JHawk 6 Documentation - JHawk Data Viewer - User Manual

Virtual Machinery
February 2020

Overview

Since its release in 1999 JHawk has been a leader in the provision of Software Metrics to Java developers. Originally released as a stand-alone application it was subsequently integrated with Visual Age for Java and has now evolved to include a command line version. Functionality has also been added over time to allow the export of the metrics gathered by JHawk in CSV, HTML and XML format.

JHawk has proved itself in many different areas and its customers have reflected that – from Fortune 500 companies to Academic institutions, from Banking to Telecommunications companies and across the globe from Norway to Brazil and from the US to China.

The JHawk Data Viewer provides the user with a number of different comparative views of data collected by JHawk over the lifetime of a project. This is done using the XML files exported by JHawk (version 5 and above only). This is an extremely powerful tool unique in the Java Software metrics area.

Each stage in the lifetime of the project is represented by a single XML file containing the basic data (see the definition in the JHawk manual) collected on the Java source at that time. Each of these stages is known as a build.

Once the builds have been selected and analysed these can be viewed in the display tabs.

Documentation

The following documents are provided with the distribution (depending on which license you have purchased). They are in PDF format and are located in the 'docs' directory of the distribution –

Document	Personal	Professional	Starter	Demo
JHawk6CommandLineManual – Documentation for the Command line version of JHawk	No	Yes	No	Yes
JHawk6CreateMetric – Documentation explaining how to create new metrics that can be added to JHawk	Yes	Yes	No	No
JHawk6DataViewerManual – Documentation for the JHawk DataViewer product	No	Yes	No	Yes
JHawk6Licensing – Licensing details for JHawk products	Yes	Yes	Yes	Yes
JHawkStarterManual – Documentation for the JHawk Starter edition	No	No	Yes	No
JHawk6UserManual – Documentation for the JHawk standalone application	Yes	Yes	No	Yes
JHawk6UsingMetrics – Documentation outlining how to get the best from JHawk and a list of the metrics implemented by JHawk with details of their calculation.. It also includes an introduction to the area of Java code metrics.	Yes	Yes	Yes	No

Installing JHawk Data Viewer

In the JHawk Distribution you will find the JHawkDataViewer.jar. You can locate this jar anywhere as long as you have a jhawk.properties file with it. You can then run the JHawk Data Viewer either by double clicking on the jar or starting it from the command line (see section below – 'Running the Data Viewer from the Command Line').

You can use any jhawk.properties file that you have created and/or modified using the JHawk stand alone application.

If you have created additional metrics that you want to use in the Data Viewer you will need to have these metrics available to the Data Viewer in the jhawk.properties file and in the CustomMetrics.jar. The CustomMetrics.jar must be in the same directory as the JHawkDataViewer.jar.

As part of the distribution you will notice the following properties files –

- jhawkbase.properties
- jhawkfull.properties
- jhawkbasepluscustom.properties

jhawkbase.properties is a version of the properties file with the base level set of metrics that are initially enabled for JHawk. This is the same as the jhawk.properties file that comes with the JHawk distribution.

jhawkfull.properties is a version of the properties file with the full set of metrics available to JHawk enabled.

jhawkbasepluscustom.properties is a version of the properties file with the base level set of metrics that are initially enabled for JHawk plus the sample metric found in the CustomMetrics.jar that comes with the distribution. For more details about this metric see the 'JHawk5CreateMetric' document.

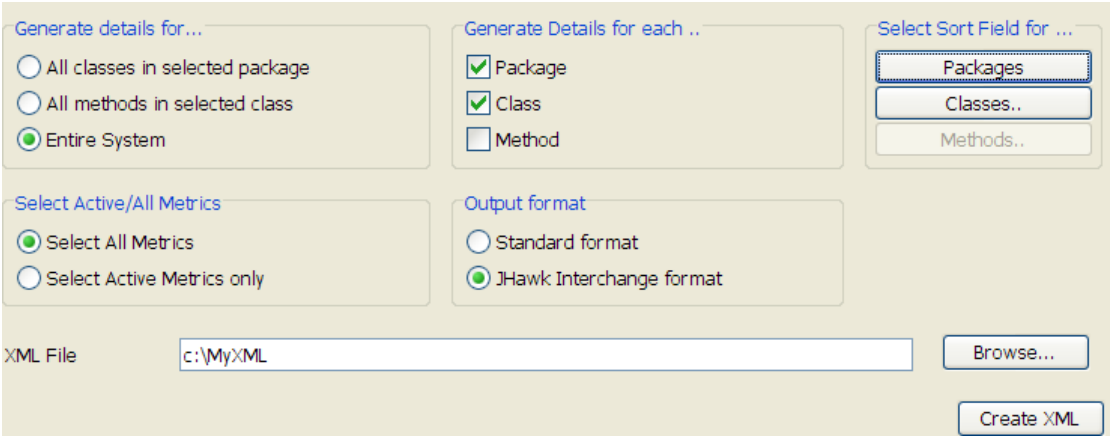
You can use any of these sets of metrics by copying them to the jhawk.properties file and starting as normal or by using the –p flag and the property file name (JhawkCommandLine only).

Properties loaded from these files can be amended in the usual way – see 'Preferences' section in the documentation.

Creating files for the Data Viewer

The files used by the Data Viewer must be in the JHawk Metric Interchange Format (JMIF). These are XML files that have been created using the JMIF option when exporting files from JHawk either using the stand alone application or the command line interface.

In the JHawk stand alone application you must first analyse the Java files, then go to the Export tab and select the 'Create XML' sub tab. On this tab you need to select the 'Entire System' and the 'JHawk Interchange Format' radio buttons. You then need to select the levels that you wish the analysis to occur at – Package, Class or Method. If you want to analyse to a particular level you will need to select the levels above so that the JHawk Data Viewer (which works on a tree basis) can 'drill down' to the lower levels. This means that if you want to see data down to the method level you will need to select the package, class and method levels and if you want to analyse to the class level you will need to select the package and class levels.



If you are using the JHawk Command line you will need to set the XML flag (-x), the JHawk Metric Interchange Format flag (-b) and the Level flag (-l). If you want to analyse to a particular level you will need to select the levels above so that the JHawk Data Viewer (which works on a tree basis) can 'drill down' to the lower levels. This means that if you want to see data down to the method level you will need to select the package, class and method levels (-l pcm) and if you want to analyse to the class level you will need to select the package and class levels (-l pc). The command line below is equivalent to the selections made in the JHawk stand alone screenshot shown above.

```
-f .*\.java -r -b -l pc -s C:\mySource -x C:\MyXML -n MySource
```

