Technical Guide 2

Using InsightToolkit (ITK) with MIPAV

August 24, 2005

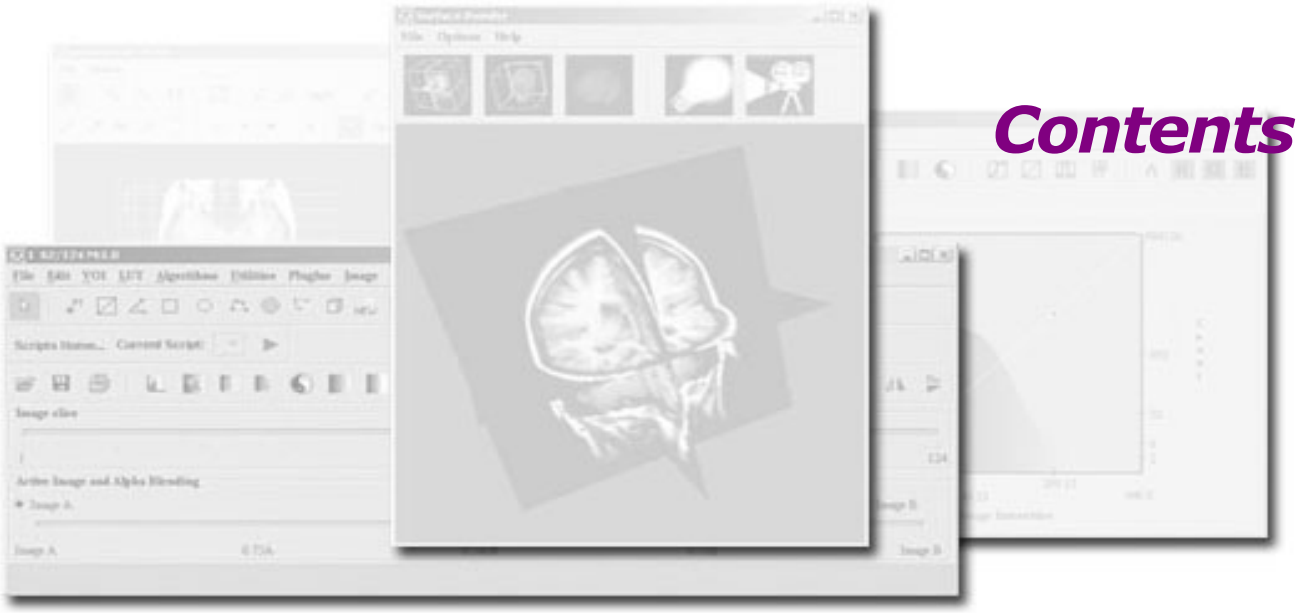
National Institutes of Health
Center for Information Technology
Rockville, Maryland

August 24, 2005

Matthew McAullife, PhD mcmatt@exchange.nih.gov
301-594-2432
Building 12A, Room 2041
National Institutes of Health
Bethesda, Maryland 20892

If you find a bug, please send e-mail to bug@mipav.cit.nih.gov. Frozen menus and JAVA exceptions dialogs are common signs. Please include as much information about what happened as you can. Please understand that we might need to get more information from you about what happened so we understand the problem.

If you have a feature idea, send an email to wishlist@mipav.cit.nih.gov.



Contents

List of Figures

List of Tables

Preface

The purpose of the MIPAV software program is to allow medical researchers to extract quantitative information from image datasets of various medical image modalities.

Scope of this guide

The primary purpose of *Technical Guide 2: Integrating InsightToolkit (ITK) with MIPAV* is to give MIPAV users who use the InsightToolkit the information they need to extend, if desired, MIPAV's capabilities through the development of new functions in plug-in applications via use of the MIPAV's application program interface (API).

Who should use this guide

Technical Guide 2: Integrating InsightToolkit (ITK) with MIPAV is designed for MIPAV users who can program in Java and who want to extend MIPAV's capabilities using the InsightToolkit.

Skills you need to perform tasks in MIPAV

To create plug-in applications for MIPAV to add new functionality, you must have software programming skills and be familiar with Java. In addition, you should be familiar with using the InsightToolkit.

