Sponsored through Framework Programme Sixth (Call 5) by the European Commission

Document Information

Version: 2.0
Date : Dec 8, 09
Pages : 29

Owning Partner:
CETIC

Author(s):
Frédéric F. Monfils (CETIC)
Jacques Flamand (CETIC)

Reviewer(s):
José Ruiz (AdaCore)
Nicolas Devos (CETIC)

To:
European Commission

Purpose of distribution:
Documentation of the QualOSS platform. Submitted to EC for review

The QUALOSS Consortium consists of: CETIC (BE), Facultés Notre Dame de la Paix à Namur (BE), Universidad Rey Juan Carlos (ES), Fraunhofer IESE (DE), ZEA Partners (BE), MERIT (NL), AdaCore (FR), PEPITe (BE)

Status:

[ ] Draft
[ ] To be reviewed
[ ] Proposal
[X] Final/Released

Confidentiality:

[X] Public - Intended for public use
[ ] Restricted - Intended for QUALOSS consortium only
[ ] Confidential - Intended for individual partner only

Deliverable ID: **D2.4**

Title:

Reference Guide to the QualOSS platform

Deliverable: D2.4

Title: Reference Guide to the QualOSS platform

Executive Summary:

The QualOSS Platform helps in following the QualOSS Methodology described in Deliverable 4.1 QualOSS Methodology. This document, in its current version, is the User Guide of the QualOSS as of version 1.1. The Specification and Requirements of the QualOSS Platform are described in the Deliverable 2.1 and the implementation of the QualOSS Platform is done in Deliverable 2.2.

The QualOSS Platform has been designed to support the methodology by implementing a user interface defined for the various roles:

- The System Administrator: this user is responsible for the installation of the platform
- The QualOSS Platform Administrator: this user is responsible for the configuration of the platform
- The various QualOSS End Users:
  - The QualOSS Basic User: this user can request a QualOSS assessment or retrieve QualOSS assessment results obtained from a previous assessment.
  - The QualOSS Advanced User: this user can import measures and indicators from intermediate files, or can alter the Quality Model to use in an assessment, and can perform an assessment by specifying the appropriate data to the QualOSS platform.
  - The QualOSS Expert User: this user can add new metrics to connectors and create a new connector.

The remaining of this document describes the different guides for the listed roles.

Section 2 describes the guide for the system administrator. It lists the operations installing the QualOSS platform as well as prerequisite tools that must be installed by a system administrator on the server before using the QualOSS Platform.

Section 3 describes the guide for the QualOSS platform administrator. This section describes how to configure the QualOSS Platform and the configuration files prior to launching the QualOSS platform on a selected assessment.

Section 4 describes the guide for the expert user. This section explains the actions that an expert user can perform. More specifically it explains the architecture of the platform and how the platform can be extended. It can be considered as a Developer's Guide.

Section 5 describes the guide for the advanced user. This section describes how to provide the needed information about an F/OSS Endeavor in order to allow its proper assessment.

Section 6 describes the guide for the basic user. This section describes how it is possible to retrieve the results of analyses.

TABLE OF CONTENTS

1. Introduction

The purpose of this document is to provide the needed instructions to install, configure and run the QualOSS Platform. This document also describes the architecture and some details about the implementation so that it is possible to extend the platform with validated tools and metrics.

QualOSS is a tooled methodology in the sense that this methodology needs a platform to store in a structured way the information gathered during an analysis, but also to perform automatic analysis when applicable. Automating an analysis requires the detailed definition of the artefact (work product on which measurements are performed), the detailed definition of the metrics (how to perform the measurement) and the location of the data sources (where the artefacts are made publicly available).

The QualOSS platform supports 2 use cases :

Use case 1 : import assessment results from separate spreadsheets

The measurements needed for the analysis are performed independently from the QualOSS platform and collected in separate spreadsheets (one per characteristic). The advantage of this approach (in comparison with a full automated process) is that the analysis process can be better controlled and that some specific aspects of the artefacts can be taken into account. Moreover, the analysis process can be performed in a decentralized way without direct access to the centralized QualOSS platform, which can be more convenient in some cases. Tools are available to ease and automate as far as possible the measurement process. The spreadsheets also contain formulas that allow to compute automatically the value and the colours of the indicators on the basis of the measurement values. The spreadsheets can then easily be converted into simple Comma Separated Values (CSV) files that can in turn be used for introducing (“importing”) the results of the analysis in the QualOSS database.

The following figure gives an overview of the QualOSS platform for the of use case of “Importing”

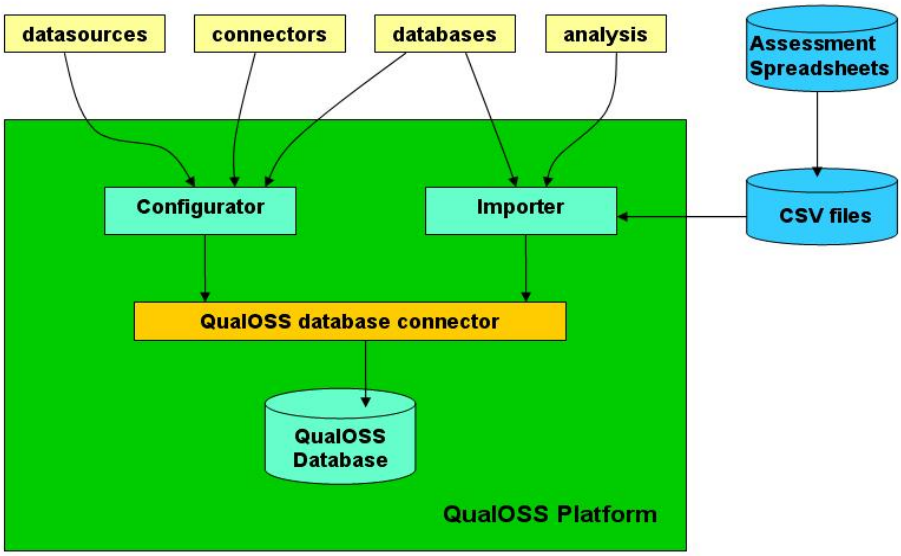


Figure 1: QualOSS platform : Importing Use Case

The yellow boxes stand for configuration files that are used for configuring the platform and/or for importing the assessment results.