The following Ant script will have the same effect –

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project basedir="." default="jhawkant" name="Task">
    <target name="jhawkant">
      <taskdef name="jhawktask"
         classname="com.virtualmachinery.jhawk.ant.JHawkAntTask"
         classpath="JHawkCommandLine.jar"/>
          <jhawktask
              filepattern=".*\.java"
              recursive="true"
              basiconly="true"
              outputlevels="pc"
              startpath="C:\ mySource"
              xmlfilename="C:\MyXML"
              namelevels="MySource" />
    </target>
</project>
```

In both cases the level to which you wish to carry out analysis will have a bearing on the size of the XML files created. This, in turn, may limit the number of files that JHawk Data Viewer can handle at a given level of memory. In practice there is little point in analysing down the method level, package and class levels are usually adequate for most purposes. As an example the XML file sizes for an analysis of the entire source for Eclipse 3.4 (16,175 source files) were 215Mb at method level, 40Mb at class level and 134k at package level.

If you wish to compare files in the tree views (Text Compare and Graph) then all of your systems should have the same name (or all have no name). You can set the system name in the 'Select Files' tab in the JHawk stand alone application. In the command line version you should use the –system flag followed by the name that you wish to give the system. The default value of the System name is 'No System Name'.

Data Viewer – A quick Start

How you start the Data Viewer application will depend on your environment. Data Viewer has been tested in a large number of different Java environments and should run in any environment that supports Java 1.5 and above.

Start up Data Viewer

Start up Data Viewer in a Windows environment

- When you look in the directory that you installed the JHawk application you will find a directory called code and in that directory you will find a jar file called JHawkDataViewer.jar Double clicking on this should start the application. If it does not start then you will have to start the application from the command line – see the section 'Starting JHawk from the command line'.

Start up Data Viewer in a Mac Environment

- When you look in the directory that you installed the JHawk application you will find a directory called code and in that directory you will find a jar file called JHawkDataViewer.jar. Double clicking on this should start the application. If it does not start then you will have to start the application from the command line – see the section 'Starting the JHawk Data Viewer from the command line'.

Start up Data Viewer in a Unix Environment

- Startup will depend on the Unix environment. In some cases you may be able to double click on the JHawk Data Viewer jar and in others you may have to start the Data Viewer from the command line. In the latter case you should see the section – 'Running the Data Viewer from the Command Line'.

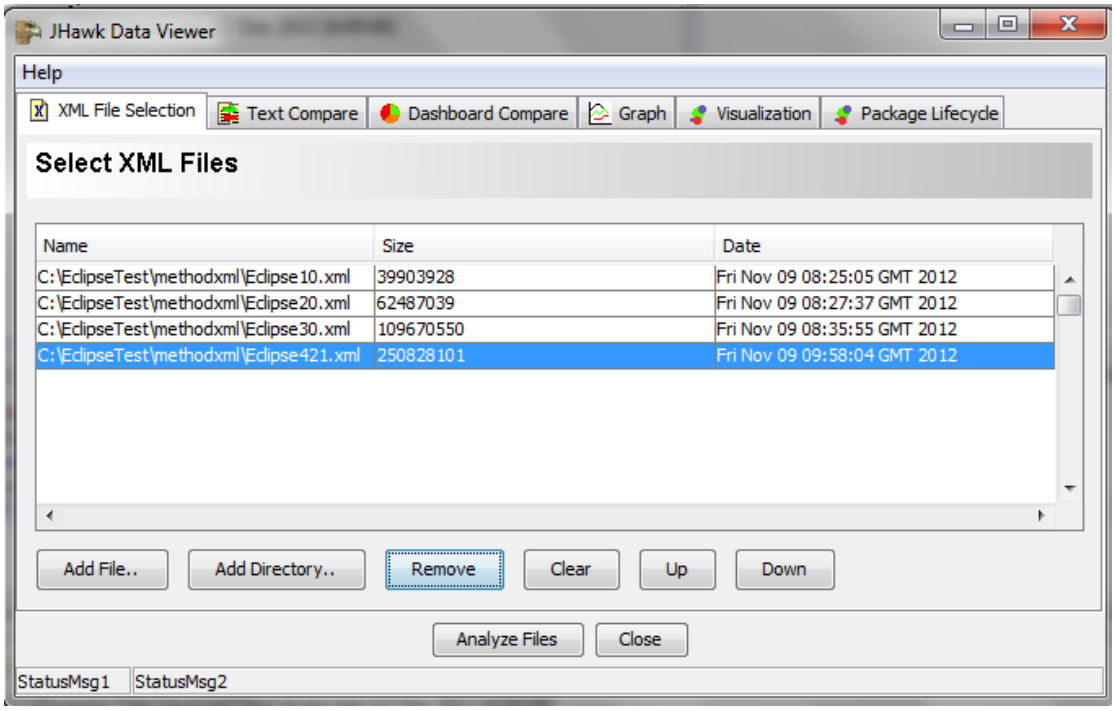
Note that if you want to start the Data Viewer with more memory than is provided in the standard Java jar startup you should use the command line. If you are using a large number of files or even a small number of large files you may have to increase the memory available to Data Viewer considerably. For more details see the section – 'Running the Data Viewer from the Command Line'.

Lets get going

Once the Data Viewer has been started there are two possible ways in which you might choose to view your data –

- **Using the full facilities of Data Viewer**. In this case you select the XML files on the opening screen, press the 'Analyze Files' button, wait for the analysis to complete and then view the data using the subsequent tabs.
- **Using Data Viewer to create Google Visualizations**. In this case you select the files to be used in the visualizations but you don't need to analyse them as this will be done at the time the visualization is created. As well as saving time there is a further advantage as one XML file is processed at a time – reducing the memory burden on the system.

The opening screen



When the Data Viewer application starts up select the files using the 'Add File'/'Add Directory' buttons to create the list of files that you want to analyse. You will probably want to analyse these files in a specific order and you can use the 'Up' and 'Down' buttons to set the order. If you don't want to analyse some of the files that you have selected you can remove them with the 'Remove' button. You can find out more about this tab in the 'XML File Selection Tab' section below.

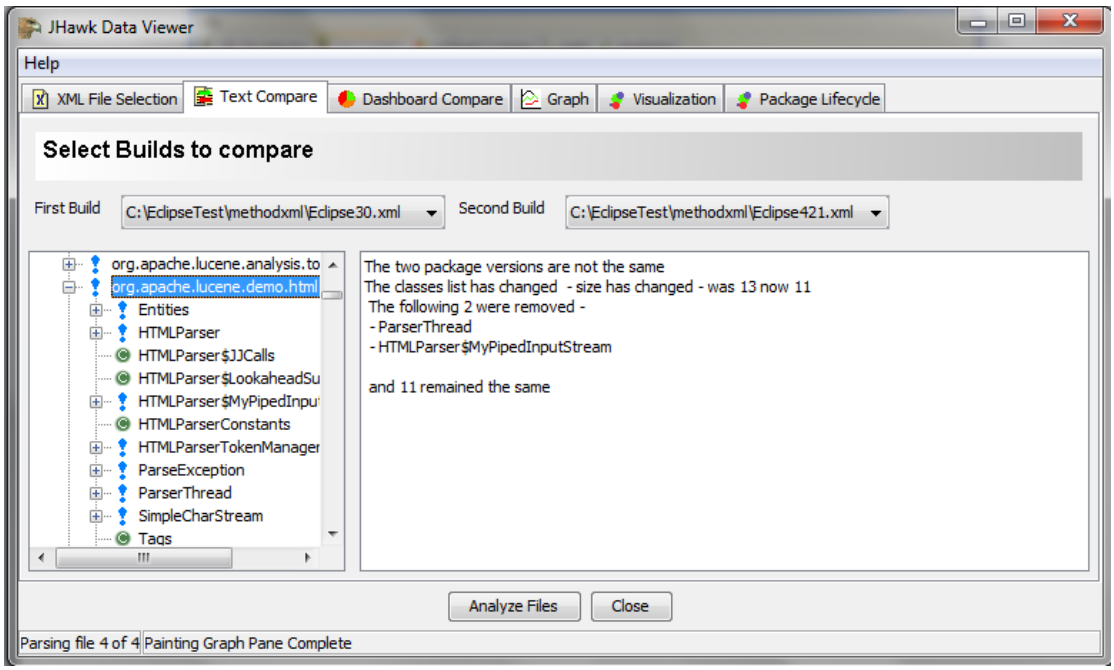
If you just want to create a Google Visualization you should now skip forward to the 'Visualization' section below.

If you are going to use the other tabs on the Data Viewer then you need to analyse the files then you need to press the 'Analyse Files' button. When the status pane along he bottom of the application shows 'Painting Graph Pane Complete' and the other tabs have been reactivated you can view the data in the other tabs.

If you are analyzing a large number of files you may wish to alter the amount of memory available to the Data Viewer. You can do this by running it from the command line – see the section – 'Running the Data Viewer from the Command Line'.

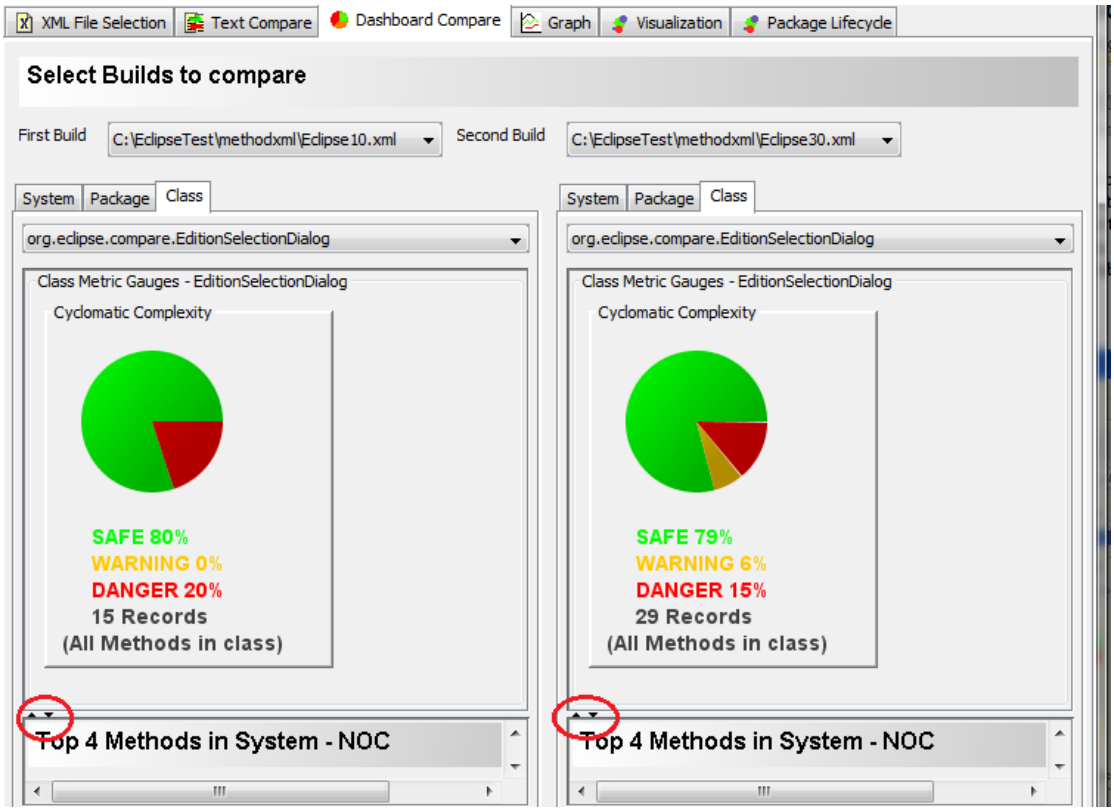
The Text Compare Tab

The Text Compare Tab shows the differences between any two builds in text format. The differences can be viewed at all levels from System down to Method. You select the two builds to be compared in the drop down boxes at the top. By default the first and last builds are selected and the differences at System level are shown. You can find out more about this tab in the main 'Text Compare Tab' section below.



