park, MD  20742
25    * (301) 405-2569
26    * (301) 793-0567 (cell)
27    *
28    *   @version    July 12, 2002
29    *   @author
30    *   @see        AlgorithmBase
31    *
32    *   $Logfile: /mipav/src/plugins/PlugInAlgorithmCT_MD.java $
33    *   $Revision: 10 $
34    *   $Date: 10/13/04 1:09p $
35    *
36    */
37    public class PlugInAlgorithmCT_MD extends AlgorithmBase {
38
39
40        private boolean     entireImage = true;
41
42        public int         fatL    = -190;
43        public int         fatH    = -30;
44
45        public int         ldmL    = 0;
46        public int         ldmH    = 30;
47
48        public int         hdmL    = 31;
49        public int         hdmH    = 100;
50
51
52        /**
53        * Constructor for 3D images in which changes are placed in a predetermined
54       * destination image.
55        *   @param destImg      Image model where result image is to stored.
56        *   @param srcImg       Source image model.
57        */
58          public PlugInAlgorithmCT_MD(ModelImage destImg, ModelImage srcImg) {
59           super(destImg, srcImg);
60          }
61
62    /**
63        * Prepares this class for destruction.
64    */
65          public void finalize(){
66              destImage   = null;
67              srcImage    = null;
68              super.finalize();
69          }
70
```

Figure 342. PlugInAlgorithmCT_MD.java (continued)

```
71        /**
72        *   Starts the algorithm.
73        */
74          public void run() {
75
76            if (srcImage  == null) {
77                displayError("Source Image is null");
78                notifyListeners(this);
79                return;
80            }
81            if (destImage  == null) {
82                displayError("Source Image is null");
83                notifyListeners(this);
84                return;
85            }
86
87
88            // start the timer to compute the elapsed time
89            setStartTime();
90
91            if (destImage != null){     // if there exists a destination image
92                if (srcImage.getNDims() == 2){
93                     calcStoreInDest2D();
94                  }
95                  else if (srcImage.getNDims() > 2) {
96                      calcStoreInDest3D();
97                  }
98            }
99
100           // compute the elapsed time
101           computeElapsedTime();
102           notifyListeners(this);
103       }
104
105       /**
106       * This function produces a new image that has been median filtered and places
107      *  filtered image in the destination image.
108       */
109       private void calcStoreInDest2D(){
110
111
112           int length;               // total number of data-elements (pixels) in image
113           float buffer[];           // data-buffer (for pixel data) which is the "heart"
114                                  // of the image
```

Figure 343. PlugInAlgorithmCT_MD.java

```
115             try {
116                 // image length is length in 2 dims
117                 length = srcImage.getExtents()[0]  * srcImage.getExtents()[1];
118                 buffer       = new float[length];
119                 srcImage.exportData(0,length, buffer); // locks and releases lock
120             }
121             catch (IOException error) {
122                 buffer = null;
123                 errorCleanUp("Algorithm CT_MD reports: source image locked", true);
124                 return;
125             }
126             catch (OutOfMemoryError e){
127                 buffer = null;
128                 errorCleanUp("Algorithm CT_MD reports: out of memory", true);
129                 return;
130             }
131 
132             int mod = length/100; // mod is 1 percent of length
133             initProgressBar();
134 
135             // Fat:  -190 to -30
136             // Low density muscle:  0 to 30
137             // High density muscle:  31 to 100
138             BitSet mask = null;
139             if (srcImage.getVOIs().size() > 0 ) {
140                 mask = srcImage.generateVOIMask();
141                 entireImage = false;
142             }
143 
144             int fat      = 0;
145             int ldMuscle = 0;
146             int hdMuscle = 0;
147             for (int i = 0; i < length && !threadStopped; i++){
148                 if (isProgressBarVisible() && (i)%mod==0)
149                     progressBar.setValue(Math.round((float)(i)/(length-1) * 100));
150 
151                 if (entireImage == true || mask.get(i) ) {
152                     if( buffer[i] >= fatL && buffer[i] <= fatH  ) {
153                         destImage.set(i, 20);
154                         fat++;
155                     }
156                     else if( buffer[i] >= ldmL && buffer[i] <= ldmH  ) {
157                         destImage.set(i, 40);
158                         ldMuscle++;
159                     }
160                     else if( buffer[i] >= hdmL && buffer[i] <= hdmH  ) {
161                         destImage.set(i, 60);
162                         hdMuscle++;
163                     }
164                     else {
165                         destImage.set(i, 0);
166                         //buffer[i] = (float)srcImage.getMin();
167 
168                 }
169                 }
170             }
171 
172 
```

Figure 343. PlugInAlgorithmCT_MD.java

