# Getting started with BioCAT development

## Packages and classes

### annotool.extract

This package contains feature extractor classes. All feature extractors must implement FeatureExtractor interface.

### annotool.select

This package contains feature selector classes. All feature selectors must implement FeatureSelector interface.

### annotool.classify

This package contains feature classifiers. All feature classifiers implement FeatureClassifier interface.

### annotool.gui

This package contains gui classes. Some of the major classes in this package are as follows:

* **AnnotatorGUI**: This is the main class that is executed at the start.
* **LandingPanel**: This is the panel that contains wizard steps.
* **LoadImageDialog**: Dialog that comes up when loading images. It includes three tabs for three different modes.
  + **TargetFileDialogPanel**: Interface for loading images from directory and associated target file.
  + **DirectoryTreeDialogPanel**: Interface for loading images from subdirectories where each directory is a class for images inside it.
  + **ROIModeDialogPanel**: Interface for loading a single image and files determining region of interests on this image.
* **AnnTablePanel** **and** **AnnImageTable**: AnnImageTable represents the table with all the images loaded on it. AnnTablePanel can hold one or two AnnImageTable.
* **ImageReadyPanel:** This is the panel that is displayed once images are loaded. This contains AnnTablePanel. This has class information loaded, radio buttons to select channels, buttons to execute Expert or AutoComparison modes and status area.
* **ExpertFrame and AutoCompFrame:** These two subclasses of PopUpFrame corresponds to the two different modes used for training. Multiple instances of these can be created.

### annotool.gui.model

This package has helper classes that interface with gui classes.

* **Chain:** This class represents a chain of algorithms. It can hold one or more extractor(s), one or more selector(s) and a single classifier.
* **ChainFilter:** FileFilter to display only files with chain extension(.ichn) when opening or saving files.
* **ChainTableModel:** TableModel for displaying chains in table.
* **Extractor:** Class that represents single extractor in a chain. This is different than feature extractor algorithm. This class has information such as extractor name, corresponding class, external path if the feature extractor is a external plugin and feature extractor parameters.
* **ImageFilter:** FileFilter to display only image files when opening or saving files.
* **ModelFilter:** FileFilter to display only model files (.imdl) when opening or saving files. Models files contains trained data.
* **ModeLoader:** This class is responsible for loading and applying saved models to problems.
* **ModelSaver:** This class is responsible for saving model files once the trained models are created.
* **PDFFilter:** FileFilter to display only pdf files when opening or saving files.
* **Selector:** Class representing a single selector in a chain. Similar to Extractor above but for feature selectors.
* **Styles:** Defines some constants used by report saver to format pdf report created.
* **Utils:** Contains some utility methods such as creating image icons, getting file extensions etc.

### annotool.io

This package contains classes responsible for input/output operations such as loading and saving files from/into disk.

* **Algorithm**: This class represents a single algorithm (extractor, selector or classifier) loaded from xml file to be displayed in drop down lists.
* **AlgoXMLParser**:This class reads the xml file (Algorithms.xml) and loads the information about the algorithms (name, type, class etc).
* **ChainIO:** This class is responsible for saving and loading chains of algorithms so that once created, a set of chains can be loaded to run again in auto comparison mode.
* **ChainModel:** This class saves model files generated after training.
* **DataInput:** This class is responsible for getting image data from input image or image sets.
* **DirectoryReader:** This class scans images in directory when using directory structure input mode.
* **LabelReaer:** This class reads the target file for class names, annotation name and to associate each image in the input set with one or more class(es).
* **Parameter:** This class represents a parameter for algorithm. It defines the name of the parameter, it's type (Boolean, integer etc), maximum/minimum values, default values etc.
* **PluginScanner:** This class is responsible for scanning plugins directory to check for algorithms.
* **ReportSaver:** This class is responsible for saving classification results as pdf file.

## Plugins

New algorithms can be introduced using plugins in BioCAT. The plugins folder is located at the root level of application i.e. the same level as the executable. Each plugin conforms to the following guideline.

* Each plugin has a subdirectory under plugins directory. This subdirectory contains the plugin files.
* Each plugin folder has an xml file called Algorithm.xml which has the same format as the Algorithms.xml file (the default list of algorithms). This xml file defines the name of the plugin (as it appears in the gui), a short description of the plugin, name of the class associated with the plugin, path of the pugin under plugins folder and the parameters it takes.
* The plugin files can be provided as a jar file or a class file (can be multiple files as long as one of them is the class specified in the xml file).

## Model Files

These are files associated with the trained models generated after running a sequence of algorithms (extractor->selector->classifier) on an image set. For each annotation target, there are two files generated.

* .imdl file: This is a plain text file that contains training information such as size of images, channel used, classes, extractor information, indices of features to select (based on the selector(s) used in training), classifier and path to the model created by the classifier.
* Model file: This is the trained model generated by the classifier used. When classifying new problems, this data is used on the selected features of the new problems.

## Libraries to include

Following libraries are used by BioCAT and should be linked to BioCAT project.

* ImageJ(ij.jar) – This is mainly for reading images.
* iText(itextpdf) – This is for writing pdf files.
* jFreeChart – This one is for generating charts.
* FeatureJ (featureJ\_.jar and imagescience.jar) – This is an imagej plugin for extracting some of the features.
* Jama
* libsvm
* weka
* jcommon