Use case 2 : automatic analysis by launching appropriate tools

During the analysis, the needed measures are automatically collected by the appropriate tools that are launched by the platform using the appropriate connectors. As an additional functionality, the reporter allows to output the values of the measures and of the indicators based on a quality model file.

The following figure gives an overview of the QualOSS Platform for the automatic analysis use case.

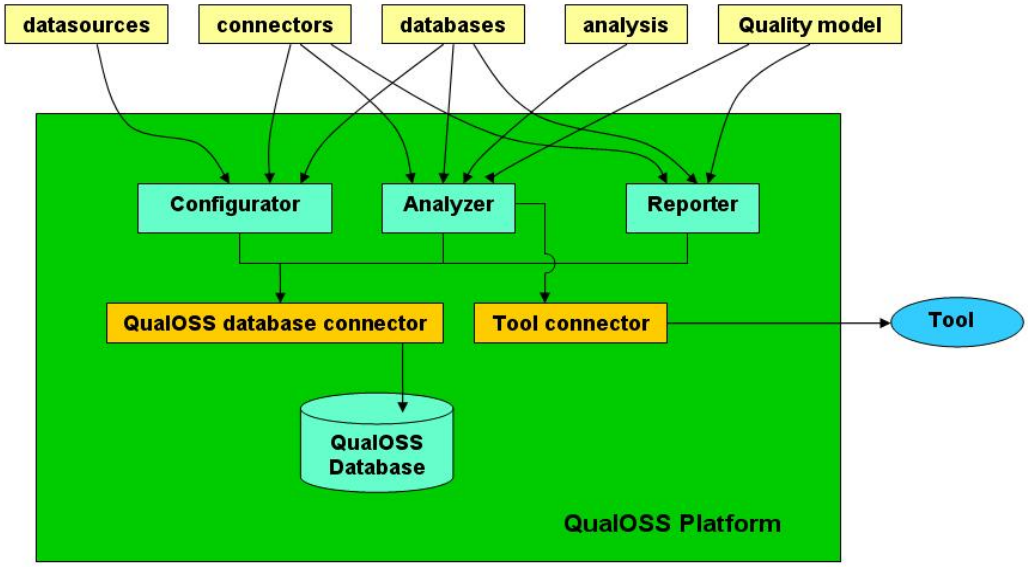


Figure 2: QualOSS platform : Automatic Analysis Use Case

Independently from the use case, various roles have been designed to use and administer the platform.

- The system administrator who is installing the platform
- The platform administrator who is using the Configurator
- The expert, advanced and basic user who are using either the Importer or the Analyzer / Reporter

The following sections describes the guides for these roles.

2. System Administrator Guide

This guide lists the pre-requisites for the installation of the platform and the various operations that need to be performed by the administrator of the system on which the QualOSS Platform will be deployed.

2.1 Pre-requisite for the installation

The QualOSS Platform is implemented in Python (Python 2.4). It has not been tested for Python 2.5 although precautions have been taken to allow the use of Python 2.5.

The QualOSS Platform is distributed as a compressed archive (zip, tar.gz).

The current release has been tested on Linux (Ubuntu, Redhat) and Windows (XP, Vista).

The installation of the platform requires the following steps:

Python and easy_install feature

- Install Python [latest version is Python 2.4.4] (http://www.python.org)
- Install the Easy Install (http://peak.telecommunity.com/DevCenter/EasyInstall)
- (Optional) Install psyco accelerator, choose version 1.5.2 for Python 2.4. (http://sourceforge.net/project/showfiles.php?group_id=41036&package_id=33183)

Database Clients and Python bindings
- Install SQLite (http://www.sqlite.org)
- Install MySQL (http://dev.mysql.com/downloads/)
- Install the GUI Tools for MySQL (http://dev.mysql.com/downloads/gui-tools/5.0.html)

Python bindings to database clients
- Install python binding to sqlite [only for Python < 2.5] (http://pypi.python.org): `$ easy_install pysqlite`
- Install python bindings to mysql (http://pypi.python.org): `$ easy_install mysql-python`
- Install python bindings to postgres (http://pypi.python.org): `$ easy_install pygresql`

Note: Same as previous note, only binding to mysql will be needed in the final version.

Python XML support
- Install LXML (http://codespeak.net/lxml/): `$ easy_install lxml`
- (Optional) Install ElementTree (http://www.effbot.org): `$ easy_install elementtree`
- (Optional) Install cElementTree (http://www.effbot.org): `$ easy_install celementtree`

Python YAML support
- Install YAML for Python (http://pyyaml.org/wiki/PyYAML): `$ easy_install pyyaml`

Once these third-party packages have been installed, the platform can be deployed by unzipping the archive on the desired location. This location is referred to as $QUALOSS_HOME in the remaining of this document.

2.2 Configuration of the system

The following groups need to be created on Linux/Unix system:
- Group QualOSS Platform Administrator
- Group QualOSS Expert User
- Group QualOSS Advanced User
- Group QualOSS Basic User

These groups should determine the files accessible for a given role. For example, the script Importer.py should only be accessible to Expert and Advanced Users but not to Basic Users.

2.3 Configuration of the database server

The system administrator should also create specific users, and for each user with the right to administrate the QualOSS platform, create a database on the MySQL server.

3. QualOSS Platform Administrator Guide

This section describes the activities needed to administer and configure the QualOSS Platform. This concerns the modification and adaptation of the needed configuration files, such as the Databases.ini, that holds the connections strings to the internal QualOSS Database where the measures are stored.

3.1 Installation of Analysis Tools

Analysis tools need to be installed on the platform.

The QualOSS Platform v1.1 is delivered with example of connectors for the following tools :

- Checkstyle v4.4
- filecounter V1.0
- sissy V0.45
- cvsanaly V1.0.1

3.2 Configuring the QualOSS Platform

Prior to any analysis, the platform needs to be configured. The script **Configurator.py** is used to configure the QualOSS Platform.

This script uses the following configuration files:

- databases.ini
- datasources.ini
- connectors.ini

Example of such files are provided in the $QUALOSS_HOME/config/default directory and in the *Appendix A*.

- databases.ini

  This configuration file contains a list of database connections. SQLite is the preferred connection during development, MySQL is easier to be used on production environments.

