JHawk 5 Documentation- Command Line Manual

Virtual Machinery
March 2012
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.

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	Demo
JHawk5CommandLineManual – Documentation for the Command line version of JHawk	No	Yes	Yes
JHawk5CreateMetric – Documentation explaining how to create new metrics that can be added to JHawk	Yes	Yes	No
JHawk5DataViewerManual – Documentation for JHawks DataViewer product	No	Yes	Yes
JHawk5EclipseManual – Documentation for the JHawk Eclipse Plugin	Yes	Yes	Yes
JHawk5Licensing – Licensing details for JHawk products	Yes	Yes	Yes
JHawk5UserManual – Documentation for the JHawk stand alone application	Yes	Yes	Yes
JHawk5UsingMetrics – 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	No

JHawk Command Line

Installation

The JHawk Command Line version is distributed as a single jar and a 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.

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.

Installing the sample Export Output

The sample export output from JHawk can be downloaded from the Virtual Machinery website (http://www.virtualmachinery.com/jhdownload.htm). You should install this in the ExportSamples subdirectory of your JHawk Distribution.

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] [-e encodingType] [-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 pattern 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 pattern 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. No argument is supplied to the –r option. All .java files in the directories selected using the –d 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 pattern which matches the files to be excluded from the analysis.

-xd Exclude directories - a pattern which matches the directories to be excluded from the analysis.

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 ",".

-e	File encoding. This can be used to set the encoding type of the Java files being analysed. E.g. UTF-8. The encoding type is sent as an argument to the option. This should only be required where files created under one file encoding regime are being analysed in a different one.

