JHawk 6.1 Documentation- Command Line Manual

Virtual Machinery
September 2015
Version 1.1

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 has now evolved to include a command line version and an Eclipse plugin. 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 Command Line interface to JHawk is distributed as a single jar. Typically the command line is used to analyse code and generate output in an environment where a GUI would be inappropriate e.g. as part of a build process.

The JHawk Command Line jar also includes a JHawk Ant task which can be used as part of any Ant script.

If you are already using] JHawk you will find details of changes made to JHawk since the last version (JHawk 5.1) in the document JHawk6Changes.pdf.

This issue refers to JHawk Version 6.1. The only significant change since JHawk 6.0 is the addition of the Filters functionality. These changes are covered in the Filters section of this document and in the individual sections where filters are relevant.

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
JHawk6EclipseManual – Documentation for the JHawk Eclipse Plugin	Yes	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

JHawk Command Line

The JHawk Command Line version is distributed as a single jar and a jhawk.properties file. These are located in the CommandLine sub-directory of the distribution. Please note the jhawk.properties file used by the JHawk command line is different from the standard jhawk.properties file For standard operation these should be located in the same directory. You can locate the properties file in a different directory (and even give it a different name) by using the –p option as described below.

You can generate a new properties file for use with the command line by exporting it from any version of JHawk that allows you to edit and save the properties file i.e. the stand alone version and the Eclipse plugin. To use the exported file with the command line you must replace the first line :

```
jhawk.factoryclass=com.virtualmachinery.jhawk.views.VMJHawkApplicationFactory
```

With:

```
jhawk.factoryclass=com.virtualmachinery.jhawk.commandlineservice.VMJHawkCommandLine
Factory
```

As part of the distribution you will notice the following properties files in the CommandLine sub-directory of the distribution –

- 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.

Usage

The format of a valid JHawk command line call is as follows –

java -jar JHawkCommandLine.jar [-p propertiesFile] [-c csvfile] [-raw level] [-v csvSeparator] [-d directories] [-f files] [-h htmlfile] [-l levels] [-r] [-s startpath] [-x xmlfile] [-b] [-xd excludedirectories] [-xf excludedfiles] [-n names] [-system name] [-a] [-hp htmlPath]

Description of Available options

The available options are preceded with a dash ("-"). These are -

Help

-? Help – this prints out help text outlining all the available options and their arguments

Source code file selection

-f File selection – a regular expression which matches the files to be selected. All matching files will be analyzed unless they have been specifically excluded using the –xf option
-d Directory selection - – a regular expression which matches the directories to be selected. All .java files in these directories will be analyzed unless they have been specifically excluded using the –xf option
-r Recurse – this is applied to the directories selected using –d or –s. No argument is supplied to the –r option. All .java files in the directories selected using the –d or –s option will be analysed unless they have been specifically excluded using the –xf option, or their directories have been specifically excluded using the –xd option.
-xf Exclude files - a regular expression which matches the files to be excluded from the analysis.
-xd Exclude directories - a regular expression which matches the directories to be excluded from the analysis.

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.

Metric selection

-a Only the active metrics will be selected. If the –a option is not set then all metrics will be output. No further arguments are required

Properties File

-p Properties file – the parameter following is the name of the properties file which will be used during the analysis of the code. If the –p parameter is not set then the first jhawk.properties file in the path of the Command Line jar file will be used.

System name

-system The text following this option will be used in the output files as the system name

Output type

Only one output type can be selected – except in the case of CSV output where raw output will be produced if both –raw and –c are selected
-x Standard XML output – the parameter following is the name of the file to which the XML output will be written.
-b JHawk Standard Interchange XML output – the parameter following is the name of the file to which the XML output will be written.
-c Standard CSV output – the parameter following is the name of the file to which the CSV output will be written.
-raw Raw CSV output – the parameter following is the artefact level which will be written to the raw CSV file – acceptable values are either "p" (for Package) , "c" (for Class) or "m" (for Method). This parameter must be used in conjunction with the –c parameter which will indicate the name of the file to which the Raw CSV is to be written.
-h HTML output - – the parameter following is the name of the main index file from which the HTML output can be accessed

Additional Qualifiers

-l Levels – These are the levels at which analysis will be carried out. Acceptable levels are any string containing the letters "p" (for Package) , "c" (for Class) or "m" (for Method).

