Table of Contents

SA2.4.3 Metrics Tools Knowledge Base: Report generator documentation....1
The reports....1
Detailed report....1
EMT report....1
EU report....1
Installation and building....1
Checkout....1
Dependencies....2
Configurations....3
Compiling....3
Example how to quickly install, build and run the Report Generator....3
Building and running with ETICS....4
Running the report generator....4
Log files....5
Data sources....6
xml file....6
.xml....7
ETICS repository - plugin data....7
.xml....7
Bug listings....7
Daily report generation by ETICS....8
XML Chart definition....8
Adding a new metric to the report generator....9
JasperReports....10
iReport....10
Dataset....10
Report templates....10
JODReports....10
Creating a data map for filling the report....10
Inserting images....10
Inserting text....11
Looping through an array....11
Component Mapping....11

SA2.4.3 Metrics Tools Knowledge Base: Report generator documentation

The report generator queries the ETICS repository with SQL and XPath which produces XML pages for each metric in each component. Then it parses the data and creates HashMaps of the data.

With the data stored as HashMaps it generates a JFreeChart dataset. This dataset is then used by JFreeChart to generate a graph. The graph is then put into a JasperReport or JODReport along with relevant data.

The reports

The report generator can create three different reports. New reports can be made with a few steps:

- Create a new chartsDefinition[ReportName].xml (for example chartsDefinitionEMT.xml or chartsDefinitionEU.xml). This file will contain the configuration of the new report.
- Include the location of the xml file in reportgenerator.properties, by defining the following key: ChartsDefinition[ReportName] (for example: ChartsDefinitionDetailed=!http://svnweb.cern.ch/world/wsvn/emisa2/org.emi.reportgenerator/trunk/project/xm

Detailed report

This is a long pdf document to be generated after each nightly build. It is aimed at the product teams so that they can see how well they are doing in the different QA metrics. The first section contains an overview of the whole project and it is followed by a section for each product team (PT), which contains overview charts for the PT and product specific charts. Here the product teams can compare its different products and components within a product. This report is generated as a pdf file, using the JasperReports library.

EMT report

The EMT report is a stripped down version of the detailed report with metrics that are useful for the weekly EMT meetings (Mondays at 15:00). The only changes from the detailed report are made in the **chartsDefinitonEMT.xml**.

EU report

This is a report aimed to be a backbone to deliverables to the EU. The main feature is that it is generated as an Open Office Document Format (ODF) so that the plots can be commented in OpenOffice or Microsoft Word. It should contain more or less the same plots as the first section of the detailed report, the product team specific charts are not included. This report is generated using the JODReports library.

Installation and building

Checkout

The report generator's source files are under SVN. You can access them with a web browser here or here.

The checkout command is:

```
svn co https://svn.cern.ch/reps/emisa2/org.emi.reportgenerator/trunk
```

you can also checkout anonymously (you won't have commit permissions):

```
svn co http://svnweb.cern.ch/guest/emisa2/org.emi.reportgenerator/trunk
```

Dependencies

At the time of the writing, the Report Generator had the following dependencies:

Name	Version	Default location where it should be installed
Axis	1.2	${ext.library-location.dir}/axis
Batik		${ext.library-location.dir}/batik
Commons-beanutils	1.8.0	${ext.library-location.dir}/commons
Commons-cli	1.2	${ext.library-location.dir}/commons
Commons-collections	3.2	${ext.library-location.dir}/commons
Commons-digester	2.1	${ext.library-location.dir}/commons
Commons-io	1.4	${ext.library-location.dir}/commons
Commons-javaflow	20060411	${ext.library-location.dir}/commons
Commons-logging	1.1	${ext.library-location.dir}/commons
Freemarker	2.3.16	${ext.library-location.dir}
itext	2.1.4	${ext.library-location.dir}
Jasperreports	4.0.0	${ext.library-location.dir}/jasperreports/dist
Jcommon	1.0.16	${ext.library-location.dir}/jfreechart
Jfreechart	1.0.13	${ext.library-location.dir}/jfreechart
Jodreports	2.4.0	${ext.library-location.dir}/jodreports
Log4j	1.2.14	${ext.library-location.dir}
Slf4j	1.5.0	${ext.library-location.dir}/slf4j-1.5.0
Xom	1.2.6	${ext.library-location.dir}

Please refer to the ETICS configuration (**org.etics.QA.report-generator**) or the project/build.properties file in SVN for an updated list of external dependencies. Also in that file, you can define ${ext.library-location.dir} and also change the default relative location of each library.