  - In the case you want to use SQLite engine :

    schema : specifies the location of the sql script that will be used for creating the QualOSS database; this file is delivered with the QualOSS platform and should not be modified for a specific QualOSS platform version

    set use = true in the [qualoss.sqlite] section (and set use = false in the other section)

    engine : identifies the current section (sqlite or mysql); provided for future extension should not be modified by the QualOSS administrator

    database : the name of the QualOSS data base (the default location can be used if appropriate)

    winclient : the pathname of the sqlite client if you use a Windows operating system - this path depends on the system configuration

    linuxclient : the pathname of the sqlite client if you use a Linux operating system - this path depends on the system configuration

  - In the case you want to use MySQL engine :

schema : specifies the location of the sql script that will be used for creating the QualOSS database; this file is delivered with the QualOSS platform and should not be modified for a specific QualOSS platform version

set use = true in the [qualoss.mysql] section (and set use = false in the other section)

engine : identifies the current section (sqlite or mysql); provided for future extension should not be modified by the QualOSS administrator

user : user name as defined by the MySQL server administrator

password : password for this user

host : hostname for the MySQL server

port : port for accessing the MySQL server (3306 is the default)

database : the name of the qualoss data base (as defined by the MySQL server administrator)

winclient : the pathname of the mysql client if you use a Windows operating system - this path depends on the system configuration

linuxclient : the pathname of the mysql client if you use a Linux operating system - this path depends on the system configuration

- datasources.ini

  List of the datasource types and, for each data source, of the artefact types taken into account on this instance of the platform. This file contains a section for each datasource type; the section contains a list of all artefacts defined for this datasource. Additionally, a short name has to be specified for each datasource type. In the example given in appendix, this configuration file tells that data source type “packaged_distribution” contains the artefact types “code“, “doc“, “code_file“, ...

- connectors.ini

  List of the connectors that are validated for the platform. This file contains a section for each connected tool. This section specifies the name of the tool, its location, its version, the format used for receiving the results of the tool, and command line associated to the tool.

The main parameters specify the configuration files to be used (databases.ini, datasources.ini and connectors.ini).

Configurator.py beeing a python script, it has to be launched using the python command, as in the following example :

```
python Configurator.py
       -c myconfig/project1/connectors.ini
       -d myconfig/project1/databases.ini
       -s myconfig/project1/datasources.ini
```

The result of the run of Configurator.py is a database initialised with the datasource types and artefacts types of interest, the connectors that are validated by the platform and the metrics that these connectors are providing.

Scripts of the QualOSS platform have a common set of options available for all of them and another set of specific options. The following Table 1 describes the common set of options. The following Table 2 describes the options specific for Configurator.py.

Table 1: Common options for all tools

Option	Mandatory	Default	Description
help	no	N/a	Prints the list of options and exits
version	no	N/a	Prints the version number of the *Qualoss Analyzer*
verbose	no	no	Indicates whether the details of the execution of the tool should be displayed to the standard output.
quiet	no	yes	Inverse function of verbose
debug	no	no	Generates debug information
fast	no	no	Indicates whether the Analyzer should optimise its running
log	no		Directory for log files
config	no		Gets options from configuration file
raw-config	no		Gets options from raw configuration file
generate-config	no	no	Generates config file based on the provided options

Table 2: Specific options for the Qualoss Configurator

Option	Mandatory	Default	Description
connectors	no	config/default/connectors.ini	Path to connectors specifications
databases	no	config/default/databases.ini	Path to databases connections
datasources	no	config/default/datasources.ini	Path to datasources definitions
init	no	no	Forces the initialization of the database

The complete description of the script Configurator.py (output of the help option) is given in Appendix B.

4. QualOSS Expert User Guide

This section describes the activities that can be performed by an expert user on the QualOSS Platform. The current version of the platform provides the script Analyzer.py. It also allows a developer to add new metrics to connectors and to create a new connector. In the latter case, the Expert user can then request the installation of its new connector to the QualOSS Platform Administrator.

4.1 Importing Measures and Indicators

Once the platform has been initialized, it is possible to insert measures and indicators from csv files using the script **Importer.py.** This corresponds to the first "Importing" use case mentioned in the introduction of this document.

This script uses the following configuration files:
- databases.ini: (see section Configuring the Platform)
- analysis.ini describes the artefacts that should be taken into account for the analysis.

An example of file "analysis.ini" is provided in the $QUALOSS_HOME/config/default directory and in the *Appendix A*.

The file analysis.ini contains in its first section [endeavour] a description of the endeavour to be analyzed (name, version, main programming language). Then for each datasource that is relevant for the analysis, general information must be provided as well as information specific for each artefact. This information will be introduced in the QualOSS database.

This script is to be used to insert "manually" measures and/or indicators from a csv files. These csv files can be created in an easy way from the spreadsheets where the result of the endeavour assessment (save as ... csv file).

The csv file starts with a header line and must contain one line for each measure or indicator to be inserted in the platform data base.
The header line identifies the name of the columns that will be used for importing the relevant information.

For the import of measures, the following information must be specified in each line of the csv file :

Column name in header	**Information**
Measure Value	value of the measure (float)
Measure Name	name of the measure (var char)
Data Source Type	data source type (var char)
Artifact Type	Artefact type (var char)
Measure Status	measure status (int)

For the import of indicators, the following information must be specified in each line of the csv file :

Column name in header	**Information**
Value	value of the indicator (float)

Color	colour of the indicator (var char)
Indicator Name	name of the indicator (var char)
Coverage	coverage of the indicator (float)
Confidence	confidence of the indicator (float)

Example of such measure and indicator files are given in Appendix E.

Note : the column name in the header can be configured by modifying the configuration file fields. ini; and example of such a file is given in Appendix A.

Importer.py being a python script, it has to be launched using the python command, as in the following examples :

```
python Importer.py
       -d myconfig/project1/databases.ini
       -a myconfig/project1/analysis.ini
       --measures=myconfig/project1/measures.csv
```

The result of the run of the Importer.py script is stored inside the internal QualOSS database.

Importer.py supports the common set of options (see Table 1). The following Table 3 describes the options specific for Importer.py.

