GenEx

Software library for object detection in genetical analysis evaluation

User manual

Jan Schier Bohumil Kovář
Michal Kuneš
Institute of Information Theory and Automation of the AS CR
schier@utia.cas.cz

Supported from the TA01010931 project
of the Technology Agency of the Czech Republic

December 19. 2014

Contents

List of Figures

1 Acknowledgement

This work has been supported by the research project TA010100931 of the Technology Agency of the Czech Republic.

2 *GenEx* library for evaluation of FISH microscopy images

In this technical report, the *GenEx* library for evaluation of FISH microscopy images is described. The purpose of the library is to find and filter objects in images from fluorescence microscope and to detect fluorescence signals contained in these objects. In the GenEx project, it is used for diagnostics of the Turner syndrome — chromosome aneuploidies in chromosome X, in other words, to find the (roughly round) interphase nuclei and to count the associated hybridization signals (see illustrative images in Figure 1; the colour channels of the image are shown also as surface plots, to better reveal the character of the hybridization signals).

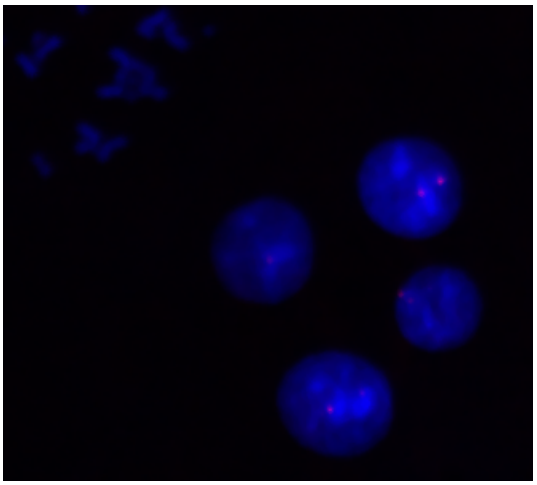
The library is built on the top of the ImageJ environment (`http://www.imagej.org`), which is widely used in bioimaging.

The library is characterized by the following features:

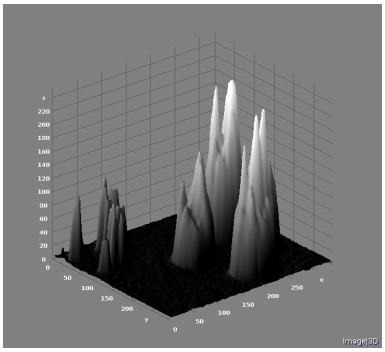
- separate processing channels for DAPI and for the fluorescence signals.
- no restrictions on the number of the signal channels
- output in the form of a CSV-file – ImageJ ResultTable export

The block diagram in Fig. 2 shows the high level functionality of the library: it takes two or more sets of the input images – one for the DAPI-stained images ("DAPI channel" in the following text), the others for the images with fluorescence markers marking specific chromosomes (such as the TxR, GFP, etc.) ("signal channel" in the following text) and performs processing of the DAPI channel to find cell nuclei contained in the image. The borders of the nuclei found in the DAPI channel, are used to select the regions of interest for the hybridization signals (bright spots) in the other channels, corresponding with the chromosomes of interest. As the output, two sets of data are provided for each channel, that is, the mathematical descriptors (see Set Measurements section of the ImageJ user guide) and the contours of the objects.

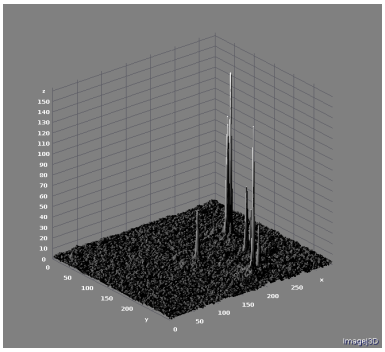
The processing flow in each channel (DAPI, signal) is schematically depicted in Figure 3. The pipeline includes background subtraction, noise suppression, image segmenation, particle analysis and object filter.