How this guide is organized

This technical guide includes the following:

- Chapter 1, “Introducing InsightToolkit Platforms and Language Bindings” on page 1 presents an introduction to Kitware’s InsightToolkit (ITK).
- Chapter 2, “Creating Windows Runtime Libraries for Java Bindings” on page 6explains how to obtain InsightToolkit source code, generate MSVC++ projects, and build ITK software and Java binding.
- Chapter 3, “Using the MIPAV Interface to Java and to the InsightToolkit” on page 10, explains how to install runtime libraries and the JAR file and use the interface support class. It also provides an example of integrating InsightToolkit with MIPAV.

Where to find more information

You may want to read the following MIPAV publications, which appear as Adobe PDFs on the MIPAV web site:

http://mipav.cit.nih.gov/documentation/

Table 1. MIPAV documentation for end users and programmers

Audience	Documentation
End users	*MIPAV User's Guide, Volume 1, Basics*
	MIPAV User's Guide, Volume 2, Algorithms
	Technical Guide 1: *Using MIPAV to Label and Measure Brain Components in Talairach Space*
	Technical Guide 3: *Using Fluorescent Resonance Energy Transfer (FRET) Algorithms in MIPAV*
	Frequently Asked Questions
Programmers	*Application Program Interface JavaDoc*—A programmer's guide detailing public classes and methods available to plug-ins and programs built against MIPAV.
	XML Format guide—A helpful guide to the MIPAV XML file formats described by XML. The raw image definition and surface definition are available in addition to the guide.

Conventions used in this guide

This guide uses the following conventions:

This convention . . .	Stands for . . .
Italics	Names of books, guides, or manuals as references
	New terms or emphasis
	Names of executable files
Bold	User input
	Names of programming commands
Bold italic	Classes
All caps	File types, such as TIFF, GIF, or JPG
Upper- and lowercase	Names of keys
name@address.com	E-mail address format

This convention . . .	Stands for . . .
Hyperlink	An internet link (position the cursor on this word and click the left mouse button)*
Monospace	Code sample, including keywords and variables within text and as separate paragraphs, and user-defined program elements within text

*All figure and table citations, such as Figure 1 or Table 1, are hyperlinks although they are not underscored. Clicking the citation displays the appropriate figure or table.

Both volumes of the *MIPAV User's Guide* include special information that briefly highlights particular features or functions or that provide clarification. Based on the type of information they convey, these notes are labeled "note," "tip," "example," "recommendation," "remember," "reference," "caution," and "disclaimer." The following examples indicate how these notes appear and the type of information they include.

Note: Notes provide additional information that is related to the subject at hand. They tend to be "by the way" types of information or asides.

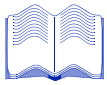
Tip: Tip paragraphs point out application shortcuts or briefly describe special features of the software.

Example: An example paragraph provides an example of a task or incident in which something of note could occur.

Recommendation: Paragraphs that are labeled "Recommendation" suggest methods of good practice or advice.

Definition: The definitions of specific words or phrases of note appear in "definition" paragraphs.

Remember: Notes labeled "Remember" present information that was previously discussed and that is pertinent in the current topic.



Reference: A reference note highlights one or more references that contain information on the current topic.

Caution: A paragraph labeled "Caution," alerts you to be very careful about avoiding some action that could harm your equipment or data.

Disclaimer: A disclaimer indicates the possible limitations or ramifications of a topic.

Introducing InsightToolkit Platforms and Language Bindings

The InsightToolkit (ITK) is an open source, cross-platform, object-oriented system for the processing, segmentation, and registration of images. ITK was developed using Federal funding and is presently maintained and managed by Kitware, Inc., (http://www.kitware.com) under the primary direction of the National Institutes of Health (NIH). Source code, documentation, support, mailing lists, etc., are all available through the website at http ://www.itk.org.

Implemented in the C++ programming language, the CMake build environment (http://www.cmake.org) manages the compilation process on Windows platforms and on a variety of Unix platforms, including Linux and Macintosh systems.

