Developing VStar Plug-ins

Table of Contents

Introduction

Some things clearly belong to the basic set of features needed by a program such as VStar such as the ability to:

- Plot light curves and phase plots
- View data in tabular format
- Load observation data from a variety of sources
- Save and print
- Filter/search
- Perform period analysis

Other features are less obvious. Even some of the items on the list above are not clear-cut. Which period analysis algorithms should be included? What format should files be saved in? What filter expressions should be permitted? It is not possible to anticipate every use to which VStar may be put.

To address this, VStar has a plug-in capability that allows anyone with a moderate knowledge of Java (or other programming languages) to add new functionality to VStar without having to modify its source code, or to have commit rights to the GitHub source code repository, or to be able to rebuild and redeploy VStar.

In addition, a plug-in may be a good way to prototype an idea or algorithm that may ultimately be included in the VStar source tree. In this way, the VStar user community can more easily engage in the tool's future development, and the tool's flexibility is enhanced.

In principle, the plug-in developer is not constrained to use only Java. Any language that can generate Java Virtual Machine class files may be used to develop plug-ins, e.g. Scala, Ruby (JRuby), Python (Jython), JavaScript (Nashorn, GraalVM, V8), Groovy, Lisp (e.g. Clojure). Having said that, Java is the most straighforward choice.

Plug-in Types

There are currently six types of VStar plug-in:

1. Observation tool
2. General tool
3. Custom filter
4. Period analysis
5. Observation source
6. Observation sink
7. Model creation
8. Observation transformer

All plug-ins must take the form of a jar (Java archive) file. Once this jar file is copied to a particular directory, the plug-in will appear as a menu item next time VStar is started.

An *observation tool plug-in* will appear in the Tool menu and is able to perform arbitrary processing on the currently loaded dataset.

A *general tool plug-in* is intended to be used for general utilities and will appear in the Tool menu.

A *custom filter plug-in* filters data in the plots and tables as per the View -> Filter… menu item, permitting an arbitrarily complex filter expression. Custom filters will appear in the View -> Custom Filter sub-menu.

A *period analysis plug-in* will appear in the Analysis -> Period Search sub-menu and applies an algorithm such as a Fourier Transform to a particular series (band) of the loaded dataset. By default, the data that is passed to the plug-in is the series that is used as the basis for the light curve mean plot. The results may then be plotted and/or stored in a table. Support is provided for creating a dialog for displaying the results of the period analysis and for selecting a period value with which to create a new phase plot.

An *observation source plug-in* will appear in the File menu or within a file chooser and permits observation datasets to be loaded from arbitrary sources, such as FITS files, web services or HTTP data streams, databases other than AAVSO's, or special file formats. Implementations of these plug-ins have been the most frequent of all so far.

An *observation sink plug-in* will appear in the file save chooser and permits the observation list to be saved in a particular format.

A *model creation plug-in* will appear in the Analysis menu and can be used to create an arbitrary model of the loaded dataset, e.g. custom curve fitting.

An *observation transformer plug-in* will appear in the Tool menu and can be used to transform currently loaded observations in arbitrary ways.

Application Programming Interface

VStar's application programming interface (API) for plug-in development is captured as Javadoc.

The file `vstar.jar` contains VStar's functionality and is used for VStar plug-in development. For a particular release of VStar, this file can be found in the:

- `VStar/dist` directory for Windows and Linux, or
- `VStar.app/Contents/Java` directory on Mac OS X.

The latest VStar documentation (Javadoc) zip file can be downloaded from a release directory:

- https://github.com/AAVSO/VStar/releases

Insofar as creating plug-ins is concerned, the key classes and interfaces are:

1. `org.aavso.tools.vstar.plugin.PluginBase`
2. `org.aavso.tools.vstar.plugin.ObservationToolPluginBase`
3. `org.aavso.tools.vstar.plugin.GeneralToolPluginBase`
4. `org.aavso.tools.vstar.plugin.ObservationSourcePluginBase`
5. `org.aavso.tools.vstar.plugin.ObservationSinkPluginBase`
6. `org.aavso.tools.vstar.plugin.CustomFilterPluginBase`
7. `org.aavso.tools.vstar.plugin.ModelCreatorPluginBase`
8. `org.aavso.tools.vstar.plugin.ObservationTransformerPluginBase`
9. `org.aavso.tools.vstar.plugin.period.PeriodAnalysisPluginBase`
10. `org.aavso.tools.vstar.plugin.period.PeriodAnalysisDialogBase`
11. `org.aavso.tools.vstar.plugin.period.PeriodAnalysisComponentFactory`
12. `org.aavso.tools.vstar.data.ValidObservation`

The `org.aavso.tools.vstar.plugin.period.impl` package contains an example period analysis plug-in implementation: `DcDftPeriodAnalysisPlugin` for the Date Compensated Discrete Fourier Transform algorithm. This class is presented internally to VStar as just another plug-in.

