



How to Make Statistic Plugins?

Prepared by
Rajmund Bocsi



September 2008, Budapest

Table of content

1	Introduction	3
2	Implementing Statistic Methods	3
3	Implementing the <i>IStatisticsPlugin</i> interface	4
4	Creating the <i>plugins</i> File	8
5	Building and Deploying JAR File	8
6	Advanced options	9
7	Notes	10
8	References	10

1 Introduction

Statistics are the main tools in the Parameter Sweep Wizard to create derived data sources from variables and methods of the selected model. (Please consult the MEME User Manual [2] – *MEME_Manual.pdf* – for further details about statistics.) The statistics shown in the Data Source Editor dialogue are defined by several JAR files known as statistic plugins. This document introduces the user-defined statistic plugins.

You will be shown how to create your own statistic plugins and how they can be used from the Parameter Sweep Wizard.

We will use an example statistic plugin package, called *MyOwnStatistic.jar* to present the creation process (the source code of the plugin package can be found in the *MyOwnStatistic.zip* archive).

Note: Plugin creation is for advanced users. It requires some programming skills in Java.

2 Implementing Statistic Methods

2.1 General

The first task is to implement the desired statistic as a *public static* method. There are some rules to be followed:

- The abovementioned static method must be defined in a *public* class.
- The return type of the method must be one of the following:
 - `byte/Byte`;
 - `short/Short`;
 - `int/Integer`;
 - `long/Long`;
 - `float/Float`;
 - `double/Double`.
- The method should have one or more parameters. The parameter type must be one of the following:
 - `byte/Byte`;
 - `short/Short`;
 - `int/Integer`;
 - `long/Long`;
 - `float/Float`;
 - `double/Double`;
 - `byte[]/Byte[]`;
 - `short[]/Short[]`;
 - `int[]/Integer[]`;
 - `long[]/Long[]`;
 - `float[]/Float[]`;
 - `double[]/Double[]`;

- o `cern.colt.list.DoubleArrayList`¹.

2.2 Example

In *MyOwnStatistic.jar* we create two “new” statistics:

- **my_sum**: returns the sum of the elements of a parameter array.
- **my_sum_if_is_in_interval**: returns the sum of the elements of a parameter array that are between **lower** bound and **upper** bound. The **lower** bound and **upper bound** are two other parameters of the method and they form a closed interval.

So we define a new public class named `MyOwnStatistics` and the two static methods. Below is the source code of the class:

```
package my.own.statistic;

public class MyOwnStatistics {

    public static double my_sum(double[] data) {
        double sum = 0;
        for (double d : data)
            sum += d;
        return sum;
    }

    public static double my_sum_if_is_in_interval(double[] data, double lower,
                                                double upper) {

        double sum = 0;
        for (double d : data) {
            if (d >= lower && d <= upper)
                sum += d;
        }
        return sum;
    }
}
```

3 Implementing the *IStatisticsPlugin* interface

3.1 General

The next step is to make a descriptor class for each statistic. These descriptor classes (which implement the abovementioned `IStatisticsPlugin` interface) are the bridge between the statistic methods and the Parameter Sweep Wizard.

The methods of this interface and its super interfaces provide some basic information about the statistic and it is very easy to implement them.

The *statistics-plugin.jar* file (attached to this document) contains these interfaces. When compiling your source code you must extend the class path with this JAR to ensure that the Java compiler finds the definition of all interfaces. In the *statistics-plugin.zip* archive file (also attached) you can find the sources of these interfaces.

The following list contains methods specified by the `ai.aitia.meme.paramsweep.plugin.IStatisticsPlugin` interface and its super interfaces:

- **String getName()**: [Inherited from `ai.aitia.meme.paramsweep.plugin.IPSScriptPlugin` interface] Returns the simple name of the statistic. The wizard

¹ This type is defined in the Colt project (<http://acs.lbl.gov/~hoschek/colt/>) which provide a lot of statistics.

uses this string during creation of a statistic instance so it must be a valid Java identifier (see Chapter 3 in [1] about Java identifiers).

- ***String getLocalizedName()***: [Inherited from `ai.aitia.meme.pluginmanager.IPlugin` interface] Returns the display name of the statistic. It is shown (see Figure 1) in the *Available statistics* list on the *Data Source Editor* dialogue. While it is not necessary to be unique, we recommend that practice..