The report generator also depends on **org.etics.QA.report-generator-jaxb-stub-java**. This is a library generated with Java Architecture for XML Binding (JAXB), from XML schema definitions (.xsd files). Basically it generates java files that represent the xml structure defined by the .xsd file, which makes possible for the report generator to easily parse XML files compliant to that structure. You can download the library from the ETICS repository or generate it locally. To generate it locally, you need to checkout the .xsd files and the build.xml and use the Ant tool. You also need to get JAXB.

To checkout the .xsd files and the build.xml:

```
svn co https://svn.cern.ch/reps/emisa2/tools/trunk/org.etics.QA.report-generator-jaxb-stub-java/
```

You can get JAXB directly from http://jaxb.java.net/ or from ETICS: http://eticssoft.web.cern.ch/eticssoft/repository//externals/jaxb/2.1.2/noarch/jaxb-2.1.2.tar.gz.

You need to point Ant to the JAXB library location. You can either modify the *jaxb* property in build.xml or, when calling Ant, by passing the argument: "-Djaxb=*[location]*".

To generate the library, run the Ant tool in the base directory (or point it to the build.xml). Afterwards, you should have in the lib folder the library, named reportGeneratorJaxbStubs.jar. This will be needed when compiling the report generator.

Configurations

Before compiling, **you need to configure the project/build.properties file** with the location of the libraries, including the library generated by JAXB. This file is used by the Ant tool when compiling or performing other actions. Alternatively, you may specify these properties when running Ant, with arguments in the form "-D*property*=*value*".

This means that you need to point, in that file, ***ext.library-directory*** **to the location of the dependency jar files**. Also, **make sure the dependencies are installed in folders as seen in the following *ext.library-location.dir* properties** (also shown in the table in *dependencies* section), or alternatively, change these properties as well. As already referred, you also need to **point where your JAXB-generated .jar file is located** , by correctly defining the property _ext.reportgenerator-stubs.dir_, either in the properties file or by specifying it as an argument when running Ant.

Compiling

The report generator can be compiled with Java version 1.5 or greater and it is built with ANT using the `build.xml` file which gets its properties from the `build.properties` file in the project folder.

There are many targets defined in the build.xml file, that can be executed with Ant, as listed in the table:

Target Name	Action
prefetch	Gets libraries from the directories pointed in the configuration file and copies them to the local endorsed/ folder
prefetch.done	Checks if all necessary libraries are in endorsed/ folder
init	Creates local directories
compile.java	compiles .java files
compile.jasper	compiles .jrxml templates to .jasper files
jar	Builds jar
install	installs the application in a specific directory. It requires defining install.dir

You can also get the target list doing: ant -p. The jasper report library uses .jasper files, which are compiled descriptions of the report structure. These .jasper files need to be generated from .jrxml templates, before running the report generator. This can be easily done using Ant. The target, defined in the build.xml file, is "compile.jasper". This only has to be done again when the jrxml files are updated.

Some of the commands trigger other, so in practice it is not necessary to type all the targets - check the following example.

Example how to quickly install, build and run the Report Generator

Before building the Report Generator, you need to download and install the dependencies. Please refer to *Dependencies* and *Configurations* sections. You need then to point *ext.library-location.dir* to the base directory where your libraries are, and *JAXB_generated_lib_location* to the location of your JAXB-generated .jar file. The libraries should be correctly installed in folders as shown in the table in the *dependencies* section (you can, alternatively, change this hierarchy in the project/build.properties file).

```
~/tmp$ sudo apt-get install ant #only if you don't have Ant installed!

~/tmp$ svn co https://svn.cern.ch/reps/emisa2/org.emi.reportgenerator/trunk
~/tmp$ cd trunk
~/tmp/trunk$ ant prefetch -Dext.reportgenerator-stubs.dir=[JAXB_generated_lib_location] -Dext.lib
~/tmp/trunk$ ant compile.jasper
~/tmp/trunk$ ant jar
~/tmp/trunk$ ant install -Dinstall.dir=[install_dir]
~/tmp/trunk$ cd [install_dir]
```

```
~/tmp/trunk$ java -jar emiReportGenerator.jar -report detailed --volatile emi_R_1_rc_metrics
```

You should now have in your install_dir the file: emiReportGenerator.jar. As previously referred, you can alternatively define the properties in project/build.properties instead of defining them when running Ant.

