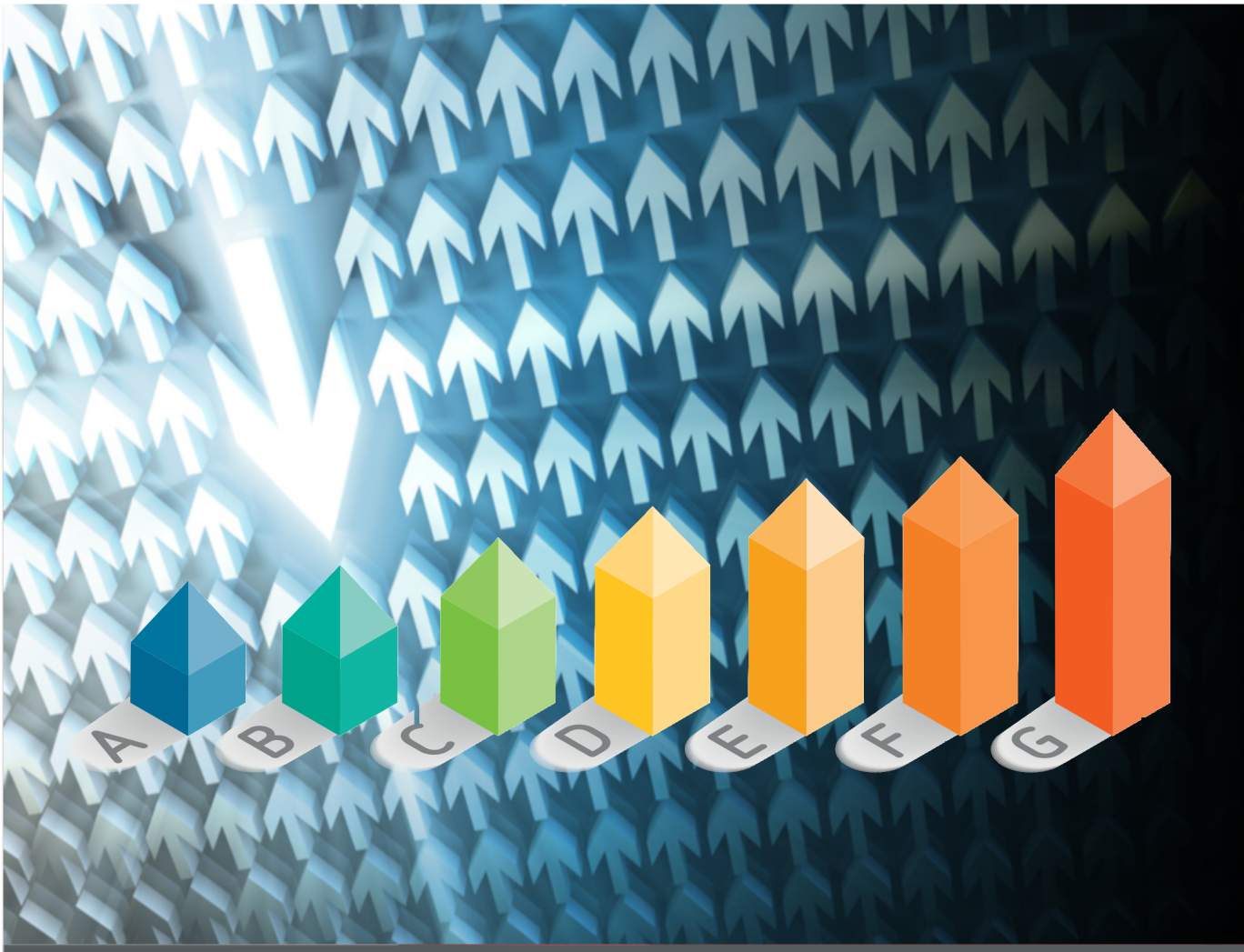
VECTOR



Command Line Interface

Squore 20.1.11

Last updated 2022-02-10

Table of Contents

Preface

Foreword

This edition of the Command Line Interface was released by Vector Informatik GmbH.

It is part of the user documentation of the Squore software product edited and distributed by Vector Informatik GmbH.

For information on how to use and configure Squore, the full suite of manuals includes:

User Manual	Target Audience
Squore Installation Checklist	New users before their first installation
Squore Installation and Administration Guide	IT personnel and Squore administrators
Squore Getting Started Guide	End users, new users wanting to discover Squore features
Squore Command Line Interface	Continuous Integration Managers
Squore Configuration Guide	Squore configuration maintainers, Quality Assurance personnel
Squore Eclipse Plugin Guide	Eclipse IDE users
Squore Reference Manual	End Users, Squore configuration maintainers
Squore API Guide	End Users, Continuous Integration Managers
Squore Software Analytics Handbook	End Users, Quality Assurance personnel

You can also use the online help from any page when using the Squore web interface by clicking **? > Help**.

Licence

Warranty

Vector Informatik GmbH makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Vector Informatik GmbH shall not be liable for errors contained herein nor for incidental or consequential damages in connection with the furnishing, performance or use of this material.

This edition of the Command Line Interface applies to Squore 20.1.11 and to all subsequent releases and modifications until otherwise indicated in new editions.

Responsabilities

Approval of this version of the document and any further updates are the responsibility of Vector Informatik GmbH.

Contacting Vector Informatik GmbH Product Support

If the information provided in this manual is erroneous or inaccurate, or if you encounter problems during your installation, contact Vector Informatik GmbH Product Support: https://portal.vector.com/

You will need a valid customer account to submit a support request. You can create an account on the support website if you do not have one already.

For any communication:

- **support@vector.com**
- **Vector Informatik GmbH Product Support**

  Vector Informatik GmbH - Holderäckerstr. 36 / 70499 Stuttgart - Germany

Getting the Latest Version of this Manual

The version of this manual included in your Squore installation may have been updated. If you would like to check for updated user guides, consult the Vector Informatik GmbH documentation site to consult or download the latest Squore manuals at https://support.squoring.com/documentation/latest. Manuals are constantly updated and published as soon as they are available.

Chapter 1. Introduction

This document is the Command Line Interface Guide for Squore.

It is indented as a follow up to the Squore Getting Started Guide and will help you understand how to use Squore CLI to create and update projects. It is divided into several chapters, as detailed below:

- Getting Started With the Squore CLI provides a basic introduction to Squore CLI and the examples provided with your Squore installation.
- Command Line Reference provides a complete reference of all the command line options and parameters for creating projects.
- Repository Connectors covers the default Repository Connectors and the parameters to pass to Squore to use them.
- Data Providers is a reference guide to all the Data Providers shipped with Squore.

Chapter 2. Getting Started With the Squore CLI

Squore CLI is a package that is installed on every client computer that needs to perform local code analyses or trigger a remote analysis on Squore Server. It contains the client (squore-engine.jar), its libraries, configuration files and some sample job files to help you get started. In this section, you will learn more about the different setup configurations supported by the CLI, its installation and integration into a Continuous Integration environment.

Squore CLI accepts commands and parameters to communicate with Squore Server. Inside the installation folder, some scripts are provided as examples to create projects, save encrypted credentials to disk, and synchronise the client's configuration with the server.

There are two ways to contemplate the deployment of Squore CLI:

1. As a way to analyse code and process data on a client machine and send the results to the server.
2. As a way to instruct the server to carry out an analysis of code and other input data.

Squore CLI and Squore Server must always be the same version in order to work together.

Installation Prerequisites

Supported Operating Systems

The following is a list of the officially supported and tested operating systems:

A 64-bit version of the OS is required

- CentOS 7
- CentOS 8
- Ubuntu 18.04 LTS
- Ubuntu 20.04 LTS
- Windows 10
- Windows Server 2016

The following is a list of the operating systems that are not regularly tested but are known to be working:

- Red Hat Enterprise Linux 7
- Fedora 29
- SuSe Linux 11.1
- Ubuntu Server 16.04
- Windows 7
- Windows 8
- Windows Server 2008 R2
- Windows Server 2012 R2

For All Systems

For a successful installation of Squore, you will need:

- The latest version of the Squore CLI installer, which can be downloaded from https://www.vector.com
- A Java Runtime Environment version 8 or 11 (64-bits) (other versions are not supported)
- At least 4 GB of space available on the disk for a full installation with demo projects
- At least 8 GB of RAM on the server machine
- At least 4 GB of RAM on the client machine
- The *java* executable should be in the machine's PATH environment variable for Squore CLI to run successfully.

Keep in mind that the requirements above are the strict minimum. In production, Squore Server generally runs on a dedicated machine. A performant configuration is usually:

- 16 threads CPU.
- 64GB of RAM.
- SSD hard drives.

Squore reserves 25% of the available RAM of the machine to the database and another 25% to the server. External processes (like Checkstyle or FindBugs) running on the same machine as Squore may add to the amount of RAM required for analysing source code. Linux is known to offer better performances than Windows when running Squore. For a production database, you should plan a minimum of 20 GB of disk space.

Packages for Windows

A JRE is required for Squore CLI. The Windows installer contains the tcl and perl runtimes needed. It will allow you to obtain the configuration needed to create projects from the server.

Packages for CentOS and Red Hat Enterprise Linux

On Red Hat Enterprise Linux and CentOS (6.5, 7.1 and 8), the dependencies are satisfied by the following packages:

Mandatory packages:

- **java-1.8.0-openjdk**
- **perl**
- **perl-Date-Calc**
- **perl-Digest-SHA**
- **perl-JSON**
- **perl-libwww-perl**
- **perl-Time-HiRes**
- **perl-XML-Parser**
- **fontconfig**
- **tcl**

Optional packages for working with RTRT:

- **perl-XML-Simple**

Optional packages for working with Microsoft Excel:

- **perl-HTML-Parser**
- **perl-CPAN**(CPAN utility requirement)
- **perl-Spreadsheet-ParseExcel**(available in the EPEL repository)
- **perl-Spreadsheet-XLSX**(available in the EPEL repository)

The module **Spreadsheet::BasicRead** is not available as a package and must therefore be installed using cpan (make sure cpan is properly configured, by running `cpan` without arguments first):

```
sudo cpan -i Spreadsheet::BasicRead
```

Optional packages for working with OSLC systems:

- **perl-TimeDate**
- **perl-WWW-Mechanize**(available in the EPEL repository)
- **perl-XML-LibXML**

Optional packages for Advanced CSV Export Management:

- **perl-Text-CSV**

Optional packages for working with Mantis, Jira and other ticket management software:

- **perl-TimeDate**
- **perl-JSON-XS**
- **perl-Spreadsheet-ParseExcel**(available in the EPEL repository)
- **perl-Text-CSV**
- **perl-WWW-Mechanize**(available in the EPEL repository)
- **perl-XML-LibXML**

The module **Spreadsheet::BasicRead** is not available as a package and must therefore be installed using cpan (make sure cpan is properly configured, by running `cpan` without arguments first):

```
sudo cpan -i Spreadsheet::BasicRead
```

For more information about how to install the Extra Packages for Enterprise Linux (EPEL) repository, consult https://fedoraproject.org/wiki/EPEL.