Table 3: Specific options for the Qualoss Importer

Option	Mandatory	Default	Description
analysis	no	config/default/analysis.ini	Path to analysis specifications
databases	no	config/default/databases.ini	Path to databases connections
measures	no	at least one or both parameters can be specified	Path to CSV file with measures
indicators	no		Path to CSV file with indicators
fields	no	config/default/fields.ini	Specifies the fields mapping of the CSV files
force	no	no	Forces the insertion of measures and / or indicators

The same measures and indicators can be imported several times in the QualOSS database : a new row will be created each time and the most recent row can be retrieved thanks to the recorded validity dates.

The complete description of the script Importer.py (output of the help option) is given in Appendix B.

4.2 Configuring an Automatic Analysis

An analysis can be launched once the platform has been initialised. The script **Analyzer.py** is used to launch the analysis.

This script uses the following configuration files:
- databases.ini: (see section Configuring the Platform)
- connectors.ini: (see section Configuring the Platform)
- analysis.ini: the artefacts that should be taken into account for the analysis.

– qualitymodel.yaml: the list of indicators and their definitions. These indicators are using metrics computed by the connectors that have been validated for the platform. The qualitymodel is using a specific syntax that is easy to understand and that is automatically transformed into an equivalent Python class located in the same directory as the .yaml file.

The advanced user has to create and specify the file analysis.ini, he can also create and specify a file qualitymodel.yaml accordingly to his own needs.

Analyzer.py being a python script, it has to be launched using the python command, as in the following examples :

```
python Analyzer.py
       -d myconfig/project1/databases.ini
       -c myconfig/project1/connectors.ini
       -a myconfig/project1/analysis.ini
       --quality-model=myconfig/project1/qualitymodel.yaml
```

The result of the run of the Analyzer.py script is stored inside the internal QualOSS database. A new analysis has been inserted and the measures are stored along with the artefacts and datasources of interest for this analysis.

Analyzer.py supports the common set of options (see Table 1). The following Table 4 describes the options specific for Analyzer.py.

Table 4: Specific options for the Qualoss Analyzer

Option	Mandatory	Default	Description
analysis	no	config/default/analysis.ini	Path to analysis specifications
databases	no	config/default/databases.ini	Path to databases connections
connectors	no	config/default/connectors.ini	Path to connectors specifications
quality-model	no	config/default/qualitymodel.yaml	Path to quality model
force	no	no	Forces the insertion of measures

The complete description of the script Analyzer.py (output of the help option) is given Appendix B.

4.3 Obtaining the result for the Analysis

To report the indicators, the script **Reporter.py** is used.

This script uses the following configuration files:
– qualitymodel.qm: (see section Configuring an Analysis)
– analysis.ini: (see section Configuring an Analysis)
– databases.ini: (see section Configuring the Platform)

Reporter.py being a python script, it has to be launched using the python command, as in the following examples :

```
python Reporter.py
       -d myconfig/project1/databases.ini
       -a myconfig/project1/analysis.ini
       --quality-model=myconfig/project1/qualitymodel.yaml
       --format=text
```

The result of the run of Reporter.py is displayed on screen and can be redirected to a file. This can be printed in textual format (close to the .qm user-friendly format) or in XML.

Reporter.py supports the common set of options (see Table 1). The following Table 5 describes the options specific for Importer.py.

Table 5: Specific options for the Qualoss Reporter

Option	Mandatory	Default	Description
analysis	no	config/default/analysis.ini	Path to analysis specifications
databases	no	config/default/databases.ini	Path to databases connections
quality-model	no	config/default/qualitymodel.yaml	Path to quality model
format	no	csv	text | xml | yaml | csv | text
output	no	stdout	Output format

The complete description of the script Reporter.py (output of the help option) is given in Appendix B.

4.4 How to extend the QualOSS Platform

The QualOSS platform can be extended by the expert user in order to integrate an additional tool and to be able to compute additional metrics. This implies the installation and configuration of the new tool, but also the creation of a new connector.

This section describes how to create a new connector and uses as example the integration of the tool SISSy.

The connector will be used to :

1. run the analysis tool
2. compute metrics (specifically provided by the tool) from the results produced by the run of the tool.

Creating a connector consists of the following steps :

1. request the installation of the needed tool to the system administrator
2. implement and test the connector
3. create the entry for the configuration file “connectors.ini”

4.4.1 Step 1: Request the installation of the needed tool

The System Administrator has to install the tool that will be launched by the QualOSS Platform. This often requires administrator rights on the server on which the QualOSS is installed. These tools should be checked against this checklist:

– be accessible at command line

- receive 'input' option
- receive 'output' option

4.4.2 Step 2: Implement and test the connector

A connector consists in a python module implementing a class derived from a specific subclass of the Connector class. The subclasses depend on the format of the output of the associated tool (e.g., the subclass XMLConnector is used for tools that produce their output as an XML document, while the subclass PostgresqlConnector is used for tools producing their output in a PostgreSQL database).

Here is the API for a Connector:

class Connector
  method call(*configuration*, *global_analysis_parameters*)
    Configure the connector with the information from the XML configuration files (connectors.ini and analysis.ini)
  method launch()
    Launch the associated tool at command line
  method load()
    Load the result produced by the tool
  getter metrics
    Retrieve the list of metrics defined by the

The subclasses of the Connector class that are available to be subclassed are:

class TextConnector(Connector)
  Connector to output saved in a file in a textual format without any syntax
  field: data
    Handle to the opened file where the output has been saved

class ParseableConnector(Connector)
  Connector to output saved in a file whose contents can be parsed with simple regular expressions, extends the class Connector
  field: data
    Handle to the opened file where the output has been saved

class HtmlConnector(Connector)
  Connector to output saved in HTML format
  field: data
    Handle to an instance of an HTML parser fed with the content of the file where the output has been saved

class XmlConnector(Connector)
  Connector to XML output
  field: data
    Handle to an instance of an XML parser fed with the content of the file where the output has been saved

class CsvConnector(Connector)
  Connector to output saved in CSV format
  field: data

Handle to an instance of a CSV reader fed with the content of the file where the output has been saved

class PostgresqlConnector(Connector)
Connector to output saved in a PostgreSQL Database
field: data
Handle to a connection to the PostgreSQL Database schema with the given user, password and host information

class MysqlConnector(Connector)
Connector to output saved in a MySQL Database
field: data
Handle to a connection to the MySQL Database schema with the given user, password and host information

class SqliteConnector(Connector)
Connector to output saved in a Sqlite Database
field: data
Handle to a connection to the Sqlite Database file

When implementing a connector to a given tool, the QualOSS Expert User must subclass the appropriate *Connector class based on the type of format of the output.