Although the core of ITK is implemented in C++, bindings for the Java, Tcl, and Python programming languages can be automatically generated so that the ITK software can be accessed from these languages. SWIG (http:// www.swig.org) is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages, but SWIG cannot parse such C++ structures as deeply nested template instantiations. Since the ITK depends on C++ templates as part of its generic programming implementation, the CableSwig set of tools (http:h// www.itk.org/HTML/CableSwig.html) was created to automatically generate the interfaces (i.e., "wrappers") to these other programming languages from the ITK C++ source.

The ITK wrapper-generation process uses the following CableSwig tools:

- **gccxml tool**—A modified version of the GNU gcc compiler that produces an XML description of an input C++ program
- **CABLE tool**—A tool (http://public.kitware.com/Cable/HTML/Index.html) that processes the generated XML information along with a configuration file to select the parts of the C++ interface that should be wrapped
- **CSWIG tool**—A modified version of SWIG that has SWIG's usual parser replaced with an XML parser (XML output from the gccxml and CABLE tools) and produces the appropriate language binding

The templated classes in ITK must be instantiated prior to wrapping, i.e., the template parameters must be specified as part of the wrapping process. While the C++ interface to the ITK can allow many data types, even user-defined ones, for any of the template parameters, the wrappers can only include instances of templates for data types known in advance. In order to avoid the exponential code explosion effect by allowing all possible template parameter types, especially when used in combinations (e.g., a filter has separate template types for the input and output images), the wrappers only include a limited number of the common types. Although more types can be added, the wrapping process would need to be redone and the generated libraries would grow in size as expected.

Java language binding for Windows platform

The Java programming language binding is needed since the MIPAV software is developed in Java. The binding is specific to a platform since the ITK C++ software must be built for execution on a particular platform. The executable ITK software is stored in dynamically loaded runtime libraries, which can be loaded by the Java ITK wrappers.

The MIPAV software runs on any Java-enabled Windows, Unix, and Mac OS10 platforms. Since ITK software can be built for any of these environments, the Microsoft Windows platform was chosen as a matter of convenience for demonstration. The generation of Java bindings for the other platforms supported by MIPAV is similar.

Required software tools

The process of building the ITK software with the Java bindings for the MS Windows platform requires that the following tools be installed:

- CMake
- C++ compiler
- CableSwig
- Java SDK

Note: Several versions may exist for each of these tools, but for demonstration purposes only one particular version was used.

CMake

ITK requires at least CMake version 1.6. You can download CMake at no cost from (http://www.cmake.org). It can be downloaded in binary form for any of the popular platforms including Windows, Solaris, IRIX, HP, Macintosh, and Linux. Alternatively, CMake source code can be downloaded and the tool can built on the target system. Either way, the CMake web page contains instructions for downloading and installing the software.

Note: For demonstration purposes, CMake version 2.0 (patch 5) was used.

C++ Compiler

There are different compilers you can use to build software from C++ source code for Windows platforms. The compiler must have full support for C++ templates.

Note: For demonstration purposes, Microsoft Visual C++ .Net 2003 (also known as version 7.1) was used.

CableSwig

The generation of Java wrappers for the ITK software was added in the version 2.0.0 of the ITK software. A version of CableSwig for ITK was made available at the same time. The CableSwig software is not available in binary form, so it must be downloaded from http://www.itk.org/HTML/Download.php and built using the CMake tool.

Once you've downloaded the CableSwig source code, use the following sequence of operations to build the set of tools:

1. Unpackage the source code into the following folder:
   `c:\itk\CableSwig-2.0.0`

   Once unpackaged, the root folder for the software should be:

   `c:\itk\CableSwig-2.0.0\CableSwig-2.0.0`

2. Launch the CMake tool and perform the following operations:
   a. Set the source folder to:

      `c:\itk\CableSwig-2.0.0\CableSwig-2.0.0`

   b. Set the binaries folder to:

      `c:\itk\CableSwig-2.0.0\CableSwig-2.0.0-bin`

   c. Select Visual Studio 7.NET 2003 to build.

   d. Enable Show Advanced Values.

   e. Set BUILD_EXAMPLES to OFF.

   f. Set BUILD_SHARED_LIBRARIES to ON.

   g. Set BUILD_TESTING to OFF.

   h. Set CMAKE_USE_RELATIVE_PATHS to ON.

   i. Click Configure.