```
173           //destImage.releaseLock();
174
175           if (threadStopped) {
176           finalize();
177           return;
178           }
179
180           float area = srcImage.getFileInfo()[0].getResolutions()[0] *
181                        srcImage.getFileInfo()[0].getResolutions()[1];
182
183           destImage.getUserInterface().getMessageFrame().append("Number of Fat pixels = " +
184              fat , ViewJFrameMessage.DATA );
185           destImage.getUserInterface().getMessageFrame().append("  Area = " + (fat*area) +
186              " mm^2\n", ViewJFrameMessage.DATA );
187
188           destImage.getUserInterface().getMessageFrame().append("Number of LDM pixels = " +
189              ldMuscle , ViewJFrameMessage.DATA );
190           destImage.getUserInterface().getMessageFrame().append("  Area = " + (ldMuscle*area) +
191              " mm^2\n", ViewJFrameMessage.DATA );
192
193           destImage.getUserInterface().getMessageFrame().append("Number of HDM pixels = " +
194              hdMuscle , ViewJFrameMessage.DATA );
195           destImage.getUserInterface().getMessageFrame().append("  Area = " + (hdMuscle*area) +
196              " mm^2\n", ViewJFrameMessage.DATA );
197
198           destImage.calcMinMax();
199           setCompleted(true);
200       }
201
202       /**
203       * This function produces a new volume image that has been median filtered.
204       * Image can be filtered by filtering each slice individually, or by filtering
205      *  using a kernel-volume.
206       */
207       private void calcStoreInDest3D(){
208
209           int totLength, imgLength;
210           float buffer[];
211
212           float vol = srcImage.getFileInfo()[0].getResolutions()[0] *
213                       srcImage.getFileInfo()[0].getResolutions()[1] *
214                       srcImage.getFileInfo()[0].getResolutions()[2];
215
216           try {
217               // image totLength is totLength in 3 dims
218               imgLength = srcImage.getSliceSize();
219               totLength = srcImage.getSliceSize() * srcImage.getExtents()[2];
220               buffer = new float[totLength];
221               srcImage.exportData(0,totLength, buffer); // locks and releases lock
222               buildProgressBar(srcImage.getImageName(), "Processing image ...", 0, 100);
223           }
224
225   catch (IOException error) {
226               buffer = null;
227               errorCleanUp("Algorithm CT_MD: source image locked", true);
228               return;
229           }
```

Figure 343. PlugInAlgorithmCT_MD.java