As a running example, we take the tool SISSy. SISSy parses the code source of a given set of directories, it creates a meta-model of the source code and stores this meta-model in a JDBC database (PostgreSQL, MySQL) based on the configuration that was provided. To create a connector for the tool SISSy, the expert user (or developer) will create a python module, for example sissy.py. The skeleton of the module will be:

```
Line
1   from eu.qualoss.connector import PostgresqlConnector, metric
2   class SissyConnector(PostgresqlConnector):
3       @metric
4       def new_sissy_metric(self, artefact):
5           "Description of this new_sissy_metric"
6           return -1
```

Explanation of each line of code:

Line1. This line imports the needed information, the PosgresqlConnector class and the metric function that will be used as a decorator (see Line 3). A decorator is a mean to annotate the code. These annotations can be used later by inspecting tools.

Line2. This line defines the connector class for SISSy. In this case, the connector for the SISSy tool will extend the functionalities provided by the PostgresqlConnector, that is, its field data will be a connection to the PostgreSQL database where the output of SISSy has been stored (see the list of provided subclasses above).

Line3. This line decorates the method declared on Line 4. This line is important because it is used to tell the QualOSS Platform that the following method has to be treated as a metric.

Line4. This line declares a method of the SissyConnector. Thanks to the previous line it will be treated as a metric. The name of this method must respect the convention described below. The first

parameter 'self' is needed as a Python convention, it refers to the current instance of the class where the method is applied; this is the equivalent of “this” in the Java programming language. The second parameter 'artefact' is also mandatory; every metric of a connector is applied on a given artefact

Line5. This line is the docstring of the metric, equivalent to the Javadoc in the Java programming language.

Line6. This line is the actual body of the method and its purpose is to compute what will be returned as the result of the metric. In this example template, we simply return the value -1.

As this SissyConnector class is subclassing the PostgresqlConnector class, its field 'data' contains handle to a cursor connected to the database. The actual scheme for implementing a metric in such a case is the following:

```
Line
3       @metric
4       def datasource_type__artefact_type__new_sissy_metric(self, artefact):
5           "Description of this new_sissy_metric"
5.1         self.data.execute(“A Select Query”)
5.2         (column1, column2, ... columnN) = self.data.fetchone()
6           return columnM
```

We have introduced lines 5.1 and 5.2 and modified line 6:

Line 5.1 executes a select query on the database
Line 5.2 maps the columns of the result of the query to Python variables
Line 6 returns the appropriate column

An example of connector for the tool SISSy is given in Annex C. This connector implements 3 metrics.

Naming convention: the name of the metric must be unique across the whole QualOSS Platform. As a convention for the QualOSS Platform, please use this scheme: <datasource type>, <artefact type>, <metric name> separated by 3 underscores.

Example: version_control_system___code___number_of_files.
or vcs__code__number_of_files

A shorthand is possible, this is where the “short” field of the datasource_type is used ('vcs' is the short name for the datasource 'version_control_system').
The QualOSS Platform will do what is necessary to store this metric in the right place.

4.4.3 Step 3: Create the entry for the XML configuration file “Connectors.ini”

The following information has to be specified in this entry :

- location of the connector module that will be called for the metrics extraction (how to create this module is described in the previous section)
- name and version of the tool
- installation path of the tool
- command line to be used
- input parameter(s) (will be concatenated with the command)
- output parameter(s) (will be concatenated with the command)

– (optionally) connections parameters; this depends on the way the tool produces its results. In the example hereby, the results are stored in a data base so the parameters are parameters for connection to the database

An example of the entry for the tool SISSy is given in Appendix C.

Once the connector has been validated, it must be included in the file “connectors.ini” by the *QualOSS Platform Administrator*.

5. QualOSS Advanced User Guide

This section describes the activities that can be performed by an advanced user on the QualOSS Platform The QualOSS Advanced User can import measures and indicators from intermediate files or can specify a QualOSS Quality Model and launch an automatic analysis.

5.1 Importing Measures and Indicators

Once the platform has been initialized, it is possible to insert measures and indicators from csv files using the script **Importer.py.** This corresponds to the first “Importing” use case mentioned in the introduction of this document.

This script uses the following configuration files:
– databases.ini: (see section Configuring the Platform)
– analysis.ini describes the artefacts that should be taken into account for the analysis.

The result of the run of the Importer.py script is stored inside the internal QualOSS database.

5.2 Configuring an Analysis

An analysis can be launched once the platform has been initialised. The script **`Analyzer.py`** is used to launch the analysis.

This script uses the following configuration files:
– `databases.ini`: (see section QualOSS Platform Administrator Guide)
– `connectors.ini`: (see section QualOSS Platform Administrator Guide)
– `analysis.ini`: the artefacts that should be taken into account for the analysis.
– `qualitymodel.qm`: the list of indicators and their definitions. These indicators are using metrics computed by the connectors that have been validated for the platform. The qualitymodel is using a specific syntax that is easy to understand and that is automatically transformed into an equivalent Python class located in the same directory as a .qm file.

The result of the run of the `Analyzer.py` script is stored inside the *Internal QualOSS Database*. A new analysis has been inserted and the measures are stored along with the artefacts and datasources of interest for this analysis.

5.3 Obtaining the result for the Analysis

To report the indicators, the script **`Reporter.py`** is used. This script uses the following configuration files:
– `qualitymodel.qm`: (see section Configuring an Analysis)
– `analysis.ini`: (see section Configuring an Analysis)
– `databases.ini`: (see section Configuring the Platform)

The result of the run of `Reporter.py` is displayed on screen and can be redirected to a file. This can be printed in textual format (close to the `.qm` user-friendly format) or in XML.

5.3.1 Defining a Custom User Quality Model

Because of the well formed rules of XML concerning the signs '<' and '>', XML was not chosen as the format for the Quality Model. In this first phase, a simple textual language has been designed. This document is then translated to actual Python code. The language is pretty straightforward and self-explanatory.