-n Names selected – These are the names which will be displayed as part of the metrics output. Acceptable values are any string containing the letters “p” (for Package) , “c” (for Class) or “s” (for System).
-s Start path - The path of the directory from which the search for files to analyze will start - the default is the current directory .
-hp HTML Path (HTML output only). The argument to this sets the name of the directory in which the HTML files will be located . The root index file will be that which ahs been sent as the argument to the –h option.
-v CSV separator (CSV output only). This sets the CSV separator to be used in the CSV files. If no CSV separator is set then it is assumed to be “,”.

Filters

-ff Filter File Path (HTML output only). The argument to this is the name of the file which contains the filters to be used. See the main section on Filters below.

Examples

The following line will analyse all the Java source code in the directory C:\MySource and will generate HTML at package, class and method level and put it in the directory C:\myhtml. The root file of the HTML files will be index.html.

java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\mysource -hp C:\myhtml -h index

The following line will analyse all the Java source code in the directory C:\MySource and will generate XML at package, class and method level and put it in the file C:\myXMLFile. The System name used in the XML file will be MySystem

java -jar jhawkCommandLine.jar -f .*\.java -r -b -l pcm -s C:\mysource -x C:\myXMLFile -system MySystem

The following line will analyse all the Java source code in the directory C:\MySource and will generate a raw csv file at method level called mysource.csv.

java -jar jhawkCommandLine.jar -f .*\.java -r -l pcm -s C:\mysource -raw m -c mysource.csv

JHawk Ant Task

The functionality provided by the JHawk Command Line jar is also available as an Ant task. Output from the Ant task will be the same as that from the JHawk Command Line for the same parameters. This section describes the parameters available and provides examples of JHawk ant tasks that illustrate the use of these parameters. The following parameters are provided in the Ant task.

Usage

Output related parameters

csvfilename - requests the data to be output as CSV - takes the output path of the file as a parameter. Equivalent to the -c command at the command line. See also the csvseparator, outputlevels and namelevels parameters.

csvseparator - separation character to be usesd between CSV fields. Takes the character as a parameter. Equivalent to the -v command at the command line.

htmlfilename - requests the data to be output as html - takes the output path of the file as a parameter. Equivalent to the -h command at the command line. See also the outputlevels and namelevels parameters.

htmlpath - the directory path for the html files created when html output is requested - takes the directory path as a parameter. Equivalent to the -hp command at the command line.

xmlfilename - requests the data to be output as xml - takes the output path of the file as a parameter. Equivalent to the -x command at the command line. See also the outputlevels and namelevels parameters.

outputlevels - sets the levels of output that will be produced. Each level is represented by a character - p(package), c(class) or m(method). More than one level can be produced by combining the characters e.g. 'pcm' will produce output at package, class and method level. Equivalent to the -l flag at the command line.

namelevels - sets the levels of naming that will be produced as additional columns in a csv file. Each level is represented by a character - s(system), p(package) or c(class). More than one level can be produced by combining the characters e.g. 'pcm' will show the package, class and method names if appropriate. Equivalent to the -n flag at the command line.

systemname- sets the string representing the system name be used when analysing the files and producing the results. Takes a string representing the system name as a parameter. Equivalent to the -system command at the command line.

startpath - sets the base directory from which the selection of java files will begin - the default is '.' (the current directory) . Takes a string representing the start path as a parameter. Equivalent to the -s command at the command line

File selection patterns

directorypattern - regular expression representing the directories whose files may be analysed by JHawk*. Takes the regular expression as a parameter. Equivalent to the -d flag at the command line.

filepattern - regular expression representing the files to be analysed by JHawk*. Takes the regular expression as a parameter. Equivalent to the -f flag at the command line.

excludefilepattern - regular expression representing the files that are to be excluded from analysis by JHawk*. Takes the regular expression as a parameter. Equivalent to the -xf flag at the command line.

excludedirectorypattern - regular expression representing the directories whose files are to be excluded from analysis by JHawk*. Takes the regular expression as a parameter. Equivalent to the -xd flag at the command line.

*File selection for analysis will be influenced by a combination of directorypattern, file pattern, excludefilepattern and excludedirectorypattern. Directorypattern is evaluated first, then excludedirectorypattern then filepattern then excludefilepattern. There can be multiple instances of each pattern which will be grouped then evaluated in the order described.