Dashboard Compare

The dashboard compare tab allows you to select two builds and compare them using the dashboard style tabs used in JHawk. By default the first and last builds are selected and the differences at System level are shown. The Package and Class sub-tabs allow you to select the package or class whose data you wish to view. If you select a package or class in one tab the data for this will be selected in the other tab if the data is available – if the data is not available then the other tab will remain on the package or class that is currently selected on the tab.

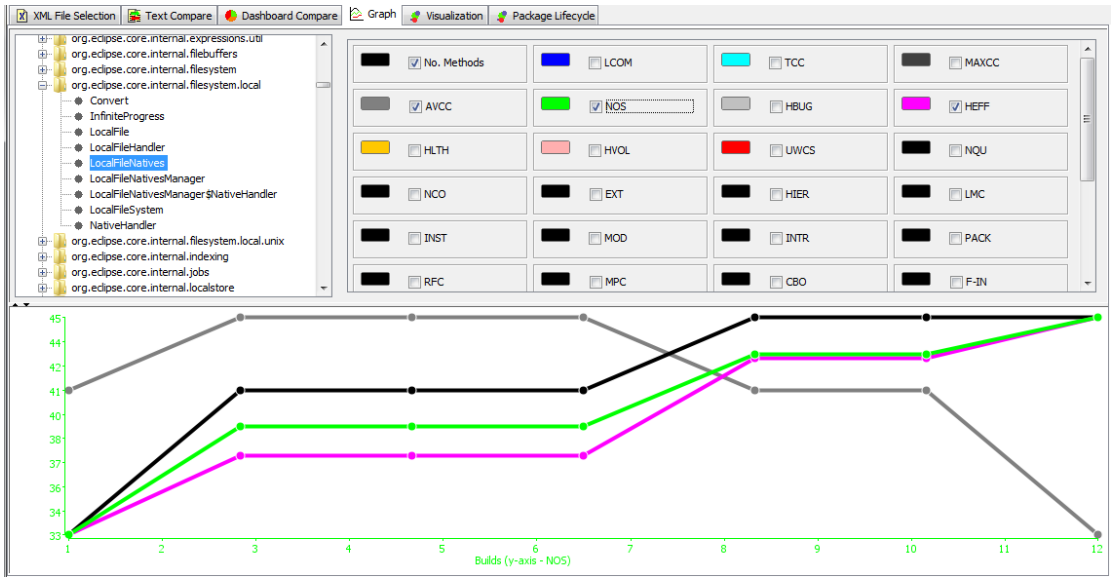


The dashboard tabs contain an upper and a lower panel. Gauges are displayed in the top panel and if any dashboard tables have been defined at that level then they will be displayed in the bottom panel. You can switch the sizes of the panels by using the One-Touch icons ringed in the image above.

You can find out more about this tab in the main ‘Dashboard Compare Tab’ section below.

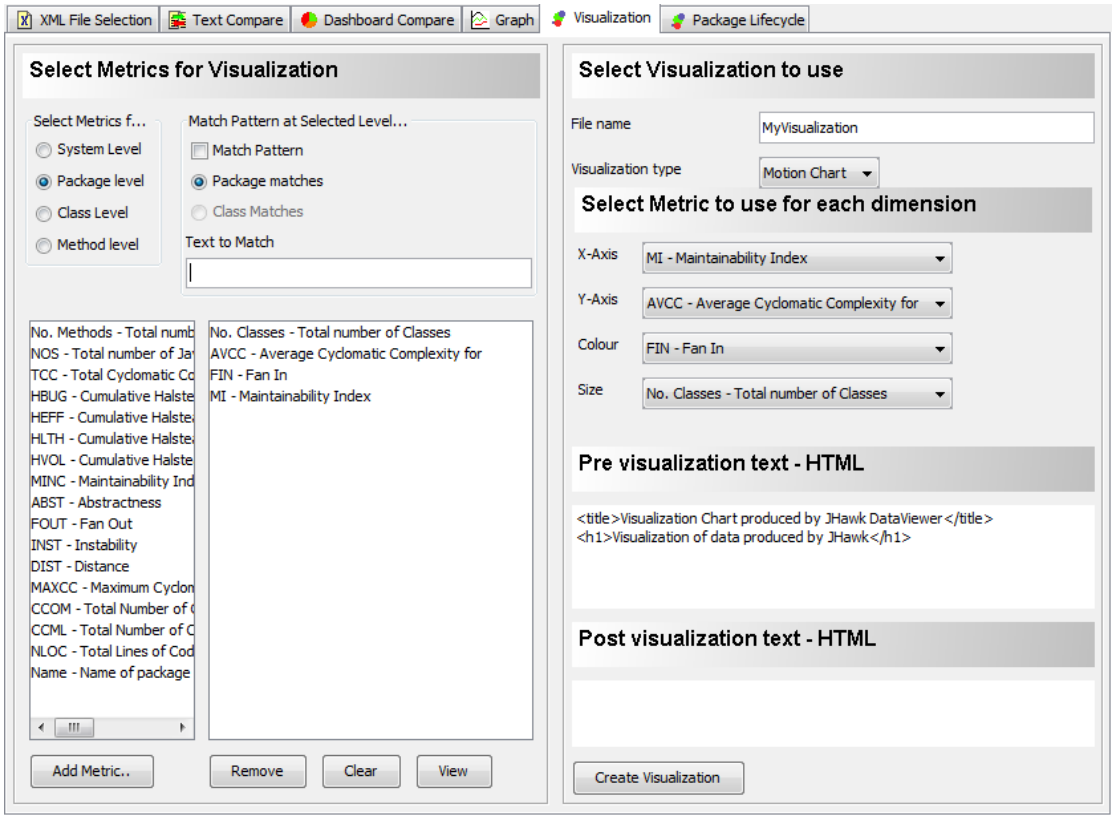
Graph Tab

The Graph tab gives a graphical representation of the changes in data over the builds selected for analysis. The Graph and the available metrics will change according to the level selected in the top left panel. Data will be displayed at System, Package, Class and Method levels. For more details see the ‘Graph Tab’ section below.



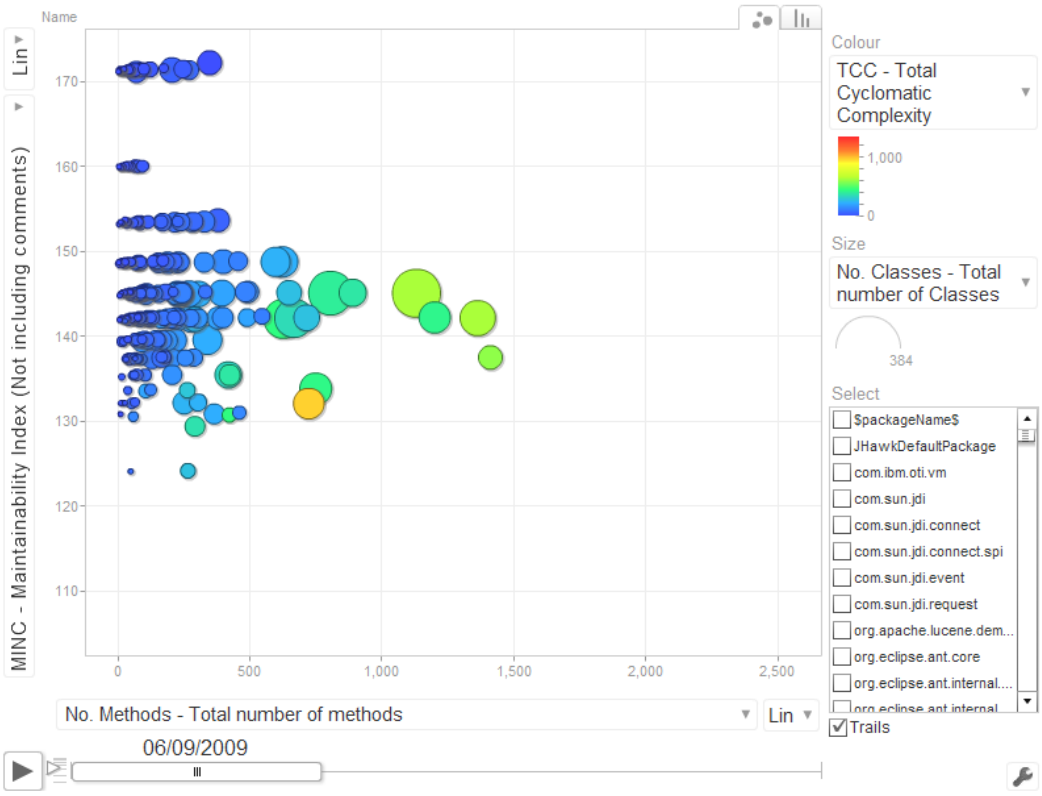
Visualization Tab

The Visualization Tab allows you to select data for display using Googles Visualization tools. .At present both the Motion Chart and Data Table visualizations are available.



When you have created the HTML you can then view it using your browser. Depending on the amount of data in the visualization it may take some time to load. You may also get an error indicating a Security Violation. If this should happen look at the section marked ‘Security Violation’ in the full section on the Visualization Tab below.

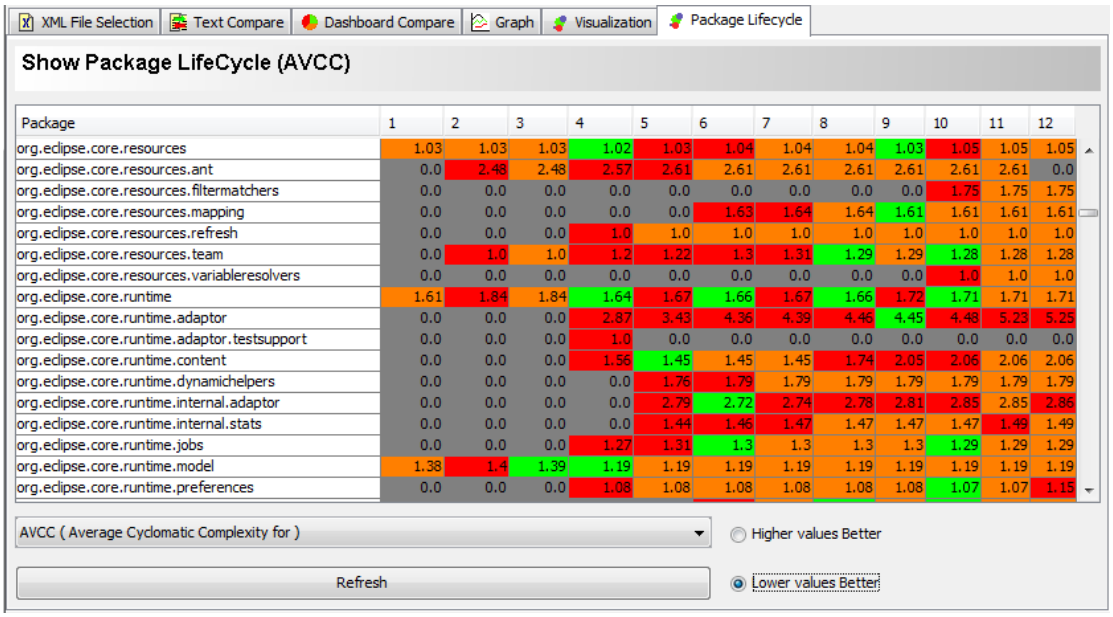
Visualization of data produced by JHawk



To find out more about the Visualization Tab see the ‘Visualization Tab’ section below.

Package Lifecycle Tab

The Package Lifecycle tab gives you a package level overview of which metrics have changed at package level in each build.

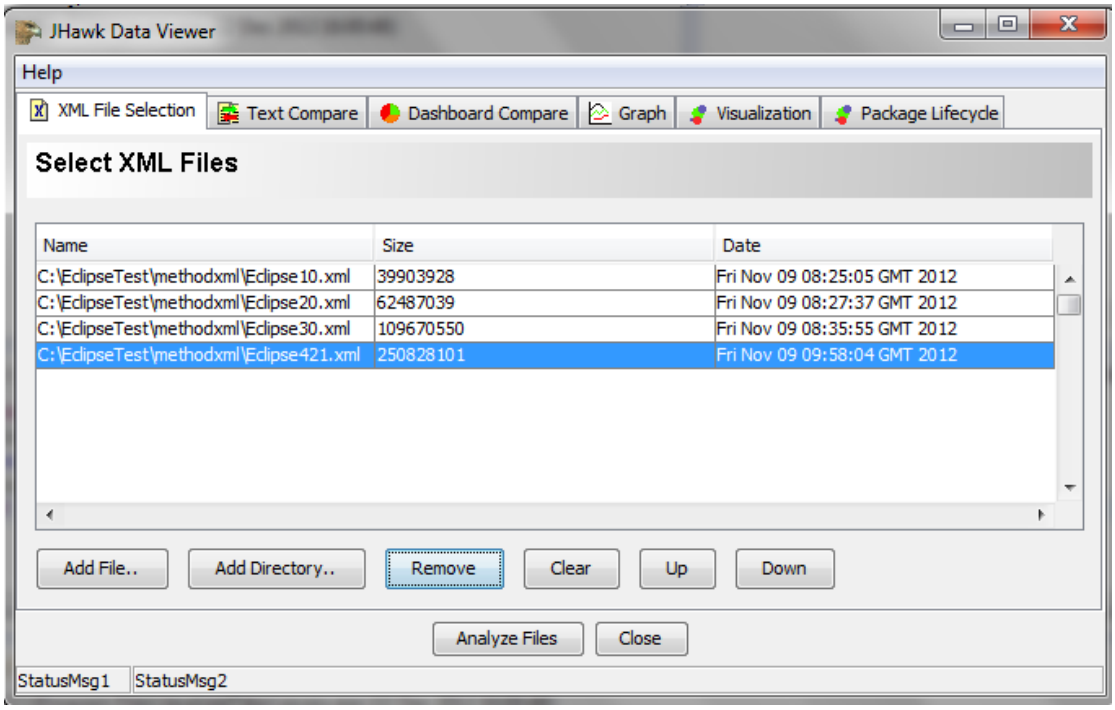


Package	1	2	3	4	5	6	7	8	9	10	11	12
org.eclipse.core.resources	1.03	1.03	1.03	1.02	1.03	1.04	1.04	1.04	1.03	1.05	1.05	1.05
org.eclipse.core.resources.ant	0.0	2.48	2.48	2.57	2.61	2.61	2.61	2.61	2.61	2.61	2.61	0.0
org.eclipse.core.resources.filtermatchers	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.75	1.75	1.75
org.eclipse.core.resources.mapping	0.0	0.0	0.0	0.0	0.0	1.63	1.64	1.64	1.61	1.61	1.61	1.61
org.eclipse.core.resources.refresh	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
org.eclipse.core.resources.team	0.0	1.0	1.0	1.2	1.22	1.3	1.31	1.29	1.29	1.28	1.28	1.28
org.eclipse.core.resources.variableresolvers	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0
org.eclipse.core.runtime	1.61	1.84	1.84	1.64	1.67	1.66	1.67	1.66	1.72	1.71	1.71	1.71
org.eclipse.core.runtime.adaptor	0.0	0.0	0.0	2.87	3.43	4.36	4.39	4.46	4.45	4.48	5.23	5.25
org.eclipse.core.runtime.adaptor.testsupport	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
org.eclipse.core.runtime.content	0.0	0.0	0.0	1.56	1.45	1.45	1.45	1.74	2.05	2.06	2.06	2.06
org.eclipse.core.runtime.dynamichelpers	0.0	0.0	0.0	0.0	1.76	1.79	1.79	1.79	1.79	1.79	1.79	1.79
org.eclipse.core.runtime.internal.adaptor	0.0	0.0	0.0	0.0	2.79	2.72	2.74	2.78	2.81	2.85	2.85	2.86
org.eclipse.core.runtime.internal.stats	0.0	0.0	0.0	0.0	1.44	1.46	1.47	1.47	1.47	1.47	1.49	1.49
org.eclipse.core.runtime.jobs	0.0	0.0	0.0	1.27	1.31	1.3	1.3	1.3	1.3	1.29	1.29	1.29
org.eclipse.core.runtime.model	1.38	1.4	1.39	1.19	1.19	1.19	1.19	1.19	1.19	1.19	1.19	1.19
org.eclipse.core.runtime.preferences	0.0	0.0	0.0	1.08	1.08	1.08	1.08	1.08	1.08	1.07	1.07	1.15

To find out more about the Package Lifecycle Tab see the ‘Package Lifecycle Tab’ section below.

The XML Selection Tab

On starting the data viewer the user can select the XML files to be analysed.



When you have completed this screen you will have an ordered list of XML files that can be used for further analysis, either within the Data Viewer itself or to prepare the data for viewing using the Google visualizations API.

Initial File Selection

You can select files individually by clicking on the 'Add File..' button. This will bring up a file dialog which will allow you to browse to the location of the file that you wish to select. The file will then be added to the list on screen.

You can also select all the XML files in a particular directory structure by clicking the 'Add Directory..' button. This will bring up a File dialog which will allow you to browse to the location of the directory the directory that you wish to select. After you have selected the directory all of the XML files in that directory and the consequent sub-directories of that directory will be added to the list. Remember that all XML files in the directory tree will be added, not just those that have been created by JHawk.

Once you have selected the XML files you can remove any that you don't want by clicking the 'Remove' button.

If you wish to start the file selection process again you can press the 'Clear' button.

Ordering the files