Composed RGB image, lense magnification 40×



DAPI channel
Cell nuclei



TxR channel
Hybridization signals

Figure 1: Example of an input image

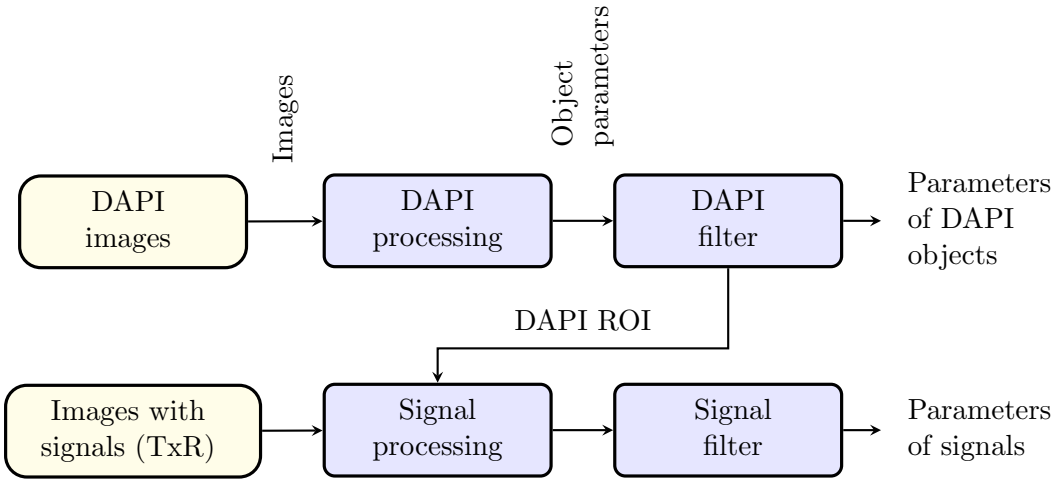


Figure 2: Main components of the GenEx plugin

3 Source of images

The library takes two options for the source of input images: either an image stack displayed in the ImageJ image window, or a set of files from a disk directory.

4 Library output

The output of the library consists of two sorts of data for each channel: the contours of the objects found in the image, and the results tables containg parameters of these objects.

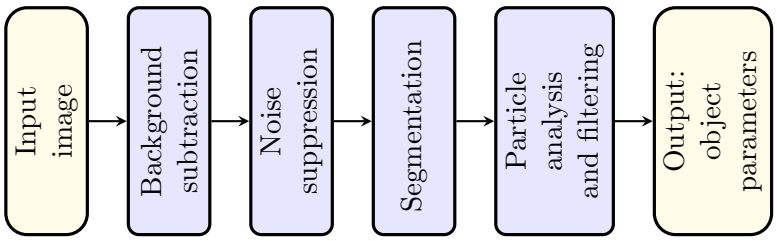


Figure 3: Image processing pipeline

5 Architecture of the library

The library consists of two main packages, `genex` and `genexd.lib` and several specialized packages, such as `genex.lib.segmentation`. Figure 4 shows, on a schematic level, the relations between the main components contained in the `genex` and `genex.lib` packages. The purpose of these components is following:

5.1 Default package

- the `GenExPlugin_` class provides the interface for using the plugin from within the *ImageJ* environment. Note that the rest of the library, while using the ImageJ classes, can be linked to any Java code. This class is placed in the `default` package.

5.2 Genex package

The `genex` package contains the high/level classes which define the processing flow:

- Either of the `GenexPipeline` and `GenexIterator` classes can be used to call the image processing pipeline.

  The `GenexPipeline` class provides more general solution, which can be configured either to process all available images in one batch, or to process them image-by-image.

  The `GenexIterator` class provides an interface conformant to the Java *Iterator* interface - the `hasNext()` method returns true if there is any image to process, while the `next()` method provides the results for single image.

- The `DapiProcessor` and `SignalProcessor` classes define the actual image processing pipelines for DAPI channel and for the channels with imagess of the hybridization signals.

5.3 Genex.lib package

The `genex.lib` package contains class that are, in general, used to cope with concrete problem, plus some helper classes. The important classes contained in this package are the following:

- The *Definitions* class, `GenExDefines.java`, is where the constants and definitions used in the code are put.

- *Image input*, that is, the `ImageReader.java` interface, provides a unified interface for all image input methods, be it the stack of images on the screen or image files.

Current implementations of this interface include `ScreenImgReader.java` and `FileImgReader.java`.

- The *Segmentation* class, `Segmentation.java`, takes the input image and performs segmentation and, if enabled, watershed segmentation (which is used to "disconnect" touching particles). The output of segmenation is binary image.

  The concrete segmentation method to use is selected using the AbstractFactory mechanism (the implementatiom is contained in `genex.lib.thresholder`). At this point, several threshold segmentation methods are included: the Snell segmentation [Snell2011] (contained in `genex.lib.thresholder`) and the ImageJ segmentation methods (contained in `ij.process.AutoThresholder`). However, it is trivial to plug in any other method with the AbstractFactory mechanism.

- *Extended regions* (`ExtRoi.java`) is an encapsulation of the *ImageJ Roi* class (`ij.gui.Roi`). This encapsulation supplies the standard `Roi` class with important additional information, such as the links between the nucleus and the fluorescence signals contained therein, the name of the image where the object is located and the validity flag for the object (only some of the objects found by the segmentation and particle analysis routines are valid objects in the sense that they represent cell nuclei or fluorescence signals).