Note: Since the features of the compiler are being tested, this step takes several minutes.

   On completion, you may need to click Configure a second time since there may be some options that are presented for the user confirm.

   j. Click OK. Doing so generates a Microsoft Visual Studio.Net solution

file and all of the projects for compiling the CableSwig source code. This step automatically closes the CMake tool.

3 Launch Microsoft Visual C++ .Net 2003 and perform the following operations:

a Open the solution file:

```
c:\itk\CableSwig-2.0.0\CableSwig-2.0.0-bin\CableSwig.sln
```

b Select the Release build configuration.

c Run the Build Solution operation.

Note: This step takes several minutes.

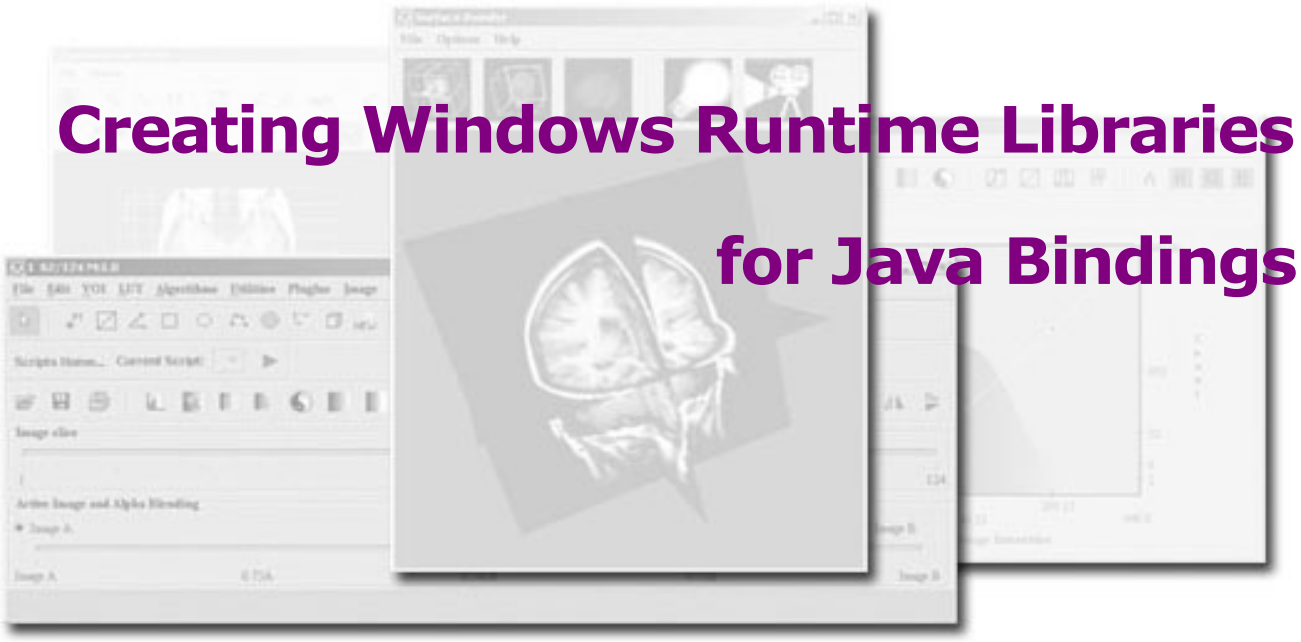
d Verify that there were no build errors and then close this tool.

Note: For demonstration purposes, CableSwig version ITK-2.0.0 was used.

Java SDK

The Java SDK is required because it includes the Java compiler and the Java Native Interface (JNI) for interfacing to standard C and C++ code. The SDK version must be compatible with the Java version used by MIPAV.

Note: For demonstration purposes, Java SDK version 1.4.2_05 was used. This version can be downloaded from http://java.sun.com/j2se/1.4.2/download.html.