The files will be analysed in the order that they appear in the selection list, each file being taken as a different build with the file at the top of the list being taken as the file relating to build number 1 and so on down the list. You can move files up and down the list by selecting the file entry in the list and clicking the 'Up' or 'Down' buttons.

Completing the analysis

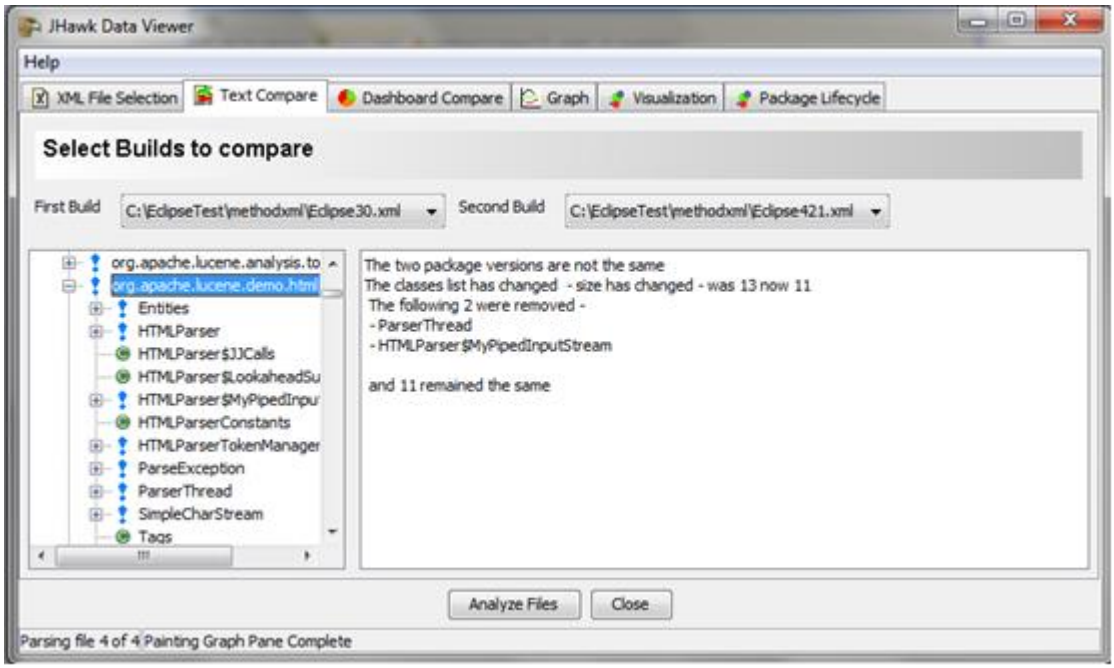
If you only wish to create a Google visualization file then you can now proceed to the 'Visualization' tab and carry on from there.

If you wish to view the results in the Data Viewer you will need to press the 'Analyze Files' button and wait for the analysis to complete. As the analysis progresses the status bar at the bottom of the screen will be updated. When the analysis is completed the status bar will read 'Painting Graph Pane Complete' and all of the tabs visible on the main screen will now be active. You can now view the files in the Data Viewer and output the data to a Google Visualization file should you wish.

If you are analysing a large number of files you may wish to alter the amount of memory available to the Data Viewer. You can do this by running it from the command line – see the section – 'Running the Data Viewer from the Command Line'.

Text Compare Tab

On the Text Compare tab you can view the changes between each build e.g. methods added, removed, changed; variables added, removed, changed, class references etc. You can view these changes at the System, Package, Class and Method level.



The Text Compare Tab has three main panels. The top panel allows you to select the builds to be compared. Two drop-down lists are provided each of which lists all of the build files analysed. The lower left panel shows a tree list of the data available. This allows data to be viewed at System, Package, Class and Method level. The lower right panel is a text box describing the differences between the builds at the chosen level.

The Text Compare Tab opens with the root (System) level selected and the first and last builds in the list of builds selected.

The text display opens with the timestamps on the builds selected and a statement stating whether the two builds are the same at this level. If there are differences then these are described in the subsequent text. A standard notation is used for lists that have changed – the difference in the size of the lists and the number of items added or removed is stated followed by the items added (preceded by a '+') or removed (preceded by a '-').

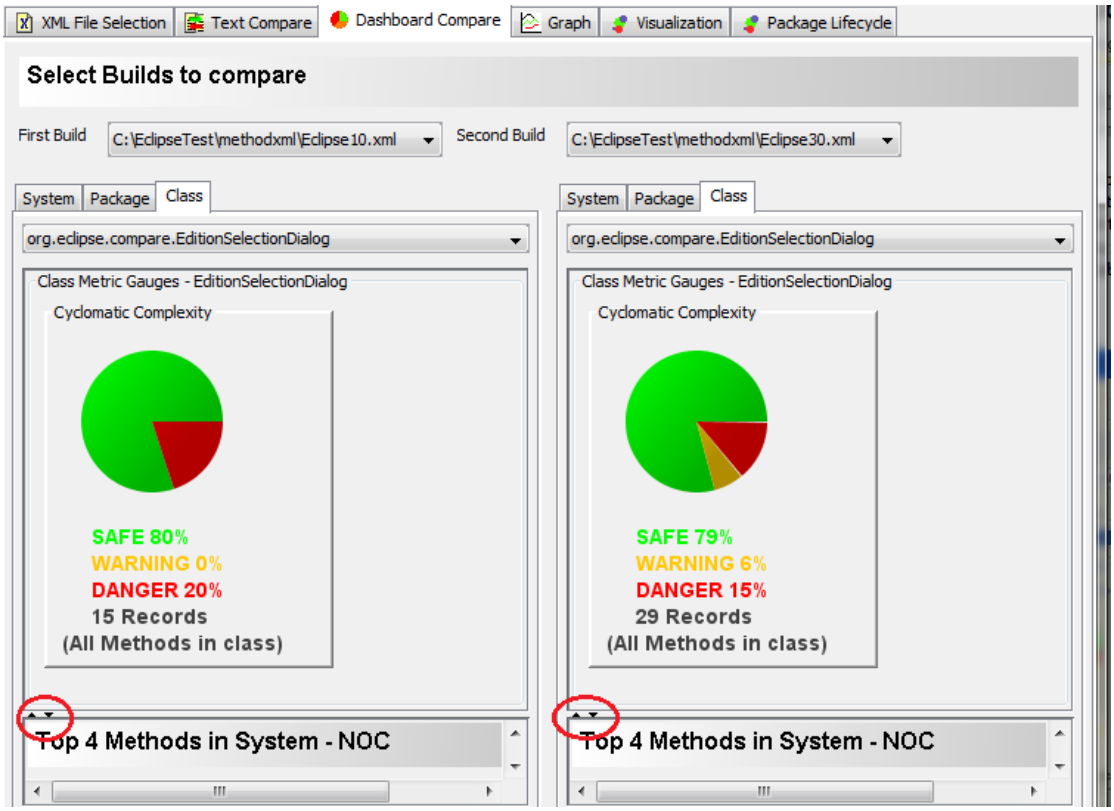
In the Tree the blue exclamation mark indicates that something has changed at this level or below. It might mean that the artefact (or one of its sublevels) has changed or it might indicate that a new code artefact has been added or removed at this level. An unchanged package will be indicated by a yellow 'P' symbol, an unchanged class by a green 'C' symbol and an unchanged method by a grey 'M' symbol.

Within the package org.apache.lucene.demo.html we can see that the class HTMLParserConstants is unchanged as it is marked with a Green C symbol.

By navigating the tree on the left to different levels, and different items within that level, the text in the right hand pane will be updated. As we can see above the package org.apache.lucene.demo.html has been highlighted in the tree and details relating to the changes in that package are indicated in the text screen. The text screen shows that 2 classes have been removed from the package between the two versions under analysis. Remember that the differences described relate only to items associated with the metrics collected and do not necessarily related to differences in the code.

Dashboard Compare Tab

On the dashboard compare tab you can compare changes in key metrics between each build using the same dashboard style tabs used in JHawk. You can view these changes at the System, Package, Class and Method level.



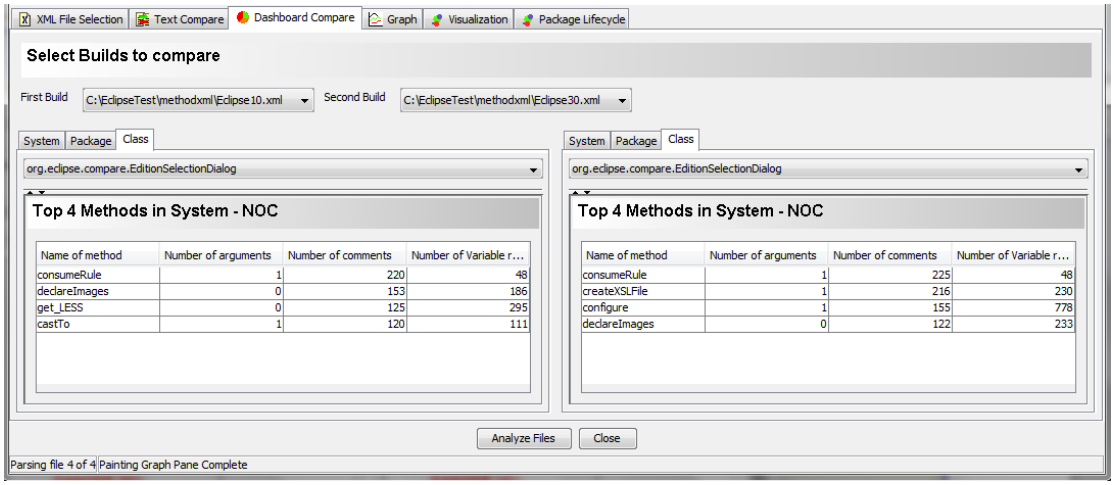
The Dashboard Compare Tab has three main panels. The top panel allows you to select the builds to be compared. Two drop-down lists are provided each of which lists all of the build files analysed. The left and right panels below show the dashboards for each build. As in the JHawk product itself each of these panels has three sub-tabs which allow the metrics selected for the dashboard at System, Package and Class method to be shown. As you switch tabs on one side of the screen the corresponding tab on the other side will be selected, keeping your comparisons synchronized at all times.

The Dashboard Compare Tab opens with the root (System) level selected and the first and last builds in the list of builds selected.

On the Package and Class tabs there is a drop down list of the Packages or Classes. Selecting a Package or Class in one list will cause it to be selected in the other list (if the Package or Class is available in both builds). If the Package or Class is not available in the other list then a message will be displayed indicating this.

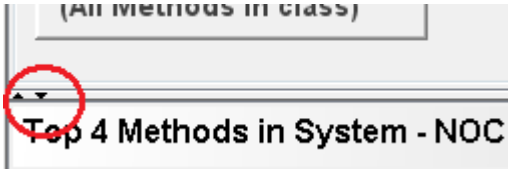
Within each Dashboard sub tab there are two panes – the upper pane contains the Gauges that you have defined in your JHawk properties and the lower pane contains the dashboard tables that you have defined in your properties. This is exactly the same as you see in the standalone version of JHawk.

In this case a ‘Number of Comments (NOC) table has been defined at the class level –



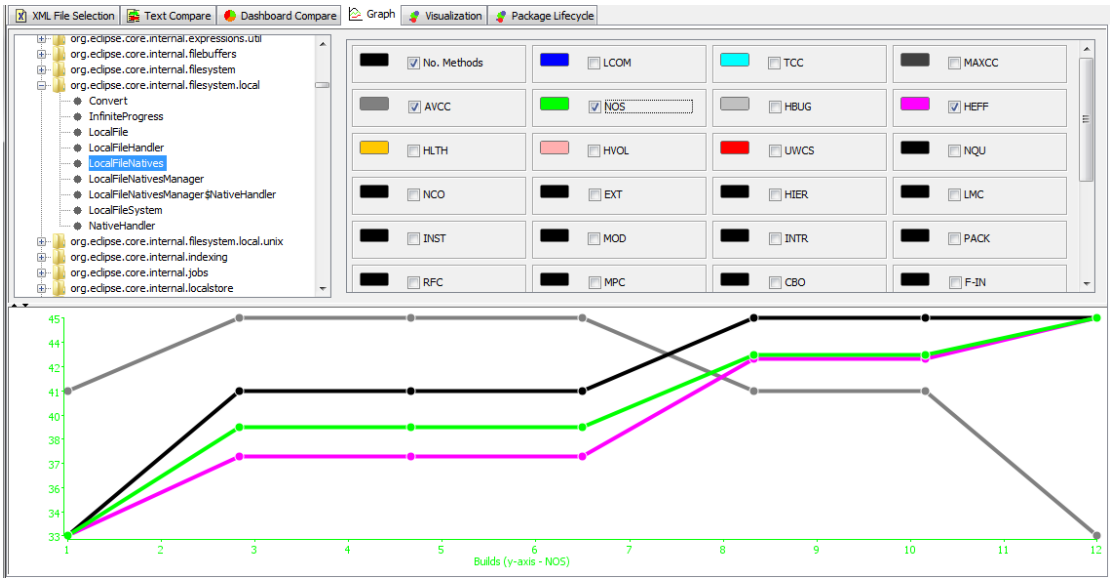
In the screenshot shown above the 4 methods in the system with the greatest number of Comments (NOC) are shown.

You can choose to display only one of the panels by using the One-Touch icon at the left hand side of the dividing bar between the panels –



Graph Tab

