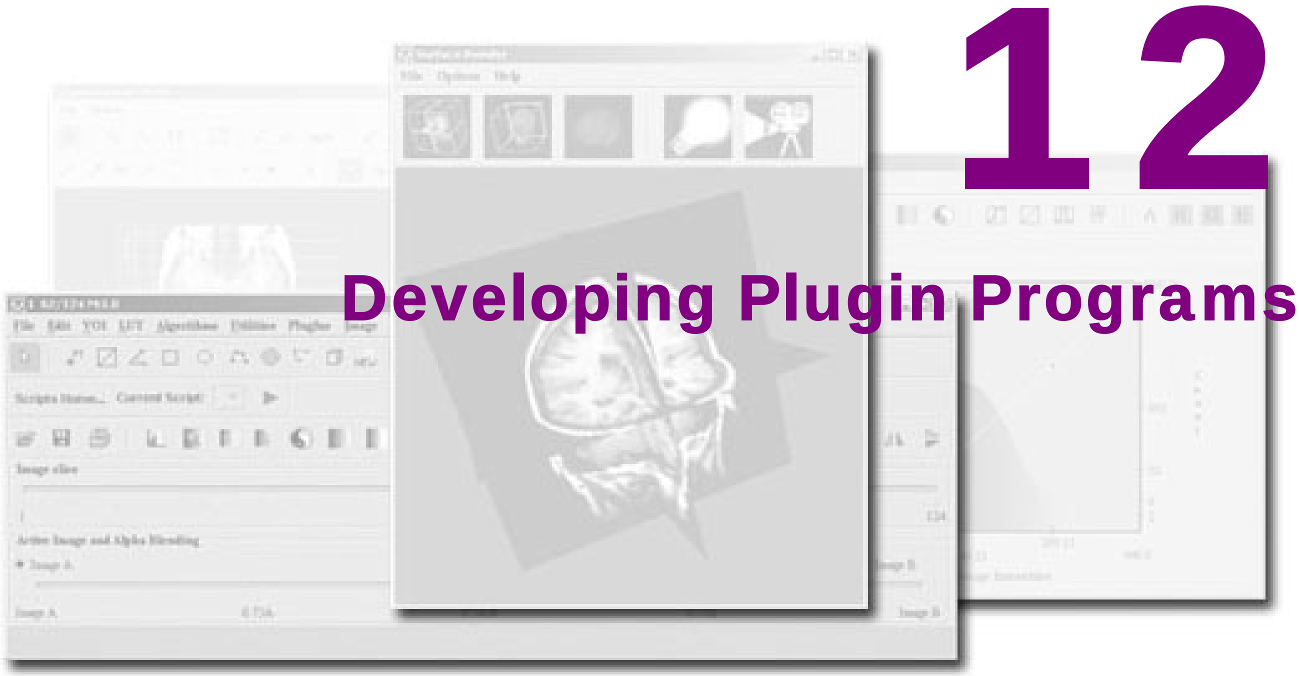


12 Developing Plugin Programs

In this chapter. . .

Users who know how to program in Java can write a *plugin 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 plugin 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 plugin types
- Include mandatory lines of code in plugin programs so that they interface correctly with MIPAV
- Install plugin programs

Understanding plugin programs

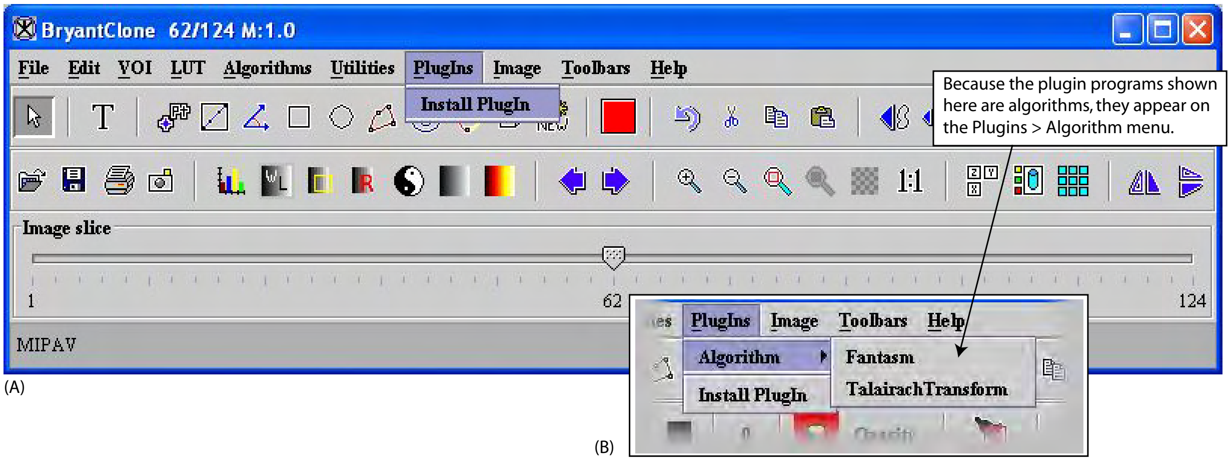
Plugin programs, also known simply as *plugins*, are utilities or sets of instructions that add functionality to a program without changing the program. In MIPAV, you use Java to write and compile plugin programs to perform specific functions, such as automatically removing all odd-numbered images from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

- **Algorithm**—An algorithm type of plugin performs a function on an image. An example is a plugin that applies a radial blur algorithm to an image. You can create plugin algorithms through Java.
- **File**—A file type of plugin allows MIPAV to support a new file format. An example is a plugin that allows MIPAV to view Kodak Photo CD files (.pcd).
- **View**—A view type of plugin 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 plugins. Most plugin programs, therefore, are algorithms.

After developing a plugin program, you can then install the plugin program into the MIPAV application and access it from the Plugins menu in the MIPAV window. The MIPAV window labeled “(A)” in Figure 310 shows the Plugins menu as it appears before any plugin programs are installed. The picture labeled “(B)” in Figure 310 shows the Plugins menu as it appears after two plugin programs—in this case, the Fantasm plugin program and the Talairach Transform plugin program—are installed. Because the Fantasm and Talairach Transform plugin programs are algorithms, they appear under the Plugins > Algorithm menu.

Note: If a plugin program is a file type of plugin, it would appear under a Plugins > File menu. If it is a view type, it would appear under a Plugins > View menu.



Because the plugin programs shown here are algorithms, they appear on the Plugins > Algorithm menu.

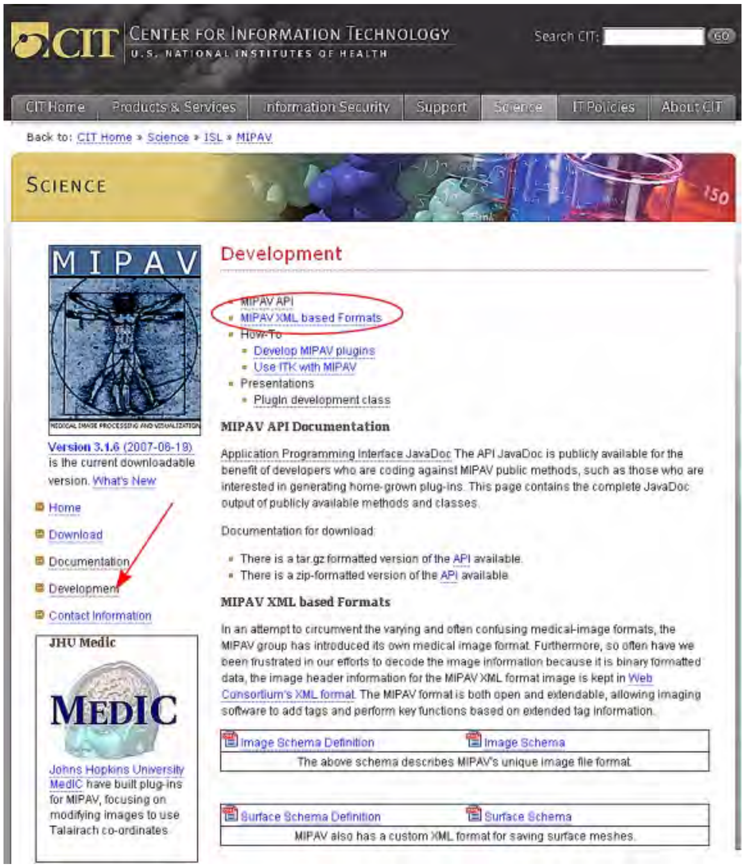
Figure 310. Plugins menu in the MIPAV window: (A) Before a plugin is installed and (B) after two algorithm plugins are installed

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 311.
3. Here, use the following links:
   - MIPAV API
   - MIPAV XML based Formats



CIT CENTER FOR INFORMATION TECHNOLOGY
U.S. NATIONAL INSTITUTES OF HEALTH
Search CIT:
GO
CIT Home
Products & Services
Information Security
Support
Science
IT Policies
About CIT
SCIENCE
MIPAV
MEDICAL IMAGE PROCESSING AND VISUALIZATION
MEDIC

Back to: CIT Home » Science » ISL » MIPAV

Development

- MIPAV API
- MIPAV XML based Formats
- How-To
  - Develop MIPAV plugins
  - Use ITK with MIPAV
- Presentations
  - Plugin development class

MIPAV API Documentation

Application Programming Interface JavaDoc The API JavaDoc is publicly available for the benefit of developers who are coding against MIPAV public methods, such as those who are interested in generating home-grown plug-ins. This page contains the complete JavaDoc output of publicly available methods and classes.