Building and running with ETICS

There are two configurations available in ETICS: HEAD and DEPLOY. They are located in org.etics.QA.report-generator.

- HEAD is used to build the report generator from scratch, including the jaxb library. It should be launched as a remote build.
- DEPLOY contains test commands that runs the binaries created by the HEAD commands and generates reports. It should be launched as a remote test.

The jobs are usually run on SL5x64. There are two options that need to be checked when submitting jobs, in order to properly build and test (run) the report generator in ETICS.

- Checkout: propagate environment and properties from *etics-dev*.
- Packaging & Repository: Publish artifacts to a Custom Volatile Area: emi-qa-reports

Running the report generator

The report generator can be run with several command line options. It uses commons CLI to parse the command line arguments. The following table gives an overview of the available options.

Description	Short option	Long option	Example
Which report to generate. It will define which chart definition xml to load.	-r	--report	-r detailed -r EMT --report EU
Black and white textures in report.	-b	--blackWhite	-b --blackWhite
Sort names alphabetically (if not used, names are sorted by values).	-s	--sort	-s --sort
Specify ETICS Project name. If not specified, 'emi' will be used. It is used when requesting plugin results and when trying to find out the last buildId.	-p	--project	--project etics -p gLite
If specified, the ETICS volatile name will be included in the query when trying to find the last buildId, filtering the results.	-v	--volatile	
If specified, the ETICS volatile name will be included in the query when when trying to find the last buildId, filtering the results.	-c	--configuration	-c emi_B_1_rc3
Specify a ETICS BuildId, from which build info and plugins results are collected. If not specified, a day-by-day search will be performed, starting in the current date.	-i	--buildId	--buildId 428b61dc-160f-434a-b935-a36a33c4faf9

Specify a build platform. It will be used once, in the request of BuildStatus xml, together with BuildId. If not specified, it will be used the one found while finding the buildId or, if this information is not available, it will use sl5_x86_64_gcc412EPEL.		--platform	--platform sl5_ia32_gcc412EPEL
Specify build date to create a report with historical data. Will override all build status and bugListing xmls and builddateplugins. Use the following fomat: yyyy-MM-dd.		--builddate	--builddate 2010-01-30
Specify build date to create a report with historical data. Only used by the plugins, bugListings and build-status.xml needs to be specified (see below). Use the following fomat: yyyy-MM-dd		--builddateplugins	--builddateplugins 2010-01-30
Specify old build-status.xml URL.		--buildstatusxmlurl	--buildstatusxmlurl http://.../build-status.xml
Specify old gLite bugListing.xml URL.		--glitebuglistingurl	--glitebuglistingurl http://.../gLite-bugListing.xml
Specify old ARC bugListing.xml URL.		--arcbuglistingurl	--arcbuglistingurl http://.../ARC-bugListing.xml
Specify old UNICORE bugListing.xml URL.		--unicorebuglistingurl	--unicorebuglistingurl http://.../UNICORE-bugListing.xml
Specify old dCache bugListing.xml URL.		--dcachebuglistingurl	--dcachebuglistingurl http://.../dCache-bugListing.xml
Specify chart configuration XML.	-x	--xml	TODO

For example, to generate the detailed report using build data from the emi_R_1_rc_metrics ETICS configuration:

```
java -jar emiReportGenerator.jar -report detailed --volatile emi_R_1_rc_metrics
```

You should now have the report in reports/Detailed_report.pdf.

The report generator will query the ETICS repository for a build to generate a report for. The build selection will only affect the metrics generated from ETICS, not the bug tracking metrics. It will generate a query that looks for any builds matching the project name done in the last 24 hours and select the newest build. If no build is found it goes 24h back until a build is found.