The Graph tab gives a graphical representation of the changes in data over the builds selected for analysis. The Graph and the available metrics changes according to the level selected in the top left panel. Data will be displayed at System, Package, Class and Method levels.



The Graph Tab is split into three panels. The top left panel displays in tree form the JHawk Metric artefacts at System, Package, Class and Method level. Selecting an artefact will cause the other two panels to change to reflect the data at that level. The top right panel displays a number of sub panels – one for each metric at that level. Each Panel contains a colour icon, a check box and the short name of the metric (the description of the metric will be displayed as you hover over the colour icon). The colour indicates the colour of the line that will be drawn in the Graph Pane below. By clicking on this icon you can change the colour to the one that you prefer to be used for that particular metric. The check box allows you to choose which metrics will appear on the Graph Pane. The Graph pane will be updated as metrics are selected or de-selected.

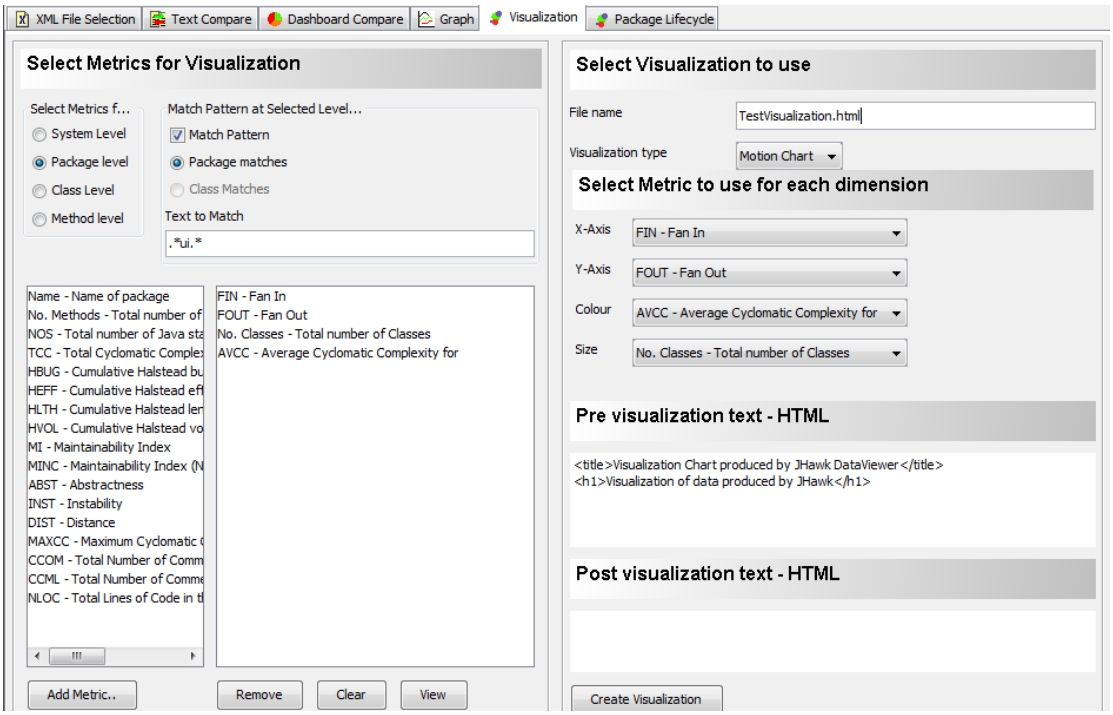
The Graph Pane shows graphs plotting the metrics selected over the builds. All metrics are displayed relative to their maximum and minimum values over the builds. The zero point on the Y-Axis is the lowest value for the metric and the top point denotes the highest value. The values shown on the Y-Axis will be in the colour of the metric to which they refer.

Visualization Tab

The Visualization Tab allows you to select data for display using Googles Visualization tools.

The Google visualization tools allow you to analyze data in a standard HTML file – provided that that file is laid out in a way that meets the standards imposed by the Google Visualizations API. You can find out more about this API here - https://developers.google.com/chart/interactive/docs/gallery.

This tab of the JHawk Data Viewer allows you to format the data created by JHawk in a way that meets this API. At present this only produces data for the Motion Chart (or Bubble Chart) and Table Chart visualizations.



When the screen is displayed there are two clear sections – on the left an area to set up the level and metrics to be selected and on the right an area to configure the HTML file to be created.

Select Metrics for Visualization

In this area you can select the level that you wish to choose the metrics from and the metrics that you wish to select for the visualization. You can only select metrics from one level.

Those of you familiar with previous versions of JHawk will notice a new area where you can limit the classes or packages from which the data will be selected. You can do this by selecting the level (package or class) to filter, checking the 'Match Pattern' check box then entering a Regular Expression into the 'Text to Match' field.

NB All regular expressions used follow the guidelines for Java regular expressions. You can find the definition of these here - http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html. There is also a useful article with examples here - http://www.vogella.com/articles/JavaRegularExpressions/article.html.

Once you have selected the level you can then select the metrics that you want to make available in the visualization. You can add as many metrics as you like but the more that you add the longer that the

visualization will take to load (there may well be a limit to the size of the visualization files that you can create but we are not yet aware of that).

You add a metric by selecting the metric in the left hand pane then press the ‘Add Metric’ button and the metric will appear in the list in the right hand pane.

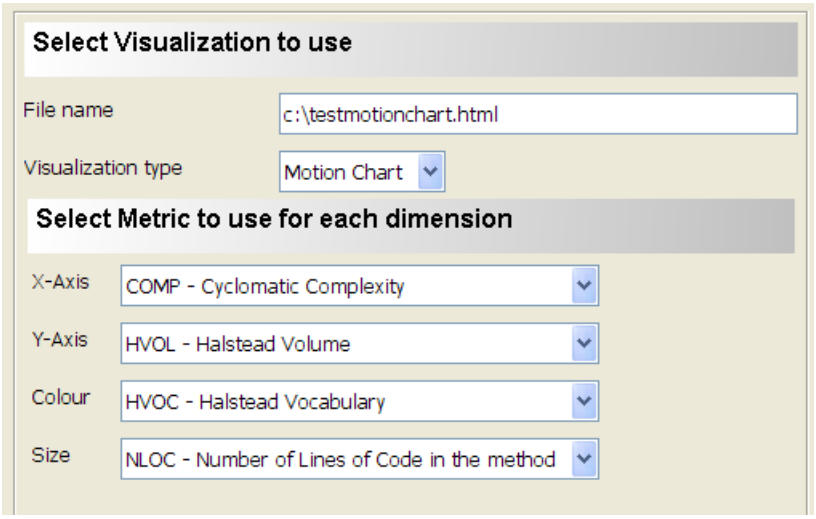
Completing the visualization configuration

On the right hand side of the screen you will see the area marked – ‘Select visualization to use’. The top part of the panel takes in two pieces of information –

- the location of the file to be created – for simplicity this filename should end in ‘.html’as this is the format expected by the Google Visualizations API
- the type of visualization to be created - either a Motion Chart or a Data Table. Depending on the option chosen a different entry screen will appear below the selection box. These screens are described below.

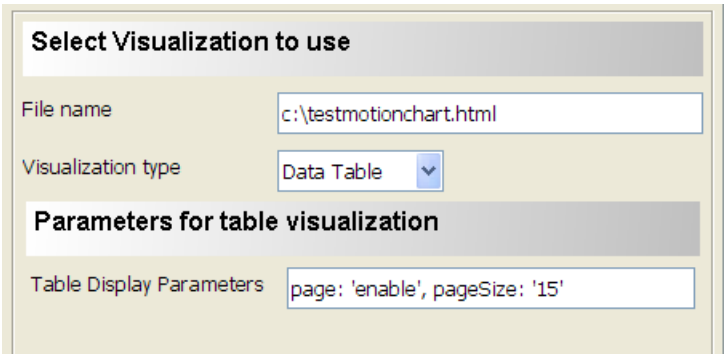
Entering data for the Motion Chart

The following fields will be displayed –



Entering data for the Data Table

In this case the following fields will be displayed –



The table display parameters are inserted as the parameters to the instruction to display the Data table. If you are familiar with the Google API you can put parameters in here that will modify the appearance of the HTML table. Parameters take the form of the parameter name followed by a colon followed by the value of the variable. In the illustration above the page formatting is enabled and the number of entries on each page is set to 15. If you look at the source of the page generated you will see that the following line is entered after the data table –

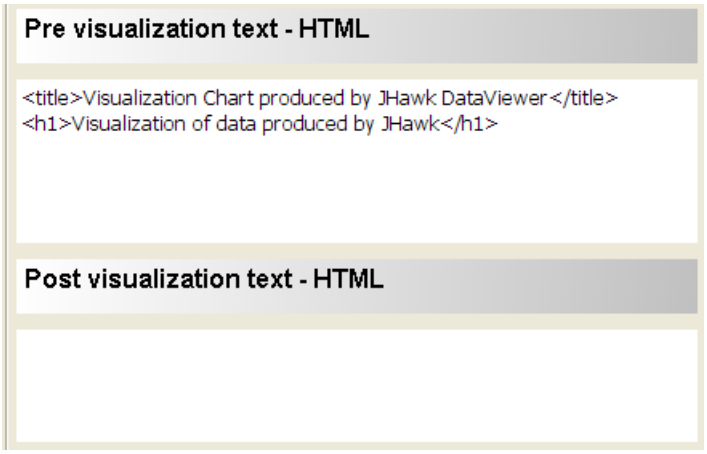
chart.draw(t, {page: 'enable', pageSize: '15'});

When you wish to set your own parameters remember that the text that you enter will be the text entered between the { } brackets. To find a complete list of parameters for the Data Table go to the Google API page for the Data table here -
http://code.google.com/apis/visualization/documentation/gallery/table.html

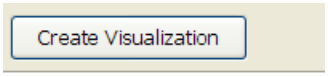
If nothing happens when you attempt to load your page after creating a Data Table then it is quite probable that there is an error in the data that you have entered for the table display parameters. See the section on 'Error handling in the Google Visualization API' below for advice in this regard.