Documentation for download:

- There is a tar.gz formatted version of the API available.
- There is a zip-formatted version of the API available.

MIPAV XML based Formats

In an attempt to circumvent the varying and often confusing medical-image formats, the MIPAV group has introduced its own medical image format. Furthermore, so often have we been frustrated in our efforts to decode the image information because it is binary formatted data, the image header information for the MIPAV XML format image is kept in Web Consortium's XML format. The MIPAV format is both open and extendable, allowing imaging software to add tags and perform key functions based on extended tag information.

Image Schema Definition	Image Schema
The above schema describes MIPAV's unique image file format.	

Surface Schema Definition	Surface Schema
MIPAV also has a custom XML format for saving surface meshes.	

Version 3.1.6 (2007-06-19) is the current downloadable version. What's New

- Home
- Download
- Documentation
- Development
- Contact Information

JHU Medic

Johns Hopkins University MedIC have built plug-ins for MIPAV, focusing on modifying images to use Talairach co-ordinates.

Figure 311. 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

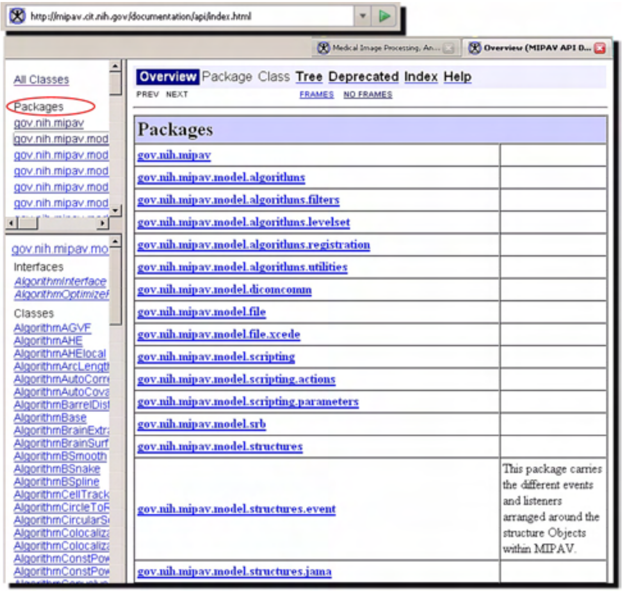
On the Development page, click the Application Programming Interface JavaDoc link <http://mipav.cit.nih.gov/documentation/api/index.html>. The API documentation page appears displaying the following three frames:

- **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 312.



Packages	
gov.nih.mipav	
gov.nih.mipav.model.algorithms	
gov.nih.mipav.model.algorithms.filters	
gov.nih.mipav.model.algorithms.levelset	
gov.nih.mipav.model.algorithms.registration	
gov.nih.mipav.model.algorithms.utilities	
gov.nih.mipav.model.dicomcomm	
gov.nih.mipav.model.file	
gov.nih.mipav.model.file.xcede	
gov.nih.mipav.model.scripting	
gov.nih.mipav.model.scripting.actions	
gov.nih.mipav.model.scripting.parameters	
gov.nih.mipav.model.srb	
gov.nih.mipav.model.structures	
gov.nih.mipav.model.structures.event	This package carries the different events and listeners arranged around the structure Objects within MIPAV.
gov.nih.mipav.model.structures.jama	

Figure 312. 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. Refer to Figure 312.

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 313.

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 313.

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 313.

- 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.

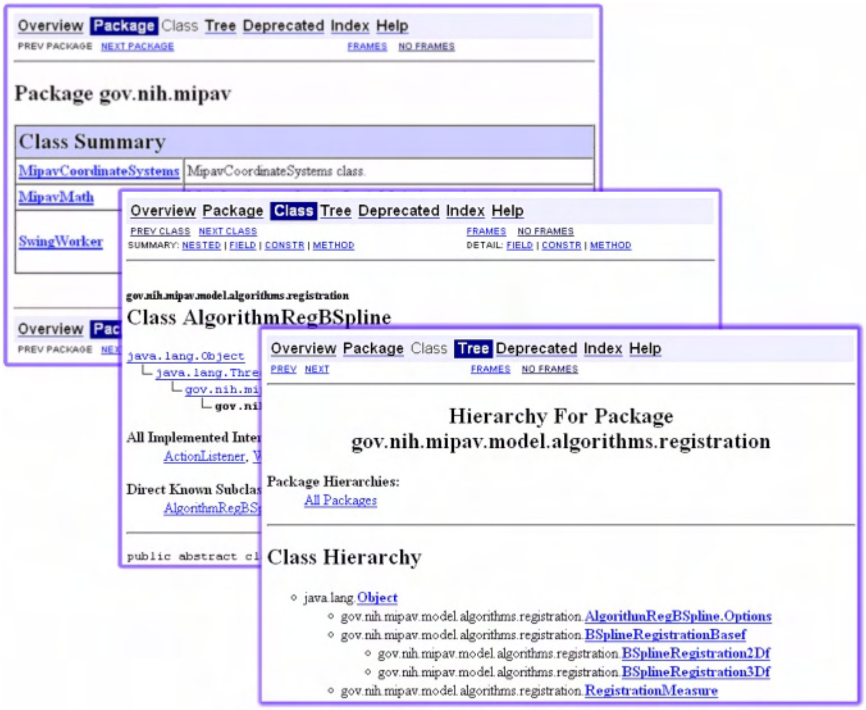
Warning: Deprecated APIs may be removed in future implementations.

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.


Overview Package Class Tree Deprecated Index Help
PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES
Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
Overview Package Class Tree Deprecated Index Help
PREV NEXT FRAMES NO FRAMES

Package gov.nih.mipav

Class Summary	
MipavCoordinateSystems	MipavCoordinateSystems class.
MipavMath	
SwingWorker	

gov.nih.mipav.model.algorithms.registration

Class AlgorithmRegBSpline

Hierarchy For Package gov.nih.mipav.model.algorithms.registration

Package Hierarchies:
All Packages

Class Hierarchy

- java.lang.Object
  - gov.nih.mipav.model.algorithms.registration.AlgorithmRegBSpline.Options
  - gov.nih.mipav.model.algorithms.registration.BSplineRegistrationBasef
    - gov.nih.mipav.model.algorithms.registration.BSplineRegistration2Df
    - gov.nih.mipav.model.algorithms.registration.BSplineRegistration3Df
  - gov.nih.mipav.model.algorithms.registration.RegistrationMeasure

Figure 313. 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 plugin programs

MIPAV provides the following classes for developing plugin programs:

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

Plugin programs are developed in the same way other Java programs are developed. The high-level steps of creating plugins follow.

- **Step 1, Determining the type of plugin program**—Before you begin to write the code for the plugin, determine the plugin type: algorithm, file, or view. Refer to page 522.
- **Step 2, Determining which version of Java to use**—Detailed instructions appear in "Step 2, Determining which version of Java to use" on page 523 and Figure 314.
- **Step 3, Writing the source code**—Some lines of code must appear in the source code so that the plugin program interfaces correctly with MIPAV. Refer to page 523.
- **Step 4, Building and compiling plugin programs**—You should keep back-up copies of the source and compiled files in case you need to update or change plugin programs. See page 528.
- **Step 5, Installing plugin programs**—This section explains how to install plugin programs. Refer to page 534.
- **Sample plugin programs**—This section provides a couple of examples of MIPAV plugins. Refer to page 535.

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

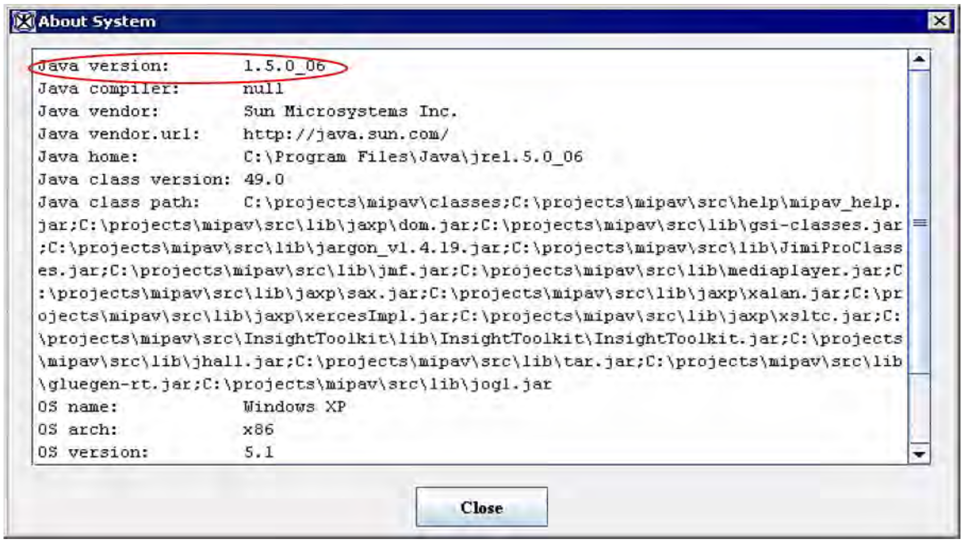
Step 1, Determining the type of plugin program

The first step of creating a plugin program is to determine the type you want to create, which depends on its purpose. As mentioned earlier, MIPAV plugin 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 plugin program.