Packages for Ubuntu

On Ubuntu 16.04 LTS, 18.04 LTS and 20.04 LTS, the dependencies are satisfied by the following packages:

Mandatory packages:

- **libdate-calc-perl**

- **libhttp-message-perl**
- **libjson-perl**
- **libwww-perl**
- **libxml-parser-perl**
- **openjdk-8-jre**
- **perl**
- **tcl**

Optional packages for working with RTRT:

- **libxml-simple-perl**

Optional packages for working with Microsoft Excel:

- **make** (CPAN utility requirement)
- **libhtml-parser-perl**
- **libspreadsheet-parseexcel-perl**
- **libspreadsheet-xlsx-perl**

The module **Spreadsheet::BasicRead** is not available as a package and must therefore be installed using cpan (make sure cpan is properly configured, by running `cpan` without arguments first):

```
sudo cpan -i Spreadsheet::BasicRead
```

Optional packages for working with OSLC systems:

- **libtimedate-perl**
- **libwww-mechanize-perl**
- **libxml-libxml-perl**

Optional packages for Advanced CSV Export Management:

- **libtext-csv-perl**

Optional packages for working with Mantis, Jira and other ticket management software:

- **libtimedate-perl**
- **libjson-perl**
- **libspreadsheet-parseexcel-perl**(available in the EPEL repository)
- **libtext-csv-perl**
- **libwww-mechanize-perl**(available in the EPEL repository)
- **libxml-libxml-perl**

The module **Spreadsheet::BasicRead** is not available as a package and must therefore be installed using cpan (make sure cpan is properly configured, by running `cpan` without arguments first):

```
sudo cpan -i Spreadsheet::BasicRead
```

Packages for other Linux distributions

On Linux platforms, the following must be installed before installing Squore:

- **Perl** version 5.10.1 or greater including the following extra-modules:
  - Mandatory packages:
    - **Date::Calc** [module details]
    - **Digest::SHA** [module details]
    - **HTTP::Request** [module details]
    - **JSON** [module details]
    - **LWP** [module details]
    - **LWP::UserAgent** [module details]
    - **Time::HiRes** [module details]
    - **XML::Parser** [module details]
  - Optional packages for working with RTRT:
    - **XML::Simple** [module details]
  - Optional packages for working with Microsoft Excel:
    - **HTML::Entities** [module details]
    - **Spreadsheet::BasicRead** [module details]
  - Optional packages for working with OSLC systems:
    - **Date::Parse** [module details]
    - **WWW::Mechanize** [module details]
    - **XML::LibXML** [module details]
  - Optional packages for Advanced CSV Export Management:
    - **Text::CSV** [module details]
  - Optional packages for working with Mantis, Jira and other ticket management software:
    - **Date::Parse** [module details]
    - **JSON::XS** [module details]
    - **Spreadsheet::ParseExcel** [module details]
    - **Spreadsheet::BasicRead** [module details]
    - **Text::CSV** [module details]
    - **WWW::Mechanize** [module details]
    - **XML::LibXML** [module details]

If some of these modules are not available as packages on your operating system, use your perl installation's cpan to install the modules. Using the OS packages is recommended, as it avoids having to reinstall via cpan after upgrading your version of perl.

- **Tcl** version 8.5 or greater,

Deploying Squore CLI

Note that Oracle's Java Runtime Environment 8 or 11 (64-bits) (other versions are not supported) is required on the client machine for the CLI to run.

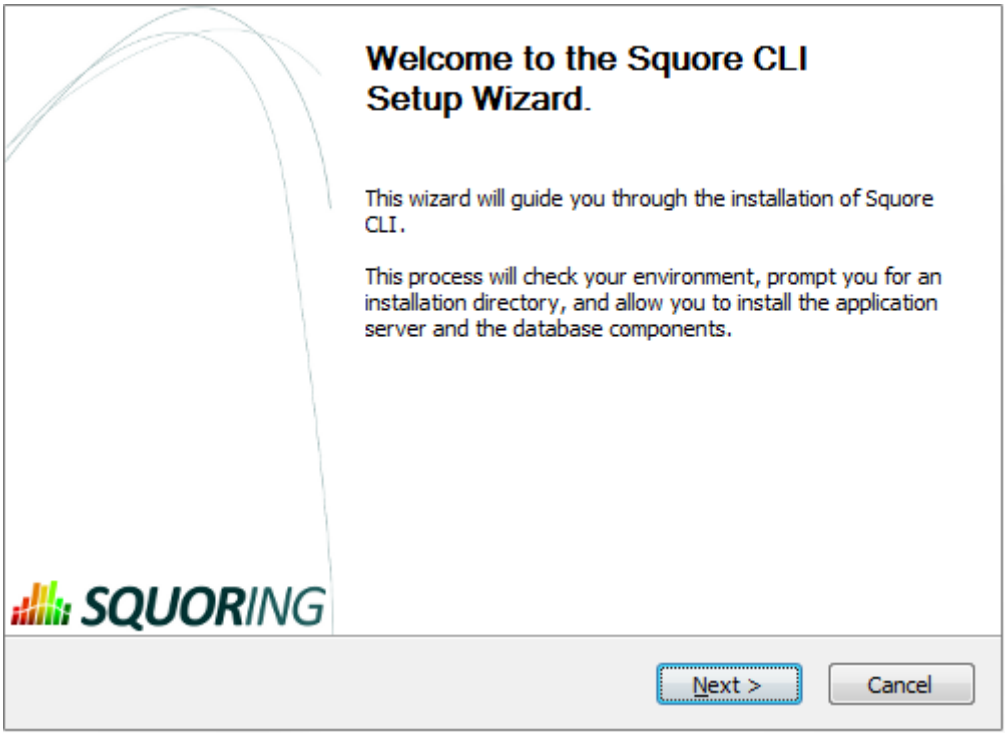
There is currently no Squore CLI installation package for Squore 17.0 and up. If you need to install Squore CLI, download the latest 16.3 version from and perform an upgrade after installing following the steps on https://wiki.squoring.com/display/HOW/Installing+a+Squore+client+for+Squore+17+and+later.

On Windows

After verifying that you meet the prerequisites detailed in Installation Prerequisites, log on with an account that has administrator privileges and launch Squore CLI installer. Each of the wizard screens is documented below in the order that you will see them.

The data and temporary folders must be excluded from the scope of virus scanners, malware protectors and search indexers to avoid any errors during an analysis.

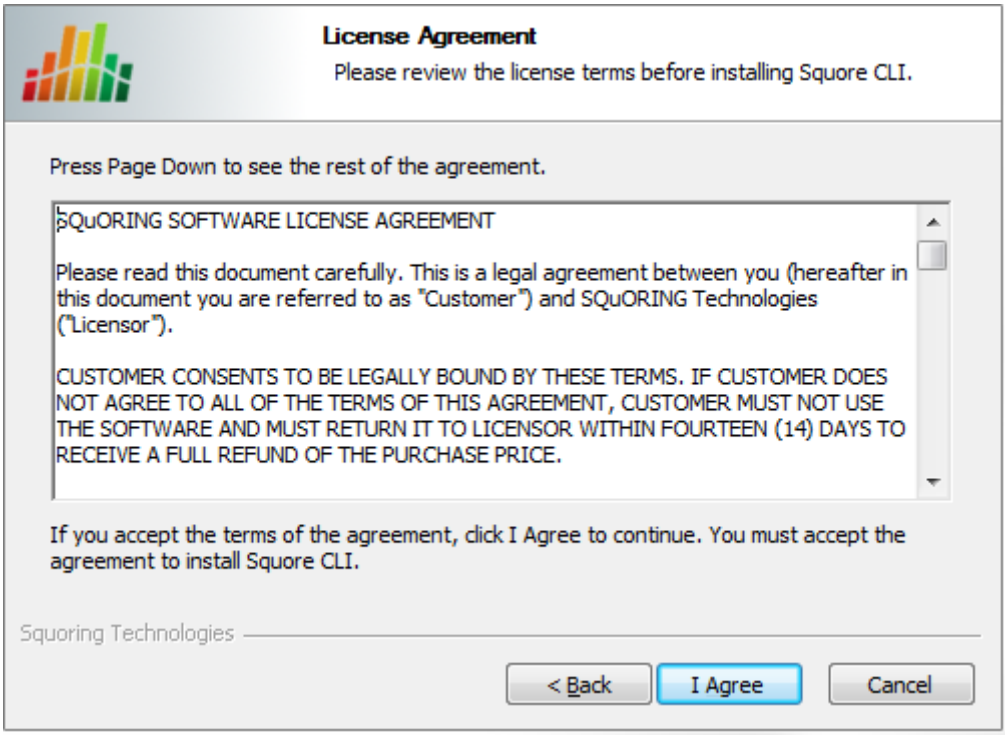
1. Squore CLI installer Welcome screen



Squore CLI installer Welcome screen

On the Welcome screen, click the Next button to start the installation.

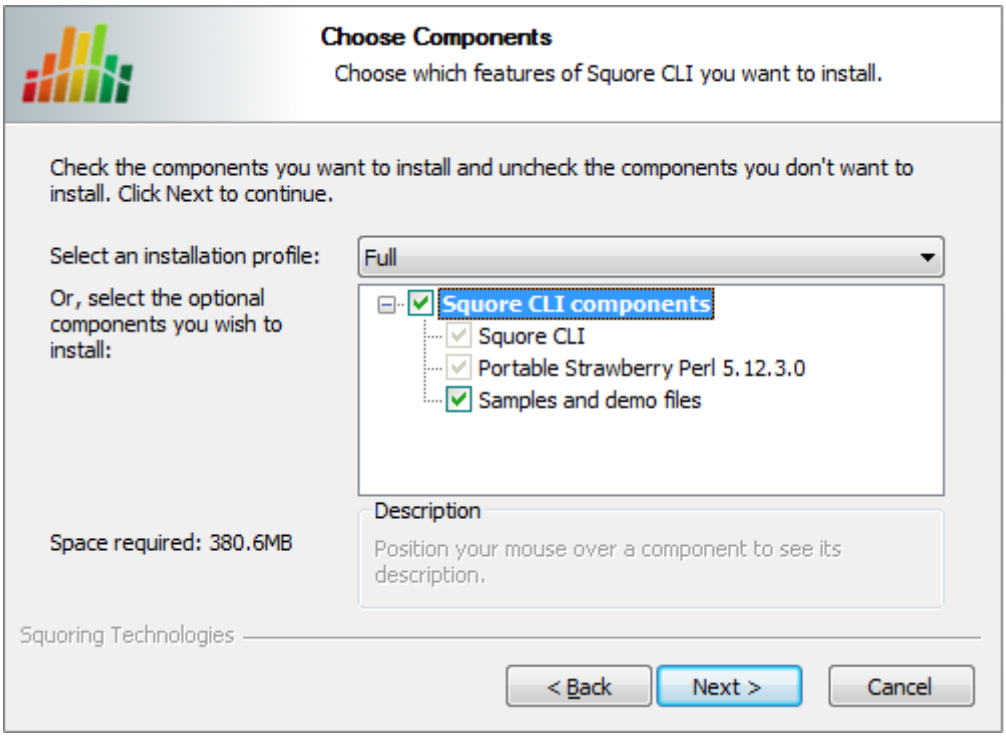
2. Squore CLI licence agreement screen



Squore CLI licence agreement screen

Click the **I Agree** button after reviewing the terms of the licence to continue the installation.