By default the only restriction in the query is that the project must match 'emi', unless overridden with a different name in the --project option. Further specifications can be made by specifying the volatile area and configuration. By default it will include all volatile areas and all configurations. If an ETICS build ID is specified, it will not query for a build, but use the build ID instead to query for metrics. If a build ID is specified it overrides --project, --volatile and --configuration.

Log files

After running the Report Generator, you will find in reports/logs many log files. Here is a short description of the information you can find in these text files.

BuildStatus.txt : Contains a listing of the components that were successfully mapped to a product team. It also indicates which components failed to build and if it could be due to some dependency failing. It also shoes the dependencies and externals of each Product team (components that do not belong to the same PT), and detects which dependencies don't correspond to any of the component found.

dupeList.txt : Contains lists of components detected as repeated from each other. In each list, the components share most probably the same code. They are found by comparing SLOC values and by comparing the component names in the cases where SLOC values equal 0.

Mapping.txt : Output of the ProductTeams/Product/ETICSsubsystem/ETICScomponent hierarchy used by the report generator, parsed from the mapping XML file.

NotInMapping.txt : Shows which components found in the build XML have no mapping to any Product Team. It does not include components found as dependencies. Components without mapping are generally excluded from the charts.

NotInBugMapping.txt : It is a list of bug components found in one of the bug XML files, that are not mapped to any product. In these cases, the component name is used as the product name. Thus, these components are not excluded from the charts, even though are probably not displayed under the correct category.

Plugins/NotInMapping[PluginName].txt : It lists components found in the data from a plugin that are not mapped to any product, and the corresponding total values in the newest build date. Hence, they will not be included in any product or product team, and are therefore excluded from charts concerning data from the plugin. It is useful to check the list corresponding to SLOC, as a first step to verify if a excluded component is "dummy" or if it really has code associated to it.

listing_*.txt : Listings of bugs. The name of the file should indicate which bugs are listed in it. The file listing_allBugs.txt contains all the bugs parsed from all the bug trackers.

Data sources.

The report generator collect data from several resources:

- ChartDefinition xml file.
- EmiEticsMapping.xml
- ETICS repository direct query for plugin data
- BuildStatus.xml generated by an ETICS build
- Bug listings (in xml) provided by each middleware's bug tracker.

The exact data collected depends on the running parameters. Next, some more details are provided for each listed resource.

xml file

This file defines the content of the report, by specifying which charts should appear in the report, among other things. It has to follow the structure defined in ChartDefinition.xsd (used by JAXB to generate the library that parses the xml files that follow this structure). A brief description of some of the xml tags is in "XML Chart definition" section. Many different definitions in different xml files may exist. The report generator checks for the --report option (see "Running the report generator" section) and reads the reportgenerator.properties file to find out which file it should use.

.xml

This file defines the mapping between ETICS components, products and product teams. It is located in the repository - In the report generator source, check the src/reportgenerator.properties file to find out where the Report Generator looks for it.

ETICS repository - plugin data

When ETICS runs a configuration and builds components, it runs as well plugins that generate data concerning each component and regarding some metric. For example, one of the plugins counts the single lines of code (SLOC) of each component. The report generator gets this data querying the ETICS repository using XPATH. In these queries, it uses the project name, plugin name and build ID. It is also possible to do queries without a build ID, specifying a date interval instead. This is used to find builds and to collect data from various builds to generate trend charts. You can find an overview of all the nodes in the repository here.

If no Build ID is given as an argument, the report generator looks for the newest build it can find, complying with any specifications made in the command line options. In this case, the first it does is to check for any SLOC count data in a period of 24 hours and backtrack until it finds a build that matches all the properties specified or until it reaches 500 loops (days). When it has found the SLOC data, it saves it along with the **build ID** and **platform name**, which are later used to fetch other data.

The build ID is used to find data related to other metrics, generated by other plugins. For example, to get data from FindBugs :
http://etics-repository.cern.ch/repository/query/metrics/1/1/xpath//etics:metrics[@etics:name='FindBugs'+and+@etics:

For PMD and Checkstyle the values are stored in the ETICS repository in categories, so the report generator loops through all the categories and saves them together.