Creating Windows Runtime Libraries for Java Bindings

This section explains how to create Microsoft Windows runtime libraries for Java bindings.

Getting InsightToolkit source code

The ITK source code normally is downloaded from http://www.itk.org/HTML/Download.htm. As of the time of this writing, however, the latest version was 2.0.1 (patch 1, March 2005), which had problems in the generation of the Java bindings. Instead, a more recent version (although not officially released) was downloaded from the ITK CVS repository that included the updates for correctly generating the Java bindings.

Access to the ITK CVS repository is made through the following commands:

```
cvs -d :pserver:anonymous@www.itk.org:/cvsroot/Insight
login
```

The password for this login is "`insight`".

The following command is used to initiate the download of the software into a subfolder named *Insight*.

cvs -d :pserver:anonymous@www.itk.org:/cvsroot/Insight co Insight

A Windows version of a CVS client that includes a graphics user interface is available from http://www.wincvs.org/.

For demonstration purposes, a snapshot of the ITK source code was retrieved from the ITK CVS repository on 17 April 2005. The root folder into which the Insight folder from the CVS repository retrieved was `c:\itk\InsightToolkit-cvs 2005-04-17`.

Generating MSVC++ Projects

The CMake tool is used to generate the Microsoft Visual Studio .Net solution file along with all of the projects for compiling the ITK source code and generating the Java bindings. The following sequence of operations are performed once the CMake tool is launched.

1. Set the source folder to

   `c:\itk\InsightToolkit-cvs 2005-04-17\Insight`

2. Set the binaries folder to

   `c:\itk\InsightToolkit-cvs 2005-04-17\Insight-bin`

3. Select to build for Visual Studio 7 .NET 2003.
4. Enable the Show Advanced Values option.
5. Set BUILD_EXAMPLES to OFF.
6. Set BUILD_SHARED_LIBRARIES to ON.
7. Set BUILD_TESTING to OFF.
8. Set CMAKE_USE_RELATIVE_PATHS to ON.
9. Set ITK_CSWIG_JAVA to ON.
10. Click Configure.

Note: Since the features of the compiler are being tested, this step takes several minutes.

11 On completion, it may be necessary to click Configure a second time as there may be some options that are presented for the user confirm. These options may have a JAVA_ prefix with the values referencing the correct version of the Java SDK to use when building the Java wrappers.

12 Click OK. This steps does the following:

- Generates a Microsoft Visual Studio .Net solution file and all of the projects for compiling the ITK source code and generating the Java bindings
- Automatically closes the CMake tool

Building ITK software and Java binding

Launch Microsoft Visual C++ .Net 2003 and perform the following operations:

1 Open the following solution file:

```
c:\itk\InsightToolkit-cvs 2005-04-17\Insight-
bin\ITK.sln
```

2 Select the Release build configuration.

3 Run the Build Solution operation.

Note: This step takes several minutes, if not an hour. Verify that there were no errors.

4 Select the INSTALL project and run the Build Only INSTALL operation. The default build configuration excludes the INSTALL project.

Note: This step takes a few minutes. Verify that there were no errors.

Distributing files

On completion of the build INSTALL project operation, the following files are generated:

```
c:\usr\local\bin\ITKAlgorithmsJava.dll
c:\usr\local\bin\ITKBasicFiltersAJava.dll
c:\usr\local\bin\ITKBasicFiltersBJava.dll
c:\usr\local\bin\ITKCommonJava.dll
c:\usr\local\bin\ITKCommonAJava.dll
c:\usr\local\bin\ITKCommonBJava.dll
c:\usr\local\bin\ITKIOJava.dll
c:\usr\local\bin\ITKNumericsJava.dll
c:\usr\local\bin\SwigRuntimeJava.dll
c:\usr\local\bin\VXLNumericsJava.dll
c:\usr\local\lib\InsightToolkit\InsightToolkit.jar
```

This path of `c:\usr\local` is hard-coded into the build for the INSTALL project.