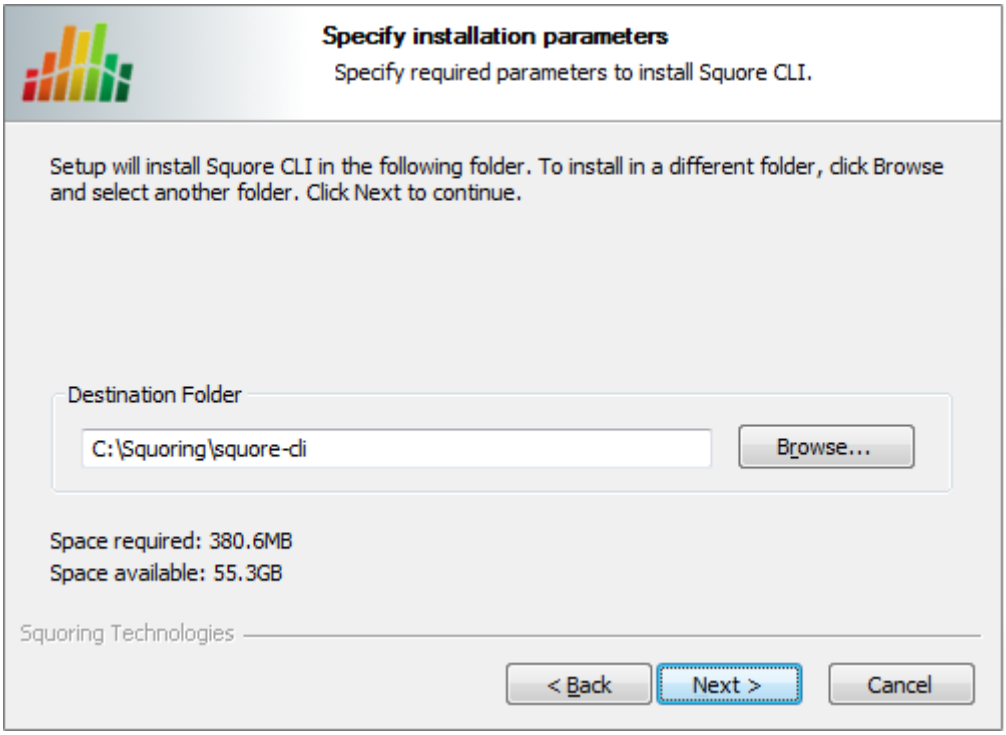
3. Squore CLI components screen



Squore CLI components screen

Select the components you want to install and click the Next button to proceed to the next step of the installation.

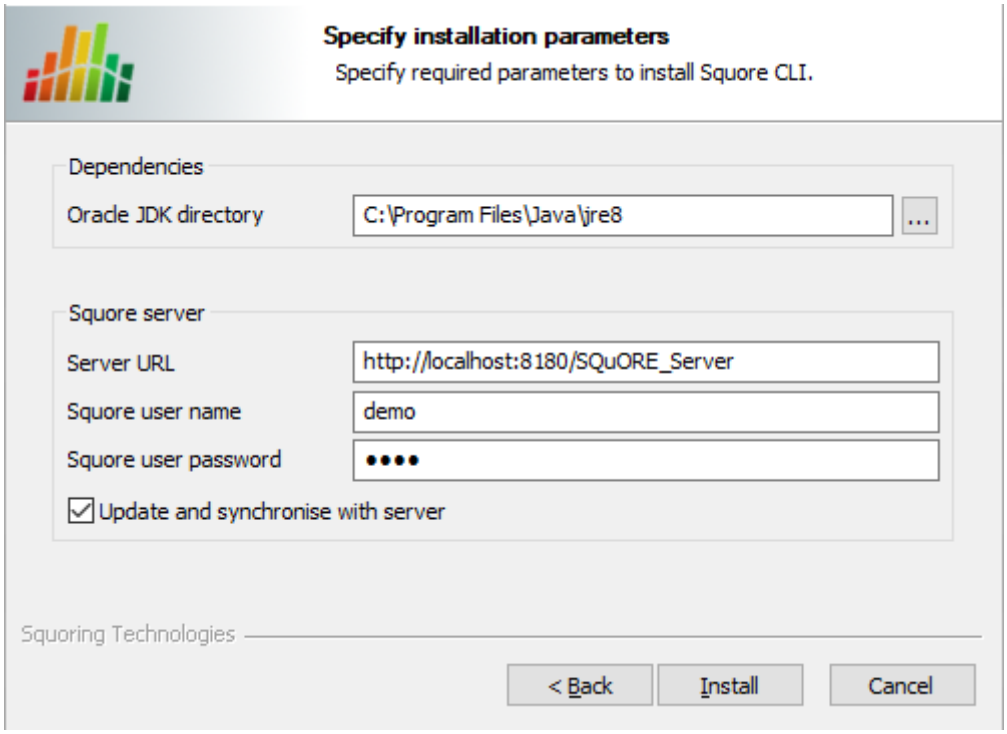
4. Squore CLI destination folder screen



Squore CLI destination folder screen

Browse for the folder where you want to deploy Squore CLI and click the Next button to proceed to the next step of the installation.

5. Squore CLI installation parameters screen



Squore CLI installation parameters screen

Specify the path of the Java installation on your system. Specify the details of Squore Server that the client should connect to. if you check the *Update and synchronise with server* box, the installer will attempt to retrieve the up-to-date client binaries from the server as well as the configuration. Click the Next button to start copying the installation files onto your hard disk.

If an error happens during the installation process, a log file is available in the destination folder you selected during the installation.

On Linux

Before installing Squore CLI on a Linux platform, verify that all prerequisites are met, as described in Installation Prerequisites

1. Copy the installation package (a compressed tar.xz archive) into the location where you want to install Squore CLI (For example: */opt/squore/*).
2. Extract the contents of the archive into the selected installation directory.

   The folder now contains a new folder called *squore-cli*, which we will refer to as ***<SQUORE_HOME>***.

3. Run the installation script in a command shell:

```
&lt;SQUORE_HOME&gt;/bin/install -v -s http://localhost:8180/SQuORE_Server -u user
-p password
```

For more details on `install` options, refer to install(2).

When installing Squore CLI, a connection to Squore Server is automatically attempted to retrieve the most up-to-date client and configuration. You can disable this synchronisation attempt by passing *-N* to the installation script.

Third-Party Plugins and Applications

If you have deployed some third-party tools on Squore Server, they will automatically be downloaded to your client when you launch the client synchronisation script.

AntiC and Cppcheck on Linux also require special attention: Cppcheck must be installed and available in the path, and antiC must be compiled with the command:

```
cd addons/Antic_auto/bin/ && gcc antic.c -o antic
```

For more information, refer to the Command Line Interface Manual, which contains the full details about special installation procedures for Data Providers and Repository Connectors.

Post Installation Actions

After the CLI installation is successful, you can familiarise yourself will the structure of the installation directory:

- **<SQUORE_HOME>/addons** A folder containing the Data Providers of the product.
- **<SQUORE_HOME>/bin** A folder containing sample projects creation scripts and utilities.
- **<SQUORE_HOME>/configuration** A configuration of the product containing the tools, wizards and analysis models.
- **<SQUORE_HOME>/docs** A folder containing the Command Line Interface manual.
- **<SQUORE_HOME>/lib** A folder containing the main engine and its client libraries.
- **<SQUORE_HOME>/samples** A folder containing sample source code to be used with the sample launchers supplied in <SQUORE_HOME>/bin.
- **<SQUORE_HOME>/share:** A folder containing specific perl libraries used by the CLI to launch

jobs.

- **<SQUORE_HOME>/tools** A folder containing the perl and tclsh distributions on Windows. This folder does not exist in the Linux version, since the system installations of perl and tclsh are used.
- **<SQUORE_HOME>/config.xml** An XML configuration file that the CLI uses to find its configuration.

After installing Squore CLI, the credentials for the user you specified during the installation have been saved, and the scripts in <SQUORE_HOME>/bin will use the username and password specified.

The file *config.xml* contains information about the Squore CLI installation.. Here is the default *config.xml*:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<squore type="client" version="1.3">
    <paths>
        <path name="perl.dir" path="/path/to/perl"/>
        <path name="tclsh.dir" path="/path/to/tclsh"/>
    </paths>
    <configuration>
        <path directory="&lt;SQUORE_HOME&gt;/configuration"/>
    </configuration>
    <addons>
        <path directory="&lt;SQUORE_HOME&gt;/addons"/>
    </addons>
</squore>
```

You can extend your *config.xml* by specifying where you want the temporary and data files to be stored on your system, as shown below:

- *Folder used to store temporary log files:* <tmp directory="${java.io.tmpdir}/squore-${user.name}"/>
- *Folder used to run analyses and store project files before they are sent to the server:* <projects directory="${user.home}/.squore/projects"/>
- *Folder used when extracting files from SCM systems:* <sources directory="${java.io.tmpdir}/sources"/>

Using java system properties to specify the paths to the *tmp*, *projects* and *sources* folders is useful if you want the Squore CLI installation to work for multiple users. Note that all three elements are optional, and will use the values shown above by default if you do not specify them in *config.xml*.

Here is an example of a full *config.xml*:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<squore type="client" version="1.3">
    <paths>
        <path name="perl.dir" path="/path/to/perl"/>
        <path name="tclsh.dir" path="/path/to/tclsh"/>
    </paths>
    <configuration>
        <path directory="<INSTALLDIR>/configuration"/>
    </configuration>
    <addons>
        <path directory="<INSTALLDIR>/addons"/>
    </addons>
    <tmp directory="${java.io.tmpdir}/squore-${user.name}"/>
    <projects directory="${user.home}/.squore/projects"/>
    <sources directory="${java.io.tmpdir}/sources"/>
</squore>
```

Note that all three folders can be cleaned up regularly when no analysis is running.