Configuration parameters

propertiesfile – path to the file which will be used during the analysis of the code. If this parameter is not set then the first jhawk.properties file in the path of the Command Line jar file will be used.

Flags

activeonly - no parameters. Only the active metrics will be selected. If this option is not set then all metrics will be output. Equivalent to the -a flag at the command line.

basiconly - no parameters. This is only effective in the case of XML output. It will ensure that the output is written in JHawk Standard Interchange XML format (see the section in 'Output Formats' below). Equivalent to the -b flag at the command line.

rawlevel - no parameters. This is only effective in the case of CSV output. It will ensure that the output is written in 'Raw' format (see the section in 'Output Formats' below). Equivalent to the -raw flag at the command line.

recursive - no parameters. This is applied to the directories selected using startpath or directorypattern. All .java files in the directories selected using the directorypattern option will be analysed unless they have been specifically excluded using the excludefilepattern option, or their directories have been specifically excluded using the excludedirectorypattern option.Equivalent to the -r flag at the command line.

Sample Ant Files

```
<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"
                 systemname="MySource" />
     </target>
</project>
```

The simple file above is equivalent to the following JHawk Command line parameters –

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

The Ant task will analyse all the Java files in the directory structure under 'C:\MySource' (**startpath** and **recursive**) and output the results to the file 'C:\MyXML.xml' (**xmlfilename**). Only the results at package and class level (**outputlevels**) will be shown and the output will be in basic (JHawk Metric

Interchange) format (**basiconly**). The system name in the output will be named as 'MySource' (**systemname**).

Anything that you can define as a command line sequence with the JHawk Command Line can be implemented as an Ant task.

While an Ant task is normally defined and run as part of a larger Ant script you can also run a JHawk Ant task in isolation by running the following at the command line –

```
ant -v -lib JHawkCommandLine.jar –f JHawkAntTaskFile.xml
```

where JHawkAntTaskFile.xml is an xml file containing a JHawk Ant task. You need to make sure that the jhawk.properties file is in the same directory (or else include a reference to another properties file using the **propertiesfile** parameter in your Ant task.

Here are a number of example Ant tasks that demonstrate some of the capabilities of the JHawk Ant Task –

The following example illustrates the creation of HTML output with some directories and files excluded –

```
<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"
                    excludedirectorypattern=".*ui"
                    excludefilepattern="C:\\SourceTestArea\\JUnitSourcetests\\
junit48\\.*Test.*"
                    startpath="C:\SourceTestArea\JUnitSourcetests\junit48"
                    htmlpath="C:\SourceTestArea\anttests\anthtml48"
                    htmlfilename="junit48ex"
                    systemname="Junit48" />
     </target>
</project>
```

This JHawk Ant task will analyse all of the Java files in the directory structure under C:\SourceTestArea\junit48 (**startpath**) except those in directories whose name ends with 'ui' (**excludedirectorypattern**) and those files whose names include 'Test' in their name (**excludefilepattern**). Note that the full start path is used in the **excludefilepattern** value this is because it contains 'Test' and the full path of the file name is taken into account when the file is tested for exclusion. The results of the analysis are output in html format in the directory 'C:\SourceTestArea\anttests\anthtml48' (as set in the **htmlpath** variable) and the root file of the html output will be the file 'junitex48.html' (set by the **htmlfilename** property) in the 'C:\SourceTestArea\anttests\anthtml48' directory.

You can find the output files for this test in the anthtml48 directory of the 'Sample Output' download which you can find at http://www.virtualmachinery.com/jhdownload.htm. As an exercise you can compare this output with that in the html48 to show that the expected files have been excluded from analysis.

Ant is a registered trade mark of the Apache Software Foundation.

Filters

Filters are a new feature of JHawk 6.1. Filters created in the JHawk Stand alone and Eclipse versions and stored to file can be used in the command line using the –ff command and the name of the filter file.

Filters can be managed at three levels:

- Filter Record Group Sets
- Filter Record Groups
- Filter Records

The Filter Record is the lowest level. It contains a record of the level of the metrics being filtered (Package, Class or Method) , the filter applied relating to the calculated value of the metric (Danger, Warning, Warning and Danger) and the Metric(s) that are to be calculated for the filter.