Given that plug-ins references `vstar.jar`, a plug-in can make use of other VStar classes (e.g. `org.aavso.tools.vstar.ui.dialog.MessageBox`).

For examples of plug-ins in use, see:

- https://github.com/AAVSO/VStar/tree/master/plugin/src/org/aavso/tools/vstar/external/plugin

Creating an Observation Tool plug-in

Suppose you wanted to create a simple plug-in to count the number of observations in the loaded dataset. While not very useful in itself, working through a simple example like this will get you started towards creating more complex, useful plug-ins. Here are the steps to follow:

- Ensure you have installed the Java Development Kit, version 1.6 or higher.
- Locate the version of `vstar.jar` associated with the release of VStar you wish to develop plug-ins for.
- Create a file with your favourite editor or Integrated Development Environment (e.g. Eclipse) with the following content:

```
package my.vstar.plugin;

import java.util.List;

import org.aavso.tools.vstar.data.ValidObservation;
import org.aavso.tools.vstar.plugin.ObservationToolPluginBase;
import org.aavso.tools.vstar.ui.dialog.MessageBox;

/**
 * This is a simple observation counting VStar tool plug-in.
 */
public class ObservationCounter extends ObservationToolPluginBase {

	@Override
	public void invoke(List<ValidObservation> obs) {
		int count = obs.size();
		MessageBox.showMessageDialog("Observation Count", String.format(
				"There are %d observations in the loaded dataset.",
count));
	}

	@Override
	public String getDescription() {
		return "Observation counting tool plug-in";
	}

	@Override
	public String getDisplayName() {
		return "Observation Counter";
	}
}
```

Some key things to note are:

- The file must be called `ObservationCounter.java` and located in the directory `my/vstar/plugin` relative to the current directory. Let's call the latter `src`.
- After saving the file, and assuming you are taking a bare-bones approach to editing and compiling Java code , from a DOS prompt or *nix shell, with `src` as your current directory, compile the source code, including the vstar.jar file

(specifying the full path to vstar.jar) in the classpath, like this:

```
javac -cp vstar.jar my/vstar/plugin/ObservationCounter.java
```

- Next, create a jar file with the same name as the fully qualified `ToolPluginBase` subclass. Note that this naming scheme is not optional. In this case, we would have:

```
jar -cf my.vstar.plugin.ObservationCounter.jar
        my/vstar/plugin/ObservationCounter.class
```

- Next, move the jar file to the `vstar_plugins` directory. Plug-in jars must be placed into the `vstar_plugins` directory in your home directory (for Mac or *nix that will be `$HOME` or `~`, "`C:\Documents and Settings\`*`user`*" under Windows).

- Finally, start VStar. A menu item called `Observation Counter` should appear in the Tool menu. Load a dataset from a file or the database and select that menu item. A simple dialog box should appear telling you how many observations are loaded.

Subsequent sections point to GitHub for plugin type examples.

See also *Plug-in Development Tool* for plug-in building support.

Creating a General Tool plug-in

This section shows an example of a simple General Tool plug-in that converts a Julian Day to a calendar date:

- https://github.com/AAVSO/VStar/blob/master/plugin/src/org/aavso/tools/vstar/external/plugin/JDToDateTool.java

The source file needs to be `JDToDateTool.java` and must be located in the directory `org/aavso/tools/vstar/external/plugin`. If the package on line 4 was changed to:

```
    package my.vstar.plugin;
```

then the directory `my/vstar/plugin` could be used instead.

Creating a Custom Filter plug-in

Here is a simple example of a custom filter plug-in

- https://github.com/AAVSO/VStar/blob/master/plugin/src/org/aavso/tools/vstar/example/plugin/CustomFilterTest.java

See the following for a more complex (and useful example):

- https://github.com/AAVSO/VStar/blob/master/plugin/src/org/aavso/tools/vstar/external/plugin/ObserverListFilter.java

Creating a Period Analysis plug-in

Here is a simple example of a period analysis plug-in:

- https://github.com/AAVSO/VStar/blob/master/plugin/src/org/aavso/tools/vstar/example/plugin/PeriodAnalysis.java

For a more complex example, see:

- https://github.com/AAVSO/VStar/blob/master/plugin/src/org/aavso/tools/vstar/external/plugin/AoVPeriodSearch.java

Creating an Observation Source plug-in

By developing an observation source plug-in, VStar can accept observations from arbitrary data sources, such as a non-AAVSO database, a web service, a file format not understood by VStar natively, and so on. These plug-ins appear as additional File menu items.