${user.home} corresponds to $HOME on linux and %APPDATA% on Windows

${java.io.tmpdir} corresponds to /tmp on linux and %TEMP% on Windows

Upgrading Squore CLI

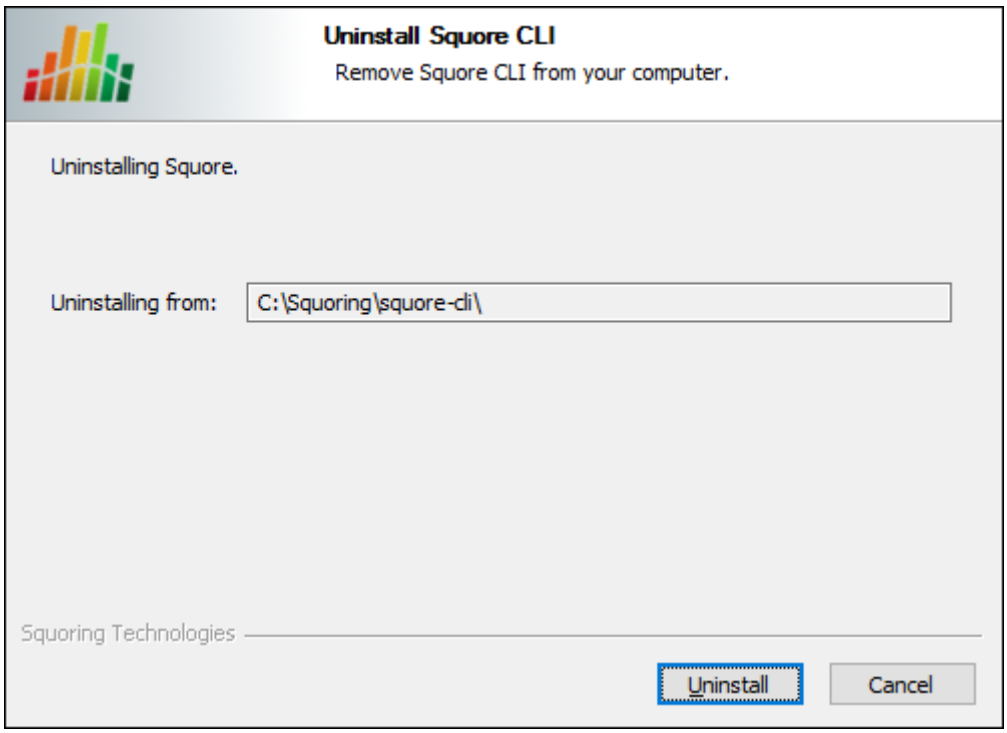
In order to upgrade Squore CLI to a new version, simply run *<SQUORE_HOME>\bin\synchronise.bat* (on Windows) or *<SQUORE_HOME>/bin/synchronise* (on Linux) script to retrieve the latest version of the binaries from Squore Server.

Removing Squore CLI

On Windows

You can remove Squore Server from your machine by going through the uninstaller wizard, as described below:

1. Launch the uninstaller wizard from the **Add/Remove Programs** dialog in the control panel or directly by double-clicking *<SQUORE_HOME>/Squore_CLI_Uninst.exe*. The wizard opens:

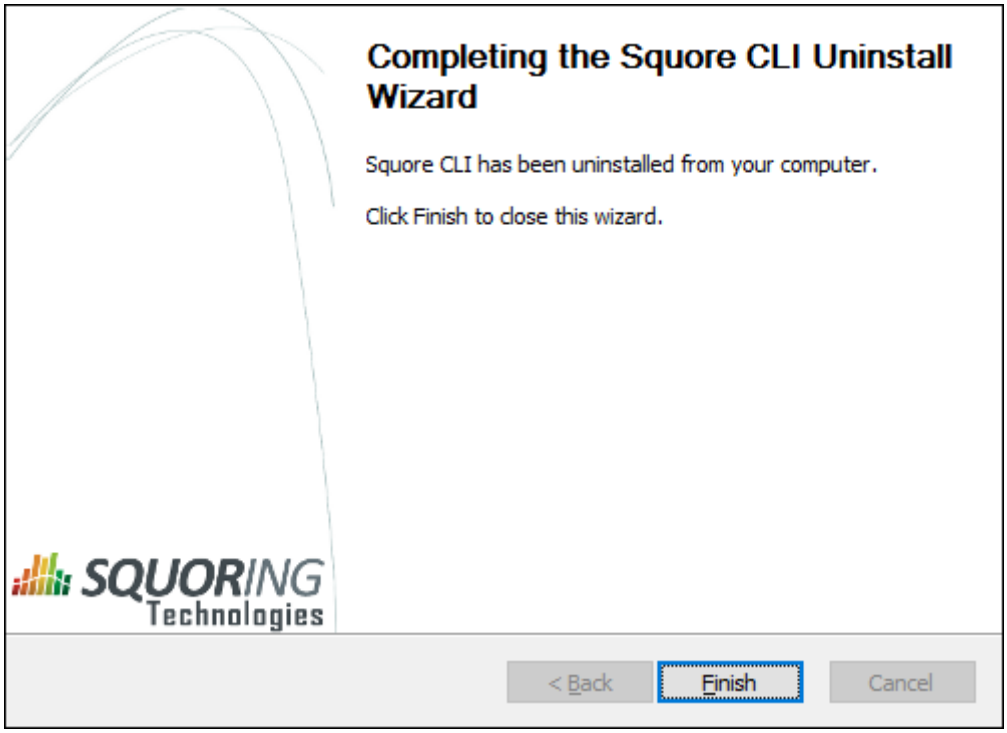


The Squore CLI uninstallation wizard

Click **Uninstall** to procede with the removal of the software.

This operation cannot be interrupted or rolled-back.

2. The wizard will notify you when the uninstallation finishes, as shown below:



The **Uninstallation Complete** *screen*

Click **Finish** to exit the wizard.

On Linux

There is no uninstallation script for Squore CLI on linux. In order to completely remove Squore CLI

from your system, delete <SQUORE_HOME>, the folder containing *config.xml* and the Squore binaries.

Setting up HTTPS

If HTTPS redirection is setup on your Squore server then the HTTPS URL must be used in the command line.

For Java to be able to trust/certify the connection with the server, you also need to add the SSL certificate in the Java truststore of the client machine hosting Squore CLI.

Java default truststore is a *cacert* file located in the *java.home/lib/security* directory. Where *java.home* is the runtime environment's directory.

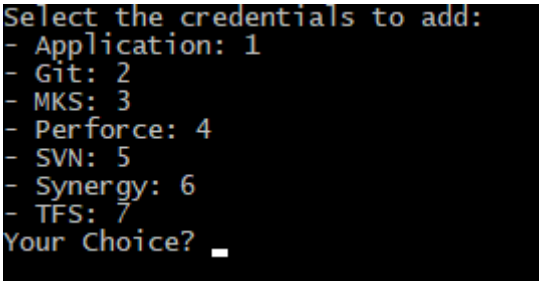
The default password of Java default truststore is *changeit*.

Refer to the Import a certificate section to know how to import the certificate in the truststore.

Saving Credentials to Disk

Squore CLI includes a small utility called add-credentials that can save your credentials to disk. This avoids typing your password every time you create a project, and also avoids having to save the password in your script files.

add-credentials is located in <SQUORE_HOME>/bin and allows saving passwords for Squore users and the various Repository Connectors known to Squore. To start saving credentials, simply run add-credentials.sh on Linux or add-credentials.bat on Windows. You are presented with a choice of several types of credentials you can save:



Available credentials types in add-credentials

In order to save user credentials for Squore Server, select **1**, then type the login and associated password.

In order to save credentials for a SVN server, select **2**. add-credentials.sh will prompt you for the URL of the SVN repository, for example *https://svnserver/var/svn*. Upon confirming, you will be prompted for your username and password to access this SVN URL.

Note that the saved credentials are only used by Squore CLI. When you use Squore's web interface, you will need to enter your password again to log in or browse source code.

Credentials are only saved for the current user. If you want to clear the credentials saved for a user profile, remove the file *$HOME/.squorerc* on linux or *%USERPROFILE%\.squorerc* on Windows.

Adding credentials can be done from the command line by running the following command:

```
java -cp /path/to/squore-engine.jar -Dsquore.home.dir=$SQUORE_HOME
com.squoring.squore.client.credentials.MakeCredentials --type squore
--login demo --password demo --url http://localhost:8180/SQuORE_Server
```

Running The Sample Scripts

The *<SQUORE_HOME>/bin* folder contains scripts that use the source code in the folder *<SQUORE_HOME>/samples* to create demo projects. You can also copy the command lines in these scripts to start creating your own projects.

A sample job instruction is a call to *squore-engine.jar* with some arguments and parameters to specify the Data Providers, Repository Connectors and attribute values you want to use, for example:

```
java -Dsquore.home.dir="<SQUORE_HOME>" -jar squore-engine.jar
 --url="<server_url>" --login="<LOGIN>" --password="<password>
 --name="myProject" --wizardId="ANALYTICS"
 -r "type=FROMPATH,path=/path/to/java/sources"
 --commands "DELEGATE_CREATION"
```

To learn more about command line parameters, refer to Command Line Reference

Squore in a Continuous Integration Environment

Squore can be used in a continuous integration environment using the commands detailed in Command Line Reference

Below is an example of a native call to the client using the ant exec task:

```
<project name="CIproject" default="build" basedir=".">
    <property name="server.url" value="http://localhost:8180/SQuORE_Server">
    <property name="cli.home" value="D:\CLI"/>

    <target name="build">
        <exec executable="java">
            <arg value="-Dsquore.home.dir=${cli.home}"/>
            <arg value="-jar"/>
            <arg value="${cli.home}\lib\squore-engine.jar"/>
            <arg value="--url=${server.url}" />
            <arg value="--version=${label}" />
            <arg value="--repository type=FROMPATH,path=${source.dir}" />
            <arg value="--color=rgb(255,0,0)" />
            <arg value="--name=${project.name}" />
            <arg value="--login=demo" />
            <arg value="--password=demo" />
            <arg value="--wizardId=Software Analytics" />
            <arg value="--tag APPLY_TO_BE_TESTED=0" />
            <arg value="--tag VOCF_THRESHOLD=10" />
            <arg value="--commands=PROCESS_CREATION" />
        </exec>
    </target>
</project>
```

You can use also java calls to **squore-engine.jar** in any automation server software.

Linking to Projects

There are two ways to build direct links to projects in Squore:

- using IDs with the **RestoreContext** page
- using names with the **LoadDashboard** page

Each method supports different parameters to build direct links to a tab of the Explorer for the specified project, as explained below.