Step 2, Determining which version of Java to use

To avoid compatibility problems when you create a plugin 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 314.



```
Java version:       1.5.0_06
Java compiler:      null
Java vendor:        Sun Microsystems Inc.
Java vendor.url:    http://java.sun.com/
Java home:          C:\Program Files\Java\jre1.5.0_06
Java class version: 49.0
Java class path:    C:\projects\mipav\classes;C:\projects\mipav\src\help\mipav_help.
jar;C:\projects\mipav\src\lib\jaxp\dom.jar;C:\projects\mipav\src\lib\gsi-classes.jar
;C:\projects\mipav\src\lib\jargon_v1.4.19.jar;C:\projects\mipav\src\lib\JimiProClass
es.jar;C:\projects\mipav\src\lib\jmf.jar;C:\projects\mipav\src\lib\mediaplayer.jar;C
:\projects\mipav\src\lib\jaxp\sax.jar;C:\projects\mipav\src\lib\jaxp\xalan.jar;C:\pr
ojects\mipav\src\lib\jaxp\xercesImpl.jar;C:\projects\mipav\src\lib\jaxp\xsltc.jar;C:
\projects\mipav\src\InsightToolkit\lib\InsightToolkit\InsightToolkit.jar;C:\projects
\mipav\src\lib\jhall.jar;C:\projects\mipav\src\lib\tar.jar;C:\projects\mipav\src\lib
\gluegen-rt.jar;C:\projects\mipav\src\lib\jogl.jar
OS name:            Windows XP
OS arch:            x86
OS version:         5.1
```

Figure 314. 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>

Step 3, 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 plugin for MIPAV, several lines must be present in the code so that it executes properly. Some mandatory code should be included in **all** plugin files. Other code might change depending on the plugin type.

INCLUDING MANDATORY CODE

The next three figures (Figure 315—Figure 317) show the mandatory source code needed for creating a file type of plugin, a view type of plugin, and an algorithm type of plugin. The plugins directory of MIPAV includes these three files (e.g. C:\[$MIPAV]\mipav\plugins):

- **PlugInFile.java**—Mandatory source code for a file type of plugin. See Figure 315;
- **PlugInView.java**—Mandatory source code for a view type of plugin. See Figure 316;
- **PlugInAlgorithm.java**—Mandatory source code for an algorithm type of plugin. See Figure 317.

```
package gov.nih.mipav.plugins;

import gov.nih.mipav.view.*;

import java.awt.*;

public interface PlugInFile extends PlugIn {

    /**
    *   run
    *   @param UI           MIPAV main user interface.
    */
    public void run(ViewUserInterface UI);
}
```

Figure 315. Mandatory code for a file type of plugin (PlugInFile.java). For readability purposes, keywords in all code reproduced in this chapter appear in bold, and comments appear in green type

```
package gov.nih.mipav.plugins;

import gov.nih.mipav.model.structures.*;
import gov.nih.mipav.view.*;

import java.awt.*;

public interface PlugInView extends PlugIn {

    /**
    *   run
    *   @param UI           MIPAV main user interface.
    *   @param parentFrame frame that displays the MIPAV image.
    *                       Can be used as a parent frame when building
    *                       dialogs.
    *   @param image        model of the MIPAV image.
    *   @see   ModelImage
    *   @see   ViewJFrameImage
    *
    */
    public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image);
}
```

Figure 316. Mandatory code for a view type of plugin (PlugInView.java). For readability purposes, keywords in all code reproduced in this chapter appear in bold, and comments appear in green type

```
package gov.nih.mipav.plugins;

import gov.nih.mipav.model.structures.*;
import gov.nih.mipav.view.*;

import java.awt.*;


public interface PlugInAlgorithm extends PlugIn {

    /**
    *   run
    *   @param UI           MIPAV main user interface.
    *   @param parentFrame frame that displays the MIPAV image.
    *                       Can be used as a parent frame when building
    *                       dialogs.
    *   @param image        model of the MIPAV image.
    *   @see   ModelImage
    *   @see   ViewJFrameImage
    *
    */
    public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image);


}

```

Figure 317. Mandatory code for an algorithm type of plugin (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 318).

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

Figure 318. 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 319).

```
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 319. Importing model structures, view functions, and [java.awt]

If you reference a class, you must include it in the plugin 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 320.

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

Figure 320. Example of placing referenced files in the \$MIPAV\plugins directory

LINES OF CODE THAT ARE DEPENDENT ON PLUGIN TYPE

Two lines of code depend on the type of plugin program being developed:

- Declaration
- Parameters for the run method

Declaration

The declaration used in a plugin depends on the type of plugin being developed. For instance, in line 9 in PlugInAlgorithm.java (Figure 317), the combination of words "**public interface** *PlugInAlgorithm*" indicates that the plugin in an Algorithm. For File or View types of plugins, simply replace *PlugInAlgorithm* with *PlugInFile* (line 7 in PlugInFile.java, see Figure 315) or *PlugInView* (line 8 in PlugInView.java, see Figure 316), respectively.

Table 5. Declarations dependent on type of plugin

Type of plugin	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 plugin type. Compare the run methods used in PlugInFile.java (Figure 315), PlugInView.java (Figure 316), and PlugInAlgorithm.java (Figure 317).

Table 6. Parameters for run methods dependent on type of plugin

Type of plugin	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);`

```
package gov.nih.mipav.plugins;

import gov.nih.mipav.model structures.*;
import gov.nih.mipav.view.*;

import java.awt *;

public interface PlugInAlgorithm extends PlugIn {

/**
* run
* @param UI           MIPAV main user interface.
* @param parentFrame Frame that displays the MIPAV image.
*                      Can be used as a parent frame when building dialogs.
* @param image        Model of the MIPAV image.
* @see ModelImage
* @see ViewJFrameImage
*/
public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image;)