A Filter Record group contains one or more filter records, it has an identifier and a description. If a number of Filter Record Groups are contained in a Filter Record Group Set then the Filter Record Group identifier must be unique within that Filter Record Group set.

A Filter Record Group set is a logical grouping of Filter Record Groups. When the Manage Filter Groups dialog is used then all the Filter Record Groups being managed in the dialog are treated as a single Filter Record Group Set. When Filter Record groups are stored in a file they are stored as a Filter Record Group Set.

Note that you must use the same JHawk properties file to run the filters as you use to create them. You can create sets of filters that include filters for different levels of artefact (package, method and class) and for different levels of metric (Warning level only, Danger level only, warning and danger level). All of the filters will be applied in turn and as soon as one of the filter criteria is met any artefact that meets the criterion will be added to the filtered set. Note that if you are analysing a hierarchy of artefacts, only the artefacts that pass the filter at the highest level will be used in the subsequent analysis. As an example let's say that you have the following artefacts that pass the filters that you have set:

- Package1
- Package1.Class1
- Package1.Class1.Method1
- Package2.Class2
- Package3.Class3.Method3

Any artefact that is not mentioned does not pass any of the filters so if we analyse at package level only the following artefacts will pass the filters:

- Package1
- Package1.Class1
- Package1.Class1.Method1

Because Package2 and Package3 do not pass the filter no further analysis will be carried out on them. If we analyse at class level the following artefacts will pass the filters:

- Package1.Class1
- Package1.Class1.Method1
- Package2.Class2

Because Package3.Class3 does not pass the filter no further analysis will be carried out on it.
Finally if we analyse at method level the following artefacts will pass the filters:

- Package1.Class1.Method1
- Package3.Class3.Method3

The Filters sections in the JHawk User Manual cover the creation, management and use of filter files in more detail.

Standard Filter Files

A number of standard filter files come with JHawk. These files can be found in the filters directory of the distribution. These filters reflect the standard Warning and Danger levels that are predefined for a number of different metrics and levels in the standard JHawk properties files. These can be used as they are or as a basis for building your own metrics filters.

File Name	Artefact	Metrics	Level
StandardPackageD.xml	Package	AVCC, MI, MAXCC	Danger
StandardPackageWD.xml	Package	AVCC, MI, MAXCC	Warning + Danger
StandardClassD.xml	Class	AVCC, MI, MAXCC	Danger
StandardClassWD.xml	Class	AVCC, MI, MAXCC	Warning + Danger
StandardMethodD.xml	Method	COMP	Danger
StandardMethodWD.xml	Method	COMP	Warning + Danger

Output Formats

Samples of the export outputs can be found in the 'Sample Output' download which you can find at http://www.virtualmachinery.com/jhdownload.htm.