Here is an example of a quality model in the .qm language.

```
# This is an example of a qualitymodel
qualitymodel {

    # we define here an indicator, it is identified by its name
    indicator:packaged_distribution.code indicator1 {

        # 'value' tells how to compute the value of the indicator
        value 'self.metric1()/self.metric2()';

        # 'is' introduces the score and the condition to give this score
        is 'black' when 'value > 15';
        is 'red' when '9 <= value <= 15';
        is 'blue' when '5 <= value < 9';
        is 'green' when 'value < 5';

        # the indicator uses a number of metrics
        metric:packaged_distribution.code metric1; # this is an atomic metric

        metric:packaged_distribution.code metric2 { # this is a compound metric
            value 'self.metric3()/self.metric4()-self.metric5()';

            metric:packaged_distribution.code metric3;
            metric:packaged_distribution.code metric4;
            metric:packaged_distribution.code metric5;
        }
    }
}
```

This quality model defines a single indicator named indicator1. This indicator is applied on the datasource type `packaged_distribution` and on the artefact type `code`.

`indicator1` also requires the following metrics: `metric1` and `metric2`
Its value is computed as described (`metric1/metric2`) and some colors have been defined for the levels (the colors are fixed for the QualOSS Platform: `black`, `red`, `blue`, `green`). The mapping from the value of the metric to the color of the indicator is given with the `when` expression.

`metric1` is an atomic metric. It will only retrieve its value from the associated metric (this metric is computed by a given connector, and the fullname of this metric must match one of the methods defined by the connectors, hence the use of artefact_type and datasource_type in the naming convention)

`metric2` is an aggregated metric, it requires the of values of the following metrics (`metric3`, `metric4`, `metric5`). Its definition is also given with the `value` expression.

6. QualOSS Basic User Guide

In case an assessment has been performed, the basic user has the right to retrieve results of the run of Reporter.py. These results should be placed at a specific location where the Basic User has the 'read only' access rights. Currently, an instance of the QualOSS platform presents the results of assessment at the URL
http://ingrid.cetic.be:33323/qualoss_assessment

7. Appendix A – Example of Configuration files

7.1 databases.ini

```
[qualoss.sqlite]
# Connector to the QUALOSS Database using Sqlite
# the 'schema' attribute is used to create the database if it is not present
# manually: launch 'sqlite3 <database>', then ".read <schema>" function of sqlite client
schema = config/default/sqlite.sql
use = false
engine = sqlite
database = data/qualoss.sqlite
# example for windows
winclient = c:/tools/sqlite/bin/sqlite3.exe
# example for unix
linuxclient = /usr/bin/sqlite3

[qualoss.mysql]
# Connector to the QUALOSS Database using MySQL
# the 'schema' attribute is used to create the database if it is not present
# manually: launch "mysql -u<user> -p<password> -h<host> -P<port> <database> < schema"
use = true
schema = config/default/mysql.sql
engine = mysql
user = qualoss
password = qualoss
host = localhost
port = 3306
database = qualoss
# example for windows
winclient = c:/mysql/bin/mysql.exe
# example for unix
linuxclient = /usr/bin/mysql
```

7.2 connectors.ini

```
[eu.qualoss.connector.tool.filecounter]
tool = filecounter.py
path = lib\filecounter
version = 1.0
format = xml

command = python %(path)s\run.py
command.option.1.-p = %($analysis.datasource.packaged_distribution.location)s
command.option.2.-o = %(__file__)s
command.option.3.-f = xml
```

7.3 DATASOURCES.INI

```
[packaged_distribution]
short = pd
# if things go well, code_test should not be used!
artefact =  code, \
            doc, \
            code_file, \
            code_package, \
            code_type, \
            code_operation, \
            code_variable, \
            code_test, \
            test

[issue_tracking_system]
short = its
artefact =  issue, \
            issue_featurerequest, \
            issue_bug, \
            issue_patch, \
            requestor, \
            assigner, \
            assignee

[version_control_system]
short = vcs
artefact =  committer, \
            commit, \
            code,\
            code_file, \
            code_test, \
            test, \
            doc

[internal_discussion_list]
short = idl
artefact =  poster, \
            thread, \
            message

[project_website]
short = pws
artefact = download_list, \
           test, \
           doc

[online]
short = online
artefact = test, \
           doc
```

Note: The data source names as well as the short name (mandatory) must be unique across all datasources.

7.4 ANALYSIS.INI

```
[endeavour]
# this section defines the name and version of the endeavour to analyze
name = azureus
version = 1.0
language = java

[datasource.packaged_distribution]
# information about the 'packaged distribution' (zip, dir, ...)
kind = dir
location = download\pd\azureus1
artefact.code = *

[datasource.version_control_system]
# information about the 'version control system' (cvs, svn, git, ...)
kind = cvs
location = :pserver:anonymous@azureus.cvs.sourceforge.net:/csvroot/azureus
artefact.committer = *

[datasource.issue_tracking_system]
# information about the 'issue tracking system' (bugzilla, sourceforge tracker, ...)
kind = sftracker
location = http://sourceforge.net/tracker/?func=browse&amp;group_id=84122&amp;atid=575154
artefact.issue = *

[datasource.internal_discussion_list]
# information about the 'internal discussion list' (mbox, ...)
kind = mbox
location = http://sourceforge.net/mailarchive/forum.php?forum_name=azureus-commitlog
artefact.poster = *

[env]
# this section defines environment parameters
module = azureus
```

7.5 FIELDS.INI

```
[indicator]
value=Value
color=Color
name=Indicator Name
coverage=Coverage
confidence=Confidence

[measure]
value=Measure Value
name=Measure Name
datasourcetype=Data Source Type
artefacttype=Artifact Type
status=Measure Status
```

8.Appendix B – Help of Platform Scripts

8.1 Help for Configurator.py

```
$ python Configurator.py --help
usage: Configurator.py [options]

This script is used to configure the QualOSS Platform with the following files:
- databases.ini, default connection to the internal QualOSS database
- datasources.ini, list of datasource types and artefact types that will be
    treated by the platform
- connectors.ini, list of connectors to be inserted inside the platform

@see Analyzer, Reporter.


options:
  --fast                      speed up [default: False]
  --log=LOG                   directory for log files
  --verbose                   trace execution [default: False]
  --quiet                     do not print anything
  --debug                     print debug information [default: False]
  --config=CONFIG             get options from configuration file
  --raw-config=RAW_CONFIG     get options from raw configuration file
  --generate-config           generate config file based on the provided
                              options [default: False]
  --version                   show program's version number and exit
  -h, --help                  show this help message and exit
  -c CONN, --connectors=CONN  Path to connectors specifications
                              [config/default/connectors.ini]
  -d DB, --databases=DB       Path to databases connections
                              [config/default/databases.ini]
  -s DS, --datasources=DS     Path to datasources definitions
                              [config/default/datasources.ini]
  --init                      Force the initialization of the database [False]
```