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
```

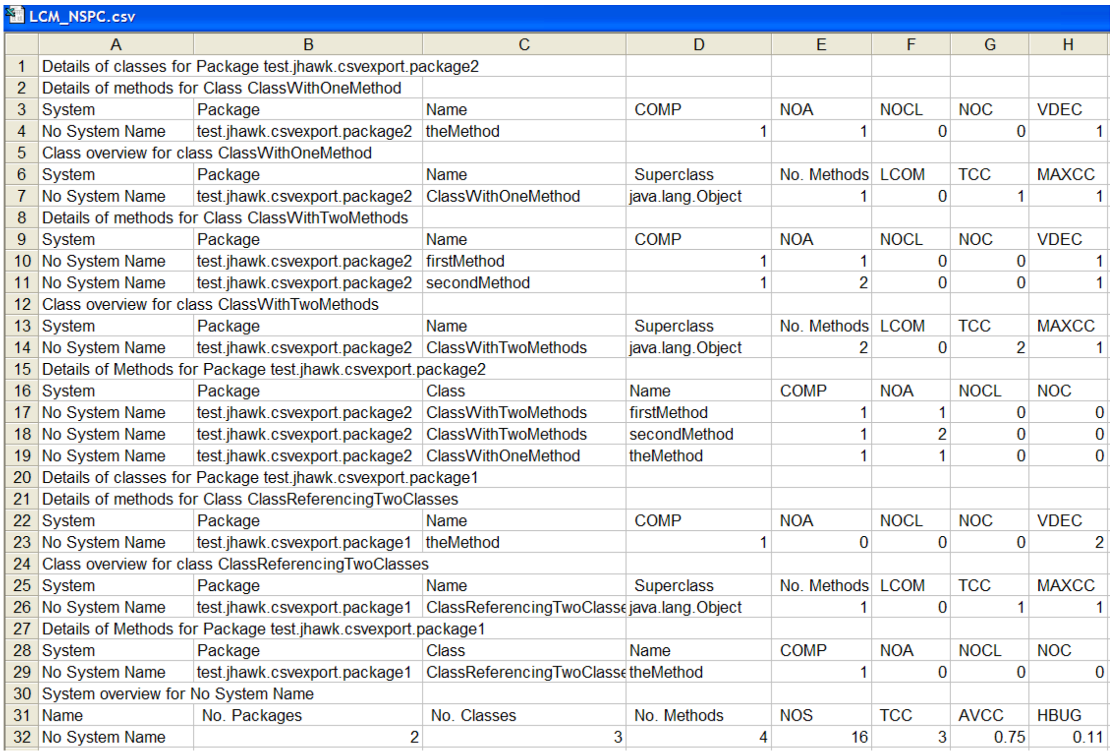
The following line will analyse all the Java source code in the directory C:\MySource and will generate a raw csv file at method level called mysource.csv.

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

Output Formats

CSV – Standard

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

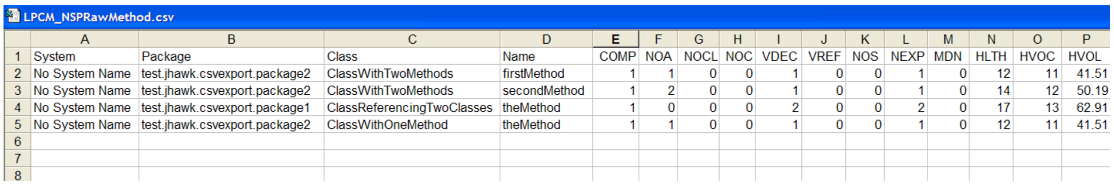


LCM_NSPC.csv

	A	B	C	D	E	F	G	H
1	Details of classes for Package test.jhawk.csvexport.package2							
2	Details of methods for Class ClassWithOneMethod							
3	System	Package	Name	COMP	NOA	NOCL	NOC	VDEC
4	No System Name	test.jhawk.csvexport.package2	theMethod	1	1	0	0	1
5	Class overview for class ClassWithOneMethod							
6	System	Package	Name	Superclass	No. Methods	LCOM	TCC	MAXCC
7	No System Name	test.jhawk.csvexport.package2	ClassWithOneMethod	java.lang.Object	1	0	1	1
8	Details of methods for Class ClassWithTwoMethods							
9	System	Package	Name	COMP	NOA	NOCL	NOC	VDEC
10	No System Name	test.jhawk.csvexport.package2	firstMethod	1	1	0	0	1
11	No System Name	test.jhawk.csvexport.package2	secondMethod	1	2	0	0	1
12	Class overview for class ClassWithTwoMethods							
13	System	Package	Name	Superclass	No. Methods	LCOM	TCC	MAXCC