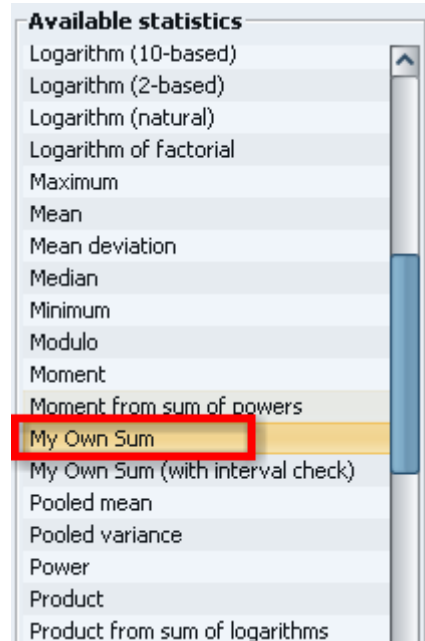


Figure 1 – Localized name of a statistic

- ***String getFullyQualifiedName()***: Returns the fully qualified name of the statistic method. The form of a method fully qualified name is the following: `<fully qualified name of the class that contains the method>.<method name without braces>`. See more about qualified names in [1]. **Implement the method carefully: if you misspell the name your statistic won't work.**
- ***String getDescription()***: [Inherited from `ai.aitia.meme.paramsweep.plugin.IPSScriptPlugin` interface] Returns the short description of the statistic. When selecting a statistic from the *Available statistics* list, its description appears in the *Description* panel (see Figure 2) of the *Data Source Editor* dialogue.

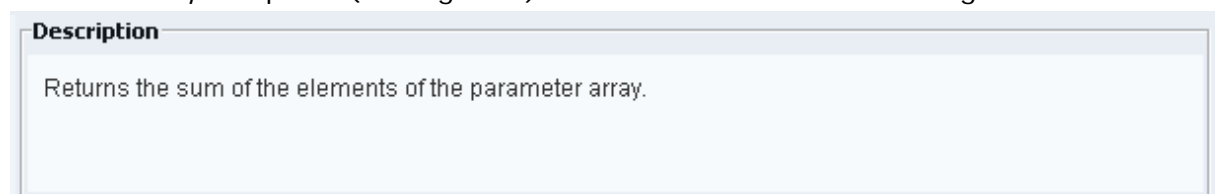


Figure 2 – Description of a statistic

- ***Class getReturnType()***: Provides the return type of the statistic method.
- ***int getNumberOfParameters()***: [Inherited from `ai.aitia.meme.paramsweep.plugin.IPSScriptPlugin` interface] Returns the number of parameters of the statistic method.
- ***List<String> getParameterNames()***: Returns a list that contains the display names of formal parameters of the statistic. These names are shown on the *Data Source Editor* dialogue next to the GUI element where you can specify the actual value of the formal parameter (see Figure 3 – Parameter names). **The size of the result list must be equal to the return value of the `getNumberOfParameters()` method described above.**