CSV – Standard

In the standard CSV export format a summary record is written at each level in addition to the data at that level.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Details of classes for Package junit.framework														
2	System	Package	Name	No. Methods	LCOM	AVCC	NOS	HBUG	HEFF	UWCS	INST	PACK	RFC	CBO	MI
3	No System Name	junit.framework	Assert	39	0	1.36	92	1.15	24438.84	39	0	0	39	4	158.29
4	No System Name	junit.framework	AssertionFailedError	2	1	1	5	0.03	344	3	1	0	2	8	192.17
5	No System Name	junit.framework	ComparisonCompactor	8	0.3	2.12	49	0.62	19152.76	16	8	0	8	2	115.19
6	No System Name	junit.framework	ComparisonFailure	4	0.25	1	15	0.12	1631.71	8	4	0	4	3	100.42
7	No System Name	junit.framework	JUnit4TestAdapter	12	0	1.42	37	0.36	6863.56	15	3	11	12	12	131.41
8	No System Name	junit.framework	JUnit4TestAdapterCache	5	0.5	2.2	27	0.32	8983.28	7	2	8	6	10	119.12
9	No System Name	junit.framework	JUnit4TestAdapterCache$Rur	3	0	1	7	0.09	1197.97	3	0	8	3	2	139.29
10	No System Name	junit.framework	JUnit4TestCaseFacade	5	0.75	1	12	0.07	762.86	6	1	2	5	5	142.58
11	No System Name	junit.framework	Protectable	1	0	1	2	0.01	72	1	0	0	1	2	97.04
12	No System Name	junit.framework	Test	2	0	1	3	0.02	91.02	2	0	0	2	19	140.07
13	No System Name	junit.framework	TestCase	13	0.67	1.62	48	0.42	11605.44	14	1	3	13	8	79.86
14	No System Name	junit.framework	TestFailure	7	0.08	1	24	0.19	2870.95	9	2	2	7	3	172.51
15	No System Name	junit.framework	TestListener	4	0	1	5	0.03	156.54	4	0	0	4	7	179.07
16	No System Name	junit.framework	TestResult	18	0	1.39	68	0.59	10410.7	23	5	4	18	17	151.99
17	No System Name	junit.framework	TestResult$Protectable	1	0	1	3	0.02	136.54	1	0	4	1	0	136.39
18	No System Name	junit.framework	TestSuite	24	0.35	2.29	102	1.36	56645.36	26	2	10	25	9	122.03
19	No System Name	junit.framework	TestSuite$TestCase	1	0	1	3	0.02	137	1	0	10	1	0	136.38
20	Package overview of package junit.framework														
21	System	Name	No. Classes	NOS	AVCC	HBUG	HEFF	HLTH	HVOL	MI	CCML	NLOC	TCC	CCOM	INST
22	No System Name	junit.framework	17	557	1.54	6.16	167133	4266	18474.07	149.7	523	792	229	125	0.33
23	Details of classes for Package org.junit.experimental.theories.internal														
24	System	Package	Name	No. Methods	LCOM	AVCC	NOS	HBUG	HEFF	UWCS	INST	PACK	RFC	CBO	MI
25	No System Name	org.junit.experimen	AllMembersSupplier	7	1	2.86	40	0.6	18084.58	8	1	12	7	6	167.81

CSV – Raw

When data is exported in the raw CSV format a single record for each artefact is written at the requested level. In the example here the data has been exported at the method level. In each case the fully qualified name will be available with the System, package, class and method name written in the columns at the left.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	System	Package	Class	Name	COMP	NOCL	NOS	HLTH	HVOC	HEFF	HBUG	CREF	XMET	LMET	NLOC	NOC	NOA	MOD
2	JUnit	junit.extensions	ActiveTestSuite	ActiveTestSuite	1	0	1	4	4	12	0	0	0	0	2	0	0	1
3	JUnit	junit.extensions	ActiveTestSuite	ActiveTestSuite	2	0	2	15	13	266.43	0.02	1	0	0	3	0	1	1
4	JUnit	junit.extensions	ActiveTestSuite	ActiveTestSuite	2	0	2	18	15	361.67	0.02	2	0	0	4	0	2	1
5	JUnit	junit.extensions	ActiveTestSuite	ActiveTestSuite	1	0	2	10	8	75	0.01	1	0	0	3	0	1	1
6	JUnit	org.junit.internal.builders	AllDefaultPossib	AllDefaultPossibilitiesBu	1	0	2	10	9	99.06	0.01	0	0	0	4	0	1	1
7	JUnit	org.junit.experimental.theories.i	AllMembersSupp	AllMembersSupplier	1	3	2	10	9	99.06	0.01	1	0	0	4	1	1	1
8	JUnit	org.junit.runners	AllTests	AllTests	2	3	2	15	13	266.43	0.02	2	0	0	3	1	1	1
9	JUnit	org.junit.internal.builders	AnnotatedBuilde	AnnotatedBuilder	1	0	2	10	9	99.06	0.01	1	0	0	4	0	1	1
10	JUnit	org.junit.internal	ArrayComparisor	ArrayComparisonFailure	1	7	4	22	15	279.34	0.03	2	0	1	5	1	3	1
11	JUnit	junit.framework	Assert	Assert	1	3	1	4	4	12	0	0	0	0	2	1	0	1
12	JUnit	org.junit	Assert	Assert	1	3	1	4	4	12	0	0	0	0	2	1	0	1
13	JUnit	junit.framework	AssertionFailedE	AssertionFailedError	1	0	1	4	4	12	0	0	0	0	2	0	0	1
14	JUnit	junit.framework	AssertionFailedE	AssertionFailedError	1	0	2	10	8	75	0.01	1	0	0	3	0	1	1
15	JUnit	org.junit.experimental.theories.i	Assignments	Assignments	1	0	4	28	18	557.25	0.04	2	0	0	6	0	3	1
16	JUnit	org.junit.internal	AssumptionViol	AssumptionViolatedExce	3	0	4	31	20	1004.85	0.04	2	0	0	5	0	2	1
17	JUnit	org.junit.internal	AssumptionViol	AssumptionViolatedExce	1	0	2	11	9	83.69	0.01	1	0	0	4	0	1	1
18	JUnit	org.junit.runners	BlockJUnit4Class	BlockJUnit4ClassRunner	2	6	2	15	13	266.43	0.02	2	0	0	4	1	1	1