14	No System Name	test.jhawk.csvexport.package2	ClassWithTwoMethods	java.lang.Object	2	0	2	1
15	Details of Methods for Package test.jhawk.csvexport.package2							
16	System	Package	Class	Name	COMP	NOA	NOCL	NOC
17	No System Name	test.jhawk.csvexport.package2	ClassWithTwoMethods	firstMethod	1	1	0	0
18	No System Name	test.jhawk.csvexport.package2	ClassWithTwoMethods	secondMethod	1	2	0	0
19	No System Name	test.jhawk.csvexport.package2	ClassWithOneMethod	theMethod	1	1	0	0
20	Details of classes for Package test.jhawk.csvexport.package1							
21	Details of methods for Class ClassReferencingTwoClasses							
22	System	Package	Name	COMP	NOA	NOCL	NOC	VDEC
23	No System Name	test.jhawk.csvexport.package1	theMethod	1	0	0	0	2
24	Class overview for class ClassReferencingTwoClasses							
25	System	Package	Name	Superclass	No. Methods	LCOM	TCC	MAXCC
26	No System Name	test.jhawk.csvexport.package1	ClassReferencingTwoClasse	java.lang.Object	1	0	1	1
27	Details of Methods for Package test.jhawk.csvexport.package1							
28	System	Package	Class	Name	COMP	NOA	NOCL	NOC
29	No System Name	test.jhawk.csvexport.package1	ClassReferencingTwoClasse	theMethod	1	0	0	0
30	System overview for No System Name							
31	Name	No. Packages	No. Classes	No. Methods	NOS	TCC	AVCC	HBUG
32	No System Name	2	3	4	16	3	0.75	0.11