Entering pre and post table html

Two text entry fields are provided to enter pre and post data fields –



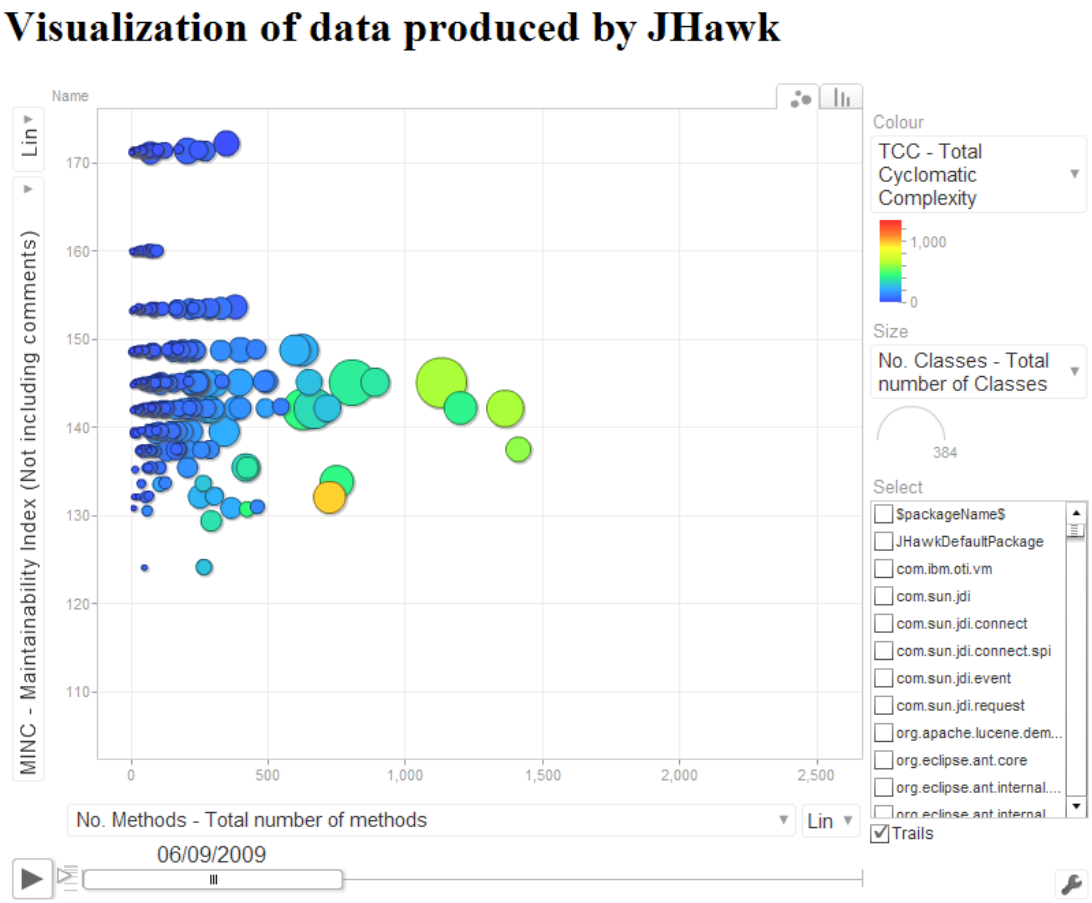
Creating the visualization and viewing the output



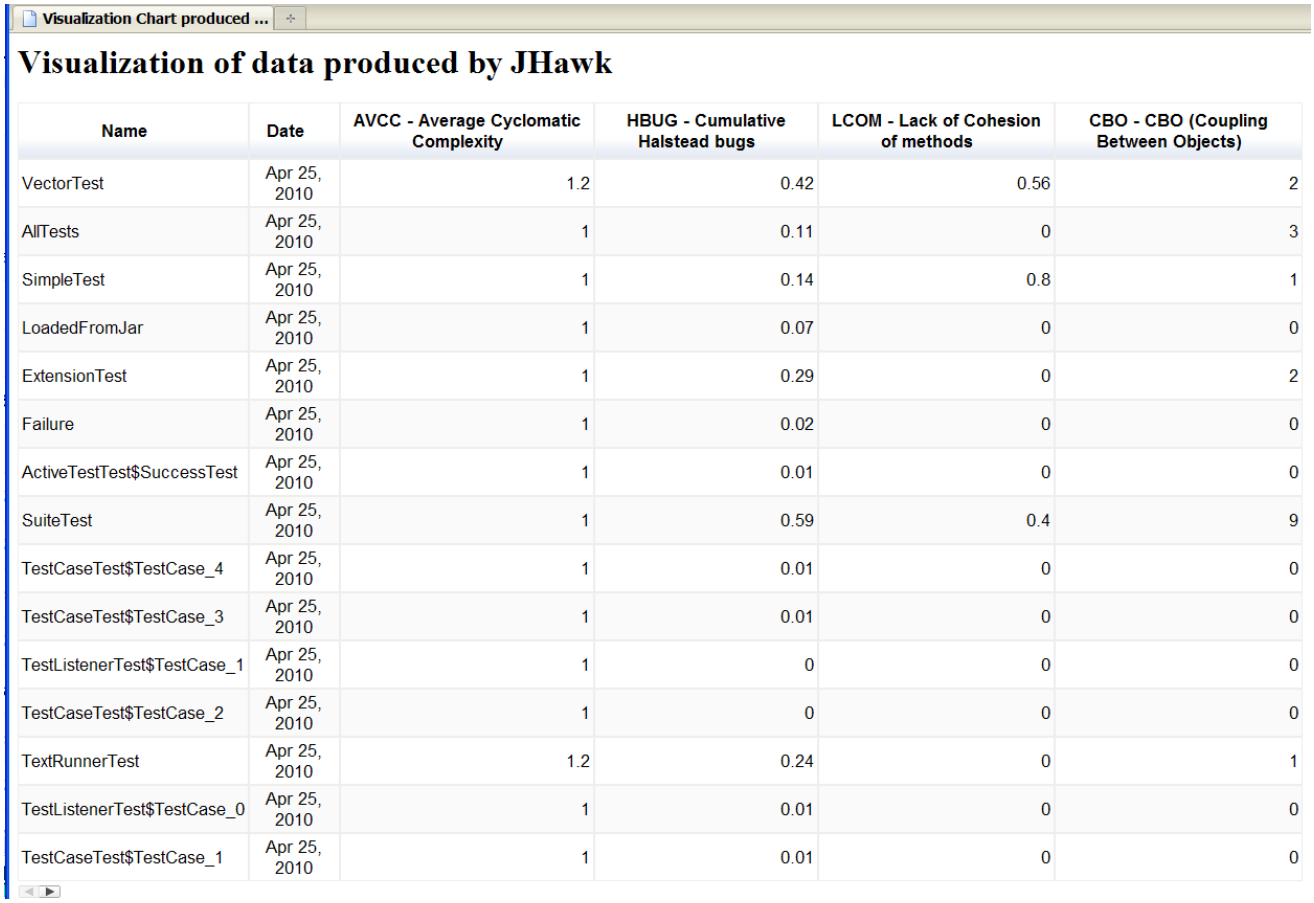
When you are ready to create your visualization press the 'Create Visualization' button - this will disable the button. When the button is re-enabled your file will be ready to view.

When you have created the HTML you can then view it using your browser. Depending on the amount of data in the visualization it may take some time to load. You may also get an error indicating a Security Violation. If you do please look at the section marked 'Security Violation' below.

Typical output from the Motion Chart Visualization



Typical output from the Data Table visualization


Visualization Chart produced ...

Visualization of data produced by JHawk

Name	Date	AVCC - Average Cyclomatic Complexity	HBUG - Cumulative Halstead bugs	LCOM - Lack of Cohesion of methods	CBO - CBO (Coupling Between Objects)
VectorTest	Apr 25, 2010	1.2	0.42	0.56	2
AllTests	Apr 25, 2010	1	0.11	0	3
SimpleTest	Apr 25, 2010	1	0.14	0.8	1
LoadedFromJar	Apr 25, 2010	1	0.07	0	0
ExtensionTest	Apr 25, 2010	1	0.29	0	2
Failure	Apr 25, 2010	1	0.02	0	0
ActiveTestTest$SuccessTest	Apr 25, 2010	1	0.01	0	0
SuiteTest	Apr 25, 2010	1	0.59	0.4	9
TestCaseTest$TestCase_4	Apr 25, 2010	1	0.01	0	0
TestCaseTest$TestCase_3	Apr 25, 2010	1	0.01	0	0
TestListenerTest$TestCase_1	Apr 25, 2010	1	0	0	0
TestCaseTest$TestCase_2	Apr 25, 2010	1	0	0	0
TextRunnerTest	Apr 25, 2010	1.2	0.24	0	1
TestListenerTest$TestCase_0	Apr 25, 2010	1	0.01	0	0
TestCaseTest$TestCase_1	Apr 25, 2010	1	0.01	0	0

Error Handling in the Google Visualization API.

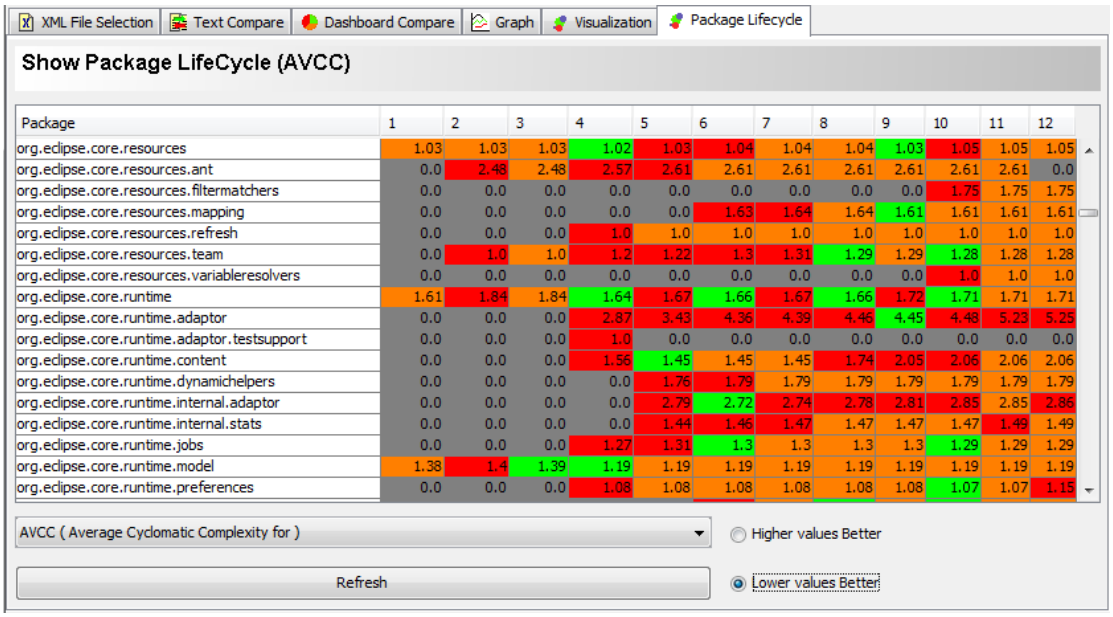
At the time of writing the Google Visualization API is still quite fragile. Sometimes a visualization will start but nothing will happen. We would suggest sending the file to Google support outlining the issue. We have done this in the past and found them responsive.

Security Violation

You may get a security violation when you open up the html page produced by the DataViewer. When you do you will need to adjust the settings on your Flash Player to allow the Google Visualization API to access the location where you have stored the Visualization HTML file. Go to the following link - http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04.html When this has displayed go to the 'Global security settings' tab and add the location of your file to the list of allowed locations.

Package Lifecycle Tab

The Package Lifecycle tab gives you a package level overview of which metrics have changed at package level in each build.



Package	1	2	3	4	5	6	7	8	9	10	11	12
org.eclipse.core.resources	1.03	1.03	1.03	1.02	1.03	1.04	1.04	1.04	1.03	1.05	1.05	1.05
org.eclipse.core.resources.ant	0.0	2.48	2.48	2.57	2.61	2.61	2.61	2.61	2.61	2.61	2.61	0.0
org.eclipse.core.resources.filtermatchers	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.75	1.75	1.75
org.eclipse.core.resources.mapping	0.0	0.0	0.0	0.0	0.0	1.63	1.64	1.64	1.61	1.61	1.61	1.61
org.eclipse.core.resources.refresh	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
org.eclipse.core.resources.team	0.0	1.0	1.0	1.2	1.22	1.3	1.31	1.29	1.29	1.28	1.28	1.28
org.eclipse.core.resources.variableresolvers	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0
org.eclipse.core.runtime	1.61	1.84	1.84	1.64	1.67	1.66	1.67	1.66	1.72	1.71	1.71	1.71
org.eclipse.core.runtime.adaptor	0.0	0.0	0.0	2.87	3.43	4.36	4.39	4.46	4.45	4.48	5.23	5.25
org.eclipse.core.runtime.adaptor.testsupport	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
org.eclipse.core.runtime.content	0.0	0.0	0.0	1.56	1.45	1.45	1.45	1.74	2.05	2.06	2.06	2.06
org.eclipse.core.runtime.dynamichelpers	0.0	0.0	0.0	0.0	1.76	1.79	1.79	1.79	1.79	1.79	1.79	1.79
org.eclipse.core.runtime.internal.adaptor	0.0	0.0	0.0	0.0	2.79	2.72	2.74	2.78	2.81	2.85	2.85	2.86
org.eclipse.core.runtime.internal.stats	0.0	0.0	0.0	0.0	1.44	1.46	1.47	1.47	1.47	1.47	1.49	1.49
org.eclipse.core.runtime.jobs	0.0	0.0	0.0	1.27	1.31	1.3	1.3	1.3	1.3	1.29	1.29	1.29
org.eclipse.core.runtime.model	1.38	1.4	1.39	1.19	1.19	1.19	1.19	1.19	1.19	1.19	1.19	1.19
org.eclipse.core.runtime.preferences	0.0	0.0	0.0	1.08	1.08	1.08	1.08	1.08	1.08	1.07	1.07	1.15

Each package is listed in the table and there is a column for each build which has been analysed. All the columns are the same width at the start so you should adjust the columns to suit the amount and type of data that you have and the length of your package names.

The table opens with the number of classes in each package displayed (although this may depend on the metrics that you have activated in your properties file).

Each entry in the table is colour coded according to the following schema –

Grey – The package did not exist in this build
Orange – The metric level is the same is that in the previous build
Red – the metric level is worse than that in the previous build
Green – the metric level is better than that in the previous build.

You can use the drop down selection and refresh button in the lower pane to change the metric used to create the data. The radio buttons on the right hand side to indicate whether lower or higher values represent an improvement in the metric. These radio buttons will be set according to the value associated with the metric but you can use the buttons to change this setting temporarily for the purposes of the Package lifecycle pane.

Running the Data Viewer from the Command Line

To run the Data Viewer from the command line all you need to do is type the following –