}
```

Figure 321. PlugInAlgorithm.java. For readability purposes, keywords in all code reproduced in this chapter appear in bold, and comments appear in green type

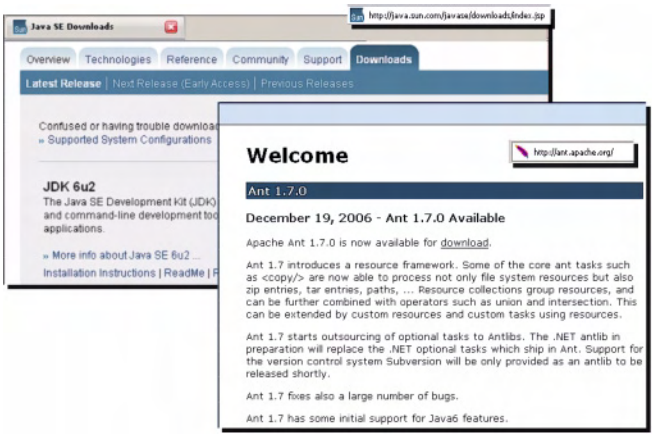
Step 4, Building and compiling plugin programs

To build a new plugin program for MIPAV, you must first install a build environment, alter the path environment variable, and compile the plugin 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/>.


http://java.sun.com/javase/downloads/index.jsp
Java SE Downloads
Overview
Technologies
Reference
Community
Support
Downloads
Latest Release | Next Release (Early Access) | Previous Releases
http://ant.apache.org/

Confused or having trouble downloa
» Supported System Configurations

JDK 6u2
The Java SE Development Kit (JDK)
and command-line development too
applications.

» More info about Java SE 6u2 ...
Installation Instructions | ReadMe | F

Welcome

Ant 1.7.0

December 19, 2006 - Ant 1.7.0 Available

Apache Ant 1.7.0 is now available for download.

Ant 1.7 introduces a resource framework. Some of the core ant tasks such as <copy/> are now able to process not only file system resources but also zip entries, tar entries, paths, ... Resource collections group resources, and can be further combined with operators such as union and intersection. This can be extended by custom resources and custom tasks using resources.

Ant 1.7 starts outsourcing of optional tasks to Antlibs. The .NET antlib in preparation will replace the .NET optional tasks which ship in Ant. Support for the version control system Subversion will be only provided as an antlib to be released shortly.

Ant 1.7 fixes also a large number of bugs.

Ant 1.7 has some initial support for Java6 features.

Figure 322. 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 323.

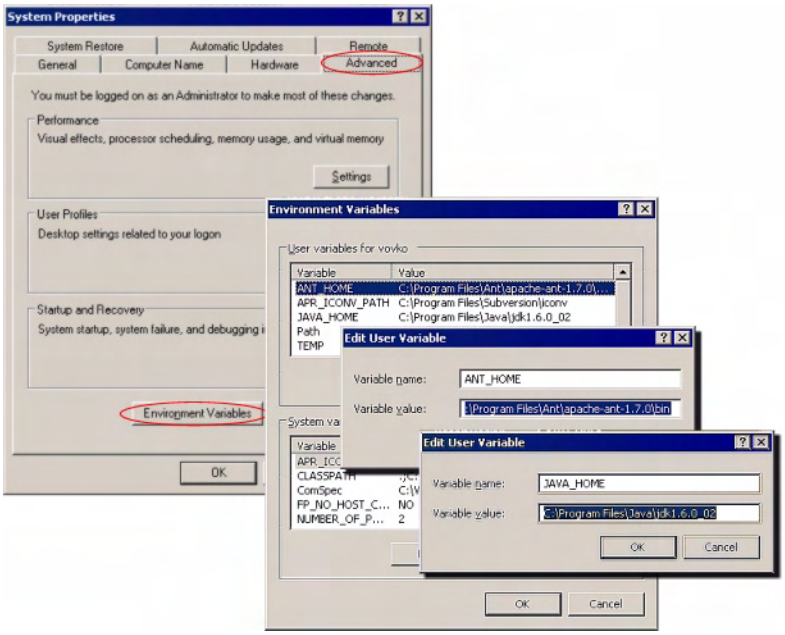
9 Open a new terminal for the change to take effect by doing the following:

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 plugin`.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.



You must be logged on as an Administrator to make most of these changes.

Visual effects, processor scheduling, memory usage, and virtual memory

Desktop settings related to your logon

System startup, system failure, and debugging i

User variables for vovko

Variable	Value
ANT_HOME	C:\Program Files\Ant\apache-ant-1.7.0\...
APR_ICONV_PATH	C:\Program Files\Subversion\iconv
JAVA_HOME	C:\Program Files\Java\jdk1.6.0_02
Path	
TEMP	

System va

Variable	
APR_ICC	
CLASSPATH	.;C:
ComSpec	C:\V
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2

Variable name: ANT_HOME
Variable value: :\Program Files\Ant\apache-ant-1.7.0\bin

Variable name: JAVA_HOME
Variable value: C:\Program Files\Java\jdk1.6.0_02

Figure 323. 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 plugin .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 324 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 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 324. The contents of the build.xml file

COMPILING THE PLUGIN FILES

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

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 plugin 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 plugin file by selecting Install Plugin in the MIPAV window.

Step 5, Installing plugin programs

Installing simple plugin 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 plugin installation tool—in the MIPAV window, select Plugins > Install Plugin.
- Use the operating system's tool for copying the files. This method requires the user to restart MIPAV so that the new plugin 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 plugins that consist of more complicated package class hierarchy, such as the Medic Talairach plugin program. To learn more about Medic Talairach plugin program, refer to MIPAV Technical Guide 1.

Examples of MIPAV plugins

To build plugin 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 plugin program(s) are included in MIPAV documentation:

- PlugInSample—a sample plugin, see "Sample plugin program" below.
- PlugInCT_MD—a typical plugin. (Refer to the MIPAV Users Guide, PDF version.)
- PlugInAlgorithm.Median—a very complicated plugin. Refer to MIPAV Volume 1 Users Guide, Appendix D.

SAMPLE PLUGIN PROGRAM

The source code for the plugin program, PlugInSample.java is an example of a simple algorithm type of plugin. This plugin opens an image in a new image frame using its own dialog box. It requires three files:

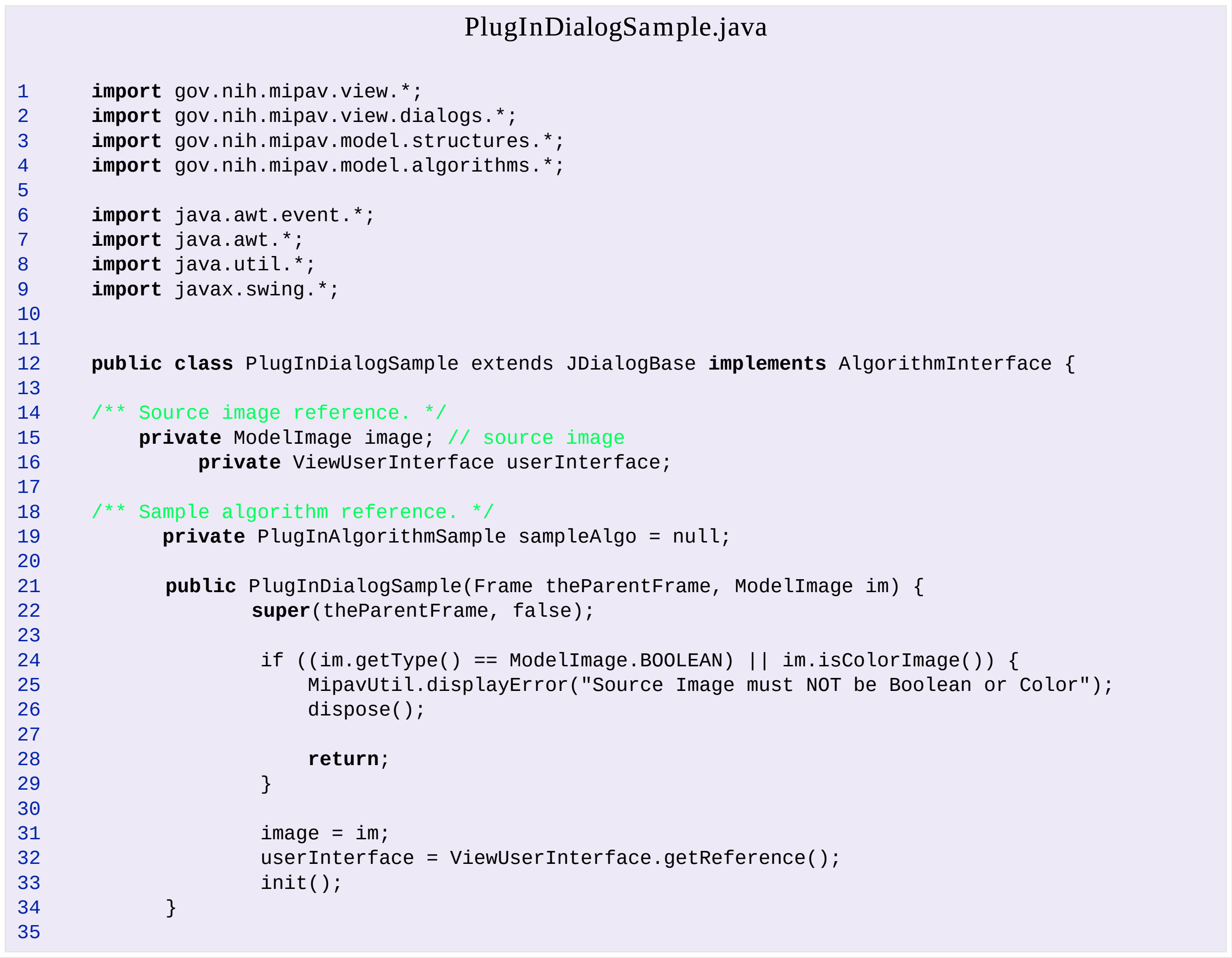
- **PlugInSample.java**—Provides an interface to MIPAV and the plugin program. See Figure 325 on page 536.
- **PlugInDialogSample.java**—Invokes the dialog to get user-supplied parameters. Refer to Figure 326 on page 537.

- **PlugInAlgorithmSample.java**—Implements the algorithm. See Figure 327.

PlugInSample.java

```
import gov.nih.mipav.plugins.*;  // needed to load PluginAlgorithm / PluginView /
                                 // PlugInFile interface
import gov.nih.mipav.view.*;
import gov.nih.mipav.model.structures.*;
import java.awt.*;

/***  This is a simple plugin to display a image in a new frame @see PlugInAlgorithm */

/** This is an Algorithm type of PlugIn and therefore must implement PlugInAlgorithm
** Implementing the PlugInAlgorithm requires this class to implement the run method
** with the correct parameters */

public class PlugInSample implements PlugInAlgorithm {
      /**
       * Defines body of run method, which was declared in the interface.
       * @param UI User Interface
       * @param parentFrame ParentFrame
       * @param image Current ModelImage--this is an image already loaded into
       * MIPAV. Can be null.
       */

      public void run (ViewUserInterface UI, Frame parentFrame, ModelImage image){
            if (parentFrame instanceof ViewJFrameImage) {
             new PlugInDialogSample(parentFrame,image);
            } else {
             MipavUtil.displayError("PlugInSample only runs on an image frame.");
          }
      }
}
```

Figure 325. PlugInSample.java



PlugInDialogSample.java

```java
import gov.nih.mipav.view.*;
import gov.nih.mipav.view.dialogs.*;
import gov.nih.mipav.model.structures.*;
import gov.nih.mipav.model.algorithms.*;

import java.awt.event.*;
import java.awt.*;
import java.util.*;
import javax.swing.*;


public class PlugInDialogSample extends JDialogBase implements AlgorithmInterface {

/** Source image reference. */
    private ModelImage image; // source image
         private ViewUserInterface userInterface;