RestoreContext

http://localhost:8180/SQuORE_Server/XHTML/RestoreContext.xhtml

Links to the Squore Explorer using IDs. The URL accepts the following parameters:

- **modelId** to link to a model node in the portfolio
- **projectId** to link to the latest version of the specified project
- **versionId** to link to a specific version of a project
- **artefactId** to link to a specific artefact in a project (can be combined with projectId and versionId)
- **tabName** to display a specific tab of the Explorer. This parameter can be combined with any of the ones above and must be one of:

- dashboard (default)
- action-items
- highlights
- findings
- documents
- attributes (Forms)
- indicators
- measures
- annotations

Users can copy a RestoreContext link from the Home page, the Projects page, or generate one using the **Share...** dialog in an artefact's context menu, which is the only way to find an artefactId. Model IDs are not exposed anywhere in the web interface.

The sharing dialog from the web UI with a full RestoreContext URL

Project and version IDs are printed in the project's output XML file, making it easy to parse and build a URL dynamically when using continuous integration to launch analyses.

LoadDashboard

http://localhost:8180/SQuORE_Server/XHTML/MyDashboard/dashboard/LoadDashboard.xhtml

Links to the Squore Explorer using names instead of IDs. The URL accepts the following parameters:

- **application (mandatory)** to specify the project to link to
- **version (optional)** to specify which version of the project to display. When not specified, the latest version fo the project is displayed
- **artefactId (optional)** to link to a specific artefact in the project
- **tabName** to display a specific tab of the Explorer. This parameter can be combined with any of the ones above and must be one of:
  - dashboard (default)
  - action-items
  - highlights
  - findings
  - documents
  - attributes (Forms)
  - indicators
  - measures

- annotations

The following is a URL that links to the version called V5 in the project called Earth. Since no artefactId and tabName are specified, the Dashboard tab will be displayed for the root node of the project: http://localhost:8180/SQuORE_Server/XHTML/MyDashboard/dashboard/LoadDashboard.xhtml?application=Earth&version=V5.

Chapter 3. Command Line Reference

In this chapter, you will learn about the commands and options you can use with squore-engine.jar

In order to run a command, you always need to specify at least:

- **-Dsquore.home.dir=<SQUORE_HOME>** to tell java there Squore CLI is installed
- **--url=http://localhost:8180/SQuORE_Server** to tell Squore CLI which Squore Server to connect to.
- **--login=demo** to tell Squore CLI which user to connect with.
- **--commands="..."** to tell Squore CLI what action you need it to perform.

squore.home.dir is used to set the location of Squore CLI's *config.xml* to *${squore.home.dir}/config.xml*. If your *config.xml* is in a different location, you can specify it on the command line with the option: **-Dsquore.configuration=/path/to/config.xml**.

Squore CLI Commands

This section details the list of commands you can use with Squore CLI and their meaning.

You will generally use a combination of these commands rather than a single command at a time.

- If you intend to use the client as a remote control to trigger project creations on the server, use **-c='DELEGATE_CREATION'**.
- A more common configuration is for the client to carry out the analysis and send the results to the server to create the project. This can be done by passing the commands **-c='SYNCHRONISE;PROCESS_CREATION'**.

Using the **SYNCHRONISE** command is optional but ensures that the client and the server are using the same model to produce analysis results.

RETRIEVE_ENGINE_PACKAGE
: Retrieves the full up-to-date package of the Engine and its libraries from the server.

SYNCHRONISE
: Retrieves the up-to-date configuration from the server.

GENERATE_CONF_PARAMETERS
: Generates the command line options associated to all parameters found in the specified configuration file. It requires the 'projectConfFile' option to be defined.

CHECK_MODELS
: Checks the validity of a model's configuration. It requires the 'outputCheckModelsFile' option.

PROCESS_CREATION
: Process project creation on client-side. With this option, Data Providers runs on the client side. the results are then sent to the server, which will performs the result analysis and will imports datas into the database.

DELEGATE_CREATION
: Sends the project settings to the server to request a project creation

GENERATE_OUTPUT
: Generates output data and statistics of the project's creation. It should always be called after

all other commands.

Squore CLI Parameters

Parameters are used to define the environment in which commands are processed. The list of parameters is as follows:

--commands="COMMAND", -c "COMMAND"
optional, default=''

The list of commands to launch. This list is a semicolon-separated string defining the commands to launch. Use *-commands="GET_COMMANDS_LIST"* to obtain the list of available commands. For more information about the available commands, refer to Squore CLI Commands.

--url="url", -s "url"
optional, default='http://localhost:8180/SQuORE_Server'

The URL of Squore Server to interact with.

--outputFile="/path/to/output.xml", --o "/path/to/output.xml"
optional, default='null'

The absolute path to the output file generated by the analysis

--outputCheckModelsFile="/path/to/validator.xml", -m "/path/to/validator.xml"
optional, default='null'

Defines the absolute path to the output check models file generated by the *CHECK_MODELS* command.

--printOutput="true|false", -print "true|false"
optional, default='false'

Redirect the engine's output to the standard output.

--help, -?
optional, default='false'

Displays help and exits.

--help:commands, -?cmd
optional, default='false'

Displays help about the available commands.

--subFoldersAsVersions="true|false", -sub "true|false"
optional, default='false'

Loops on the repository path to create a version for each sub-folder using the sub-folder name as the version name. This options is only supported when using the FROMPATH Repository Connector.

--projectConfFile="/path/to/project_conf.xml", -x "/path/to/project_conf.xml"
optional, default='null'

The XML file defining the project settings. When using a combination of a project file and some parameters passed from the command line, the command line parameters override the project file ones.

--updateModelFile="/path/to/ruleset.xml", -um "/path/to/ruleset.xml"
optional, default='null'

The XML file listing the changes to be applied to the standard analysis model for this analysis. The XML file contains a list of rules with their status and categories, as shown below:

```
<UpdateRules>
    <UpdateRule measureId="R_NOGOTO" disabled="true" categories=
"SCALE_SEVERITY.CRITICAL"/>
</UpdateRules>
```

This parameter is only read and applied when creating the first version of a project, for models where editing the ruleset is allowed. You may find it more flexible to work with named templates created in the Analysis Model Editor and specified on the command line with the **--rulesetTemplate** parameter, as described in Project Parameters.

Project Parameters

In order to create a project, you need to pass project parameters to Squore CLI. The following is a list of the parameters and their meaning:

--login="demo", -u "demo"
mandatory

The ID of the user requesting the project creation

--password="demo", -k "demo"
optional, default: ''

The password of the user requesting the project creation. If you do not want to specify a password in your command line, refer to Saving Credentials to Disk.

--name="MyProject", -n "MyProject"
mandatory

Defines the name of the project that will be created.

--WizardId="ANALYTICS" -w "ANALYTICS"
mandatory

The id of the wizard used to create the project.

--group="MyGroup"
optional, default: ''

Defines the group that the project belongs to. Projects from the same group are displayed together in the project portfolios and the group can optionally be rated as a whole. Note that you can specify subgroups by adding a **/** in your group name: --group="prototype/phase1" will create a **phase1** group under a **prototype** group.

--color="rgb(130,196,240)"
optional, default: randomly assigned

Defines the color used to identify the project in the Squore user interface after its creation. The numbers define the numbers define the values for red, green and blue respectively. Note that if

you do not specify a colour on the command line, a random colour will be picked.

--autoBaseline="true|false", -b "true|false"

optional, default: true

Instructs Squore CLI to build a baseline version that will not be overwritten by a subsequent analysis. When set to false, every analysis overwrites the previous one, until a new baseline is created. If not set, this parameter defaults to true.

--keepDataFiles="true|false"

optional, default: false

Instructs Squore to keep or discard analysis files from old versions or only for the latest analysis. Note that this behaviour only affects disk space on the server, not the analysis results

--version="V1", -v "V1"

optional, default: null

Defines the label used for the version of this project. If not specified, the version pattern parameter is used to generate the version label instead.

--versionPattern="V#.N#"

optional, default: null

Defines the pattern used to label the version automatically if no version parameter was passed.

The versionPattern parameter allows specifying a pattern to create the version name automatically for every analysis. It supports the following syntax:

- **#N#**: A number that is automatically incremented
- **#Nn#**: A number that is automatically incremented using n digits
- **#Y2#**: The current year in 2-digit format
- **#Y4#**: The current year in 4-digit format
- **#M#**: The current month in two digit format
- **#D#**: The current day in two digit format
- **#H#**: The current hour in 24 hour format
- **#MN#**: The current minute in two digit format
- **#S#**: The current second in two digit format

Any character other than **#** is allowed in the pattern. As an example, if you want to produce versions labelled *build-198.2013-07-28_13h07m* (where 198 is an auto-incremented number and the date and time are the timestamp of the project creation), you would use the pattern: **build-#N3#.#Y4#-#M#-#D#_#H#h#MN#m**

--versionDate="YYYY-MM-DDTHH:MM:SS"

optional, default: actual analysis time

Allows specifying a date for the version that is different from the current date. This is useful when the charts on your dashboard haves axes or intervals that show dates instead of version names. Note that for every new analysis, the date must be after the date of the previous analysis.

--teamUser="mike,DEVELOPER;john,TESTER;peter,PROJECT_MANAGER", -q "mike,DEVELOPER;john,TESTER;peter,PROJECT_MANAGER"

optional, default: ''

This semicolon-separated list of *login,projectRoleID* pairs is used to define a list of users who will be able to access the project when it is created.

Note that this option is taken into account when creating a new project but is ignored when creating a new version. In order to edit the list of users in a project team, you must use the Squore web interface.

Refer to the list of available projectRoleIDs in Squore by clicking **Administration > Roles**. This option can be combined with the **teamGroup** parameter if needed.

--teamGroup="devUsers,DEVELOPER;management,GUEST", -g "devUsers,DEVELOPER;management,GUEST"

optional, default: ''

This semicolon-separated list of *group,projectRoleID* pairs used to define a list of groups who will be able to access the project when it is created.

Note that this option is taken into account when creating a new project but is ignored when creating a new version. In order to edit the list of groups in a project team, you must use the Squore web interface.

Refer to the list of available projectRoleIDs in Squore by clicking **Administration > Roles**. This option can be combined with the **teamUser** parameter if needed.

--rulesetTemplate="my template"

optional, default: null

The name of the ruleset template created in the Analysis Model Editor that should be used for this analysis. For more information about ruleset templates, consult the Getting Started Guide.

--tag="TAGNAME=tagValue", -t "TAGNAME=tagValue"

optional, multiple

If the wizard allows tags (i.e. project attributes), then use the this parameter to inform the CLI of the tag values to use for this project.

--repository="type=REPOTYPE,opt1=value1,opt2=value2", -r "type=REPOTYPE,opt1=value1,opt2=value2"

optional, multiple

Used to specify repositories for sources. For more information about repositories syntax, refer to Repository Connectors. When using multiple source code repositories, each one musy have an **alias=NodeName** parameter that is used to create a folder containing the source code for the repository in the Artefact Tree.