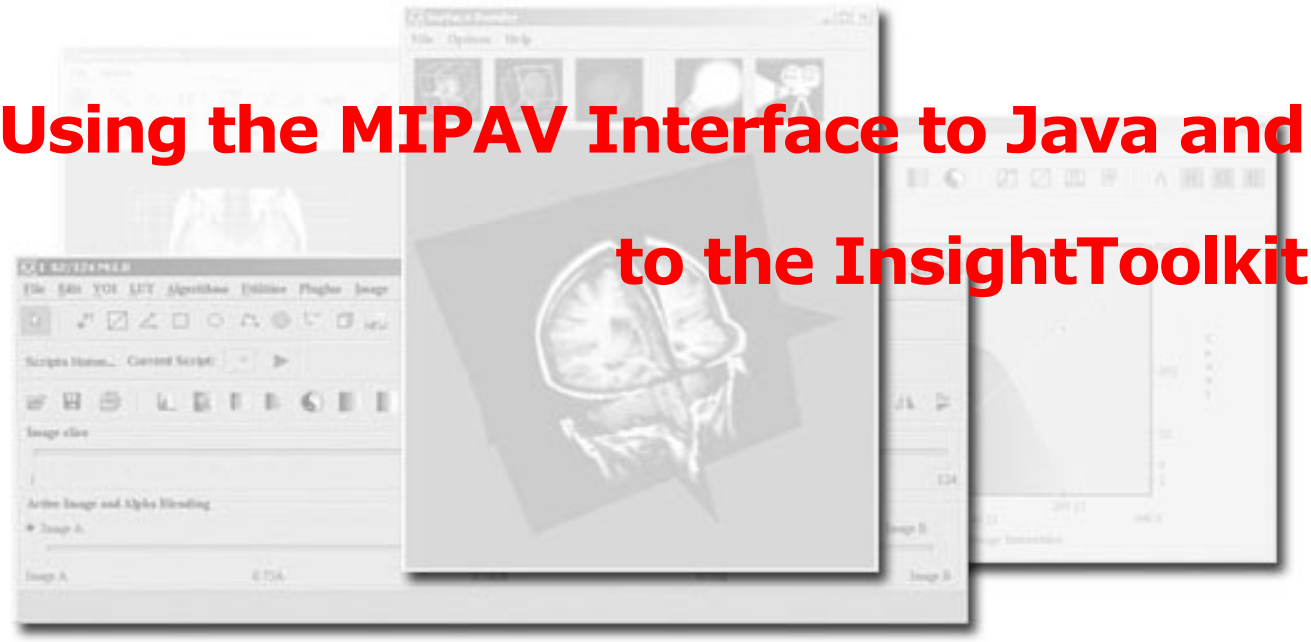
You can relocate these files in a different folder and on a different machine, but you must keep the relative path locations between the `ITK*Java.dll` files and `InsightToolkit.jar` file the same. The reason for keeping the path relative is because the software in the InsightToolkit.jar file searches for the runtime DLLs using the relative path of "`..\..\bin`".

Because the InsightToolkit was built using Microsoft Visual C++ .Net 2003 compiler (also known as version 7.1), the following C++ and C runtime libraries are required for distribution:

- `msvcp71.dll`
- `msvcr71.dll`

These files are normally located in the Windows System folder if the Microsoft Visual C++ .Net 2003 compiler is installed on the system. These files may also be placed there by another installed application, or they may not be present at all.

Using the MIPAV Interface to Java and to the InsightToolkit



This section explains where to place InsightToolkit runtime libraries and JAR file, the interface support class that was added to MIPAV, and provides an example of integrating the InsightToolkit with MIPAV.

Installing runtime libraries and JAR

The MIPAV source in the Visual SourceSafe repository is located at the following folder:

```
$/mipav/src
```