- *Particle Analysys* is part of the `Channel.java` class. Using the segmentation output, it determines the particles contained in the image. The output of the implementation in the *GenEx* library is complemented also with the data of the *Extended regions* class – the links between the objects and the name of the image containing given particle.

  Also, as part of paticle analysis, the particles are measured and the *Results table* (`ij.measure.ResultsTable`) is filled with object parameters.

- The purpose of the *Object filter* (`ObjectFilter.java`) is to classify the objects to valid and invalid ones, based on selected criterion.

  At this point, simple filter based on area and circularity of the particles is used. The filter is connected to the code using AbstractFactory, making addition of other methods fairly simple. An adapter method for the WEKA data mining tool is in preparation.

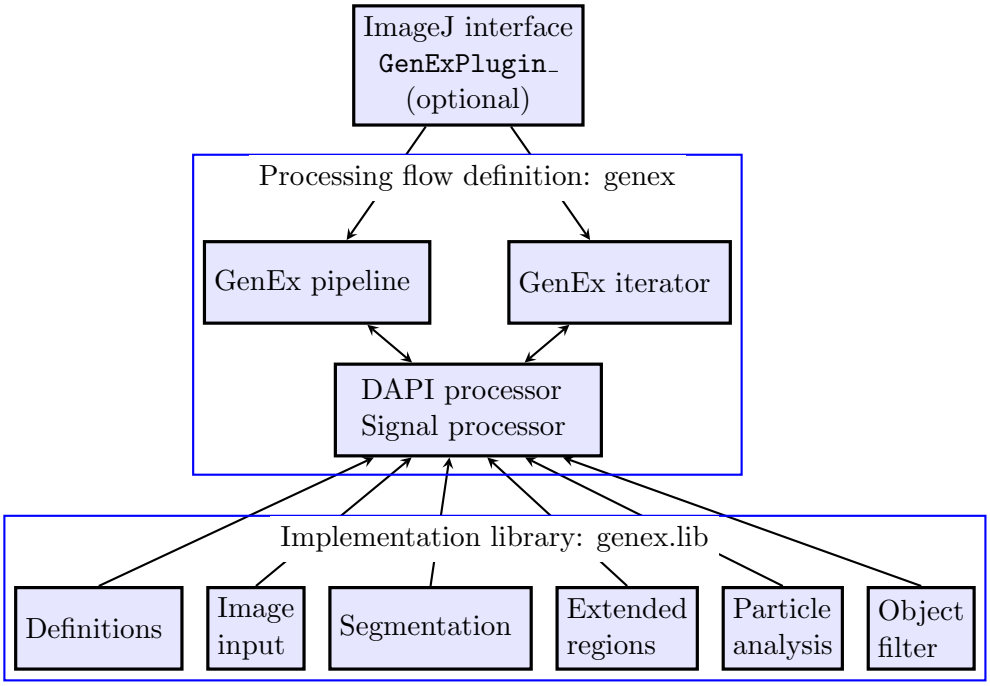


Figure 4: Plugin architecture

6 Library parameters

The library uses several parameters to define basic settings of the environment. The parameters are stored using the java *Preferences* library (`java.util.prefs.Preferences`), in the form of *key–value* pairs. The Preferences library automatically cares for the proper approach to the parameter storage.

The parameters employed in the *GenEx* library are used to describe which color channels are used in the essay (at this point, DAPI, RED and GREEN are used), what are the file patterns for these channels, the location of the image directory and of the directory for the CSV[1] result files. The parameters are summarized in Table 1.

6.1 Parameter adjustment methods

The `GenExParams_` plugin provides simple GUI for input of parameters. The relation between the plugin, parameter storage and the GenEx library is schematically shown in Figure 5. This GUI can be used to set the channel usage flags for DAPI, GREEN, RED, the file patterns and the image and results directories. The IS_PLUGIN and SAVE_BY_IMAGE parameters are only set in the code.

[1] Comma-separated values format

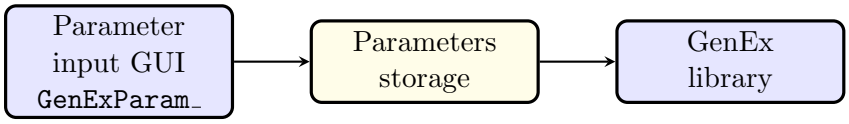


Figure 5: Setting parameters

Other option is to set the parameters directly using the `java.util.prefs.Preferences` API.

6.2 File patterns and input files

The file patterns are used in the case of separate image files for each channel, to identify the files corresponding to the particular channel. Example: with `iXXX-dapi.tiff` filenames, where `XXX` is the number of image, the string "`dapi`" can be used to identify the DAPI channel.

In the case when the images comes in the form of RGB files, only one common pattern is used in all channels, and this pattern may be empty.