```
230             catch (OutOfMemoryError e){
231                 buffer = null;
232                 errorCleanUp("Algorithm CT_MD: Out of memory creating process buffer", true);
233                 return;
234             }
235 
236             int totFat      = 0;
237             int totLdMuscle = 0;
238             int totHdMuscle = 0;
239             initProgressBar();
240 
241             for (int i = 0; i < srcImage.getExtents()[2] && !threadStopped; i++){
242                 int fat      = 0;
243                 int ldMuscle = 0;
244                 int hdMuscle = 0;
245 
246                 if ( isProgressBarVisible() )
247                         progressBar.setValue(Math.round((float)(i)/(srcImage.getExtents()[2]-1) *
248                         100));
249 
250                 for (int j = 0; j < imgLength && !threadStopped; j++){
251                     //System.out.println(" j = " + j);
252                     int index = i*imgLength+j;
253                     if( buffer[index] >= fatL && buffer[index] <= fatH  ) {
254                         destImage.set(index, 60);
255                         totFat++;
256                         fat++;
257                     }
258                     else if( buffer[index] >= ldmL && buffer[index] <= ldmH  ) {
259                         destImage.set(index, 120);
260                         totLdMuscle++;
261                         ldMuscle++;
262                     }
263                     else if( buffer[index] >= hdmL && buffer[index] <= hdmH  ) {
264                         destImage.set(index, 200);
265                         totHdMuscle++;
266                         hdMuscle++;
267                     }
268                     else {
269                         destImage.set(index, 0);
270                         //buffer[i] = -1024;
271                     }
272                 }
273                destImage.getUserInterface().getMessageFrame().append("\n\n ***************** Slice
274                     " + i + " totals ***************\n",
275                 ViewJFrameMessage.DATA);
276                 destImage.getUserInterface().getMessageFrame().append("Number of fat pixels = " +
277                     fat , ViewJFrameMessage.DATA );
278                 destImage.getUserInterface().getMessageFrame().append("  Volume = " + (fat*vol) +
279                     " mm^3\n", ViewJFrameMessage.DATA );
280 
281                  destImage.getUserInterface().getMessageFrame().append("Number of LDM pixels = " +
282                     ldMuscle , ViewJFrameMessage.DATA );
283                 destImage.getUserInterface().getMessageFrame().append("  Volume = " +
284                     (ldMuscle*vol) + " mm^3\n", ViewJFrameMessage.DATA );
285 
```

Figure 343. PlugInAlgorithmCT_MD.java

```
286                     destImage.getUserInterface().getMessageFrame().append("Number of HDM pixels
287                          = " + hdMuscle , ViewJFrameMessage.DATA );
288                 destImage.getUserInterface().getMessageFrame().append("  Volume = " +
289                         (hdMuscle*vol) + " mm^3\n", ViewJFrameMessage.DATA );
290             }
291
292             destImage.releaseLock();
293
294             if (threadStopped) {
295             finalize();
296             return;
297             }
298
299             destImage.getUserInterface().getMessageFrame().append("\n **********************
300                 Totals ********************\n",
301             ViewJFrameMessage.DATA);
302             destImage.getUserInterface().getMessageFrame().append("Number of totFat pixels = " +
303                 totFat , ViewJFrameMessage.DATA );
304             destImage.getUserInterface().getMessageFrame().append("  Volume = " + (totFat*vol) +
305                 " mm^3\n", ViewJFrameMessage.DATA );
306
307             destImage.getUserInterface().getMessageFrame().append("Number of LDM pixels = " +
308                  totLdMuscle , ViewJFrameMessage.DATA );
309            destImage.getUserInterface().getMessageFrame().append("  Volume = " + (totLdMuscle*vol)
310                  + " mm^3\n", ViewJFrameMessage.DATA );
311
312             destImage.getUserInterface().getMessageFrame().append("Number of HDM pixels = " +
313                  totHdMuscle , ViewJFrameMessage.DATA );
314            destImage.getUserInterface().getMessageFrame().append("  Volume = " + (totHdMuscle*vol)
315                  + " mm^3\n", ViewJFrameMessage.DATA );
316
317             destImage.calcMinMax();
318             progressBar.dispose();
319             setCompleted(true);
320         }
321     }
```

Figure 343. PlugInAlgorithmCT_MD.java

PlugInCT_MD.java