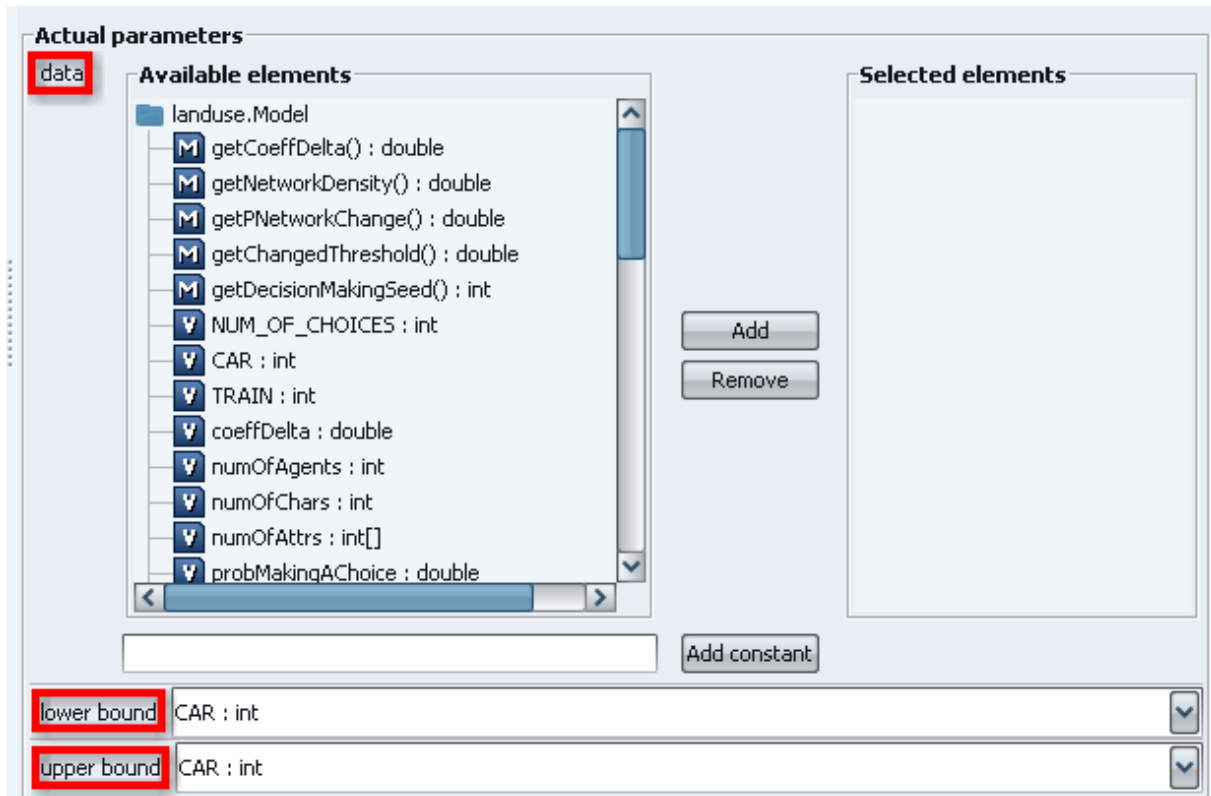


Figure 3 – Parameter names

- *List<Class>* `getParameterTypes()`: Returns a list that contains the types of formal parameters of the statistic method. **The size of the result list must be equal to the return value of the `getNumberOfParameters()` method described above.**
- *List<String>* `getParameterDescriptions()`: Returns a list that contains the short descriptions of formal parameters of the statistic. Each description appears as a tool-tip text when you move the cursor over the name of a formal parameter. **The size of the result list must be equal to the return value of the `getNumberOfParameters()` method described above.**

3.2 Example

We defined two statistic methods previously, so we have to create two descriptor classes. First we define the descriptor class of the `my_sum()` method. Here is the source:

```
package my.own.statistic;

import java.util.Arrays;
import java.util.List;

import ai.aitia.meme.paramsweep.plugin.IStatisticsPlugin;

public class MyStatistics_Sum implements IStatisticsPlugin {

    public String getName() { return "my_sum"; }

    public String getLocalizedName() { return "My Own Sum"; }

    public String getFullyQualifiedName() {
        return "my.own.statistic.MyOwnStatistics.my_sum";
    }

    public String getDescription() {
        return "Returns the sum of the elements of the parameter array.";
    }
}
```

```

public Class getReturnType() { return Double.TYPE; }

public int getNumberOfParameters() { return 1; }

public List<String> getParameterNames() {
    return Arrays.asList(new String[] { "data" });
}

public List<Class> getParameterTypes() {
    return Arrays.asList(new Class[] { double[].class });
}

public List<String> getParameterDescriptions() {
    return Arrays.asList(new String[] { "Operands of the sum." });
}
}

```

Then we create the descriptor class for the `my_sum_if_is_in_interval()` method:

```

package my.own.statistic;

import java.util.Arrays;
import java.util.List;

import ai.aitia.meme.paramsweep.plugin.IStatisticsPlugin;

public class MyStatistics_SumIf implements IStatisticsPlugin {

    public String getName() { return "my_sum_if_is_in_interval"; }

    public String getLocalizedName() {
        return "My Own Sum (with interval check)";
    }

    public String getFullyQualifiedName() {
        return "my.own.statistic.MyOwnStatistics.my_sum_if_is_in_interval";
    }

    public String getDescription() {
        return "Returns the sum of the elements of the parameter array " +
            "that are between 'lower bound' and 'upper bound'.";
    }

    public Class getReturnType() { return Double.TYPE; }

    public int getNumberOfParameters() { return 3; }

    public List<String> getParameterNames() {
        return Arrays.asList(new String[] { "data",
            "lower bound",
            "upper bound"
        });
    }

    public List<Class> getParameterTypes() {
        return Arrays.asList(new Class[] { double[].class,
            Double.TYPE,
            Double.TYPE
        });
    }

    public List<String> getParameterDescriptions() {
        return Arrays.asList(new String[] { "Operands of the sum.",
            "The lower bound of the closed interval.",
            "The upper bound of the closed interval."
        });
    }
}

```

4 Creating the *plugins* File

4.1 General

The third step is to create a simple text file named *plugins* (without any extension). Every line of this file specifies the fully qualified name of a descriptor class (which implements the `IStatisticsPlugin` interface). The plugins file contains as many rows as the number of plugins in the plugin package.