The RGB_INPUT is used only in the `GenExParams_` plugin, the Genex library uses autodetection of the file contents (the file patterns must be set properly, though).

6.3 Input checks for plugin

In the case that the IS_PLUGIN parameter is set, the parameters entered by user in the `GenExParams_` plugin are checked for the following:

- the file patterns for illegal characters (regular expressions are not allowed, for simple usage).
- if RGB_INPUT is not set, the file patterns must be non-empty.
- the input directory must be non-empty and must contain images.
- if RGB_INPUT is not set, the number of images contained in the directory must be the same for each channel.

7 Definitions

The `GenExdefines` class is used to introduce important definitions and constants, used throughout the code. These include:

ChannelNames is an enumerated type used to define the name of each channel and the transformation of this name to a string. This type should be extended if more signal channels are needed:

Key	Value	Comment
DAPI	boolean	is the channel used? (always true)
DAPI_PATTERN	String	the pattern used to select the DAPI image files
GREEN, RED	boolean	is the channel used?
GREEN_PATTERN RED_PATTERN	String	the pattern used to select image files of the given channel
IS_PLUGIN	boolean	set if the library is run as ImageJ plugin (switches on some correctness checks on parameter input)
RGB_INPUT	boolean	the input images are stored in RGB files (used only in the parameter input plugin)
SAVE_BY_IMAGE	boolean	if `true`, the data for each image are saved immediately after this image is processed. If `false`, the images are processed in batch and data is saved after the batch is processed.
LAST_INPUT_DIR	String	the directory, where the input images are stored
CSV_DIR	String	the directory, where the files with results are stored

Table 1: Parameters used by the GenEx library

```
public static enum ChannelNames {
   DAPI("DAPI"),
   RED("RED"),
   GREEN("GREEN"),
   UNSPECIFIED("");
}
```

signals is used to define which ChannelNames are reserved for signals:

```
ChannelNames[] signals = {ChannelNames.RED,
    ChannelNames.GREEN};
```

Note that which signal is actually used depends on the setting of the corresponding RED and GREEN parameters (see Section 6).

Filters is an enumerated type used to identify the denoising filters:

```
public static enum Filters {
```

```
    FILT_MEDIAN, FILT_GAUSSIAN, FILT_DESPECKLE, FILT_NONE
}
```

objectMeasurements is an integer constants which defines the parameters evaluated for particles in the image:

```
public static int objectMeasurements =
      ParticleAnalyzer.SLICE +
      ParticleAnalyzer.AREA +
      ParticleAnalyzer.CIRCULARITY +
      ParticleAnalyzer.PERIMETER +
      ParticleAnalyzer.MEAN +
      ParticleAnalyzer.CIRCULARITY +
      ParticleAnalyzer.RECT +
      ParticleAnalyzer.CENTER_OF_MASS +
      ParticleAnalyzer.SHAPE_DESCRIPTORS +
      ParticleAnalyzer.LABELS;
```

8 Image input

Image input methods are defined by the `ImageReder` interface, which is implemented by two classes, `ImgScreenReader` and `ImgFileRedaer`. `ImgScreenReader` is used to read images from a stack already opened in ImageJ and displayed on computer screen, while `imgFileReader` is used to read images directly from disk files. The relation between these classes is shown by the UML graph in Figure 6.

9 DAPI Segmentation

DAPI segmentation is performed in the `Segmentation` class, using an AbstractFactory mechanism to switch between the segmentation methods – currently either the thresholding methods contained in ij.process.Autothresholder or the Snell segmentation method.

This mechanism is illustrated by the following code snippet:

```
AbstractThresholderFactory thresholderFactory = new
    SnellThresholderFactory();
Thresholder thresholder = thresholderFactory.createThresholder();
int[] thr = thresholder.getThreshold(img);
ImageProcessor ip = img.getProcessor();
for (int image=0; image<img.getStackSize(); image++) {
      ip.threshold(thr[image]);
    }
```

Note the array used for the thresholds: for greater flexibility, ImageJ *ImageStack* mechanism (ij.ImageStack) is used throughout the code. This al-

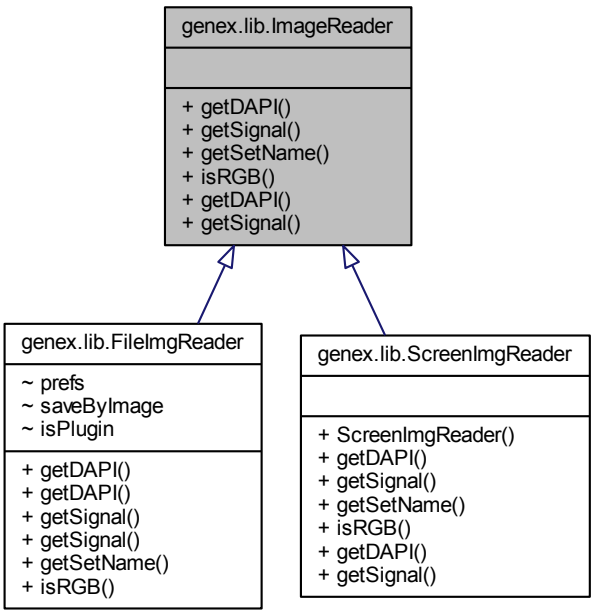


Figure 6: ImageReader interface and its implementations

lows to use the same code no matter whether a single image or the complete stack should be processed in one turn.