/** Sample algorithm reference. */
      private PlugInAlgorithmSample sampleAlgo = null;

      public PlugInDialogSample(Frame theParentFrame, ModelImage im) {
             super(theParentFrame, false);

              if ((im.getType() == ModelImage.BOOLEAN) || im.isColorImage()) {
                  MipavUtil.displayError("Source Image must NOT be Boolean or Color");
                  dispose();

                  return;
              }

              image = im;
              userInterface = ViewUserInterface.getReference();
              init();
      }

```

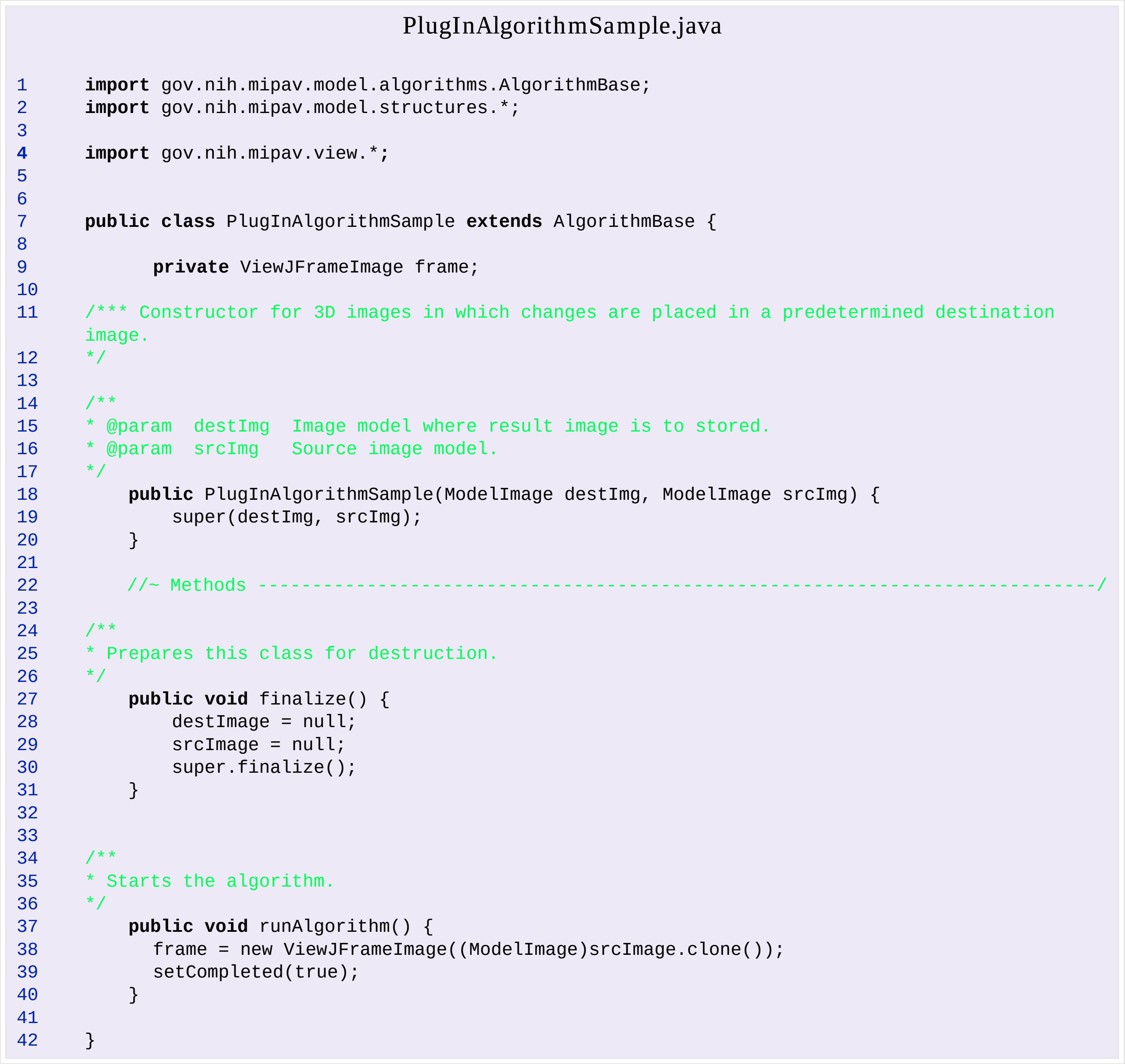
Figure 326. PlugInDialogSample.java

```
     // **********************************************************************
     // ************************** Event Processing ****************************
     // **********************************************************************

     /**
      * Closes dialog box when the OK button is pressed and calls the algorithm.
      * @param  event  Event that triggers function.
      */

    public void actionPerformed(ActionEvent event) {
         String command = event.getActionCommand();

         if (command.equals("OK")) {
         callAlgorithm();
         } else if (command.equals("Cancel")) {
             dispose();
         }
     }

     /**
      * Sets up the GUI (panels, buttons, etc) and displays it on the screen.
     */
     private void init() {

         // Build the Panel that holds the OK and CANCEL Buttons
         JPanel OKCancelPanel = new JPanel();

         JLabel questionLabel = new JLabel("Display Images?");

         // size and place the OK button
         buildOKButton();
         OKCancelPanel.add(OKButton, BorderLayout.WEST);

         // size and place the CANCEL button
         buildCancelButton();
         OKCancelPanel.add(cancelButton, BorderLayout.EAST);
         getContentPane().add(questionLabel, BorderLayout.NORTH);
         getContentPane().add(OKCancelPanel, BorderLayout.SOUTH);

         pack();
         setVisible(true);
         setResizable(false);
         System.gc();
     }
```

Figure 326. PlugInDialogSample.java (continued)

```
/*** 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. */

/** @param  algorithm  Algorithm that caused the event. */

    public void algorithmPerformed(AlgorithmBase algorithm) {
     if (algorithm instanceof PlugInAlgorithmCT_MD) {
       if ( sampleAlgo.isCompleted() ) {
             dispose();
       }
       }
    }


/*** 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. */

    protected void callAlgorithm() {
        sampleAlgo = new PlugInAlgorithmSample(null, image);
        sampleAlgo.addListener(this);
        setVisible(false); // Hide dialog

        if (isRunInSeparateThread()) {

//*** Start the thread as a low priority because we wish to still have user interface work fast.*/
            if (sampleAlgo.startMethod(Thread.MIN_PRIORITY) == false) {
                MipavUtil.displayError("A thread is already running on this object");
            }
        } else {
        sampleAlgo.run();
        }
    }

}
```

Figure 326. PlugInDialogSample.java (continued)



PlugInAlgorithmSample.java

```
import gov.nih.mipav.model.algorithms.AlgorithmBase;
import gov.nih.mipav.model.structures.*;

import gov.nih.mipav.view.*;


public class PlugInAlgorithmSample extends AlgorithmBase {

      private ViewJFrameImage frame;

/*** Constructor for 3D images in which changes are placed in a predetermined destination image.
*/

/**
* @param  destImg  Image model where result image is to stored.
* @param  srcImg   Source image model.
*/
    public PlugInAlgorithmSample(ModelImage destImg, ModelImage srcImg) {
        super(destImg, srcImg);
    }

    //~ Methods ------------------------------------------------------------------------------/

/**
* Prepares this class for destruction.
*/
    public void finalize() {
        destImage = null;
        srcImage = null;
        super.finalize();
    }


/**
* Starts the algorithm.
*/
    public void runAlgorithm() {
      frame = new ViewJFrameImage((ModelImage)srcImage.clone());
      setCompleted(true);
    }

}
```

Figure 327. PlugInAlgorithmSample.java

PlugInCT_MD, a typical plugin program

PlugInCT_MD is a typical example of a plugin program. It consists of three files:

- **PlugInCT_MD.java**—Provides an interface to MIPAV and the plugin 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 328 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 329 on page 543.

PlugInAlgorithmCT_MD.java

Figure 331 on page 553 shows the content of PlugInAlgorithmCT_MD.java.

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

PlugInCT_MD.java

```
import plugins.PlugInDialogCT_MT;     //associated class file
import gov.nih.mipav.plugins.*;       //needed to load PlugInAlgorithm / PlugInView /
                                      //PlugInFile interface
import gov.nih.mipav.view.*;
import gov.nih.mipav.model.structures.*;

import java.awt.*;

/**
*  This is a simple plugin for the University of Maryland to simple segment an
*  imagebased on CT Hounsfield units.
*
* @see PlugInAlgorithm
*/

//This is an Algorithm type of PlugIn, and therefore must implement PlugInAlgorithm
//Implementing the PlugInAlgorithm requires this class to implement the run method
//with the correct parameters
public class PlugInCT_MD implements PlugInAlgorithm {

   /**
     * Defines body of run method, which was declared in the interface.
     * @param UI            User Interface
     * @param parentFrame   ParentFrame
     * @param image         Current ModelImage--this is an image already loaded into
     *                      MIPAV. Can be null.
     */
     public void run (ViewUserInterface UI, Frame parentFrame, ModelImage image){

         if (parentFrame instanceof ViewJFrameImage)
           new PlugInDialogCT_MD (parentFrame,image);

         else
           MipavUtil.displayError (“PlugIn CT_MD only runs on an image frame.”);
         }
     }
}
```

Figure 328. PlugInCT_MD.java

PlugInDialogCT_MD.java

```java
import gov.nih.mipav.view.*;
import gov.nih.mipav.view.dialogs.*;
import gov.nih.mipav.model.structures.*;
import gov.nih.mipav.model.algorithms.*;

import java.awt.event.*;
import java.awt.*;
import java.util.*;

import javax.swing.*;