```
1     import plugins.PlugInDialogCT_MT;     //associated class file
2     import gov.nih.mipav.plugins.*;       //needed to load PlugInAlgorithm / PlugInView /
3                                           //PlugInFile interface
4     import gov.nih.mipav.view.*;
5     import gov.nih.mipav.model.structures.*;
6
7     import java.awt.*;
8
9     /**
10    *  This is a simple plugin for the University of Maryland to simple segment an
11    *  imagebased on CT Hounsfield units.
12    *
13    * @see PlugInAlgorithm
14    */
15
16    //This is an Algorithm type of PlugIn, and therefore must implement PlugInAlgorithm
17    //Implementing the PlugInAlgorithm requires this class to implement the run method
18    //with the correct parameters
19    public class PlugInCT_MD implements PlugInAlgorithm {
20
21       /**
22         * Defines body of run method, which was declared in the interface.
23         * @param UI            User Interface
24         * @param parentFrame   ParentFrame
25         * @param image         Current ModelImage--this is an image already loaded into
26         *                      MIPAV. Can be null.
27         */
28        public void run (ViewUserInterface UI, Frame parentFrame, ModelImage image){
29
30            if (parentFrame instanceof ViewJFrameImage)
31              new PlugInDialogCT_MD (parentFrame,image);
32
33            else
34              MipavUtil.displayError ("PlugIn CT_MD only runs on an image frame.");
35            }
36        }
37    }
```

Figure 344. PlugInCT_MD.java

PlugInDialogImageVOIDisplay.java

```
1     import java.awt.*;
2     import java.awt.event.*;
3     import java.util.Vector;
4
5     import javax.swing.*;
6     import gov.nih.mipav.model.file.FileInfoBase;
7     import gov.nih.mipav.model.structures.ModelImage;
8     import gov.nih.mipav.model.structures.ModelLUT;
9     import gov.nih.mipav.model.structures.ModelRGB;
10    import gov.nih.mipav.model.structures.VOI;
11    import gov.nih.mipav.view.*;
12    import gov.nih.mipav.view.dialogs.*;
13
14
15    /**
16     * Plugin example class for creating a simple, self-contained frame that extends ViewJFrame
      Image
17     * Contains a subset of the VOI functions, as well as the message frame contained within the
      frame itself
18     * @author linkb
19     *
20     */
21    public class PlugInDialogImageVOIDisplay extends ViewJFrameImage implements MouseListener,
      AdjustmentListener {}
22     //~ Constructors -----------------------------------------------------------------------------
      -----------------------
23
24       /**
25        * Default constructor
26        */
27        public PlugInDialogImageVOIDisplay(ModelImage image) {
28          super(image, null, null, false, false);
29            init();
30        }
31
32
33        /**
34         * ViewOpenFrameInterface function for opening a model image (result) into a new frame
35         */
36        public PlugInDialogImageVOIDisplay openFrame(ModelImage image) {
37          return new PlugInDialogImageVOIDisplay(image);
38    //~ Methods ----------------------------------------------------------------------------------
      ----------------------
39
40        // ************************************************************************
41        // ************************** Event Processing ***************************
42        // ************************************************************************
43     public void adjustmentValueChanged(AdjustmentEvent e) {
44          updateImages(true);
45        }
46
```

Figure 345. PlugInDialogImageVOIDisplay.java