```
java -jar JHawkDataViewer.jar
```

Just make sure that the jhawk.properties file that you want to use is in the same directory.

The most likely reasons that you would choose to run the Data Viewer from the command line are –

(a) You can't run the jar any other way in the particular environment in which you are operating
(b) You wish to increase the amount of memory available to the Data Viewer to perform its analysis.

In the latter case you should consult the section below

Increasing the Memory available to the Data Viewer

When analysing results in the Data Viewer each file is taken into memory for analysis and the results stored in memory. As a result the analysis of a large number of XML files (or a small number of very large files) can lead to an OutOfMemoryException being thrown. To overcome this you can set the parameters on the call to the Data Viewer jar at start up. The format of the call is –

```
java –Xss<S>m –Xms<MIN>m –Xmx<MAX>m -jar JHawkDataViewer.jar
```

where <S> is the stack size in Mb (usually 4 Mb is more than enough), <MIN> is the minimum Java heap size in Mb (you don't really need to set this but it can be useful if you want to reserve RAM when multiple processes are running) and <MAX> is the Maximum Java heap size in Mb. The maximum figure is not necessarily the amount of RAM in your system as some Operating Systems have their own limitations e.g. Windows XP restricts you to around 3.2Gb even if you have more RAM than that.

.As a rule of thumb you should total the number of Mb in your XML files and treble then add 200Mb. Remember that if you are going to create a Google Visualization straight from the XML files you only need to triple the size of the largest file and add 200Mb.

If you are only using the Data Viewer to create a Google Visualization file the above rule of thumb does not apply as each file is processed individually and results are not retained in memory. This means that the memory requirement will be dictated by the largest file to be analysed.

JHawk DataViewer Service

This is a separate jar that can be used to create a motion chart visualization from multiple XML files at the command line.

This means that you don't need to start up the JHawk DataViewer application to create a motion chart visualization. The JHawkDataViewerService.jar is provided as part of the JHawk product distribution and the command line invocation of the JHawk DataViewer Service is as follows:

java -jar JHawkDataViewerService.jar <outputFile> <inputDirectory > <metrics> [options]

The parameters and options are as follows:

- outputFile - this is the full path of the output file to be created. This will be an HTML file
- inputDirectory – this is the full path of the directory containing all the XML files to be analysed to create the motion chart. These files must be in the standard JHawk Interchange format. Files will be analysed in date order unless the –oa parameter is used (see Options section below)
- metrics must be a string of metric codes separated by the '^' character e.g. RVF^NOS^AVCC^MI

java – jar JHawkDataViewerService.jar <outputFile> <inputDirectory> < metrics>
(i.e. no options supplied) is equivalent to:
JHawkDataViewerService outputFile inputDirectory metrics -lp -tm –od

Options

- -p propertiesfile Use the JHawk properties file with the name propertiesfile (the default is to use jhawk.properties in the current directory)
- -r regex Regular expression to be used when matching classes or packages

- -lp Analyze at package level (you can only supply package level metrics)
- -lc Analyze at class level (you can only supply package level metrics)
- -lm Analyze at method level (you can only supply package level metrics)
- -lpm Analyze packages that meet the regular expression supplied with -r
- -lcm Analyze classes that meet the regular expression supplied with -r
- -lcmp Analyze classes in packages that meet the regular expression supplied with -r
- -a <post-html> HTML to be evaluated below the chart
- -b <pre-html> HTML to be evaluated above the chart
- -od The files in the input directory are to be analysed in date order (default)
- -oa The files in the input directory are to be analysed in alphabetic order
- -tm The output type is a motion chart (default – this is the only option available at present)

java – jar JHawkDataViewerService.jar <outputFile> <inputDirectory> < metrics>

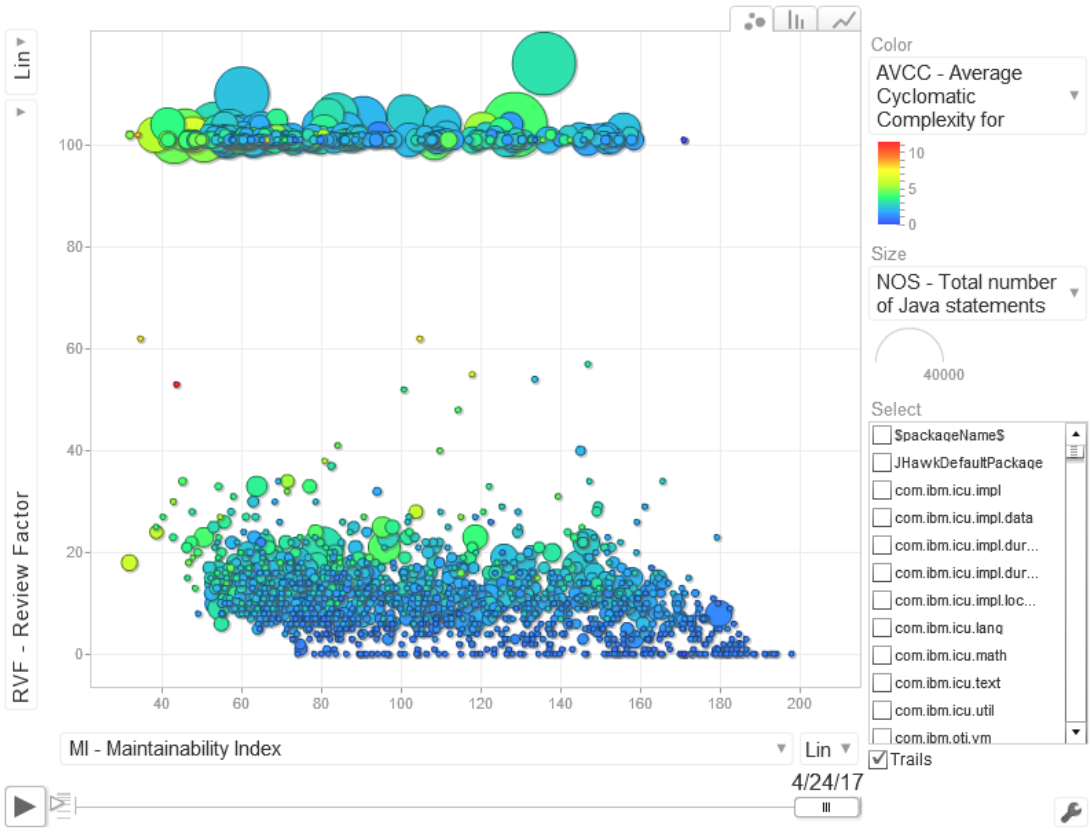
(i.e. no options supplied) is equivalent to:

java –jar JHawkDataViewerService.jar <outputFile> <inputDirectory> <metrics> -lp -tm –od –p "jhawk.properties")

The Chart below was generated using the following call to JHawk DataViewer Service:

java -jar -Xss4m -Xms800m -Xmx8000m JHawkDataViewerService.jar "D:\MyTests\Test1.html" "D:\MyTests\xmlfiles" "NOS^MI^RVF^AVCC" -tm -lp -b "<h1>Before Text</h1>" -a "<h1>After Text</h1>"

Before Text



After Text

Automating the process

It's very easy to automate the process of creating a visualization. The following example was done as an exercise at Virtual Machinery.

The aim of the test

What we set out to do was to create a less labour intensive means of comparing the source code of a project over a number of iterations. As part of the prototyping and testing of JHawk we had already undertaken this using the source code of the Eclipse project but we felt a smaller example would be more relevant as an example to our users.

Gathering the resources

We decided to use the source of the JUnit project as our example. We downloaded the zip files from Github. We downloaded the following versions –

Junit3.4.zip
Junit3.5.zip
Junit3.6.zip
Junit3.7.zip
Junit3.8.zip
Junit4.0.zip
Junit4.1.zip
Junit4.2.zip
Junit4.3.1.zip
Junit4.4.zip
Junit4.4.zip
Junit4.5.zip
Junit4.6.zip
Junit4.7.zip
Junit4.8.zip

These zip files contained a number of different kinds of files but we were only interested in the Java files so we had to extract these. We did so using the '7zip' open source tool which is available here - http://www.7-zip.org/. We put the 7z.exe file (which is the command line version of the tool) into the same directory as the zip files.

We then wrote a batch file called runextract.bat which extracted the Java files from the src.jar file in each of the zip files, maintaining the directory structure as it did so. Our batch file therefore looked like this –