The files listed in “Distributing files” on page 8 “Distributing files” for a Windows distribution must be located as follows within the repository:

```
$/mipav/src/msvcp71.dll
$/mipav/src/msvcr71.dll
$/mipav/src/InsightToolkit/bin/ITKAlgorithmsJava.dll
$/mipav/src/InsightToolkit/bin/ITKBasicFiltersAJava.dll
$/mipav/src/InsightToolkit/bin/ITKBasicFiltersBJava.dll
$/mipav/src/InsightToolkit/bin/ITKCommonJava.dll
$/mipav/src/InsightToolkit/bin/ITKCommonAJava.dll
$/mipav/src/InsightToolkit/bin/ITKCommonBJava.dll
```

```
$/mipav/src/InsightToolkit/bin/ITKIOJava.dll

$/mipav/src/InsightToolkit/bin/ITKNumericsJava.dll

$/mipav/src/InsightToolkit/bin/SwigRuntimeJava.dll

$/mipav/src/InsightToolkit/bin/VXLNumericsJava.dll

$/mipav/src/InsightToolkit/lib/
InsightToolkit\InsightToolkit.jar
```

The `msvc*71.dll` files must exist in the same folder where the `MipavMain.java` source file is located. The remainder of the DLL files and the JAR file must remain in the same relative paths to each other.

It is not necessary to distribute the InsightToolkit Windows runtime DLLs along with the rest of a MIPAV installation. However, if one of the DLLs is distributed ITK Java DLLs, then all must all be distributed. Certainly, if the target installation is a Unix or Macintosh system, then none of these DLLs should be included in the installation.

The MIPAV application automatically checks the presence of the InsightToolkit Windows runtime DLLs. If they are not present, then all user menu options for accessing InsightToolkit operations are disabled.

Using the interface support class

A new class called ***InsightToolkitSupport*** was added to MIPAV to facilitate interfacing MIPAV to the InsightToolkit with Java bindings. This class has no constructor since all of the methods are declared to be static.

Images in MIPAV are stored in the ***ModelImage*** class. These images can have any number of dimensions, but, for most processing performed in MIPAV, these images are usually 2D or 3D. These images can also have a variety of storage formats (i.e., short, unsigned short, float, etc.), and blocks of pixel values can be retrieved and converted into buffers of a different storage format.

Images in the C++ version of the InsightToolkit are stored in the ***itk::Image<Type,Dim>*** class, which uses templates. Because the Java bindings required template instantiation, the Java version of the InsightToolkit has images being stored in fixed classes based on the Type and the Dim. For instance, the ***itkImageF2*** class is used for a 2D image of

`float` values and the ***itkImageUC3*** is used for a 3D image of `unsigned char` values, and the ***`itkImageVF3`*** class is used for a 3D image of a vector of `float` values.

For an initial implementation, 2D and 3D images of float values were selected as the common types for converting between MIPAV images and InsightToolkit images. One issue that factored into this selection is that an InsightToolkit filter such as the recursive Gaussian image filter was only implemented for 2D and 3D images of `float` and `unsigned short` values.

The interface to the InsightToolkit Gaussian blur filter was selected as the operation to use for demonstration of the MIPAV to InsightToolkit interface. It should also be noted that the InsightToolkit implementation of the Gaussian blur uses separable filters, and so it was not necessary to store a multichannel image in the InsightToolkit (for this demonstration) since each channel could be processed separately.

Table 9 lists a brief description of each static method that is provided by the ***InsightToolkitSupport*** class.

Table 9. Description of static methods provided by *InsightToolkitSupport* class

Static Method	Description
isLibraryPresent	This method returns `true` if the InsightToolkit runtime library is present for this platform. If not, do *not* use the InsightToolkit java classes; any access to them causes the `java.lang.UnsatisfiedLinkError` exception.
itkCreateImageSingle3D	This method creates a single-channel 3D InsightToolkit image of `float` values with the properties and values from the specified single channel 3D `ModelImage` instance.
itkCreateImageColor3D	This method creates a single-channel 3D InsightToolkit image of `float` values with the properties from the specified multichannel 3D `ModelImage` instance and the values from the specified channel.
itkCreateImageSingle2D	This method creates a single-channel 2D InsightToolkit image of `float` values with the properties and values from the specified single channel 2D *ModelImage* instance.
itkCreateImageColor2D	This method creates a single-channel 2D InsightToolkit image of `float` values with the properties from the specified multichannel 2D *ModelImage* instance and the values from the specified channel.