```
47      /**
48           * Closes dialog box when the OK button is pressed and calls the algorithm.
49           *
50           * @param  event  Event that triggers function.
51           */
52          public void actionPerformed(ActionEvent event) {
53              String command = event.getActionCommand();
54              System.err.println("command: " + command);
55
56              //run through toggle buttons to see if a menu selected one (updates the button status)
57              getControls().getTools().setToggleButtonSelected(command);
58
59              if (command.equals("Gaussian blur")) {
60                  new JDialogGaussianBlur(this, getActiveImage());
61              } else if (command.equals("Gradient magnitude")) {
62                  new JDialogGradientMagnitude(this, getActiveImage());
63              } else if (command.equals("Open")) {
64              //ViewUserInterface.getReference().openImageFrame();
65              } else if (command.equals(CustomUIBuilder.PARAM_VOI_DEFAULT_POINTER)) {
66                  componentImage.setCursorMode(ViewJComponentEditImage.DEFAULT);
67              } else if (command.equals(CustomUIBuilder.PARAM_VOI_POINT.getActionCommand())) {
68
69
70                  if
        (!componentImage.getVOIHandler().checkForVOICompatibility(getActiveImage().getVOIs(),
        VOI.POINT, getControls())) {
71                      componentImage.setCursorMode(ViewJComponentEditImage.NEW_VOI);
72                  }
73
74                  componentImage.setCursorMode(ViewJComponentEditImage.POINT_VOI);
75              } else if (command.equals(CustomUIBuilder.PARAM_VOI_LINE.getActionCommand())) {
76
77                  if
        (!componentImage.getVOIHandler().checkForVOICompatibility(getActiveImage().getVOIs(),
        VOI.LINE, getControls())) {
78                      componentImage.setCursorMode(ViewJComponentEditImage.NEW_VOI);
79                  }
80
81                  componentImage.setCursorMode(ViewJComponentEditImage.LINE);
82              } else if (command.equals("SplitVOI")) {
83                  componentImage.setCursorMode(ViewJComponentEditImage.SPLIT_VOI);
84              } else if (command.equals(CustomUIBuilder.PARAM_VOI_POLY_SLICE.getActionCommand())) {
85
86               if
        (!componentImage.getVOIHandler().checkForVOICompatibility(getActiveImage().getVOIs(),
        VOI.POLYLINE_SLICE, getControls())) {
87                       componentImage.setCursorMode(ViewJComponentEditImage.NEW_VOI);
88                   }
89
90                  componentImage.setCursorMode(ViewJComponentEditImage.POLYLINE_SLICE_VOI);
91              } else if (command.equals("protractor")) {
92
93               if
        (!componentImage.getVOIHandler().checkForVOICompatibility(getActiveImage().getVOIs(),
        VOI.PROTRACTOR, getControls())) {
94                       componentImage.setCursorMode(ViewJComponentEditImage.NEW_VOI);
95                   }
```

Figure 345. PlugInDialogImageVOIDisplay.java (continued)

```
96      componentImage.setCursorMode(ViewJComponentEditImage.PROTRACTOR);
97              } else if (command.equals("Polyline")) {
98
99              if
        (!componentImage.getVOIHandler().checkForVOICompatibility(getActiveImage().getVOIs(),
        VOI.POLYLINE, getControls())) {
100                 componentImage.setCursorMode(ViewJComponentEditImage.NEW_VOI);
101             }
102
103             componentImage.setCursorMode(ViewJComponentEditImage.POLYLINE);
104         } else if (command.equals(CustomUIBuilder.PARAM_VOI_TEXT.getActionCommand())) {
105
106         componentImage.setCursorMode(ViewJComponentEditImage.NEW_VOI);
107
108
109             componentImage.setCursorMode(ViewJComponentEditImage.ANNOTATION);
110         } else if (command.equals("RectVOI")) {
111
112         if
        (!componentImage.getVOIHandler().checkForVOICompatibility(getActiveImage().getVOIs(),
        VOI.CONTOUR, getControls())) {
113                 componentImage.setCursorMode(ViewJComponentEditImage.NEW_VOI);
114             }
115
116             componentImage.setCursorMode(ViewJComponentEditImage.RECTANGLE);
117         } else if (command.equals("EllipseVOI")) {
118         if
        (!componentImage.getVOIHandler().checkForVOICompatibility(getActiveImage().getVOIs(),
        VOI.CONTOUR, getControls())) {
119                 componentImage.setCursorMode(ViewJComponentEditImage.NEW_VOI);
120             }
121
122             componentImage.setCursorMode(ViewJComponentEditImage.ELLIPSE);
123         } else if (command.equals("LevelSetVOI")) {
124         componentImage.getVOIHandler().checkForVOICompatibility(getActiveImage().getVOIs(),
        VOI.CONTOUR, getControls());
125             componentImage.setCursorMode(ViewJComponentEditImage.LEVELSET);
126         } else if (command.equals("Rect3DVOI")) {
127         componentImage.getVOIHandler().checkForVOICompatibility(getActiveImage().getVOIs(),
        VOI.CONTOUR, getControls());
128             componentImage.setCursorMode(ViewJComponentEditImage.RECTANGLE3D);
129         } else if (command.equals("LiveWireVOI")) {
130         componentImage.getVOIHandler().checkForVOICompatibility(getActiveImage().getVOIs(),
        VOI.CONTOUR, getControls());
131
132             if (componentImage.getVOIHandler().isLivewireNull()) {
133                 JDialogLivewire dialog = new JDialogLivewire(this);
134
135                 if (!dialog.isCancelled()) {
136                     componentImage.getVOIHandler().setModeLivewire(dialog.getSelection());
137                     componentImage.setCursorMode(ViewJComponentEditImage.LIVEWIRE);
138                 }
139             } else {
140                 componentImage.setCursorMode(ViewJComponentEditImage.LIVEWIRE);
141             }
142         } else if (command.equals("NewVOI")) {
143             componentImage.setCursorMode(ViewJComponentEditImage.NEW_VOI);
```

Figure 345. PlugInDialogImageVOIDisplay.java (continued)