.xml

The build-status.xml is made by ETICS after each build. It is used to be able to tell if building a component was successful or not. It also has information regarding dependencies and externals. The build-status.xml is found by adding the build ID and platform name to a predefined URL, for example:

http://etics-repository.cern.ch/repository/reports/id/9a85233b-0c2a-4f4a-9152-bfb27efc5e2a/sl5_ia32_gcc412EPEL/-/r

The BuildId and Platform can be provided as a argument to the program, otherwise it is found while querying the ETICS repository as described in the previous sub-section.

After 30 days or 5 builds the build-status.xml of a build is no longer available in ETICS and is therefore transferred every day to the AFS space. This will allow in the future to create trend charts with historical information regarding the builds. The build status xml for a specific day can be found in the following link, by replacing [DATE] with the date in the "yyyy-mm-dd" format:
http://emiqa.web.cern.ch/emiqa/reports/qa-report_[DATE]/EticsQAReports/xml/build-status.xml For example:
http://emiqa.web.cern.ch/emiqa/reports/qa-report_2011-05-17/EticsQAReports/xml/build-status.xml

Bug listings

The bug listings are provided by the middleware's bug trackers, following a schema provided by SA 2.3. The XML files contain both bug reports and feature requests for all the time period.

The bug tracker URLs are specified in the reportgenerator.properties file:

- http://lxbra1902.cern.ch/EMI/BugListing/gLite-BugListing.xml
- http://lxbra1902.cern.ch/EMI/BugListing/UNICORE-BugListing.xml
- http://bugzilla.nordugrid.org/ARC-BugListing.xml
- http://www.dcache.org/downloads/emi/buglisting/dCache-BugListing.xml

Some additional notes: Each bug is mapped to a **comma separated list** of products that have bugs. Bugs that are not successfully mapped to one or more products will have the product name replaced with the component name (and their names dumped to NotInBugMapping.txt). Bugs from all middlewares are merged into a common list.

Daily report generation by ETICS

There is an emi-qa-reports volatile area in ETICS where the report generator is built and reports are generated every morning at 07:00. It first builds the report generator's HEAD configuration, then it uses the DEPLOY test configuration. This runs the jar file that was just built. It then moves all the files into the reports directory so that they are reachable from the report. The report index page is named with an "index-custom-*" prefix which makes it show up in the left menu of the main build report (QA-reports).

There is also created an EMT report for use in the weekly EMT meetings.

Logs and build reports are also moved into the reports directory so that they can be used as documentation or data later (Currently, ETICS only keeps the latest 5 build reports.).

The entire reports directory is moved to an AFS location and is kept there. After thirty days the reports are only available in a tar.gz file.

XML Chart definition

To make it easier to maintain and add new metrics to the report, there are XML files that define the structure of the report and which charts are generated.

Here, some of the key XML tags are presented. For a complete list, look at the XSD schema in the repository: org.etics.QA.report-generator-jaxb-stub-java.

chart

- Main tag for a chart.

metadata

- Gives information on the chart's title, subtitle, axis titles, description and link.

dataset

- Specifies what kind of dataset to create and what data to use to create that dataset. The correct classes are selcted from a series of if sentences in chart_definition.DatasetDefinition.java. If a new data type or dataset is created it needs to be added here.

plot

- Specifies what kind of plot which should be used. It also has the following boolean parameters:

- noTitle - Don't include a title
- integer - All chart values are integers, use integer values in the axis values.
- stacked - Bar chart should be stacked.
- averageLine - creates an average line in the chart.

configuration

- Each chart tag can have multiple configurations. For each configuration the reportgenerator will create a chart. The configuration tag allows for specification of priorities to be included and the colors used on each series.

group

- The group tag is mandatory for all charts. It is used to group together different charts that belong together. It allows us to create several configurations for agroup of charts that will follow each other in the same order. An example is the PriorityBug metric where a Box and Whisker, Scatter plot and two bar charts are drawn for each priority. The group tag makes sure that all immediate priority charts are created and together before it starts with the high priority charts.

pageBreak

- Set this to true if you want a page break after the last configuration of this chart (i.e after all possible priorities and zoomed charts).

zoom

- Set this to true if you want a zoomed version of the chart. Only works for bar charts.