With the code above, an individual threshold is applied to each image. With only one image in the stack, the threshold array `thr` will contain only one element.

10 Obtaining results

10.1 Export to CSV files

The results are internally stored in the ImageJ *ResultsTable* class. At the end of processing, or after processing single file (depends on the setting of the `processByImage` variable in the `GenExPlugin_` class), they saved into a CSV file, located in a directory set in the CSV_DIR key (see Section 6, *library parameters*). If this key does not exist, the file is saved into the image directory.

The format of the file is shown in the example in Table 2. Concrete parameters included in the file depend on the setting of the `objectMeasurements` variable in the `GenExDefines` class (see Section 7).

Additional keys The following keys are added to those used by default in ImageJ:

Label unique label of the particle

Image filename of the image containing the particle

Valid particle validity as determined by the object filter

Expert particle validity as set by expert with the tagger GUI. This key will be used to train the WEKA classifier and in comparative evaluation of the library.

The following keys are used for signals:

Nucleus index of nucleus containing given signal

Color color of given signal (RED|GREEN). The data for signals in all channels are stored in one common table. channels

	Label	Area	Mean	Perim.	Circ.	AR	Round	Solidity	Image	Valid	Expert
1	10001-0001-0059	1834	77.234	160.610	0.893	1.080	0.926	0.967	01a	true	true
2	10001-0002-0092	2570	119.447	188.752	0.906	1.139	0.878	0.974	01a	true	true
3	10001-0003-0144	2587	52.545	190.652	0.894	1.133	0.882	0.972	01a	true	true
4	10001-0004-0153	3522	54.973	221.380	0.903	1.041	0.961	0.971	01a	true	true
5	10001-0005-0246	2384	77.049	181.581	0.909	1.105	0.905	0.972	01a	true	true

Note: not all data is included

Table 2: Example of the result table

Filename format of the CSV file The filename of the CSV file is constructed using the filename of first (`fnFirst`) and last (`fnLast`) image in the directory as `fnFirst-fnLast-dapi.csv` (or `-signal`, correspondingly). In the case that only one image is processed, the filename is simplified to `filename-dapi.csv`.

10.2 Getting results directly

The results can be obtained directly from the code of GenEx library, using the `GenExPipeline` class. It contains the `runPipeline` method in several variants, which returns an object of the `Results` class:

```
public Results runPipeline(int objectMeasures)
```

This is a composed object, containing all data obtained from the images:

```
public class Results {
 public FISHdata getDapi();
 public FISHdata getSignal();
 public Results(FISHdata dapi, FISHdata signal) {
    this.dapi = dapi;
    this.signal = signal;
 }
}
```

where the `FISHData` class contains the results table and the ROIs of the objects:

```
public class FISHdata {
 public FISHdata(ResultsTable results, ArrayList<ExtRoi> rois) {
    this.results = results;
    this.rois = rois;
 }

 public ResultsTable getResultsTable();

 public ArrayList<ExtRoi> getRois();
}
```

The data can be accessed using the getter methods of these classes, combined with the access methods of ij.measure.ResultsTable class.

11 Basic processing loop

In this section, we describe the processing code of the library at the highest level. It is fairly simple:

```
boolean processByImage = false;
Results results = null;
```

```
GenexPipeline pipeline = new GenexPipeline(channels);
if (pipeline.readImages()>0) {
   results = pipeline.runPipeline(GenExDefines.objectMeasurements);
}
// process results
```

In the first line, the processing pipeline is instantiated. Then, if a non-zero number of images is read from the pipeline, the processing method `runPipeline` is called. This code is used for batch processing of images.

For processing of individual images, a while-loop is used instead of the if-condition:

```
boolean processByImage = true;
GenexPipeline pipeline = new GenexPipeline(channels);
while (pipeline.readImages()>0) {
   Results results =
       pipeline.runPipeline(GenExDefines.objectMeasurements);
   // process results
}
```

12 Data tagger

To be able to collect input from expert and to compare it with the library performance, a graphical data tagger has been designed. It is contained in the `genex.lib.RoiLabeler` class. The tagger displays the microscope image with the contours of detected nuclei and signals overlaid. It allows to enable/disable these contours using a menu-driven interface. The results of tagging are stored in the "Expert" column of the result table. An illustrative screenshot is given in Figure 7.