```
144   int id = (getActiveImage().getVOIs().size() > 0)
145                       ? (((VOI) (getActiveImage().getVOIs().lastElement())).getID() + 1) : -1;
146
147               getControls().setVOIColor(id);
148           } else if (command.equals("cutVOI")) {
149
150               if (componentImage.getVOIHandler().copyVOItoClipBrd()) {
151                   componentImage.getVOIHandler().deleteSelectedVOI(true);
152               }
153           } else if (command.equals("copyVOI")) {
154               componentImage.getVOIHandler().copyVOItoClipBrd();
155           } else if (command.equals("pasteVOI")) {
156               componentImage.getVOIHandler().pasteVOI();
157           } else if (command.equals("selectAllVOIs")) {
158               componentImage.getVOIHandler().selectAllVOIs(true);
159           } else if (event.getActionCommand().equals("voiSelectNone")) {
160               componentImage.getVOIHandler().selectAllVOIs(false);
161           } else if (command.equals("deleteVOI")) {
162               componentImage.getVOIHandler().deleteSelectedVOI(true);
163           } else if (command.equals("BringToFront")) {
164               componentImage.getVOIHandler().changeVOIOrder(false, VOIHandler.FRONT);
165           } else if (command.equals("SendToBack")) {
166               componentImage.getVOIHandler().changeVOIOrder(false, VOIHandler.BACK);
167           } else if (command.equals("BringContourToFront")) {
168               componentImage.getVOIHandler().changeVOIOrder(true, VOIHandler.FRONT);
169           } else if (command.equals("SendContourToBack")) {
170               componentImage.getVOIHandler().changeVOIOrder(false, VOIHandler.BACK);
171           } else if (command.equals("PropVOIUp")) {
172
173               // It appears JButtons don't pass key modifiers
174               // if((event.getModifiers() & ActionEvent.SHIFT_MASK) != 0) {}
175               if (componentImage.getVOIHandler().propVOI(1, false) == true) {
176                   incSlice();
177               }
178           } else if (command.equals("PropVOIDown")) {
179
180               if (componentImage.getVOIHandler().propVOI(-1, false) == true) {
181                   decSlice();
182               }
183           } else if (command.equals("PropVOIActiveUp")) {
184
185               // It appears JButtons don't pass key modifiers
186               // if((event.getModifiers() & ActionEvent.SHIFT_MASK) != 0) {}
187               if (componentImage.getVOIHandler().propVOI(1, true) == true) {
188                   incSlice();
189               }
190   } else if (command.equals("PropVOIActiveDown")) {
191
192               if (componentImage.getVOIHandler().propVOI(-1, true) == true) {
193                   decSlice();
194               }
195   } else if (command.equals("PropVOIAll")) {
196               componentImage.getVOIHandler().propVOIAll();
197           } else if (command.equals("BringForward")) {
198               componentImage.getVOIHandler().changeVOIOrder(false, VOIHandler.FORWARD);
199           } else if (command.equals("SendBackward")) {
```

Figure 345. PlugInDialogImageVOIDisplay.java (continued)