Adding a new metric to the report generator

This information is outdated! Here you will find how to add metrics data generated by a new plugin to the reports.
The first step is to make the report generator load the data. For this, you need to edit the reportgenerator.properties file. Besides specifying the name of the plugin, some plugins have multiple categories. If the metric is stored in the ETICS repository in several different categories, the report generator needs to query all of them separately. So these are the required modifications:

- Add the plugin name to the "PluginNames" property.
- Add the ETICS categories to the "[PluginName]_Categories", in a comma separated list. Note that these strings are the ones used when querying ETICS and is case sensitive.
- If you want to ignore some of the categories (and have their data summed under a common category named "other"), create a property with the name "[PluginName]_Include", listing the categories which should be created and therefore not ignored.

Then, you need to modify the chart definition xml file, to effectively use the plugin data. The easiest way is to copy the definition of an existing chart similar to the way you want to create, and just modify the data to be used:

- There should be a dataset/dataSource tag, which should be set to "plugin".
- Change the string specified under dataset/dataSelect tag to the name of the new plugin, in lower case. If the chart uses trend data, it should be "[pluginName]_trend".
- In the link/suffix tag you need to add the link to the report generated by the ETICS build from the **reports** directory. For example, the link for PMD plugin is pmd/index.html.

JasperReports

http://jasperforge.org/projects/jasperreports

JasperReports is a report generation library for Java. It takes in JFreeChart objects as JCommonDrawableRenderer and Strings and fills them into a predefinied report template.

JasperReports has to different file types.

- **.jrxml** is the uncompile report template with an xml structure
- **.jasper** is the file that JasperReports fills with data. This is a compiled version of the jrxml file. The jrxml can be compiled by iReport, but in the report generator it is compiled in the build.xml target **compile.jasper**

iReport

http://jasperforge.org/projects/ireport

iReport is used as a graphical user interface to create report templates for JasperReports. It also has an xml viewer to edit the file directly and a preview tab were you can fill the report template with a dataset. Look in the ireport package for datasets you can use.

Dataset

Report templates

The report generator uses three report templates (jrxml files). One is the main report templates. The others are subreports. Subreports can be called for every dataset instance in the main report. One subreport creates the table of contents, the other creates a list of missing components that should be in the graph based on the language they are written in.

JODReports

http://jodreports.sourceforge.net/

JODReports was previously known as JOOReports, please observe that you have to use the keyword **jooscript** in the report template, not jodscript.

The JODReport template is made from an EMI Template, converted to Open Document Format (ODF). It is then filled with placeholders for the data that is to be filled by the report generator.

Creating a data map for filling the report.

The data map contains png images, strings and string arrays.

Inserting images

For graphs it is necessary to insert a picture, the picture then needs to be named like this so the JODReports engine can recognize it:

```
jooscript.image(imageName)
```

The *imageName* must correspond to the key in the map given to JODReports createDocument method.

Inserting text

For text fields it is sufficient to insert an **input field** with **jooscript** as the **reference** and ${textKey} in the textfield. The textKey must again correspond to the key in the data map. The input field can be found in OpenOffice by going to Insert -> Fields -> Other.. or press Ctrl+F2. You will find it under the Functions tab.

Looping through an array.

It is also possible to loop through an array.

```
@table:table-row
[#list nameList as name]
@/table:table-row
[/#list]
```

Creates a table row for each item in nameList. In the report you need to have an inputField as descibed above that references name. This field needs to be put in a table (with one row). To insert the script, go to Insert -> Script.. the script type needs to be **jooscript** and the radio button should remain at text. The script will be insert at the text cursor. Make sure that it is placed before the reference to the name variable.

I was able to loop lists of strings of failed or wrongly configured components, but creating nested loops was not successful. The same goes for looping over image files as they were in some cases put on top of each other and had to be adjusted manually in order to be visible.

Component Mapping

The categorymapping package has an interface that returns a list of ETICS components when given a category name from a bug tracker as input. This can be tested with the CategoryMappingTest JUnit test case.

The schemas for the component mapping can be found at
https://svnweb.cern.ch/trac/emisa2/browser/metrics/trunk/src/schema

-- DuarteMeneses - 12-07-2011 -- LarsBarlindhaug - 11-Mar-2011

This topic: EMI > SA243KnowledgeBase
Topic revision: r52 - 2011-08-03 - DuarteMeneses