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.



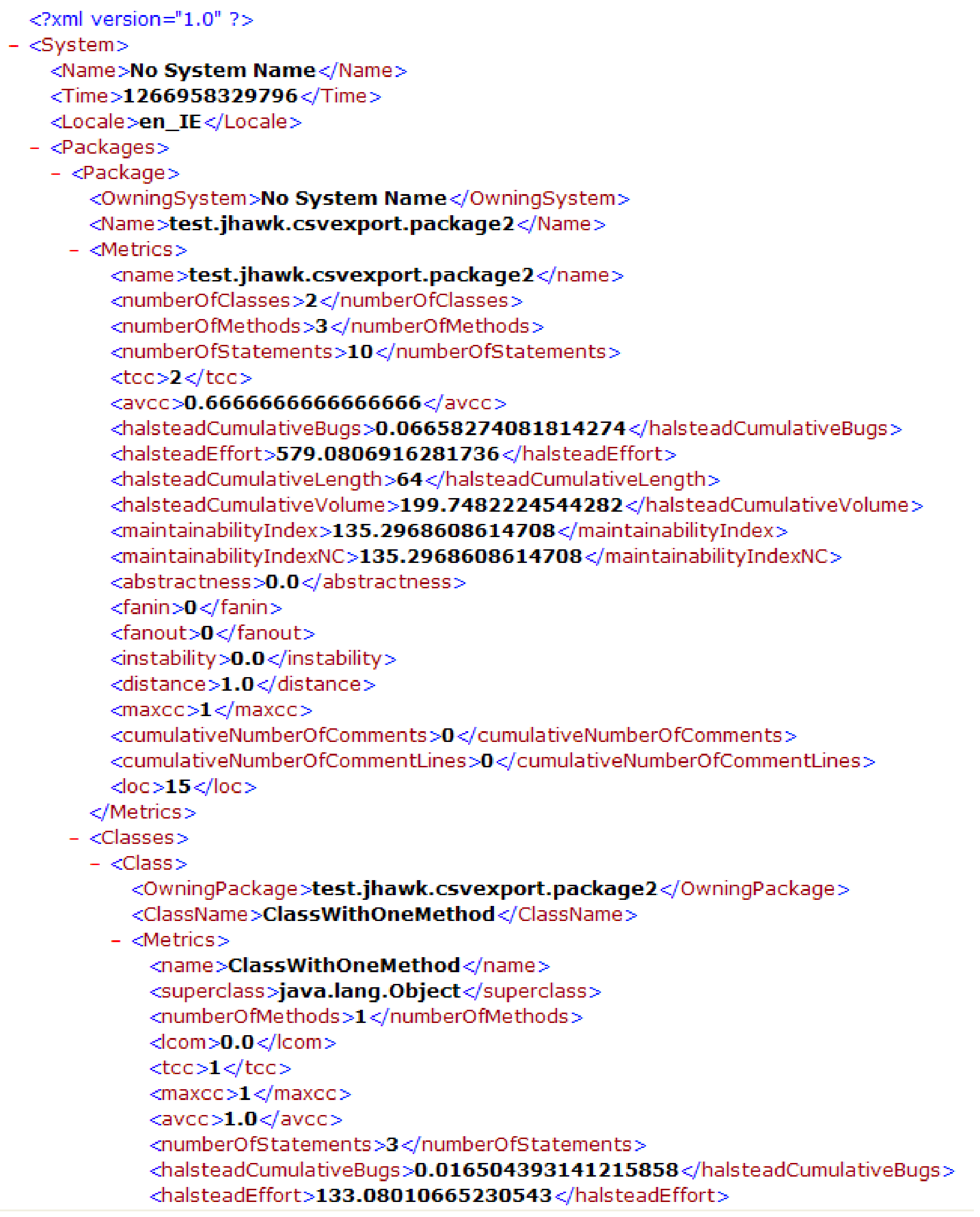
LPCM_NSPRawMethod.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	System	Package	Class	Name	COMP	NOA	NOCL	NOC	VDEC	VREF	NOS	NEXP	MDN	HLTH	HVOC	HVOL
2	No System Name	test.jhawk.csvexport.package2	ClassWithTwoMethods	firstMethod	1	1	0	0	1	0	0	1	0	12	11	41.51
3	No System Name	test.jhawk.csvexport.package2	ClassWithTwoMethods	secondMethod	1	2	0	0	1	0	0	1	0	14	12	50.19
4	No System Name	test.jhawk.csvexport.package1	ClassReferencingTwoClasses	theMethod	1	0	0	0	2	0	0	2	0	17	13	62.91
5	No System Name	test.jhawk.csvexport.package2	ClassWithOneMethod	theMethod	1	1	0	0	1	0	0	1	0	12	11	41.51
6																
7																
8																

You can find an example of a raw csv file in the ‘ExportSamples’ directory of the distribution (this has to be downloaded and installed separately from the product – see the section ‘Installing the Sample Export output’ above). 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

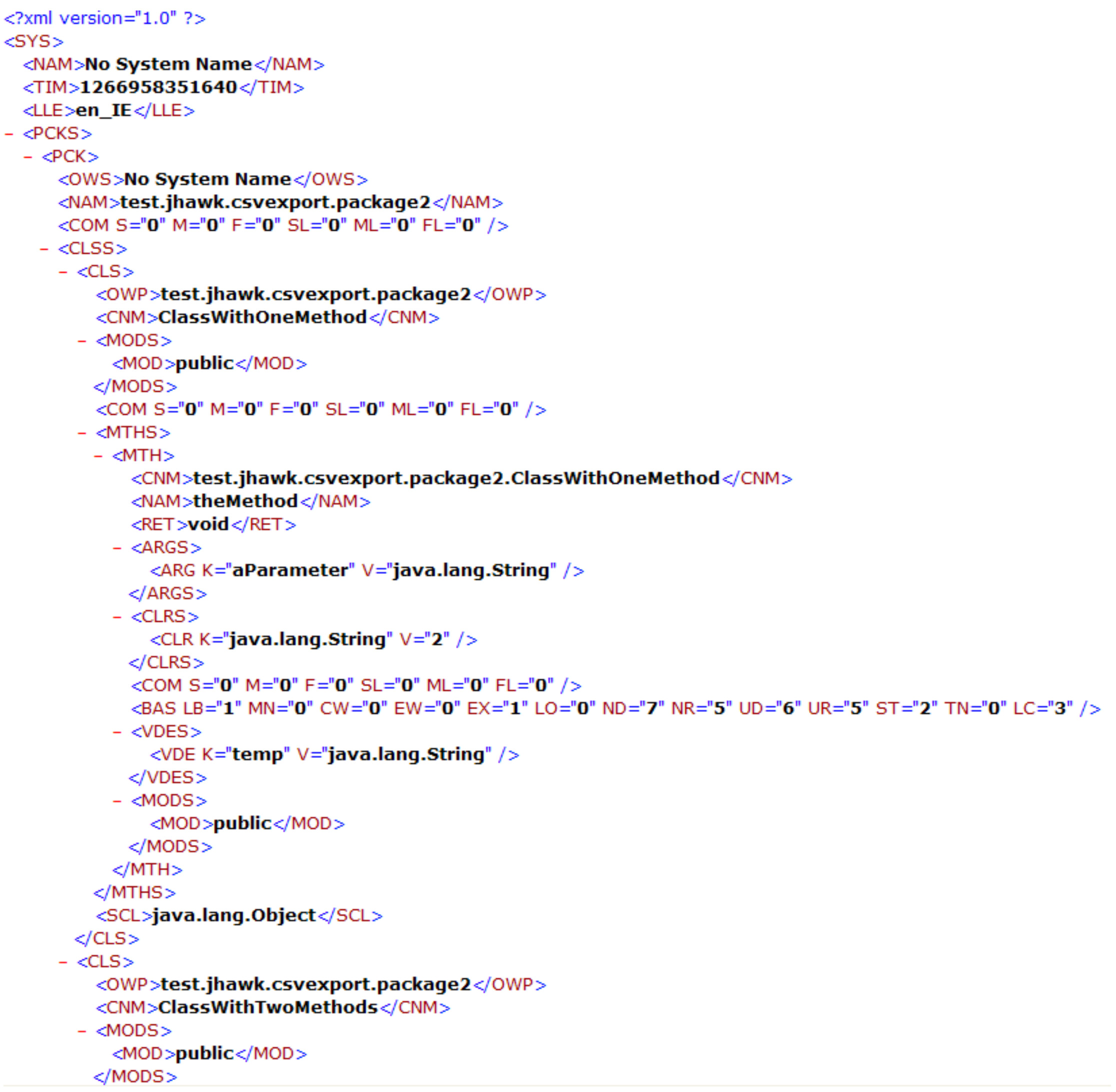
XML output can be written in two forms – Standard or JHawk Interchange . In the standard format xml output is written for the selected metrics (active or all) and at the relevant levels (System, Package, Class, Method etc). Sample output is shown below.