--dp="type=DPName,dp_opt=dp_opt_value", -d "type=DPName,dp_opt=dp_opt_value"

optional, multiple

Used to specify information for Data Providers. For more information about individual Data Provider syntax, refer to Data Providers.

--filter="FILTER_OPTS", -f "FILTER_OPTS"

optional, default: ''

This semicolon-separated string of triplets {artefactType,filterType,filterValue}. In order to export the measure *LC*, a *DESCR* info the indicator *MAIN* at application level, pass **-f "APPLICATION,MEASURE,LC;APPLICATION,INFO,DESCR;APPLICATION,INDICATOR_LEVEL,MAIN;"**.

The artefact type **ALL_TYPES** and the filter types **ALL_DEFECT_REPORTS**, **ALL_MEASURES**,

ALL_INDICATORS_LEVELS, **ALL_INDICATORS_RANKS**, **ALL_BASE_FINDINGS**, **ALL_BASE_LINKS**, **ALL_COMPUTED_LINKS** and **ALL_INFOS** can also be used, followed by an empty filter value. In order to export all measures at application level in the output file, pass the parameter **--filter="APPLICATION,ALL_MEASURES,;"**. In order to export all indicators for all artefact types in the output file, pass the parameter **--filter="ALL_TYPES,ALL_INDICATORS_LEVELS,;"**

--exportZip="/path/to/project/data", -ez "/path/to/project/data"

optional, default: null

This parameter can be used to import a debug zip file as a new project. When this parameter is used, a new project is created using the paremeters in the *conf.xml* file inside the debug package, with any other parameter passed on the command line overriding the ones from the configuration file. Instead of a debug zip file, you can pass an absolute path to a project data folder to launch an analysis of all the version folders contained in that folder.

When using this option, you should pass **--strictMode="false", -S false** to disable some internal data integrity checks and change the project owner if needed using the **--owner="admin", -O "admin"** to se the new project owner (requires admin privileges).

This functionality is mostly used to test out how a project is rated in a different model, however it is not recommended for use in production since it will not replicate the following data from the old project to the new project:

- All comments and discussion threads
- Action Item statuses
- History of changes in forms and relaxation comments
- Relaxations and exclusions statuses of artefacts and findings

--milestone="id=BETA_RELEASE,date=2015/05/31,PROGRESS=95", -M "id=BETA_RELEASE,date=2015/05/31,PROGRESS=95"

optional, multiple

Allows you to define a milestone in the project. This parameter accepts a date and a series of metrics with their values to specify the goals for this milestone. Note that this parameter allows you to add milestones or modify existing ones (if the ID provided already exists), but removing a milestone from a project can only be done from the web interface.

You can also define milestones in your project file using a `Milestones` element in the `Wizard` section:

```
<SquoreProjectSettings>
    <Wizard>
        <Milestones>
            <Milestone id="BETA_RELEASE" date="2015-05-31">
                <Goal id="PROGRESS" value="95" />
            </Milestone>
        </Milestones>
    </Wizard>
</SquoreProjectSettings>
```

The rest of the parameters that you will pass to the Engine to create projects are specific to Repository Connectors and Data Providers and are detailed respectively in the Repository Connectors and Data Providers.

Exit Codes

After a successful or unsuccessful run, the CLI returns an exit code from this list:

0

OK - The operation completed successfully.

1

Client creation error - There was an error launching the client process.

2

Configuration error - This could be due to an unreachable configuration file or a parameter set to an invalid value.

3

Problem while launching one of the commands - One of the commands failed to complete successfully. The console should provide information about what exactly failed.

4

Engine error - The client failed to launch the analysis. More details about this error are available in the client console and in the server logs.

Chapter 4. Repository Connectors

Folder Path

Description

The simplest method to analyse source code in Squore is to provide a path to a folder containing your code.

Remember that the path supplied for the analysis is a path local to the machine running the analysis, which may be different from your local machine. If you analyse source code on your local machine and then send results to the server, you will not be able to view the source code directly in Squore, since it will not have access to the source code on the other machine. A common workaround to this problem is to use UNC paths (*Server**Share*, *smb://server/share*...) or a mapped server drive in Windows.

Usage

Folder Path has the following options:

- **Datapath (path, mandatory)**:
  - **Absolute Path**: the absolute path to the folder containing the files you want to include in the analysis. The path specified must be accessible from the server and user must have **Access Server Resources** permission.
  - **Authorized Paths**: a list of server paths accessible for all users, regardless of the **Access Server Resources** permission. This list can only be configured by a Squore administrator : Configure Authorized Server Paths.

The full command line syntax for Folder Path is:

```
-r "type=FROMPATH,path=[text]"
```

Zip Upload

Description

This Repository Connector allows you to upload a zip file containing your sources to analyse. Select a file to upload in the project wizard and it will be extracted and analysed on the server.

The contents of the zip file are extracted into Squore Server's temp folder. If you want to uploaded files to persist, contact your Squore administrator so that the uploaded zip files and extracted sources are moved to a location that is not deleted at each server restart.

Usage

This Repository Connector is only available from the web UI, not from the command line interface.

CVS

Description

The Concurrent Versions System (CVS), is a client-server free software revision control system in the field of software development.

For more details, refer to http://savannah.nongnu.org/projects/cvs.

The following is a list of commands used by the CVS Repository Connector to retrieve sources:

```
cvs -d $repository export [-r $branch] $project
```

```
cvs -d $repository co -r $artefactPath -d $tmpFolder
```

Usage

CVS has the following options:

- **Repository (`repository`, mandatory)**: Specify the location of the CVS Repository.
- **Project (`project`, mandatory)**: Specify the name of the project to get files from.
- **Tag or Branch (`branch`)**: Specify the tag or branch to get the files from.

The full command line syntax for CVS is:

```
-r "type=CVS,repository=[text],project=[text],branch=[text]"
```

ClearCase

Description

IBM Rational ClearCase is a software configuration management solution that provides version control, workspace management, parallel development support, and build auditing. The command executed on the server to check out source code is: $cleartool $view_root_path $view $vob_root_path.

For more details, refer to http://www-03.ibm.com/software/products/en/clearcase.

The ClearCase tool is configured for Linux by default. It is possible to make it work for Windows by editing the configuration file

Usage

ClearCase has the following options:

- **View root path (`view_root_path`, mandatory, default: /view)**: Specify the absolute path of the ClearCase view.
- **Vob Root Path (`vob_root_path`, mandatory, default: /projets)**: Specify the absolute path of the ClearCase vob.
- **View (`view`)**: Specify the label of the view to analyse sources from. If no view is specified, the current ClearCase view will be used automatically, as retrieved by the command cleartool pwv -s.
- **Server Display View (`server_display_view`)**: When viewing source code from the Explorer after building the project, this parameter is used instead of the view parameter specified earlier. Leave this field empty to use the same value as for view.
- **Sources Path (`sub_path`)**: Specify a path in the view to restrict the scope of the source code to analyse. The value of this field must not contain the vob nor the view. Leave this field empty to analyse the code in the entire view. This parameter is only necessary if you want to restrict to a directory lower than root.

The full command line syntax for ClearCase is:

```
-r
"type=ClearCase,view_root_path=[text],vob_root_path=[text],view=[text],server_display_
view=[text],sub_path=[text]"
```

Folder (use GNATHub)

Description

Retrieve Sources from a folder on the server, use GNATHub to limit the files (compatible with GNAT Pro versions 7.4.2 up to 18.2).

This Repository Connector will only be available after you configure your server or client *config.xml* with the path to your gnathub executable with a *<path name="gnatub" path="C:\tools\GNAThub\gnathub.exe" />* definition. Consult the Configuration Manual for more information about referencing external executables.

Usage

Folder (use GNATHub) has the following options:

- **Path of the source files (`path`)**: Specify the absolute path to the files you want to include in the analysis. The path specified must be accessible from the server.
- **Path of the gnathub.db file (`gnatdb`)**: Specify the absolute path of the gnathub.db file.
- **Root path for sources in the GNAT DB (`gnat_root`)**: Specify the root path for sources in the GNAT DB

The full command line syntax for Folder (use GNATHub) is:

```
-r "type=GNAThub,path=[text],gnatdb=[text],gnat_root=[text]"
```

Git

Description

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

For more details, refer to http://git-scm.com/.

The following is a list of commands used by the Git Repository Connector to retrieve sources:

```
git clone [$username:$password@]$url $tmpFolder
```

```
git checkout $commit
```

```
git log -1 "--format=%H"
```

```
git config --get

remote.origin.url
```

```
git clone [$username:$password@]$url $tmpFolder
```

```
git checkout $commit
```

```
git fetch
```

```
git --git-dir=$gitRoot show $artefactPath
```

Git 1.7.1 is known to fail with a *fatal: HTTP request failed* error on CentOS 6.9. For this OS, it is recommended to upgrade to git 2.9 as provided by software collections on https://www.softwarecollections.org/en/scls/rhscl/rh-git29/ and point to the new binary in *git_config.tcl* or make the change permanent as described on https://access.redhat.com/solutions/527703.

Usage

Git has the following options:

- **URL (`url`, mandatory)**: URL of the git repository to get files from. The local, HTTP(s), SSH and Git protocols are supported.
- **Branch or commit (`commit`)**: This field allows specifying the SHA1 of a commit or a branch name. If a SHA1 is specified, it will be retieved from the default branch. If a branch label is specified, then its latest commit is analysed. Leave this field empty to analyse the latest commit of the default branch.
- **Sub-directory (`subDir`)**: Specify a subfolder name if you want to restrict the analysis to a subpath of the repository root.
- **Authentication (`useAccountCredentials`, default: NO_CREDENTIALS)**: Possible values for authentication are
  - **No credentials**: Used when the underlying Git is open and does not require authetication
  - **Use my Squore credentials**: If the login/password are the same between Squore and the underlying git
  - **Define credentials**: To be prompted for login/password
- **Username (`username`)**:
- **Password (`password`)**:

The full command line syntax for Git is:

```
-r
"type=Git,url=[text],commit=[text],subDir=[text],useAccountCredentials=[multipleChoice
],username=[text],password=[password]"
```

PTC Integrity

Description

This Repository Connector allows analysing sources hosted in PTC Integrity, a software system lifecycle management and application lifecycle management platform developed by PTC.

For more details, refer to http://www.ptc.com/products/integrity/.

You can modify some of the settings of this repository connector if the si.exe and mksAPIViewer.exe binaries are not in your path. For versions that do not support the --xmlapi option, you can also turn off this method of retrieving file information. These settings are available by editing *mks_conf.tcl* in the repository connector's configuration folder.

Usage

PTC Integrity has the following options:

- **Server Hostname (`hostname`, mandatory)**: Specify the name of the Integrity server. This value is passed to the command line using the parameter --hostname.