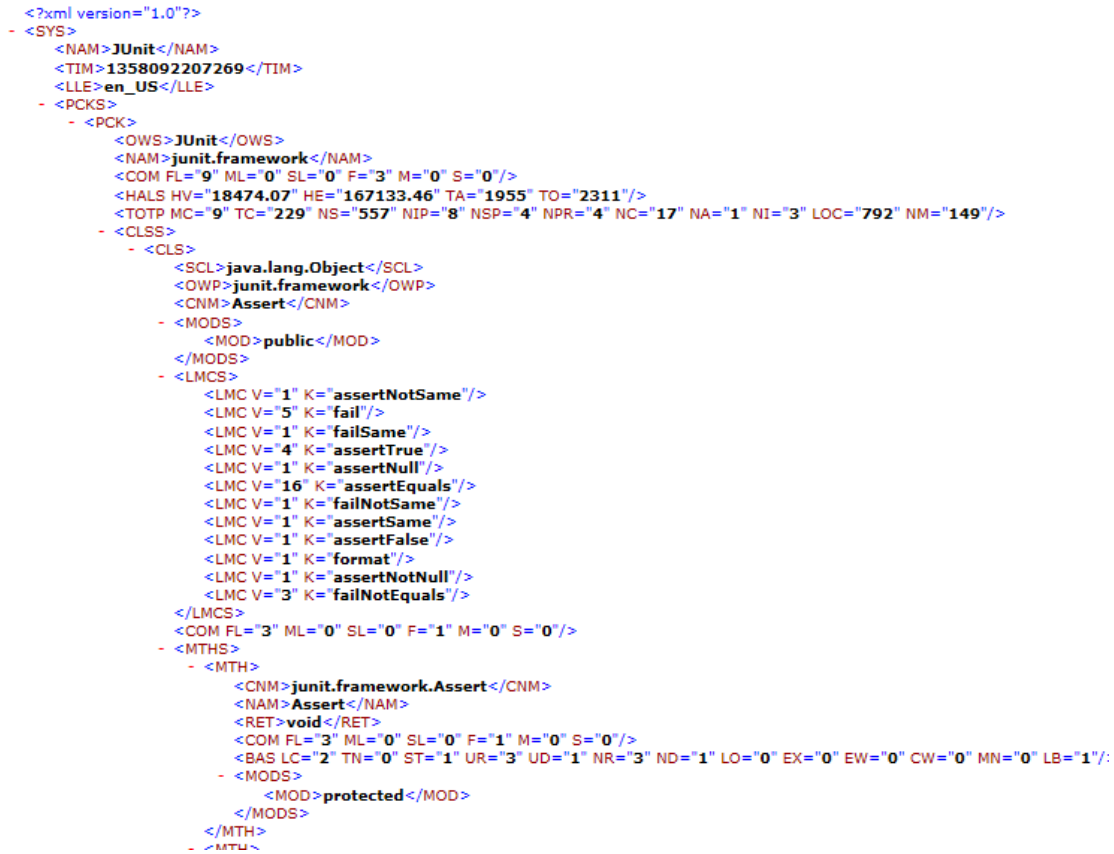
You can find an example of a raw csv file in the 'ExportSamples' directory of the distribution. It is called junit48.csv and is a raw method output of the methods in the source code of JUnit version 4.8.

XML – Standard

XML output can be written in two forms – Standard or JHawk Interchange . In the standard format xml output is written for the selected metrics (active or all) and at the relevant levels (System, Package, Class, Method etc). Sample output is shown below.