/**
*
*   JDialogBase class.
*
*   Note:
*
*   @version    July 12, 2002
*   @author
*   @see        JDialogBase
*   @see        JDialogMedian
*   @see        AlgorithmInterface
*
*   $Logfile: /mipav/src/plugins/PlugInDialogCT_MD.java $
*   $Revision: 6 $
*   $Date: 8/05/04 5:44p $
*
*/
public class PlugInDialogCT_MD extends JDialogBase implements AlgorithmInterface {

    private     PlugInAlgorithmCT_MD ctSegAlgo = null;
    private     ModelImage  image;                // source image
    private     ModelImage  resultImage = null;   // result image
     private ViewUserInterface userInterface;

    private     String       titles[];

    private     float        correctionVal;
    private     JTextField   fatLValTF;
    private     JTextField   fatHValTF;
    private     JTextField   ldmLValTF;
    private     JTextField   ldmHValTF;
    private     JTextField   hdmLValTF;
    private     JTextField   hdmHValTF;

    private     int         fatLVal;
    private     int         fatHVal;
    private     int         ldmLVal;
    private     int         ldmHVal;
    private     int         hdmLVal;
    private     int         hdmHVal;

```

Figure 329. PlugInDialogCT_MD.java

```
/**
 *  Creates new dialog for Median filtering using a plugin.
 *  @param parent          Parent frame.
 *  @param im              Source image.
 */

  public PlugInDialogCT_MD(Frame theParentFrame, ModelImage im) {
   super(theParentFrame, true);
   if (im.getType() == ModelImage.BOOLEAN || im.isColorImage()) {
     MipavUtil.displayError("Source Image must NOT be Boolean or Color");
     dispose();
     return;
   }
   image = im;
   userInterface = ((ViewJFrameBase)(parentFrame)).getUserInterface();
   init();
  }

 /**
 * Used primarily for the script to store variables and run the algorithm.  No
 * actual dialog will appear but the set up info and result image will be stored
 *  here.
 * @param UI   The user interface, needed to create the image frame.
 * @param imSource image.
 */
 public PlugInDialogCT_MD(ViewUserInterface UI, ModelImage im) {
   super();
   userInterface = UI;
   if (im.getType() == ModelImage.BOOLEAN || im.isColorImage()) {
     MipavUtil.displayError("Source Image must NOT be Boolean or Color");
     dispose();
     return;
   }

   image = im;
 }

 /**
 * Sets up the GUI (panels, buttons, etc) and displays it on the screen.
 */
  private void init(){

setForeground(Color.black);
   setTitle("CT_segmentation");

     JPanel inputPanel = new JPanel(new GridLayout(3, 3));
     inputPanel.setForeground(Color.black);
      inputPanel.setBorder(buildTitledBorder("Input parameters"));

     JLabel labelFat = new JLabel("Fat thresholds: ");
     labelFat.setForeground(Color.black);
     labelFat.setFont(serif12);
     inputPanel.add(labelFat);

```

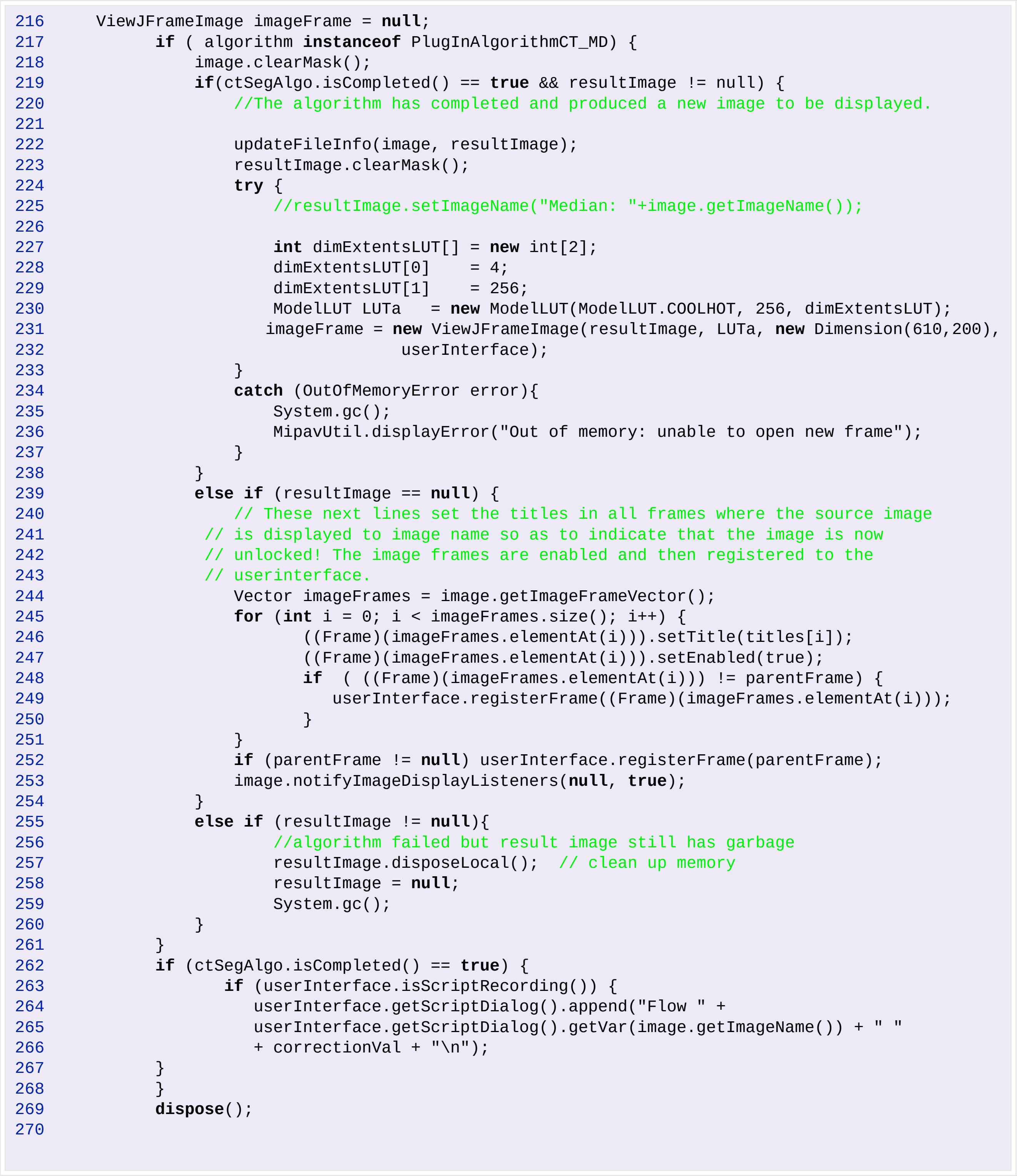
Figure 329. PlugInDialogCT_MD.java (continued)

```
                fatLValTF = new JTextField();
                fatLValTF.setText("-190");
                fatLValTF.setFont(serif12);
                inputPanel.add(fatLValTF);

                fatHValTF = new JTextField();
                fatHValTF.setText("-30");
                fatHValTF.setFont(serif12);
                inputPanel.add(fatHValTF);

                JLabel labelLDM = new JLabel("Low density muscle thresholds: ");
               labelLDM.setForeground(Color.black);
                labelLDM.setFont(serif12);
                inputPanel.add(labelLDM);

                ldmLValTF = new JTextField();
                ldmLValTF.setText("0");
                ldmLValTF.setFont(serif12);
                inputPanel.add(ldmLValTF);

                ldmHValTF = new JTextField();
                ldmHValTF.setText("30");
                ldmHValTF.setFont(serif12);
                inputPanel.add(ldmHValTF);

                JLabel labelHDM = new JLabel("High density muscle thresholds: ");
                  labelHDM.setForeground(Color.black);
                labelHDM.setFont(serif12);
                inputPanel.add(labelHDM);

                hdmLValTF = new JTextField();
                hdmLValTF.setText("31");
                hdmLValTF.setFont(serif12);
                inputPanel.add(hdmLValTF);

                hdmHValTF = new JTextField();
                hdmHValTF.setText("100");
                hdmHValTF.setFont(serif12);
                inputPanel.add(hdmHValTF);

           getContentPane().add(inputPanel, BorderLayout.CENTER);

        // Build the Panel that holds the OK and CANCEL Buttons
           JPanel OKCancelPanel = new JPanel();

           // size and place the OK button
           buildOKButton();
             OKCancelPanel.add(OKButton, BorderLayout.WEST);
           // size and place the CANCEL button
           buildCancelButton();
             OKCancelPanel.add(cancelButton, BorderLayout.EAST);
             getContentPane().add(OKCancelPanel, BorderLayout.SOUTH);
```

Figure 329. PlugInDialogCT_MD.java (continued)

```
         pack();
         setVisible(true);
         setResizable(false);
   System.gc();

   } // end init()

  /**
 *  Accessor that returns the image.
 *  @return          The result image.
 */
 public ModelImage   getResultImage(){return resultImage;}



   /**
   *      Accessor that sets the correction value
   *      @param num   Value to set iterations to (should be between 1 and 20).
   */
   public void  setCorrectionValue(float num){correctionVal = num;}

 //************************************************************************
 //************************** Event Processing ****************************
 //************************************************************************