Table 9. Description of static methods provided by *InsightToolkitSupport* class (continued)

Static Method	Description
itkCreateImageSingleSlice	This method creates a single-channel 2D InsightToolkit image of `float` values with the properties and values from the specified slice of the single channel 3D *ModelImage* instance.
itkCreateImageColorSlice	This method creates a single-channel 2D InsightToolkit image of `float` values with the properties from the specified multichannel 3D *ModelImage* instance and the values from the specified slice and channel.
itkTransferImageSingle3D	Given two InsightToolkit single-channel 3D images of `float` values, pixel values from either image are transferred into the specified single-channel 3D *ModelImage* instance. The transfer occurs on a pixel-by-pixel basis and is determined by the flag in the corresponding pixel in a binary mask image.
itkTransferImageColor3D	Given two InsightToolkit single-channel 3D images of `float` values, pixel values from either image are transferred into the specified multichannel 3D *ModelImage* instance. The transfer occurs on a pixel-by-pixel basis and is determined by the flag in the corresponding pixel in a binary mask image.
itkTransferImageSingle2D	Given two InsightToolkit single-channel 2D images of `float` values, pixel values from either image are transferred into the specified single-channel 2D *ModelImage* instance. The transfer occurs on a pixel-by-pixel basis and is determined by the flag in the corresponding pixel in a binary mask image.
itkTransferImageColor2D	Given two InsightToolkit single-channel 2D images of `float` values, pixel values from either image are transferred into the specified multichannel 2D *ModelImage* instance. The transfer occurs on a pixel-by-pixel basis and is determined by the flag in the corresponding pixel in a binary mask image.
itkTransferImageSingleSlice	Given two InsightToolkit single-channel 2D images of float values, pixel values from either image are transferred into the specified slice of the multichannel 3D *ModelImage* instance. The transfer occurs on a pixel-by-pixel basis and is determined by the flag in the corresponding pixel in a binary mask image.
itkTransferImageColorSlice	Given two InsightToolkit single-channel 2D images of `float` values, pixel values from either image are transferred into the specified slice and channel of the multichannel 3D *ModelImage* instance. The transfer occurs on a pixel-by-pixel basis and is determined by the flag in the corresponding pixel in a binary mask image.

Demonstrating InsightToolkit integration

To demonstrate the integration of the InsightToolkit into MIPAV, this section uses the InsightToolkit Gaussian blur filter. Classes that were added are:

- ***AlgorithmGaussianBlurITK*** class—This class contains the code that instantiates the Java InsightToolkit objects, uses the MIPAV ***InsightToolkitSupport*** class to convert between *MIPAV ModelImage* instances and *InsightToolkit itkImageF2* and *itkImageF3* instances, and performs the Gaussian blur operation.
- ***JDialogGaussianBlurITK*** class—This class contains the Gaussian Blur dialog (Figure 373) to allow users to select the options for performing the InsightToolkit version of the Gaussian blur operation.

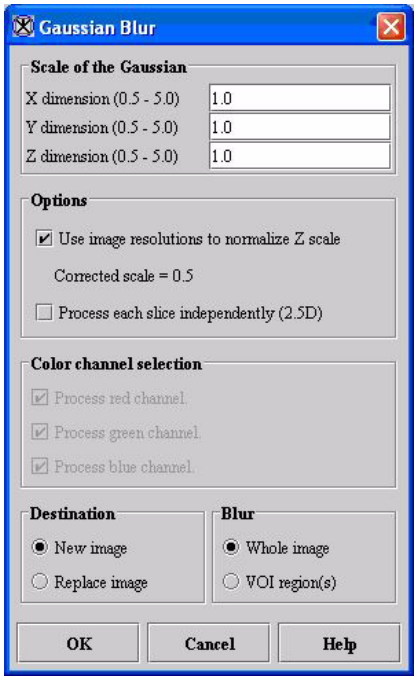


Figure 373. Gaussian Blur dialog box

The options for the InsightToolkit Gaussian blur operation are almost identical to those for the MIPAV Gaussian blur operation with the following exceptions:

- The InsightToolkit Gaussian blur can only be performed using recursive separable filters.
- There is no choice to perform the operation any other way (e.g., multidimensional convolution).