8.2 Help for Importer.py

```
$ python Importer.py --help
usage: Importer.py [options]

The Importer imports metrics from various sources. It takes:
- databases.ini, default connection to the internal QualOSS database
- connectors.ini, list of connectors to be inserted inside the platform
- analysis.ini, description of the endeavor to analyse

The data are stored in a the internal QualossDatabase.

@see Configurator, Reporter.


options:
  --fast                      speed up [default: False]
  --log=LOG                   directory for log files
  --verbose                   trace execution [default: False]
  --quiet                     do not print anything
  --debug                     print debug information [default: False]
  --config=CONFIG             get options from configuration file
  --raw-config=RAW_CONFIG     get options from raw configuration file
  --generate-config           generate config file based on the provided
                              options [default: False]
  --version                   show program's version number and exit
  -h, --help                  show this help message and exit
  -i IND, --indicators=IND    Path to CSV file with indicators
  -m MEAS, --measures=MEAS    Path to CSV file with measures
  -d DB, --databases=DB       Path to databases connections [default:
                              config/default/databases.ini]
  -a ANA, --analysis=ANA      Path to analysis specifications [default:
                              config/default/analysis.ini]
  -f FIELDS, --fields=FIELDS  Specify the fields mapping of the CSV files
                              [default: config/default/fields.ini]
  --force                     Force the insertion of measures [default: False]
```

8.3 Help for Analyzer.py

```
$ python Analyzer.py --help
usage: Analyzer.py [options]

The Analyzer launches the connectors and collect the metrics. It takes:
- databases.ini, default connection to the internal QualOSS database
- connectors.ini, list of connectors to be inserted inside the platform
- qualitymodel(.qm|.yaml|.py), qualitymodel to be evaluated
- analysis.ini, description of the endeavor to analyse

The data are stored in a the internal QualossDatabase.

@see Configurator, Reporter.


options:
  --fast                    speed up [default: False]
  --log=LOG                 directory for log files
  --verbose                 trace execution [default: False]
  --quiet                   do not print anything
  --debug                   print debug information [default: False]
  --config=CONFIG           get options from configuration file
  --raw-config=RAW_CONFIG   get options from raw configuration file
  --generate-config         generate config file based on the provided
                            options [default: False]
  --version                 show program's version number and exit
  -h, --help                show this help message and exit
  -c CON, --connectors=CON  Path to connectors specifications
                            [config/default/connectors.ini]
  -d DB, --databases=DB     Path to databases connections
                            [config/default/databases.ini]
  -a ANA, --analysis=ANA    Path to analysis specifications
                            [config/default/analysis.ini]
  -q QM, --quality-model=QM Path to quality model
                            [config/default/qualitymodel.yaml]
  --force                   Force the computation of metrics [False]
```

8.4 Help for Reporter.py

```
$ python Reporter.py --help
usage: Reporter.py [options]

This script loads the measures computed by the Analyzer and computes
the indicators defined in the provided QualityModel:
- databases.ini, default connection to the internal QualOSS database
- connectors.ini, list of connectors to be inserted inside the platform
- qualitymodel(.qm|.yaml|.py), qualitymodel to be evaluated

Output of the quality model as XML, TEXT or YAML.

@see Analyzer, Configurator.


options:
  --fast                      speed up [default: False]
  --log=LOG                   directory for log files
  --verbose                   trace execution [default: False]
  --quiet                     do not print anything
  --debug                     print debug information [default: False]
  --config=CONFIG             get options from configuration file
  --raw-config=RAW_CONFIG     get options from raw configuration file
  --generate-config           generate config file based on the provided
                              options [default: False]
  --version                   show program's version number and exit
  -h, --help                  show this help message and exit
  -d DB, --databases=DB       Path to databases connections
                              [config/default/databases.ini]
  -a ANA, --analysis=ANA      Path to analysis specification
                              [config/default/analysis.ini]
  -q QM, --quality-model=QM   Path to quality model
                              [config/default/qualitymodel.yaml]
  -f FORMAT, --format=FORMAT  Output format (text|xml|yaml|csv|text) [csv]
  -o OUTPUT, --output=OUTPUT  Path to output file [stdout]
```

9. Appendix C – Connector to the tool SISSy

Extension of the file “connectors.ini” :

```
    <connector name="eu.qualoss.connector.tool.sissy">
        <tool>sissy</tool>
        <path>c:\jf\sissy\sissy</path>
        <version>0.45</version>
        <cmd>%(this.path)s\sissy_analysis.bat</cmd>
        <input> %(analysis.packaged_distribution.location)s %(analysis.environment.sissy_language)s</input>
        <output format="postgresql">%(analysis.environment.schema)s</output>
        <connection>uid=%(analysis.environment.user)s;pwd=%(analysis.environment.password)s;server=%
(analysis.environment.host)s;port=%(analysis.environment.port)s;database=%
(analysis.environment.schema)s</connection>
    </connector>
```