A Call graphs for important classes

In this appendix, the call- and collaboration graphs for several important classes of the library are shown. Full details are given in the API description.

References

[Snell2011] Violet Snell, William Christmas, and Josef Kittler. Segmentation and shape classification of nuclei in DAPI images. In *The 22nd British Machine Vision Conference*, pages 1–9, Dundee, 2011.

Figure 7: Data tagger – example

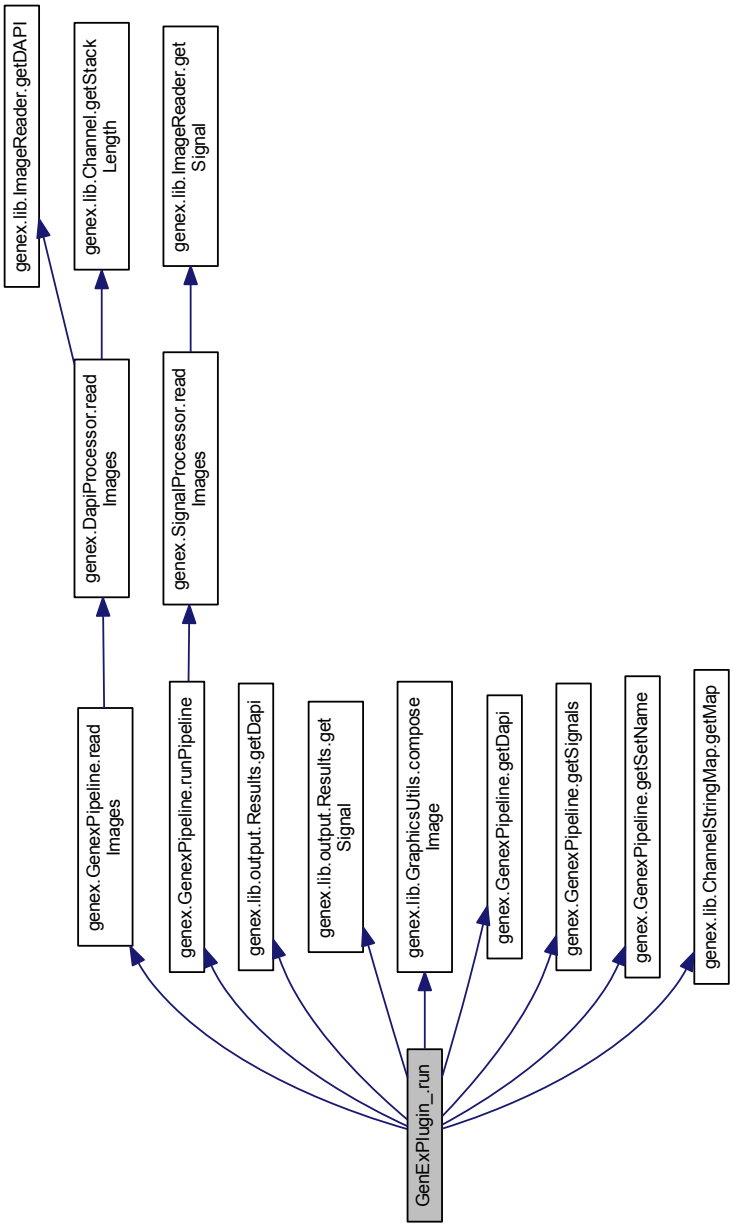


Figure 8: Call graph of the GenExPlugin_ class

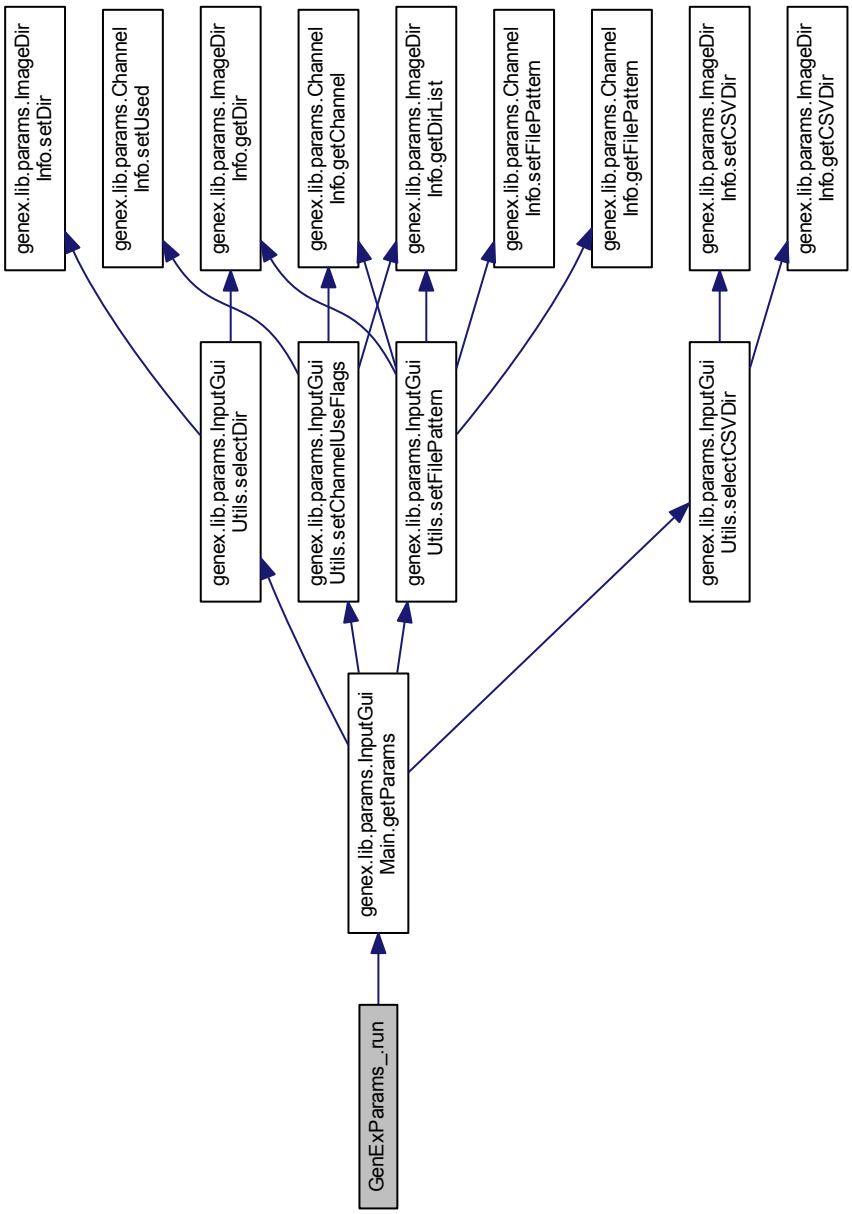


Figure 9: Call graph of the GenExParam_ class