- **Port (`port`)**: Specify the port used to connect to the Integrity server. This value is passed to the command line using the parameter --port.
- **Project (`project`)**: Specify the name of the project containing the sources to be analysed. This value is passed to the command line using the --project parameter.
- **Revision (`revision`)**: Specify the revision number for the sources to be analysed. This value is passed to the command line using the --projectRevision parameter.
- **Scope (`scope`, default: name:**.c,name:*.h)*: Specifies the scope (filter) for the Integrity sandbox extraction. This value is passed to the command line using the --scope parameter.
- **Authentication (`useAccountCredentials`, default: NO_CREDENTIALS)**:
- **Username (`username`)**:
- **Password (`password`)**:

The full command line syntax for PTC Integrity is:

```
-r
"type=MKS,hostname=[text],port=[text],project=[text],revision=[text],scope=[text],useA
ccountCredentials=[multipleChoice],username=[text],password=[password]"
```

Perforce

Description

The Perforce server manages a central database and a master repository of file versions. Perforce supports both Git clients and clients that use Perforce's own protocol.

For more details, refer to http://www.perforce.com/.

The Perforce repository connector assumes that the specified depot exists on the specified Perforce server, that Squore can access this depot and that the Perforce user defined has the right to access it.

The host where the analysis takes place must have a Perforce command-line client (p4) installed and fully functional.

The P4PORT environment variable is not read by Squore. You have to set it in the form. The path to the p4 command can be configured in the *perforce_conf.tcl* file located in the *configuration/repositoryConnectors/Perforce* folder.

The following is a list of commands used by the Perforce Repository Connector to retrieve sources:

```
p4 -p $p4port [-u username] [-P password] client -i
<$tmpFolder/p4conf.txt
```

```
p4 -p $p4port [-u username] [-P password] -c $clientName sync
"$depot/...@$label"
```

```
p4 -p $p4port [-u username] [-P password] client -d $clientName
```

```
p4 -p $p4port [-u username] [-P password] print -q -o $outputFile
$artefactPath
```

The format of the *p4conf.txt* file is:

```
Client: $clientName

Root: $tmpFolder

Options: noallwrite noclobber nocompress unlocked nomodtime normdir

SubmitOptions: submitunchanged

view:

$depot/... //$clientName/...
```

Usage

Perforce has the following options:

- **P4PORT (`p4port`, mandatory)**: Specify the value of P4PORT using the format [protocol:]host:port (the protocol is optional). This parameter is necessary even if you have specified an environment variable on the machine where the analysis is running.
- **Depot (`depot`, mandatory)**: Specify the name of the depot (and optionnally subforders)

containing the sources to be analysed.

- **Revision (`label`)**: Specify a label, changelist or date to retrieve the corresponding revision of the sources. Leave this field empty to analyse the most recent revision fo the sources.
- **Authentication (`useAccountCredentials`, default: NO_CREDENTIALS)**:
- **Username (`username`)**:
- **Password (`password`)**:

The full command line syntax for Perforce is:

```
-r
"type=Perforce,p4port=[text],depot=[text],label=[text],useAccountCredentials=[multiple
Choice],username=[text],password=[password]"
```

SVN

Description

Connecting to an SVN server is supported using svn over ssh, or by using a username and password.

For more details, refer to https://subversion.apache.org/.

The following is a list of commands used by the SVN Repository Connector to retrieve sources (you can edit the common command base or the path to the executable in *<SQUORE_HOME>/configuration/repositoryConnectors/SVN/svn_conf.tcl* if needed):

```
svn info --xml --non-interactive --trust-server-cert --no-auth-cache
[--username $username] [--password $password] [-r $revision] $url
```

```
svn export --force --non-interactive --trust-server-cert --no-auth
-cache [--username $username] [--password $password] [-r $revision]
$url
```

This Repository Connector includes a hybrid SVN mode saves you an extra checkout of your source tree when using the `local_path` attribute. Consult the reference below for more details.

Usage

SVN has the following options:

- **URL (`url`, mandatory)**: Specify the URL of the SVN repository to export and analyse. The following protocols are supported: svn://, svn+ssh://, http://, https:// .

- **Revision (`rev`)**: Specify a revision number in this field, or leave it blank to analyse files at the HEAD revision.
- **External references (`externals`, default: exclude)**: Specify if when extracting sources from SVN the system should also extract external references.
- **Sources are already extracted in (`local_path`)**: Specify a path to a folder where the sources have already been extracted. When using this option, sources are analysed in the specified folder instead of being checked out from SVN. At the end of the analysis, the url and revision numbers are attached to the analysed sources, so that any source code access from the web interface always retrieves files from SVN. This mode is mostly used to save an extra checkout in some continuous integration scenarios.
- **Authentication (`useAccountCredentials`, default: NO_CREDENTIALS)**:
- **Username (`username`)**:
- **Password (`password`)**:

The full command line syntax for SVN is:

```
-r
"type=SVN,url=[text],rev=[text],externals=[multipleChoice],local_path=[directory],useA
ccountCredentials=[multipleChoice],username=[text],password=[password]"
```

Synergy

Description

Rational Synergy is a software tool that provides software configuration management (SCM) capabilities for all artifacts related to software development including source code, documents and images as well as the final built software executable and libraries.

For more details, refer to http://www-03.ibm.com/software/products/en/ratisyne.

The Synergy Repository Connector assumes that a project already exists and that the Synergy user defined has the right to access it.

The host where the analysis takes place must have Synergy installed and fully functional. Note that, using credentials is only supported on Windows, so use the NO_CREDENTIALS option when Synergy runs on a Linux host (consult IBM's documentation at http://pic.dhe.ibm.com/infocenter/synhelp/v7m2r0/index.jsp?topic=%2Fcom.ibm.rational.synergy.manage.doc%2Ftopics%2Fsc_t_h_start_cli_session.html for more details).

The following is a list of commands used by the Synergy Repository Connector to retrieve sources:

```
ccm start -d $db -nogui -m -q [-s $server] [-pw $password] [-n $user
-pw password]
```

```
ccm prop "$path@$projectSpec"
```

```
ccm copy_to_file_system -path $tempFolder -recurse $projectSpec
```

```
ccm cat "$artefactPath@$projectSpec"
```

```
ccm stop
```

Usage

Synergy has the following options:

- **Server URL (`server`)**: Specify the Synergy server URL, if using a distant server. If specified, the value is used by the Synergy client via the -s parameter.
- **Database (`db`, mandatory)**: Specify the database path to analyse the sources it contains.
- **Project Specification (`projectSpec`, mandatory)**: Specify the project specification for the analysis. Source code contained in this project specification will be analysed recursively.
- **Subfolder (`subFolder`)**: Specify a subfolder name if you want to restrict the scope of the analysis to a particular folder.
- **Include Subprojects (`subProject`, default: yes)**: This option creates work area copies for the specified projects and all subprojects. If this option is not on, subprojects are ignored.
- **Ignore links (`ignoreLinks`, default: no)**: This option is used to ignore links to subprojects. This option is valid only on Linux systems.
- **Authentication: (`useAccountCredentials`, default: NO_CREDENTIALS)**: Note that, as stated in IBM's documentation, using credentials is only supported on Windows. The "No Credentials" option must be used when Synergy runs on a Linux host. For more information, consult http://pic.dhe.ibm.com/infocenter/synhelp/v7m2r0/index.jsp?topic=%2Fcom.ibm.rational.synergy.manage.doc%2Ftopics%2Fsc_t_h_start_cli_session.html.

- **Username (`username`)**:
- **Password (`password`)**:

The full command line syntax for Synergy is:

```
-r
"type=Synergy,server=[text],db=[text],projectSpec=[text],subFolder=[text],subProject=[
multipleChoice],ignoreLinks=[multipleChoice],useAccountCredentials=[multipleChoice],us
ername=[text],password=[password]"
```

TFS

Description

Team Foundation Server (TFS) is a Microsoft product which provides source code management, reporting, requirements management, project management, automated builds, lab management, testing and release management capabilities. This Repository Connector provides access to the sources hosted in TFS's revision control system.

For more details, refer to https://www.visualstudio.com/products/tfs-overview-vs.

The TFS repository connector (Team Foundation Server - Team Foundation Version Control) assumes that a TFS command-line client is installed and fully functional on the machine where the analysis runs. Two types of clients are supported: Team Explorer Everywhere (the java client, enabled by default) and Visual Studio Client (tf.exe).

Prior to using this repository connector, ensure that you have configured it to use the right client by adjusting settings in *<SQUORE_HOME>/configuration/repositoryConnectors/TFS/tfs_conf.tcl* file.

The Repository Connector form must be filled according to the TFS standard (eg. the Project Path must begin with the '$' character...). Note that this repository connector works with a temporary workspace that is deleted at the end of the analysis. The following is a list of commands used by the TFS Repository Connector to retrieve sources (this example uses the Windows client):

```
tf.exe workspace [/login:$username,$password] /server:$url /noprompt
/new $workspace
```

```
tf.exe workfold [/login:$username,$password] /map $path $tempFolder
/workspace:$workspace
```

```
tf.exe get [/login:$username,$password] /version:$version /recursive
/force $path
```

```
tf.exe workspace [/login:$username,$password] /delete $workspace
```

The following command is used when viewing sources in the web interface:

```
tf.exe view [/login:$username,$password] /server:$artefactPath
```

When using the Java Team Explorer Everywhere client, **/** is replaced by **-** and the **view** command is replaced by **print**.

Usage

TFS has the following options:

- **URL (`URL`, mandatory)**: Specify the URL of the TFS server.
- **Path (`path`, mandatory)**: Path the project to be analysed. This path usually starts with $.
- **Version (`version`)**: Specify the version of the sources to analyse. This field accepts a changeset number, date, or label. Leave the field empty to analyse the most recent revision of the sources.
- **Authentication (`useAccountCredentials`, default: NO_CREDENTIALS)**:
- **Username: (`username`)**:

- **Password (`password`)**:

The full command line syntax for TFS is:

```
-r
"type=TFS,URL=[text],path=[text],version=[text],useAccountCredentials=[multipleChoice]
,username=[text],password=[password]"
```

Using Multiple Nodes

Squore allows using multiple repositories in the same analysis. If your project consists of some code that is spread over two distinct servers or SVN repositories, you can set up your project so that it includes both locations in the project analysis. This is done by labelling each source code node before specifying parameters, as shown below

```
-r "type=FROMPATH,alias=Node1,path=/home/projects/client-code"
-r "type=FROMPATH,alias=Node2,path=/home/projects/common/lib"
```

Note that only alpha-numeric characters are allowed to be used as labels. In the artefact tree, each node will appear as a separate top-level folder with the label provided at project creation.

Using multiple nodes, you can also analyse sources using different Repository Connectors in the same analysis:

```
-r "type=FROMPATH,alias=Node1,path=/home/projects/common-config"
-r "type=SVN,alias=Node2,url=svn+ssh://10.10.0.1/var/svn/project/src,rev=HEAD"
```

Chapter 5. Data Providers

This chapter describe the available Data Providers and the default parameters that they accept via the Command Line Interface.