```
  <?xml version="1.0" ?>
- <System>
    <Name>No System Name</Name>
    <Time>1266958329796</Time>
    <Locale>en_IE</Locale>
  - <Packages>
    - <Package>
        <OwningSystem>No System Name</OwningSystem>
        <Name>test.jhawk.csvexport.package2</Name>
      - <Metrics>
          <name>test.jhawk.csvexport.package2</name>
          <numberOfClasses>2</numberOfClasses>
          <numberOfMethods>3</numberOfMethods>
          <numberOfStatements>10</numberOfStatements>
          <tcc>2</tcc>
          <avcc>0.6666666666666666</avcc>
          <halsteadCumulativeBugs>0.06658274081814274</halsteadCumulativeBugs>
          <halsteadEffort>579.0806916281736</halsteadEffort>
          <halsteadCumulativeLength>64</halsteadCumulativeLength>
          <halsteadCumulativeVolume>199.7482224544282</halsteadCumulativeVolume>
          <maintainabilityIndex>135.2968608614708</maintainabilityIndex>
          <maintainabilityIndexNC>135.2968608614708</maintainabilityIndexNC>
          <abstractness>0.0</abstractness>
          <fanin>0</fanin>
          <fanout>0</fanout>
          <instability>0.0</instability>
          <distance>1.0</distance>
          <maxcc>1</maxcc>
          <cumulativeNumberOfComments>0</cumulativeNumberOfComments>
          <cumulativeNumberOfCommentLines>0</cumulativeNumberOfCommentLines>
          <loc>15</loc>
        </Metrics>
      - <Classes>
        - <Class>
            <OwningPackage>test.jhawk.csvexport.package2</OwningPackage>
            <ClassName>ClassWithOneMethod</ClassName>
          - <Metrics>
              <name>ClassWithOneMethod</name>
              <superclass>java.lang.Object</superclass>
              <numberOfMethods>1</numberOfMethods>
              <lcom>0.0</lcom>
              <tcc>1</tcc>
              <maxcc>1</maxcc>
              <avcc>1.0</avcc>
              <numberOfStatements>3</numberOfStatements>
              <halsteadCumulativeBugs>0.016504393141215858</halsteadCumulativeBugs>
              <halsteadEffort>133.08010665230543</halsteadEffort>
```

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 DataViewer add on.