XML – JHawk Interchange Format

If you select the JHawk Interchange format for your XML output then the data is exported for the entire system at all levels. As this is the base data collected for your system this data can be used to completely rebuild the dataset inside either JHawk or the JHawk Data Viewer add on.



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, The JHawk Eclipse Plugin or the command line interface.

When 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). E.g.

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

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. 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. You can analyse at class or package level but if you wish to analyse metrics that are ultimately based on data collected at method level (and this includes many of the important metrics such as the Halstead Metrics and Cyclomatic Complexity) you will need to analyse right down to the method level.

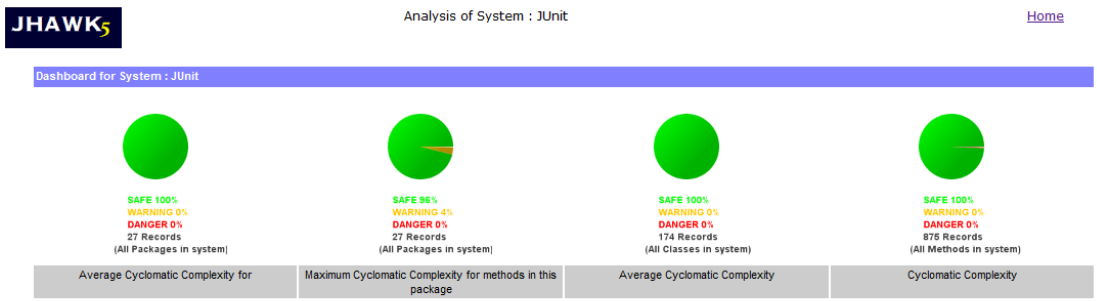
If you wish to compare files in the tree views (Text Compare and Graph) then all of your systems should have the same name (or all have no name). In the command line version you should use the –system flag followed by the name that you wish to give the system.

Creating files to load back into JHawk Stand Alone

If you are creating files to view again in the JHawk stand alone version the same criteria apply as for the JHawk Data Viewer

HTML

HTML data is output at the levels selected and the structure of the pages output will depend on the levels selected. At the very least a root page will be output showing the highest level select. The data from all the other levels selected can be accessed from this.



Summary of JUnit

Feature	Number	Values
Name	JUnit	-
Number of packages	27	-
Number of Statements	3832	-
Average Cyclomatic Complexity	1.57	-
Halstead Cumulative Bugs	46.38	-
Halstead Effort	1284331.07	-

If you have elected to select all levels then the first page of the HTML output will be as above. All the active dashboard gauges will be displayed and a table setting out the system level metrics will be displayed.

Below this a list of all the packages that constitute the system being analysed will be displayed.