```
7z x junit3.4.zip src.jar -r
7z x junit3.4\src.jar -oC:\JUnitSourceTests\junit34 *.java -r
7z x junit3.5.zip src.jar -r
7z x junit3.5\src.jar -oC:\JUnitSourceTests\junit35 *.java -r
7z x junit3.6.zip src.jar -r
7z x junit3.6\src.jar -oC:\JUnitSourceTests\junit36 *.java -r
7z x junit3.7.zip src.jar -r
7z x junit3.7\src.jar -oC:\JUnitSourceTests\junit37 *.java -r
7z x junit3.8.zip src.jar -r
7z x junit3.8\src.jar -oC:\JUnitSourceTests\junit38 *.java -r
7z x junit4.0.zip junit-4.0-src.jar -r
7z x junit4.0\junit-4.0-src.jar -oC:\JUnitSourceTests\junit40 *.java -r
7z x junit4.1.zip junit-4.1-src.jar -r
7z x junit4.1\junit-4.1-src.jar -oC:\JUnitSourceTests\junit41 *.java -r
7z x junit4.2.zip junit-4.2-src.jar -r
```

```
7z x junit4.2\junit-4.2-src.jar -oC:\JUnitSourceTests\junit42 *.java -r
7z x junit4.3.1.zip junit-4.3.1-src.jar -r
7z x junit4.3.1\junit-4.3.1-src.jar -oC:\JUnitSourceTests\junit43 *.java -r
7z x junit4.4.zip junit-4.4-src.jar -r
7z x junit4.4\junit-4.4-src.jar -oC:\JUnitSourceTests\junit44 *.java -r
7z x junit4.5.zip junit-4.5-src.jar -r
7z x junit4.5\junit-4.5-src.jar -oC:\JUnitSourceTests\junit45 *.java -r
7z x junit4.6.zip junit-4.6-src.jar -r
7z x junit4.6\junit-4.6-src.jar -oC:\JUnitSourceTests\junit46 *.java -r
7z x junit4.7.zip junit-4.7-src.jar -r
7z x junit4.7\junit-4.7-src.jar -oC:\JUnitSourceTests\junit47 *.java -r
7z x junit4.8.zip junit-4.8-src.jar -r
7z x junit4.8\junit-4.8-src.jar -oC:\JUnitSourceTests\junit48 *.java -r
```

Running this to extract the files took about 10 seconds. 1198 Java files were extracted over the 14 versions of the code.

We now had the Java files separated out from the zip files into the directory ready for analysis using JHawk. As we wanted to create a Motion Chart visualization we needed to get the data into the JHawk Metric interchange format. To do this we created another batch file – this time calling the command line version of JHawk. This file was called buildxml.bat and looked like this –

```
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit34 -x
C:\JUnitSourceTests\xmlFiles\junit34 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit35 -x
C:\JUnitSourceTests\xmlFiles\junit35 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit36 -x
C:\JUnitSourceTests\xmlFiles\junit36 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit37 -x
C:\JUnitSourceTests\xmlFiles\junit37 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit38 -x
C:\JUnitSourceTests\xmlFiles\junit38 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit40 -x
C:\JUnitSourceTests\xmlFiles\junit40 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit41 -x
C:\JUnitSourceTests\xmlFiles\junit41 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit42 -x
C:\JUnitSourceTests\xmlFiles\junit42 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit43 -x
C:\JUnitSourceTests\xmlFiles\junit43 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit44 -x
C:\JUnitSourceTests\xmlFiles\junit44 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit45 -x
C:\JUnitSourceTests\xmlFiles\junit45 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit46 -x
C:\JUnitSourceTests\xmlFiles\junit46 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit47 -x
C:\JUnitSourceTests\xmlFiles\junit47 -system JUnitSource
java -jar JHawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\JunitSourceTests\junit48 -x
C:\JUnitSourceTests\xmlFiles\junit48 -system JUnitSource
```

We had to create the xmlFiles directory prior to running this file. Running the file took about 20 seconds. Now we had all of our interchange files ready for analysis using the JHawk DataViewer. At this point we have two options –analyze the files manually using the JHawk DataViewer standalone application, or analyse the files automatically using the JHawkDataViewer Command Line Service. Since it's simpler to use the command line we'll use that first

Analyzing the XML Files using the JHawk DataViewer command line

Using the JHawk Data Viewer Command Line Service we used the following parameters:

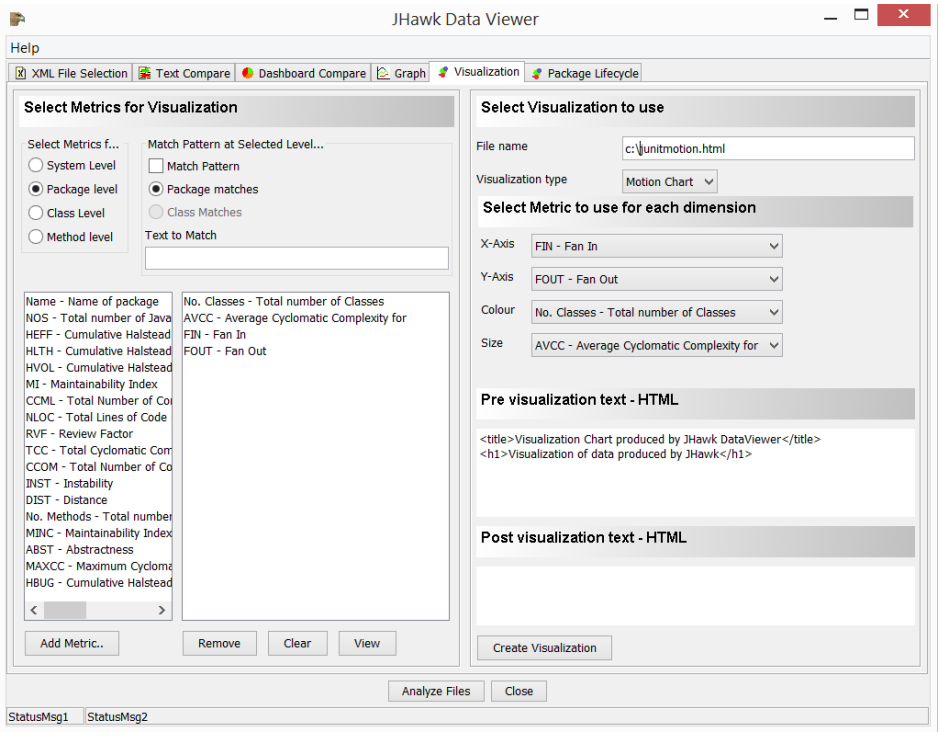
java -jar JHawkDataViewerService.jar "C:\junitmotion.html" " C:\JUnitSourceTests\xmlFiles" "AVCC^FIN^FOUT^No. Classes" -tm -lp –b "<title>Visualization Chart produced by JHawk Data Viewer Command Line</title><h1>Visualization of data produced by JHawk</h1>"

So we selected a Motion Chart visualization (-tm) , package level metrics (-lp) and the AVCC, FIN, FOUT and No. Classes metrics. We also made sure that the 'before' text was the same as that that is created by default in the standalone version by using the –b parameter.

This will give exactly the same results as those obtained using the standalone JHawk Data Viewer with the criteria selected below.

Analyzing the XML Files using the standalone JHawk DataViewer

We started this up and selected the xmlFiles directory containing the XML interchange files. As we were going straight to the creation of the visualization we did not need to analyse the files at this point. We went to the visualization tab of the Data Viewer and set up our parameters for the Motion Chart visualization –



Creating the visualization took about 15 seconds. You can find the visualization that we produced in the root directory of the 'Sample Output' download which you can find at http://www.virtualmachinery.com/jhdownload.htm. It is called visualization.html. If you wish to view this file and have not already read the section entitled 'Security Violation' above you should do so.