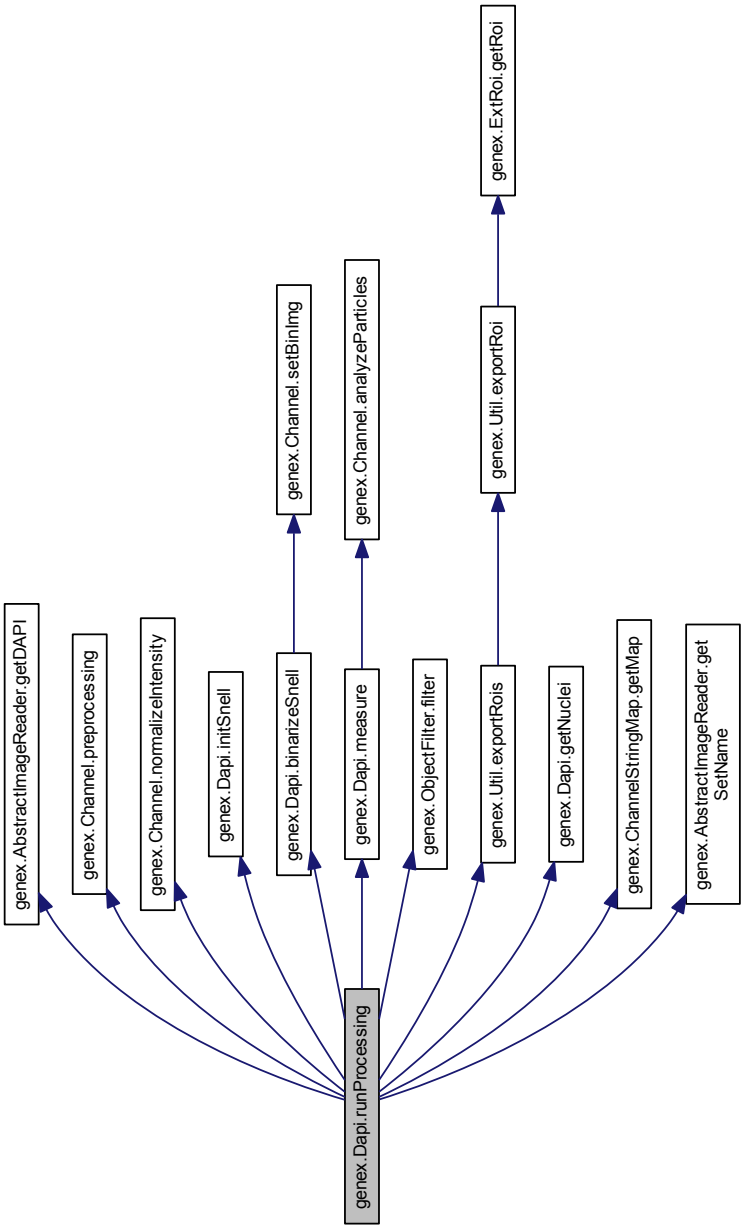


Figure 10: Call graph of the Dapi class

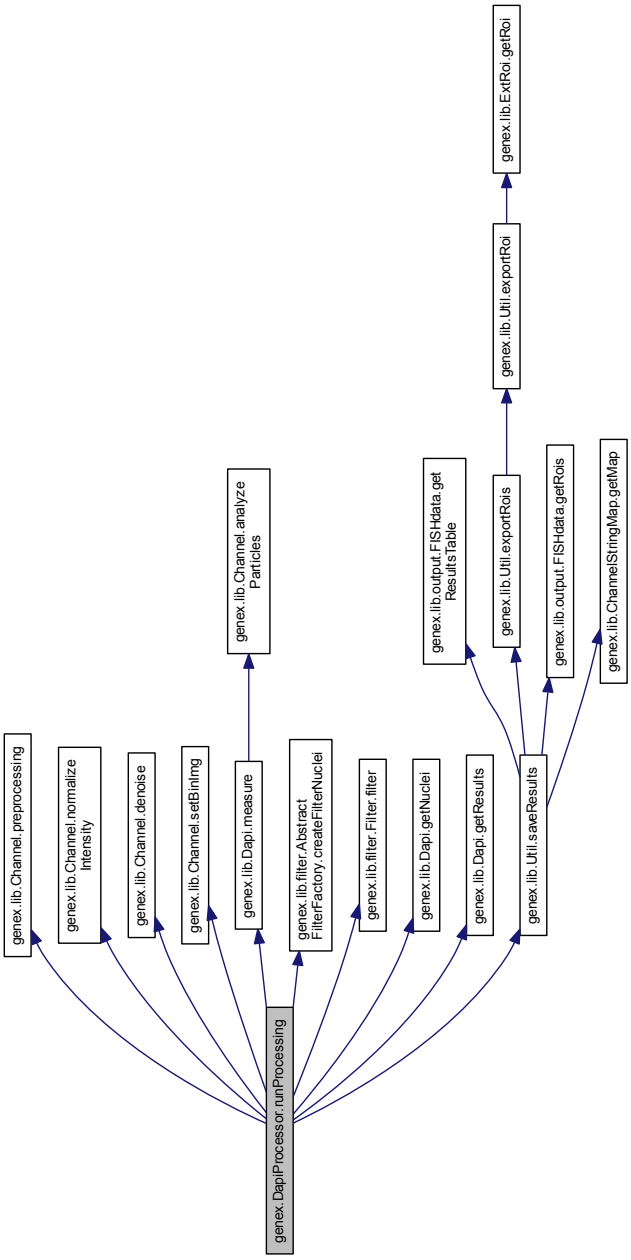


Figure 11: Call graph of the DapiProcessor class

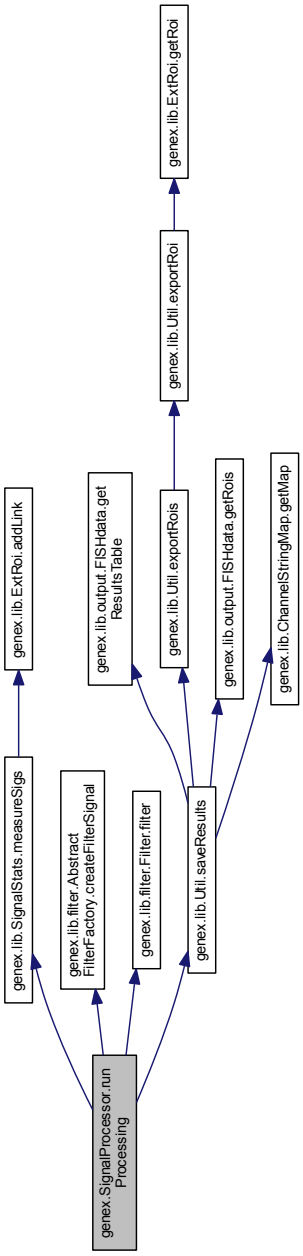


Figure 12: Call graph of the SignalProcessor class

Figure 13: Collaboration graph of the GenexPipeline class