Packages in System: JUnit

Name	No. Classes	NOS	AVCC	HBUG	HEFF	MI	CCML	NLOC	TCC	CCOM	HLTH	INST	DIST	FIN	No. Methods	MINC	ABST	HVOL	MAXCC	FOUT
junit.extensions	6	90	1.29	0.66	11767.43	181.96	37	131	31	11	540	0.50	0.50	2	24	131.96	0.00	1989.50	3	2
junit.framework	17	557	1.54	6.16	167133.46	149.70	523	792	229	125	4266	0.33	0.43	8	149	129.09	0.24	18474.07	9	4
junit.runner	3	184	1.76	2.07	69255.09	172.73	63	258	67	24	1341	0.40	0.07	3	38	122.73	0.67	6205.11	11	2
junit.textui	2	156	1.57	1.91	57310.68	163.01	76	217	55	20	1247	0.75	0.25	1	35	123.13	0.00	5720.95	9	3
org.junit	5	251	1.39	3.26	98323.41	84.06	801	368	93	68	2248	0.38	0.62	5	67	128.19	0.00	9789.64	5	3
org.junit.experimental	3	47	1.89	0.57	11140.98	120.43	0	65	17	0	395	1.00	0.00	0	9	120.43	0.00	1698.43	3	3
org.junit.experimental.categories	0	3	0.00	0.03	266.43	171.00	35	3	0	1	27	0.00	1.00	0	0	171.00	0.00	99.91	0	0
org.junit.experimental.max	7	158	1.62	2.06	46964.34	163.38	83	227	52	25	1350	0.88	0.12	1	32	123.49	0.00	6168.63	6	7
org.junit.experimental.results	6	71	1.22	0.74	12283.09	170.72	42	96	22	10	541	0.83	0.17	1	18	130.84	0.00	2221.51	3	5
org.junit.experimental.runners	1	6	2.00	0.07	960.06	151.71	20	8	2	2	57	1.00	0.00	0	1	118.33	0.00	205.76	2	1
org.junit.experimental.theories	10	172	1.62	2.28	58181.68	123.69	11	274	68	5	1533	0.67	0.13	3	42	123.69	0.20	6825.59	5	6
org.junit.experimental.theories.internal	4	160	1.83	2.16	59805.58	118.69	25	239	53	9	1391	0.60	0.40	2	29	118.69	0.00	6469.97	6	3
org.junit.experimental.theories.suppliers	1	20	2.00	0.29	7983.50	91.58	0	22	2	0	186	1.00	0.00	0	1	91.58	0.00	868.45	2	1
org.junit.internal	8	163	1.65	1.89	59839.62	168.94	41	227	51	7	1262	0.33	0.42	8	31	118.94	0.25	5681.58	7	4
org.junit.internal.builders	8	104	2.10	1.31	41184.19	168.66	24	163	42	8	892	0.70	0.30	3	20	118.66	0.00	3931.83	5	7
org.junit.internal.matchers	7	120	1.67	1.59	46632.08	123.29	21	174	50	6	1097	0.17	0.55	5	30	123.29	0.29	4766.24	6	1
org.junit.internal.requests	3	49	1.43	0.56	11200.98	154.05	17	66	10	4	391	0.67	0.33	2	7	114.16	0.00	1671.28	2	4
org.junit.internal.runners	9	291	1.75	3.99	151453.48	111.17	26	370	70	7	2350	0.53	0.47	7	40	111.17	0.00	11962.96	7	8
org.junit.internal.runners.model	3	48	1.45	0.46	8421.09	125.75	10	65	16	3	343	0.44	0.22	5	11	125.75	0.33	1394.64	3	4

This is presented in tabular form with a column for each of the metrics calculated at this level (remember that you can choose whether your HTML output shows all metrics or only those that you have marked as active). There is a row for each package and if any metric in the row has a valid range, and lies outside that range then either a red danger or an orange warning indicator will be displayed to the left of the row. The background of the offending metric(s) will also be set to either red or orange.

You can click on the link in the name column to drill down to the next level. This may or may not be active depending on the levels that you have selected to be displayed in the table.

All pages are basically laid out the same – one or more lines of dashboard gauges (if these have been activated) followed by a list of metrics at that particular level. Below this will be a table of data relating to the level below this with links to that level.

Batch Process – an example using HTML output

Using the JHawk command line jar it's very easy to automate the process of creating output from a very large number of Java files. The following example was done as an exercise at Virtual Machinery. You can also find a similar example for XML output in the Data Viewer documentation.

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 the SourceForge page here . 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 HTML output for each version in a separate directory 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 generatehtml.bat and looked like this –

```
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit34 -hp
C:\JUnitSourceTests\html34 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit35 -hp
C:\JUnitSourceTests\html35 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit36 -hp
C:\JUnitSourceTests\html36 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit37 -hp
C:\JUnitSourceTests\html37 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit38 -hp
C:\JUnitSourceTests\html38 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit40 -hp
C:\JUnitSourceTests\html40 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit41 -hp
C:\JUnitSourceTests\html41 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit42 -hp
C:\JUnitSourceTests\html42 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit43 -hp
C:\JUnitSourceTests\html43 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit44 -hp
C:\JUnitSourceTests\html44 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit45 -hp
C:\JUnitSourceTests\html45 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit46 -hp
C:\JUnitSourceTests\html46 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit47 -hp
C:\JUnitSourceTests\html47 -h index
java -jar JHawkCommandLine.jar -f .*\.java -r -l pcm -s C:\JunitSourceTests\junit48 -hp
C:\JUnitSourceTests\html48 -h index
```

Running the file took about 4 minutes. Now we had a directory containing the full HTML version of the JHawk analysis for each version of the software.

You can see the results of one of these tests in the html48 directory of the Export Samples download . The root file of the HTML file set is index.html in that directory.