   /**
   *  Closes dialog box when the OK button is pressed and calls the algorithm.
   *  @param event       Event that triggers function.
   */
   public void actionPerformed(ActionEvent event) {
         String command = event.getActionCommand();

         if (command.equals("OK")) {
               if (setVariables()) {
                     callAlgorithm();
               }
         }
         else if (command.equals("Script")) {
               callAlgorithm();
         }
         else if (command.equals("Cancel")) {
               dispose();
         }
}

 //************************************************************************
 //************************** Algorithm Events ****************************
 //************************************************************************

 /**
 * 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.
 *   @param algorithm   Algorithm that caused the event.
 */
 public void algorithmPerformed(AlgorithmBase algorithm) {
```

Figure 329. PlugInDialogCT_MD.java (continued)



```
ViewJFrameImage imageFrame = null;
      if ( algorithm instanceof PlugInAlgorithmCT_MD) {
          image.clearMask();
          if(ctSegAlgo.isCompleted() == true && resultImage != null) {
              //The algorithm has completed and produced a new image to be displayed.

              updateFileInfo(image, resultImage);
              resultImage.clearMask();
              try {
                  //resultImage.setImageName("Median: "+image.getImageName());

                  int dimExtentsLUT[] = new int[2];
                  dimExtentsLUT[0]    = 4;
                  dimExtentsLUT[1]    = 256;
                  ModelLUT LUTa   = new ModelLUT(ModelLUT.COOLHOT, 256, dimExtentsLUT);
                 imageFrame = new ViewJFrameImage(resultImage, LUTa, new Dimension(610,200),
                               userInterface);
              }
              catch (OutOfMemoryError error){
                  System.gc();
                  MipavUtil.displayError("Out of memory: unable to open new frame");
              }
          }
          else if (resultImage == null) {
              // These next lines set the titles in all frames where the source image
           // is displayed to image name so as to indicate that the image is now
           // unlocked! The image frames are enabled and then registered to the
           // userinterface.
              Vector imageFrames = image.getImageFrameVector();
              for (int i = 0; i < imageFrames.size(); i++) {
                     ((Frame)(imageFrames.elementAt(i))).setTitle(titles[i]);
                     ((Frame)(imageFrames.elementAt(i))).setEnabled(true);
                     if  ( ((Frame)(imageFrames.elementAt(i))) != parentFrame) {
                        userInterface.registerFrame((Frame)(imageFrames.elementAt(i)));
                     }
              }
              if (parentFrame != null) userInterface.registerFrame(parentFrame);
              image.notifyImageDisplayListeners(null, true);
          }
          else if (resultImage != null){
                  //algorithm failed but result image still has garbage
                  resultImage.disposeLocal();  // clean up memory
                  resultImage = null;
                  System.gc();
          }
      }
      if (ctSegAlgo.isCompleted() == true) {
             if (userInterface.isScriptRecording()) {
                userInterface.getScriptDialog().append("Flow " +
                userInterface.getScriptDialog().getVar(image.getImageName()) + " "
                + correctionVal + "\n");
      }
      }
      dispose();
```

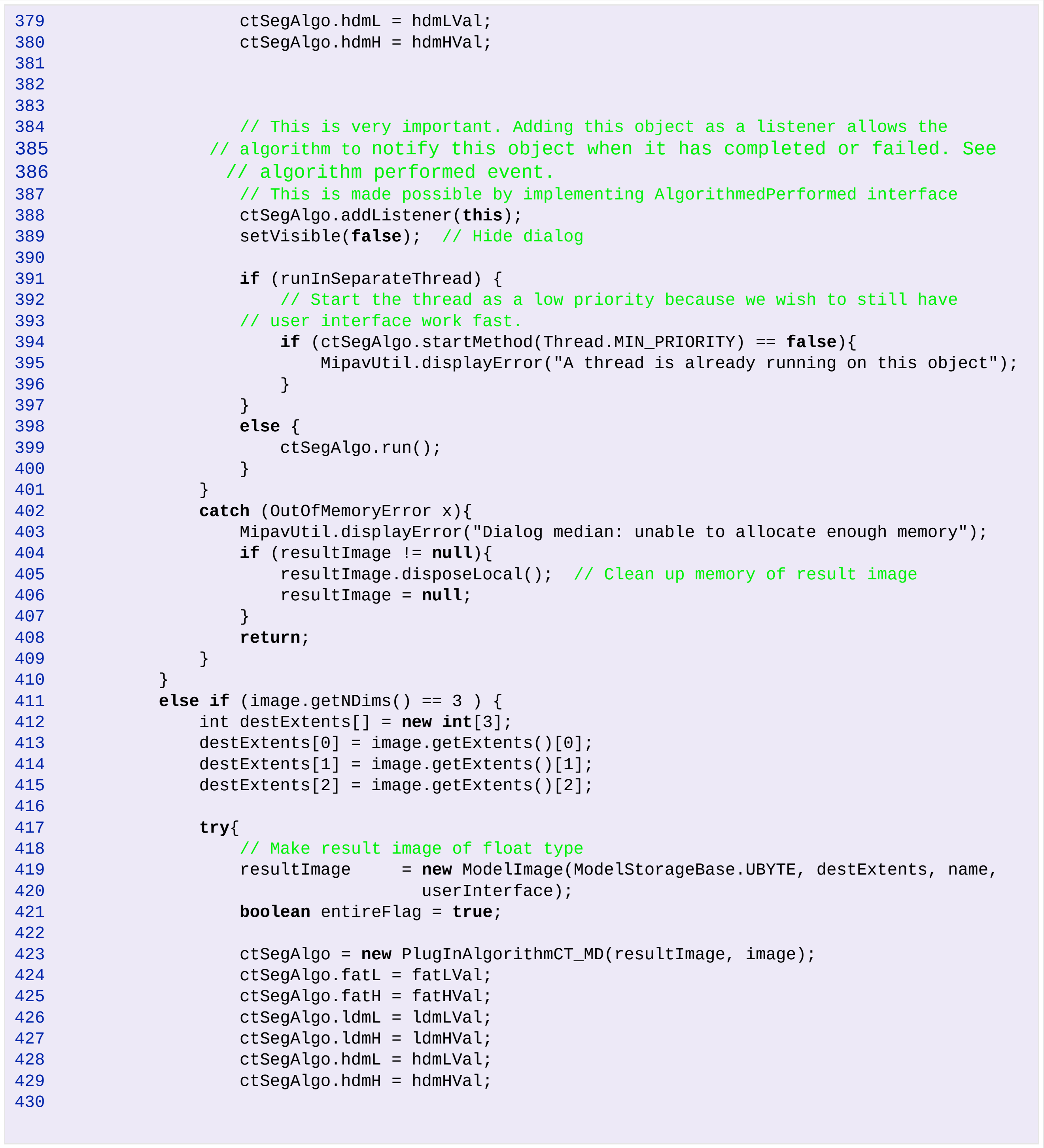
Figure 329. PlugInDialogCT_MD.java (continued)

```
    } // end AlgorithmPerformed()


       /**
       * Use the GUI results to set up the variables needed to run the algorithm.
       * @return      <code>true</code> if parameters set successfully, <code>false
      *  </code> otherwise.
       */
       private boolean setVariables() {
         String tmpStr;


          // verify iteration is within bounds
          tmpStr = fatLValTF.getText();
           if ( testParameter(tmpStr, -4000, 4000) ){
               fatLVal = Integer.valueOf(tmpStr).intValue();
           }
           else{
               fatLValTF.requestFocus();
               fatLValTF.selectAll();
               return false;
           }

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

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

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


```

Figure 329. 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 329. 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 329. PlugInDialogCT_MD.java (continued)

```
                 // 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 image memory
                    resultImage = null;
                }
                return;
            }
        }
      } // end callAlgorithm()

}
```

Figure 329. PlugInDialogCT_MD.java (continued)

PlugInAlgorithmCT_MD.java

```
import gov.nih.mipav.model.algorithms.*;
import gov.nih.mipav.model.structures.*;
import gov.nih.mipav.view.*;

import java.io.*;
import java.util.*;


/**
*
*   This shows how to extend the AlgorithmBase class.
*
*     Supports the segmentation
*     CT scans:
```

Figure 330. PlugInAlgorithmCT_MD.java

```
*     Fat:                   -190 to -30
*     Low density muscle:       0 to  30
*     High density muscle:     31 to 100
*     If you have any questions, please drop me a line.
* =====
* Matthew J. Delmonico, MS, MPH
* Graduate Research Assistant, Exercise Physiology
* 2132 HHP Building
* University of Maryland
* College Park, MD  20742
* (301) 405-2569
* (301) 793-0567 (cell)
*
*   @version    July 12, 2002
*   @author
*   @see        AlgorithmBase
*
*   $Logfile: /mipav/src/plugins/PlugInAlgorithmCT_MD.java $
*   $Revision: 10 $
*   $Date: 10/13/04 1:09p $
*
*/
public class PlugInAlgorithmCT_MD extends AlgorithmBase {


    private boolean     entireImage = true;

    public int         fatL    = -190;
    public int         fatH    = -30;

    public int         ldmL    = 0;
    public int         ldmH    = 30;

    public int         hdmL    = 31;
    public int         hdmH    = 100;


    /**
    * Constructor for 3D images in which changes are placed in a predetermined
   * destination image.
    *   @param destImg      Image model where result image is to stored.
    *   @param srcImg       Source image model.
    */
      public PlugInAlgorithmCT_MD(ModelImage destImg, ModelImage srcImg) {
       super(destImg, srcImg);
      }