```
componentImage.getVOIHandler().changeVOIOrder(false, VOIHandler.BACKWARD);
        } else if (command.equals("SendContourForward")) {
            componentImage.getVOIHandler().changeVOIOrder(true, VOIHandler.FORWARD);
        } else if (command.equals("SendContourBackward")) {
            componentImage.getVOIHandler().changeVOIOrder(true, VOIHandler.BACKWARD);
        } else if (command.equals("VOIProperties")) {

            componentImage.getVOIHandler().showVOIProperties(false);

        } else if (command.equals("VOIPropertiesColor")) {

            if (getActiveImage().getVOIs().size() > 0) {

                ViewVOIVector VOIs = getActiveImage().getVOIs();

                int i;
                int nVOI = VOIs.size();

                for (i = 0; i < nVOI; i++) {

                    if ((VOIs.VOIAt(i).isActive() == true) &&
                            ((VOIs.VOIAt(i).getCurveType() == VOI.CONTOUR) ||
                                 (VOIs.VOIAt(i).getCurveType() == VOI.POLYLINE) ||
                                 (VOIs.VOIAt(i).getCurveType() == VOI.POINT) ||
                                 (VOIs.VOIAt(i).getCurveType() == VOI.LINE) ||
                                 (VOIs.VOIAt(i).getCurveType() == VOI.PROTRACTOR))) {
                        break;
                    } else if ((VOIs.VOIAt(i).isActive() == true) &&
    (VOIs.VOIAt(i).getCurveType() == VOI.ANNOTATION)) {
                        MipavUtil.displayInfo("Double-click annotation to change properties");
                        i = -1;

                        break;
                    }
                }

                if (i == nVOI) {
                    MipavUtil.displayError("Please select VOI");
                } else if (i == -1) { // there was an annotation selected, do nothing
                } else {
                    componentImage.getVOIHandler().showVOIProperties(true);
                }
            } else {
                MipavUtil.displayWarning("Image has no VOIs!");
            }

        }
    }

    /**
     * Can handle actions for the resizing of the frame
     */
    public synchronized void componentResized(ComponentEvent event) {

    }

```

Figure 345. PlugInDialogImageVOIDisplay.java (continued)