```
<?xml version="1.0" ?>
<SYS>
  <NAM>No System Name</NAM>
  <TIM>1266958351640</TIM>
  <LLE>en_IE</LLE>
- <PCKS>
  - <PCK>
      <OWS>No System Name</OWS>
      <NAM>test.jhawk.csvexport.package2</NAM>
      <COM S="0" M="0" F="0" SL="0" ML="0" FL="0" />
    - <CLSS>
      - <CLS>
          <OWP>test.jhawk.csvexport.package2</OWP>
          <CNM>ClassWithOneMethod</CNM>
        - <MODS>
            <MOD>public</MOD>
          </MODS>
          <COM S="0" M="0" F="0" SL="0" ML="0" FL="0" />
        - <MTHS>
          - <MTH>
              <CNM>test.jhawk.csvexport.package2.ClassWithOneMethod</CNM>
              <NAM>theMethod</NAM>
              <RET>void</RET>
            - <ARGS>
                <ARG K="aParameter" V="java.lang.String" />
              </ARGS>
            - <CLRS>
                <CLR K="java.lang.String" V="2" />
              </CLRS>
              <COM S="0" M="0" F="0" SL="0" ML="0" FL="0" />
              <BAS LB="1" MN="0" CW="0" EW="0" EX="1" LO="0" ND="7" NR="5" UD="6" UR="5" ST="2" TN="0" LC="3" />
            - <VDES>
                <VDE K="temp" V="java.lang.String" />
              </VDES>
            - <MODS>
                <MOD>public</MOD>
              </MODS>
            </MTH>
          </MTHS>
          <SCL>java.lang.Object</SCL>
        </CLS>
      - <CLS>
          <OWP>test.jhawk.csvexport.package2</OWP>
          <CNM>ClassWithTwoMethods</CNM>
        - <MODS>
            <MOD>public</MOD>
          </MODS>
```

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.

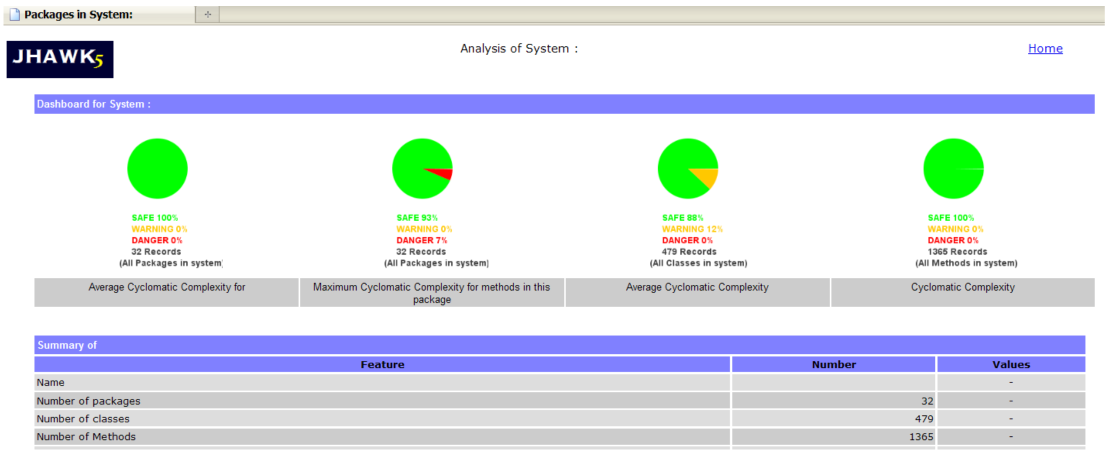
If you wish to compare files in the tree views (Text Compare and Graph) then all of your systems should have the same name (or all have no name). In the command line version you should use the –system flag followed by the name that you wish to give the system.

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.


Packages in System:

JHAWK5 Analysis of System : Home

Dashboard for System :

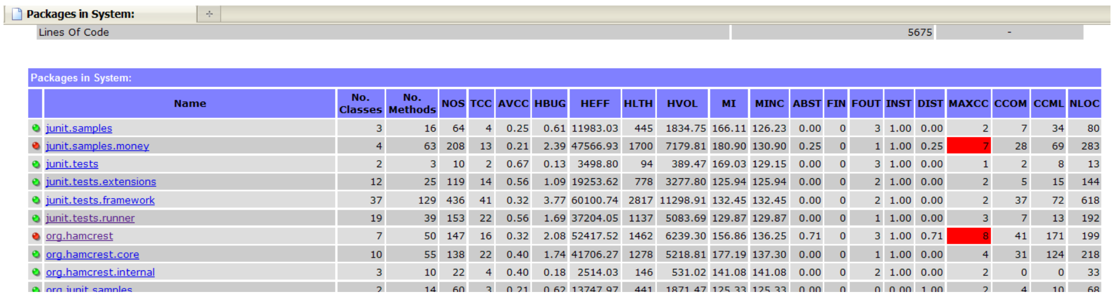