/**
    * Prepares this class for destruction.
*/
      public void finalize(){
          destImage   = null;
          srcImage    = null;
          super.finalize();
      }

```

Figure 330. PlugInAlgorithmCT_MD.java (continued)

```
    /**
    *   Starts the algorithm.
    */
      public void run() {

        if (srcImage  == null) {
            displayError("Source Image is null");
            notifyListeners(this);
            return;
        }
        if (destImage  == null) {
            displayError("Source Image is null");
            notifyListeners(this);
            return;
        }


        // start the timer to compute the elapsed time
        setStartTime();

        if (destImage != null){     // if there exists a destination image
            if (srcImage.getNDims() == 2){
                 calcStoreInDest2D();
              }
              else if (srcImage.getNDims() > 2) {
                  calcStoreInDest3D();
              }
        }

        // compute the elapsed time
        computeElapsedTime();
        notifyListeners(this);
    }

    /**
    * This function produces a new image that has been median filtered and places
   *  filtered image in the destination image.
    */
    private void calcStoreInDest2D(){


        int length;               // total number of data-elements (pixels) in image
        float buffer[];           // data-buffer (for pixel data) which is the "heart"
                           // of the image
```

Figure 331. PlugInAlgorithmCT_MD.java

```java
        try {
            // image length is length in 2 dims
            length = srcImage.getExtents()[0]  * srcImage.getExtents()[1];
            buffer       = new float[length];
            srcImage.exportData(0,length, buffer); // locks and releases lock
        }
        catch (IOException error) {
            buffer = null;
            errorCleanUp("Algorithm CT_MD reports: source image locked", true);
            return;
        }
        catch (OutOfMemoryError e){
            buffer = null;
            errorCleanUp("Algorithm CT_MD reports: out of memory", true);
            return;
        }

        int mod = length/100; // mod is 1 percent of length
        initProgressBar();

        // Fat:  -190 to -30
        // Low density muscle:  0 to 30
        // High density muscle:  31 to 100
        BitSet mask = null;
        if (srcImage.getVOIs().size() > 0 ) {
            mask = srcImage.generateVOIMask();
            entireImage = false;
        }

        int fat      = 0;
        int ldMuscle = 0;
        int hdMuscle = 0;
        for (int i = 0; i < length && !threadStopped; i++){
            if (isProgressBarVisible() && (i)%mod==0)
                progressBar.setValue(Math.round((float)(i)/(length-1) * 100));

            if (entireImage == true || mask.get(i) ) {
                if( buffer[i] >= fatL && buffer[i] <= fatH  ) {
                    destImage.set(i, 20);
                    fat++;
                }
                else if( buffer[i] >= ldmL && buffer[i] <= ldmH  ) {
                    destImage.set(i, 40);
                    ldMuscle++;
                }
                else if( buffer[i] >= hdmL && buffer[i] <= hdmH  ) {
                    destImage.set(i, 60);
                    hdMuscle++;
                }
                else {
                    destImage.set(i, 0);
                    //buffer[i] = (float)srcImage.getMin();

            }
            }
        }


```

Figure 331. PlugInAlgorithmCT_MD.java

```
        //destImage.releaseLock();

        if (threadStopped) {
        finalize();
        return;
        }

        float area = srcImage.getFileInfo()[0].getResolutions()[0] *
                     srcImage.getFileInfo()[0].getResolutions()[1];

        destImage.getUserInterface().getMessageFrame().append("Number of Fat pixels = " +
           fat , ViewJFrameMessage.DATA );
        destImage.getUserInterface().getMessageFrame().append("  Area = " + (fat*area) +
           " mm^2\n", ViewJFrameMessage.DATA );

        destImage.getUserInterface().getMessageFrame().append("Number of LDM pixels = " +
           ldMuscle , ViewJFrameMessage.DATA );
        destImage.getUserInterface().getMessageFrame().append("  Area = " + (ldMuscle*area) +
           " mm^2\n", ViewJFrameMessage.DATA );

        destImage.getUserInterface().getMessageFrame().append("Number of HDM pixels = " +
           hdMuscle , ViewJFrameMessage.DATA );
        destImage.getUserInterface().getMessageFrame().append("  Area = " + (hdMuscle*area) +
           " mm^2\n", ViewJFrameMessage.DATA );

        destImage.calcMinMax();
        setCompleted(true);
    }

    /**
    * This function produces a new volume image that has been median filtered.
    * Image can be filtered by filtering each slice individually, or by filtering
   *  using a kernel-volume.
    */
    private void calcStoreInDest3D(){

        int totLength, imgLength;
        float buffer[];

        float vol = srcImage.getFileInfo()[0].getResolutions()[0] *
                    srcImage.getFileInfo()[0].getResolutions()[1] *
                    srcImage.getFileInfo()[0].getResolutions()[2];

        try {
            // image totLength is totLength in 3 dims
            imgLength = srcImage.getSliceSize();
            totLength = srcImage.getSliceSize() * srcImage.getExtents()[2];
            buffer = new float[totLength];
            srcImage.exportData(0,totLength, buffer); // locks and releases lock
            buildProgressBar(srcImage.getImageName(), "Processing image ...", 0, 100);
        }

catch (IOException error) {
            buffer = null;
            errorCleanUp("Algorithm CT_MD: source image locked", true);
            return;
        }
```

Figure 331. PlugInAlgorithmCT_MD.java

```
        catch (OutOfMemoryError e){
            buffer = null;
            errorCleanUp("Algorithm CT_MD: Out of memory creating process buffer", true);
            return;
        }

        int totFat      = 0;
        int totLdMuscle = 0;
        int totHdMuscle = 0;
        initProgressBar();

        for (int i = 0; i < srcImage.getExtents()[2] && !threadStopped; i++){
            int fat      = 0;
            int ldMuscle = 0;
            int hdMuscle = 0;

            if ( isProgressBarVisible() )
                    progressBar.setValue(Math.round((float)(i)/(srcImage.getExtents()[2]-1) *
                    100));

            for (int j = 0; j < imgLength && !threadStopped; j++){
                //System.out.println(" j = " + j);
                int index = i*imgLength+j;
                if( buffer[index] >= fatL && buffer[index] <= fatH  ) {
                    destImage.set(index, 60);
                    totFat++;
                    fat++;
                }
                else if( buffer[index] >= ldmL && buffer[index] <= ldmH  ) {
                    destImage.set(index, 120);
                    totLdMuscle++;
                    ldMuscle++;
                }
                else if( buffer[index] >= hdmL && buffer[index] <= hdmH  ) {
                    destImage.set(index, 200);
                    totHdMuscle++;
                    hdMuscle++;
                }
                else {
                    destImage.set(index, 0);
                   //buffer[i] = -1024;
                }
            }
           destImage.getUserInterface().getMessageFrame().append("\n\n ***************** Slice
                " + i + " totals ***************\n",
            ViewJFrameMessage.DATA);
            destImage.getUserInterface().getMessageFrame().append("Number of fat pixels = " +
                fat , ViewJFrameMessage.DATA );
            destImage.getUserInterface().getMessageFrame().append("  Volume = " + (fat*vol) +
                " mm^3\n", ViewJFrameMessage.DATA );

             destImage.getUserInterface().getMessageFrame().append("Number of LDM pixels = " +
                ldMuscle , ViewJFrameMessage.DATA );
            destImage.getUserInterface().getMessageFrame().append("  Volume = " +
                (ldMuscle*vol) + " mm^3\n", ViewJFrameMessage.DATA );

```

Figure 331. PlugInAlgorithmCT_MD.java

```
                   destImage.getUserInterface().getMessageFrame().append("Number of HDM pixels
                         = " + hdMuscle , ViewJFrameMessage.DATA );
            destImage.getUserInterface().getMessageFrame().append("  Volume = " +
                    (hdMuscle*vol) + " mm^3\n", ViewJFrameMessage.DATA );
        }

        destImage.releaseLock();

        if (threadStopped) {
        finalize();
        return;
        }

        destImage.getUserInterface().getMessageFrame().append("\n ************************
            Totals ********************\n",
        ViewJFrameMessage.DATA);
        destImage.getUserInterface().getMessageFrame().append("Number of totFat pixels = " +
            totFat , ViewJFrameMessage.DATA );
        destImage.getUserInterface().getMessageFrame().append("  Volume = " + (totFat*vol) +
            " mm^3\n", ViewJFrameMessage.DATA );

        destImage.getUserInterface().getMessageFrame().append("Number of LDM pixels = " +
             totLdMuscle , ViewJFrameMessage.DATA );
       destImage.getUserInterface().getMessageFrame().append("  Volume = " + (totLdMuscle*vol)
             + " mm^3\n", ViewJFrameMessage.DATA );

        destImage.getUserInterface().getMessageFrame().append("Number of HDM pixels = " +
             totHdMuscle , ViewJFrameMessage.DATA );
       destImage.getUserInterface().getMessageFrame().append("  Volume = " + (totHdMuscle*vol)
             + " mm^3\n", ViewJFrameMessage.DATA );

        destImage.calcMinMax();
        progressBar.dispose();
        setCompleted(true);
    }
}
```

Figure 331. PlugInAlgorithmCT_MD.java