The `ValidObservation` fields displayed in plots or tables for datasets loaded by observation source plug-ins are:

- Julian Day
- Magnitude
- Band

Band is optional. Additional detail can be added to each observation via ValidObservation's `addDetail()` method. For examples of this, see the Kepler/TESS, SuperWASP, and Catalina Sky Survey observation source plug-ins here:

- https://github.com/AAVSO/VStar/tree/master/plugin/src/org/aavso/tools/vstar/external/plugin

Here is an example of an observation data source plug-in that generates a synthetic dataset each time it is selected from the File menu:

- https://github.com/AAVSO/VStar/blob/master/plugin/src/org/aavso/tools/vstar/example/plugin/ExampleObSource.java

In a real plug-in, such observations would come from a file, a web service, an http URL stream, or some database other than the AAVSO International Database. In such cases, `InputType.FILE` or `InputType.URL` would be returned by `getInputType()` rather than `InputType.NONE`. For such input types, VStar will open a dialog for the appropriate user input and pass an input stream and name to the plug-in, accessible via `getInputStream()` and `getInputName()`.

Creating an Observation Sink plug-in

Here is an example of an observation sink plug-in that saves observations in a simple XML format:

- https://github.com/AAVSO/VStar/blob/master/plugin/src/org/aavso/tools/vstar/example/plugin/ObservationSink.java

Creating a Model Creator plug-in

See the following two source files for examples of model creator plug-ins:

1. https://github.com/AAVSO/VStar/blob/master/plugin/src/org/aavso/tools/vstar/external/plugin/FourierModelCreator.java
2. https://github.com/AAVSO/VStar/blob/master/plugin/src/org/aavso/tools/vstar/external/plugin/ApacheCommonsLoessFitter.java

Create an Observation Transformer plug-in

See the following two source files for examples of observation transformation plug-ins:

1. https://github.com/AAVSO/VStar/blob/master/plugin/src/org/aavso/tools/vstar/external/plugin/MagnitudeBaselineShifter.java
2. https://github.com/AAVSO/VStar/blob/master/plugin/src/org/aavso/tools/vstar/external/plugin/VeLaObservationTransformer.java

Dependencies upon Additional Libraries

A non-trivial plug-in may be dependent upon other jar files not known to VStar. These jars must be placed into the vstar_plugin_libs directory for use with VStar. This directory has the same root as the vstar_plugins directory.

In addition, when compiling the plug-in source code, it will be necessary to specify the corresponding jar in the classpath passed to javac. Supposing the jar file in question was called mylib.jar, then your javac command invocation would need to look like this under Windows:

```
javac -cp vstar.jar;mylib.jar my/vstar/plugin/ObservationCounter.java
```

and like this under *nix (Linux, MacOS X, Solaris etc):

```
javac -cp vstar.jar:mylib.jar my/vstar/plugin/ObservationCounter.java
```

the only difference being the semi-colon vs the colon separating the jar files.

Note again that the full path to each jar file must be specified here.

Plug-in Development Tool

See here for a tool to assist in the plug-in development process:

https://github.com/AAVSO/VStar/tree/master/plugin/plugin-dev

Limitations and Constraints

VStar constructs a single instance of each plug-in class at start up.

If a plug-in class has a constructor, it must be public and parameter-less.

When a plug-in object executes within the context of a WebStart-launched VStar, the WebStart sandbox places heavy constraints upon I/O. In that context, only signed jar files may open file dialogs, for example. Observation source plug-ins work around this by telling VStar what kind of input is expected (file, URL). VStar then opens the appropriate dialog box and passes an input stream back to the plugin-in. Further refinements may need to be made to the plug-in base classes to generalize I/O handling.

Closing Remarks

Plug-ins, if redistributed, must strictly be licensed under VStar's license, AGPL 3.0, because they make use of some of VStar's classes. Future consideration may be given to dual-licensing of vstar.jar under LGPL to lift this restriction.

Over time, some plug-in source code may appear in the VStar GitHub repository (by the author and others) if for no other reason than it can be made easily accessible and version controlled in that location. Such code will not be located in the main source tree however, but in a plugin directory at the same level as src.