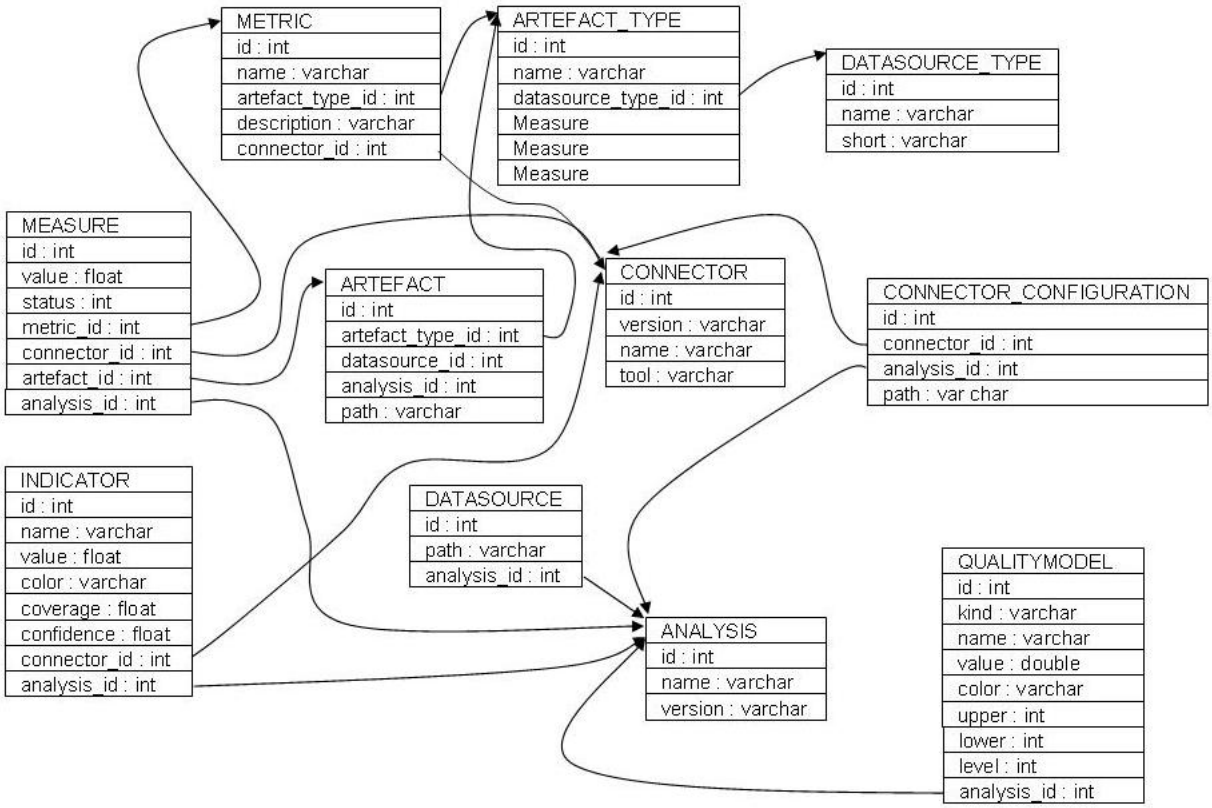
The value for input, output and connection are substituted from the file “analysis.ini” defining the endeavour to be analysed.

Example of connector module “sissy.py” :

```
from eu.qualoss.connector import PostgresqlConnector, metric

class SissyConnector(PostgresqlConnector):
    _Q_PACKAGED_DISTRIBUTION__CODE__CYCLO = """
      SELECT
            'Cyclomatic Complexity' as Metric,
            sum(func.numberOfEdges - func.numberOfNodes + 1) as cyclomatic_complexity
      FROM
            TFunctions func INNER JOIN
            TModelElements meFunc on (meFunc.id = func.id) INNER JOIN
            TConstants cmeFunc on (cmeFunc.value = meFunc.status and cmeFunc.name = 'STATUS_NORMAL')
    """
    _Q_PACKAGED_DISTRIBUTION__CODE__CALLS = """
      SELECT
            sum(calls) as calls
      FROM
            calls_by_function
    """
    _Q_PACKAGED_DISTRIBUTION__CODE__FANOUT = """
      SELECT
            sum(fanout) as fanout
      FROM
            fanout_by_function
    """

    @metric
    def packaged_distribution__code__cyclo(self, artefact):
        """Total cyclomatic complexity of the project source"""
        self.data.execute(self._Q_PACKAGED_DISTRIBUTION__CODE__CYCLO)
        # the select query returns 2 fields, use fetchone and extract these 2 fields
        (metric, number,) = self.data.fetchone()
        return number

    @metric
    def packaged_distribution__code__calls(self, artefact):
        """Number of operation calls in the project source"""
        self.data.execute(self._Q_PACKAGED_DISTRIBUTION__CODE__CALLS)
        (number,) = self.data.fetchone()
        return number

    @metric
    def packaged_distribution__code__fanout(self, artefact):
        """Number of called classes in the project source"""
        self.data.execute(self._Q_PACKAGED_DISTRIBUTION__CODE__FANOUT)
        (number,) = self.data.fetchone()
        return number
new = SissyConnector
```

This example of connector module implements 3 metrics corresponding to 3 SQL queries defined at the beginning of the class definition. The 3 methods execute such a query using a cursor object, the result is returned in a number.

10. Appendix D – Schema of the database



This figure shows the database schema for the QualOSS Platform (v1.1).
Only the relevant fields are shown (the fields dealing with the tracking of activities on the tables were not reported (these fields are: user, fromdate, todate and comment)

In the current version some additional tables are present but not used because their functionalities have not been implemented:

11. Appendix E – Input Files for Importer.py

11.1 File for import of measures

```
"Measure Name","Data Source Type","Artifact Type","Measure Value","Measure Status"
"mean_number_of_bugs_in_stable_versions","issue_tracking_system","issue_bug","1935.5",0
"slope_number_of_bugs_in_stable_versions","issue_tracking_system","issue_bug","-874.6",0
"number_of_bugs_by_number_of_stable_versions
","issue_tracking_system","issue_bug"," ",-1
"slope_number_of_bugs_by_stable_versions_lifetime","issue_tracking_system","issue_bug"," ",-1
"mean_number_of_bugs_in_stable_versions_specific","issue_tracking_system","issue_bug","1935,5",0
"slope_number_of_bugs_in_stable_versions_specific","issue_tracking_system","issue_bug","-874,6",0
"mean_number_of_new_and_ modified_files","packaged_distribution","code",586,0
"mean_percentage_of_new_and_ modified_files","packaged_distribution","code",53,0
"mean_number_of_ coding_convention_violation","packaged_distribution","code",,-3
```

11.2 File for import of indicators

```
"Indicator Name","Value","Color","Coverage","Confidence"
"Stability_Evolution","0,5","Black",100,1
"Stability_Evolution_Specific_Version","2,6","Yellow",100,1
"Importance_of_Corrections","0,5","Black",100,1
"coding_convention_violation","3,5","Green",0,0
```