Title

stata.com

java — Java plugins

Description Remarks and examples References Also see

Description

Java plugins are Java programs that you can call from Stata. When called from Stata, a Java plugin can interact with Stata's datasets, matrices, macros, scalars, and more. Programmers familiar with Java can use Java's extensive language features. There are also many third-party libraries available. If you are not an experienced Java programmer and you intend to implement a Java plugin, you should start by learning Java. Once you become a proficient Java programmer, writing a Java plugin for Stata should be relatively easy.

If you are interested in writing plugins for Stata in another language such as C or C++, see [P] **plugin**.

Remarks and examples

stata.com

A Java plugin is called or executed using Stata's `javacall` command. For a Java plugin to be useful, it needs to have access to a set of Stata's core features. Stata provides Java packages that allow plugins to interact with Stata; refer to Java-Stata API Specification for details. When compiling your Java plugin to use these features, the `sfi-api.jar` file needs to be added to the class-path of your Java compiler. This file is located in `utilities/jar/`, which can be found in the directory where Stata was installed.

Once Java source code has been compiled, a Java plugin can be executed from Stata by bundling your plugin in a JAR file and then placing the JAR file in Stata's ado-path. See [P] **javacall** for examples and additional details about loading plugins.

Java plugins can be redistributed just like any other Stata program. To redistribute your Java plugin through Stata's `net` command, you must bundle your compiled program into a JAR file because `net` copies `.class` files as text instead of binary. Additionally, you should always write a Stata ado-program as a wrapper to `javacall` to parse your syntax. The ado-program's `version` command should always be set to match the Stata version from which you obtained the `sfi-api.jar` file that you compile against.

A typical Java standalone program has an entry point through a `main()` method, which looks like this:

```
static void main(String[] args)
```

To call a Java plugin from Stata, you must define a similar entry point. However, there are two important distinctions. First, you may name your method whatever you like, as long as it conforms to standard Java naming requirements. Second, your method must have a return type of `int` instead of `void`. Here is an example of a valid method signature that Stata can call:

```
static int mymethod(String[] args)
```

Let's assume that `mymethod()` exists and that the compiled Java files have been placed in an appropriate location. To call `mymethod()`, we use Stata's `javacall` command. `javacall` allows you to invoke any static method in the class-path if that method follows the correct signature as described above. For details on class-path behavior, see [P] **javacall**.

References

Crow, K. 2017a. Working with Java plugins (Part 1). *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2017/10/11/working-with-java-plugins-part-1/.

——. 2017b. Working with Java plugins (Part 2). *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2017/10/24/working-with-java-plugins-part-2/.

Drukker, D. M. 2018a. Programming an estimation command in Stata: Preparing to write a plugin. *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2018/02/15/programming-an-estimation-command-in-stata-preparing-to-write-a-plugin/.

——. 2018b. Programming an estimation command in Stata: Writing a Java plugin. *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2018/02/23/programming-an-estimation-command-in-stata-writing-a-java-plugin/.

Also see

[P] **javacall** — Call a Java plugin