SAFE 100% WARNING 0% DANGER 0% 32 Records (All Packages in system)	SAFE 93% WARNING 0% DANGER 7% 32 Records (All Packages in system)	SAFE 88% WARNING 12% DANGER 0% 479 Records (All Classes in system)	SAFE 100% WARNING 0% DANGER 0% 1365 Records (All Methods in system)
Average Cyclomatic Complexity for	Maximum Cyclomatic Complexity for methods in this package	Average Cyclomatic Complexity	Cyclomatic Complexity

Summary of

Feature	Number	Values
Name		-
Number of packages	32	-
Number of classes	479	-
Number of Methods	1365	-

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:

Lines Of Code	5675	-

Packages in System:

	Name	No. Classes	No. Methods	NOS	TCC	AVCC	HBUG	HEFF	HLTH	HVOL	MI	MINC	ABST	FIN	FOUT	INST	DIST	MAXCC	CCOM	CCML	NLOC
🟢	junit.samples	3	16	64	4	0.25	0.61	11983.03	445	1834.75	166.11	126.23	0.00	0	3	1.00	0.00	2	7	34	80
🔴	junit.samples.money	4	63	208	13	0.21	2.39	47566.93	1700	7179.81	180.90	130.90	0.25	0	1	1.00	0.25	7	28	69	283
🟢	junit.tests	2	3	10	2	0.67	0.13	3498.80	94	389.47	169.03	129.15	0.00	0	3	1.00	0.00	1	2	8	13
🟢	junit.tests.extensions	12	25	119	14	0.56	1.09	19253.62	778	3277.80	125.94	125.94	0.00	0	2	1.00	0.00	2	5	15	144
🟢	junit.tests.framework	37	129	436	41	0.32	3.77	60100.74	2817	11298.91	132.45	132.45	0.00	0	2	1.00	0.00	2	37	72	618
🟢	junit.tests.runner	19	39	153	22	0.56	1.69	37204.05	1137	5083.69	129.87	129.87	0.00	0	1	1.00	0.00	3	7	13	192
🔴	org.hamcrest	7	50	147	16	0.32	2.08	52417.52	1462	6239.30	156.86	136.25	0.71	0	3	1.00	0.71	8	41	171	199
🟢	org.hamcrest.core	10	55	138	22	0.40	1.74	41706.27	1278	5218.81	177.19	137.30	0.00	0	1	1.00	0.00	4	31	124	218
🟢	org.hamcrest.internal	3	10	22	4	0.40	0.18	2514.03	146	531.02	141.08	141.08	0.00	0	2	1.00	0.00	2	0	0	33
🟢	org.junit.samples	2	14	60	3	0.21	0.62	13747.97	441	1871.47	125.33	125.33	0.00	0	0	0.00	1.00	2	4	10	68

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 these tests in the ExportSamples directory of the distribution which contains the html48 directory. The root file of the HTML file set is index.html in that directory.