4.2 Example

In our example, the *plugins* file contains two lines:

```
my.own.statistic.MyStatistics_Sum
my.own.statistic.MyStatistics_SumIf
```

5 Building and Deploying JAR File

5.1 General

The next task is to compile the classes and then create a JAR file that contains all required classes. Required classes are:

- The class that defines the statistic methods;
- The descriptor classes;
- Other classes referred from the above-mentioned classes (except the ones bundled either into the JRE, *colt.jar*, *repast.jar*, *statistics-plugin.jar*).

The JAR file must also contain the *plugins* text file at the root of the JAR file.

The final step is to place the created JAR file into the *Plugins* subdirectory of MEME's installation folder². **You must restart the MEME to use the new plugins.**

5.2 Example

First we have to compile the classes created before. Let's assume we are in the project directory (here is the plugins file and statistics-plugin.jar too). We can compile the three classes with the following command:

```
javac -cp ./statistics-plugin.jar my/own/statistic/MyOwnStatistics.java
my/own/statistic/MyStatistics_Sum.java my/own/statistic/MyStatistics_SumIf.java
```

Next we create the JAR file. We don't use any third party classes, so the JAR file must contain only the *MyOwnStatistics.class*, the *MyStatistics_Sum.class*, the *MyStatistics_SumIf.class* and the *plugins* file. This can be achieved with the following command:

```
jar cvf MyOwnStatistic.jar my plugins
```

After that all we need to do is to copy the *MyOwnStatistic.jar* to the *Plugins* subdirectory of MEME's installation folder. Then you need to start MEME and in the *Data Source Editor* dialogue of the Parameter Sweep Wizard both new statistics appear in the *Available statistics* list (see Figure 4).

² By default this is the c:\Program Files\MASS\MEME\ directory on Windows.

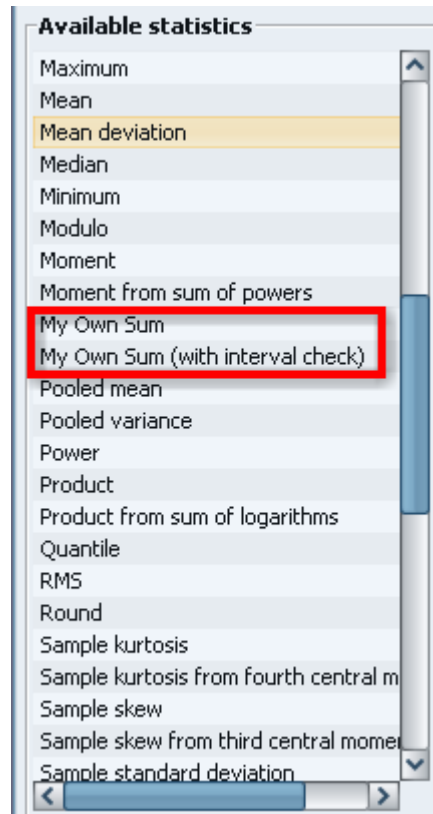


Figure 4 – The two new statistics in the list

6 Advanced options

6.1 About plugin instantiation

Let's denote by P the class name of the plugin (which appears in the *plugins* file). When loading the plugin, this class is instantiated as follows:

- if the following method exists in P , it is called:

```
public static P onPluginLoad()
```

This method should instantiate the plugin and returns a reference to it. Returning `null` means that the plugin should be ignored (the program will not know about it). A plugin can use this mechanism, for example to check whether required classes are available in the application (say, a class from an other user-defined statistic package). and make the plugin completely invisible (unavailable to the user) when it is inappropriate.

- otherwise — by default — the plugin is instantiated by calling the default constructor of the plugin's class.

6.2 About plugin destructing

Before the Parameter Sweep Wizard is closed the following method of the plugin's class is called:

```
public static void onPluginUnload()
```

This is optional: if this method doesn't exist, the plugin gets no notification before exiting.

This mechanism can be used, for example, to save user settings, release external resources (like channels, temporary files, spawned processes) etc.

7 Notes

Please note that there are statistics in the *Data Source Editor* of FABLES IME, too. The JAR archive described above can also be used there without any modification. However the process of deployment is different. Please consult with the *How to Make Statistic Plugins for FABLES IME?* document for further details.

8 References

- [1] The Java Language Specification
<http://java.sun.com/docs/books/jls/>
- [2] MEME User Manual
<http://mass.aitia.ai/>