```
255
256         /**
257          * Override MouseListener functions to prevent MouseEvent catching in ViewJFrameImage
258          */
259          public void mousePressed(MouseEvent e) {}
260          public void mouseReleased(MouseEvent e) {}
261          public void mouseEntered(MouseEvent e) {}
262          public void mouseExited(MouseEvent e) {}
263          public void mouseClicked(MouseEvent e) {}
264
265
266          /**
267           * Initialize the frame using a lut (can be null)
268           * @param LUTa the ModelLUT
269           * @throws OutOfMemoryError
270           */
271          private void init() throws OutOfMemoryError {
272
273              try {
274                  setIconImage(MipavUtil.getIconImage("davinci_32x32.gif"));
275              } catch (Exception error) {
276                  Preferences.debug("Exception ocurred while getting <" + error.getMessage() +
277                                    ">.  Check that this file is available.\n");
278              }
279
280              setResizable(true);
281
282              // initialize logMagDisplay
283              this.LUTa = initLUT(imageA);
284
285              initResolutions();
286              initZoom();
287
288              int[] extents = createBuffers();
289
290              initComponentImage(extents);
291              initExtentsVariables(imageA);
292
293              // create and build the menus and controls
294              controls = new ViewControlsImage(this); // Build controls used in this frame
295              menuBuilder = new ViewMenuBuilder(this);
296
297              // build the menuBar based on the number of dimensions for imageA
298              menuBarMaker = new ViewMenuBar(menuBuilder);
299
300              //create a custom menu bar using Vectors of UIParams
301              JMenuBar menuBar = new JMenuBar();
302
303              //add pre-defined UIParams to the vector (will be added to both menu and toolbar)
304              Vector<CustomUIBuilder.UIParams> voiParams = new Vector<CustomUIBuilder.UIParams>();
305              voiParams.addElement(CustomUIBuilder.PARAM_VOI_DEFAULT_POINTER);
306              voiParams.addElement(CustomUIBuilder.PARAM_VOI_POINT);
307              voiParams.addElement(CustomUIBuilder.PARAM_VOI_ELLIPSE);
308              voiParams.addElement(CustomUIBuilder.PARAM_VOI_RECTANGLE);
309              Vector<CustomUIBuilder.UIParams> algoParams = new Vector<CustomUIBuilder.UIParams>();
310              algoParams.add(new CustomUIBuilder.UIParams("Gaussian blur", null, null));
311              algoParams.add(new CustomUIBuilder.UIParams("Gradient magnitude", null, null));
```

Figure 345. PlugInDialogImageVOIDisplay.java (continued)

```
312   menuBar.add(menuBarMaker.makeCustomMenu("VOI example", voiParams));
313           menuBar.add(menuBarMaker.makeCustomMenu("Simple algorithm menu", algoParams));
314
315           //create a simple toolbar (rather than the default ViewJFrameImage specific toolbar)
316           //buttons will be added to the toolbar with the function call .addCustomToolBar()
317           controls.buildSimpleToolBar();
318
319           controls.addCustomToolBar(voiParams);
320
321           setTitle();
322
323           JPanel centerPanel = new JPanel();
324           centerPanel.add(componentImage, BorderLayout.CENTER);
325
326           // The component image will be displayed in a scrollpane.
327         scrollPane = new JScrollPane(centerPanel, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
328                                         JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
329
330         JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT, scrollPane,
331                 ViewUserInterface.getReference().getMessageFrame().getTabbedPane());
332         splitPane.setDividerLocation(350);
333
334         getContentPane().add(splitPane);
335         scrollPane.setBackground(Color.black);
336
337         setBackground(Color.black);
338
339
340
341           // MUST register frame to image models
342           imageA.addImageDisplayListener(this);
343
344           if (imageB != null) {
345               imageB.addImageDisplayListener(this);
346           }
347
348           windowLevel = new JDialogWinLevel[2];
349
350           this.setLocation(100, 50);
351
352           setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
353    pack();
354
355           scrollPane.setPreferredSize(new Dimension(800,800));
356           scrollPane.getVerticalScrollBar().addAdjustmentListener(this);
357           scrollPane.getHorizontalScrollBar().addAdjustmentListener(this);
358           scrollPane.addComponentListener(this);
359
360           setSize(1000,750);
361           // User interface will have list of frames
362           userInterface.registerFrame(this);
363   this.updateImages(true);
364           addComponentListener(this);
365
366           this.setJMenuBar(menuBar);
367           getContentPane().add(controls, BorderLayout.NORTH);
```

Figure 345. PlugInDialogImageVOIDisplay.java (continued)

```
368   this.addWindowListener(new WindowAdapter() {
369                public void windowClosing(WindowEvent we) {
370                    System.exit(0);
371                }
372             });
373
374             setVisible(true);
375          } // end init()
376
377
378           /**
379            * Sets the title of the frame
380            */
381           public void setTitle() {
382                 this.setTitle("Simple Image Frame: " + imageA.getImageName());
383           }
384
385
386
387   }
```

Figure 345. PlugInDialogImageVOIDisplay.java (continued)