AntiC

Description

AntiC is a part of the jlint static analysis suite and is launched to analyse C and C++ source code and produce findings.

For more details, refer to http://jlint.sourceforge.net/.

On Linux, the antiC executable must be compiled manually before you run it for the first time by running the command:

```
# cd {squore_home}/addons/tools/Antic_auto/bin/ && gcc antic.c -o antic
```

Usage

AntiC has the following options:

- **Source code directory to analyse (`dir`)**: Leave this parameter empty if you want to analyse all sources specified above.

The full command line syntax for AntiC is:

```
-d "type=Antic_auto,dir=[directory]"
```

Automotive Coverage Import

Description

Automotive Coverage Import provides a generic import mechanism for coverage results at function level.

Usage

Automotive Coverage Import has the following options:

- **CSV file (`csv`)**: Enter the path to the CSV containing the coverage data.

  The expected format of each line contained in the file is **PATH;NAME;TESTED_C1;OBJECT_C1;TESTED_MCC;OBJECT_MCC;TESTED_MCDC;OBJECT_MCDC**

The full command line syntax for Automotive Coverage Import is:

```
-d "type=Automotive_Coverage,csv=[file]"
```

Automotive Tag Import

Description

This data provider allows setting values for attributes in the project.

Usage

Automotive Tag Import has the following options:

- **CSV file (`csv`)**: Specify the path to the file containing the metrics.

The full command line syntax for Automotive Tag Import is:

```
-d "type=Automotive_Tag_Import,csv=[file]"
```

BullseyeCoverage Code Coverage Analyzer

Description

BullseyeCoverage is a code coverage analyzer for C++ and C. The coverage report file is used to generate metrics.

For more details, refer to http://www.bullseye.com/.

Usage

BullseyeCoverage Code Coverage Analyzer has the following options:

- **HTML report (`html`)**: Specify the path to the HTML report file generated by BullseyeCoverage.

The full command line syntax for BullseyeCoverage Code Coverage Analyzer is:

```
-d "type=BullseyeCoverage,html=[file]"
```

CANoe

Description

Import data from CANoe XML test results

For more details, refer to https://www.vector.com/int/en/products/products-a-z/software/canoe/.

Usage

CANoe has the following options:

- **Results folder (`dir`)**: Specify the folder containing XML test results files from CANoe.
- **File suffix (`suff`, default: .xml)**: Provide the suffix of CANoe test results files.
- **Create Test artefacts? (`createTests`, default: YES)**: Shoult Test artefacts be created?
- **Test path (`testPath`, default: Tests)**: Define test path (for example Test/HIL Test), by default the value is Tests.

The full command line syntax for CANoe is:

```
-d
"type=CANoe,dir=[directory],suff=[text],createTests=[multipleChoice],testPath=[text]"
```

CPD

Description

CPD is an open source tool which generates Copy/Paste metrics. The dectection of duplicated blocks is set to 100 tokens. CPD provides an XML file which can be imported to generate metrics as well as findings.

For more details, refer to http://pmd.sourceforge.net/pmd-5.3.0/usage/cpd-usage.html.

Usage

CPD has the following options:

- **CPD XML results (`xml`)**: Specify the path to the XML results file generated by CPD. The minimum supported version is PMD/CPD 4.2.5.

The full command line syntax for CPD is:

```
-d "type=CPD,xml=[file]"
```

Cppcheck

Description

Cppcheck is a static analysis tool for C/C++ applications. The tool provides an XML output which can be imported to generate findings.

For more details, refer to http://cppcheck.sourceforge.net/.

Usage

Cppcheck has the following options:

- **Cppcheck XML results (`xml`)**: Specify the path to the XML results file from Cppcheck. Note that the minimum required version of Cppcheck for this data provider is 1.61.

The full command line syntax for Cppcheck is:

```
-d "type=CPPCheck,xml=[file]"
```

Cppcheck (plugin)

Description

Cppcheck is a static analysis tool for C/C++ applications. The tool provides an XML output which can be imported to generate findings.

For more details, refer to http://cppcheck.sourceforge.net/.

On Windows, this data provider requires an extra download to extract the Cppcheck binary in *<SQUORE_HOME>/addons/tools/CPPCheck_auto/* and the MS Visual C++ 2010 Redistributable Package available from http://www.microsoft.com/en-in/download/details.aspx?id=5555. On Linux, you can install the cppcheck application anywhere you want. The path to the Cppcheck binary for Linux can be configured in *config.tcl*. For more information, refer to the Installation and Administration Guide's 'Third-Party Plugins and Applications' section.

Usage

Cppcheck (plugin) has the following options:

- **Source code folder (`dir`)**: Specify the folder containing the source files to analyse. If you want to analyse all of source repositories specified for the project, leave this field empty.

- **Ignore List (`ignores`)**: Specify a semi-colon-separated list of source files or source file directories to exclude from the check. For example: "lib/;folder2/". Leave this field empty to deactivate this option and analyse all files with no exception.

The full command line syntax for Cppcheck (plugin) is:

```
-d "type=CPPCheck_auto,dir=[directory],ignores=[text]"
```

CPPTest

Description

Parasoft C/Ctest is an integrated solution for automating a broad range of best practices proven to improve software development team productivity and software quality for C and C. The tool

provides an XML output file which can be imported to generate findings and metrics.

For more details, refer to http://www.parasoft.com/product/cpptest/.

Usage

CPPTest has the following options:

- **Directory which contains the XML results files (`results_dir`)**: Specify the path to the CPPTest results directory. This data provider is compatible with files exported from CPPTest version 7.2.10.34 and up.
- **Results file extensions (`pattern`, default: *.xml)**: Specify the pattern of the results files

The full command line syntax for CPPTest is:

```
-d "type=CPPTest,results_dir=[directory],pattern=[text]"
```

Cantata

Description

Cantata is a Test Coverage tool. It provides an XML output file which can be imported to generate coverage metrics at function level.

For more details, refer to http://www.qa-systems.com/cantata.html.

Usage

Cantata has the following options:

- **Cantata XML results (`xml`)**: Specify the path to the XML results file from Cantata 6.2

The full command line syntax for Cantata is:

```
-d "type=Cantata,xml=[file]"
```

CheckStyle

Description

CheckStyle is an open source tool that verifies that Java applications adhere to certain coding standards. It produces an XML file which can be imported to generate findings.

For more details, refer to http://checkstyle.sourceforge.net/.

Usage

CheckStyle has the following options:

- **CheckStyle results file (`xml`)**: Point to the XML file that contains Checkstyle results. Note that the minimum supported version is Checkstyle 5.3.

The full command line syntax for CheckStyle is:

```
-d "type=CheckStyle,xml=[file]"
```

CheckStyle (plugin)

Description

CheckStyle is an open source tool that verifies that Java applications adhere to certain coding standards. It produces an XML file which can be imported to generate findings.

For more details, refer to http://checkstyle.sourceforge.net/.

This data provider requires an extra download to extract the CheckStyle binary in *<SQUORE_HOME>/addons/tools/CheckStyle_auto/*. For more information, refer to the Installation and Administration Guide's 'Third-Party Plugins and Applications' section. You may also deploy your own version of CheckStyle and make the Data Provider use it by editing *<SQUORE_HOME>/configuration/tools/CheckStyle_auto/config.tcl*.

Usage

CheckStyle (plugin) has the following options:

- **Configuration file (`configFile`)**: A Checkstyle configuration specifies which modules to plug in and apply to Java source files. Modules are structured in a tree whose root is the Checker module. Specify the name of the configuration file only, and the data provider will try to find it in the CheckStyle_auto folder of your custom configuration. If no custom configuration file is found, a default configuration will be used.
- **Xmx (`xmx`, default: 1024m)**: Maximum amount of memory allocated to the java process launching Checkstyle.
- **Excluded directory pattern (`excludedDirectoryPattern`)**: Java regular expression of directories to exclude from CheckStyle, for example: ^test|generated-sources|.*-report$ or ou ^lib$

The full command line syntax for CheckStyle (plugin) is:

```
-d "type=CheckStyle_auto,configFile=[text],xmx=[text],excludedDirectoryPattern=[text]"
```

CheckStyle for SQALE (plugin)

Description

CheckStyle is an open source tool that verifies that Java applications adhere to certain coding standards. It produces an XML file which can be imported to generate findings.

For more details, refer to http://checkstyle.sourceforge.net/.

This data provider requires an extra download to extract the CheckStyle binary in *<SQUORE_HOME>/addons/tools/CheckStyle_auto_for_SQALE/*. For more information, refer to the Installation and Administration Guide's 'Third-Party Plugins and Applications' section.

Usage

CheckStyle for SQALE (plugin) has the following options:

- **Configuration file (`configFile`, default: config_checkstyle_for_sqale.xml)**: A Checkstyle configuration specifies which modules to plug in and apply to Java source files. Modules are structured in a tree whose root is the Checker module. Specify the name of the configuration file only, and the data provider will try to find it in the CheckStyle_auto folder of your custom configuration. If no custom configuration file is found, a default configuration will be used.
- **Xmx (`xmx`, default: 1024m)**: Maximum amount of memory allocated to the java process launching Checkstyle.

The full command line syntax for CheckStyle for SQALE (plugin) is:

```
-d "type=CheckStyle_auto_for_SQALE,configFile=[text],xmx=[text]"
```

Cobertura format

Description

Cobertura is a free code coverage library for Java. Its XML report file can be imported to generate code coverage metrics for your Java project.

For more details, refer to http://cobertura.github.io/cobertura/.

Usage

Cobertura format has the following options:

- **XML report (`xml`)**: Specify the path to the XML report generated by Cobertura (or by a tool able to produce data in this format).

The full command line syntax for Cobertura format is:

```
-d "type=Cobertura,xml=[file]"
```

CodeSonar

Description

Codesonar is a static analysis tool for C and C++ code designed for zero tolerance defect environments. It provides an XML output file which is imported to generate findings.

For more details, refer to http://www.grammatech.com/codesonar.

Usage

CodeSonar has the following options:

- **XML results file (`xml`)**: Specify the path to the XML results file generated by Codesonar. The minimum version of Codesonar compatible with this data provider is 3.3.

The full command line syntax for CodeSonar is:

```
-d "type=CodeSonar,xml=[file]"
```

Compiler

Description

Compiler allows to import information from compiler logs.

Usage

Compiler has the following options:

- **Compiler output file(s) (`txt`, mandatory)**: Specify the path(s) to CSV compiler log file(s). To provide multiple files click on '+'.

  Each line needs to match the following format: **Path;Line;Rule;Descr** where Rule is one of COMP_ERR, COMPILER_WARN or COMPILER_INFO.

The full command line syntax for Compiler is:

```
-d "type=Compiler,txt=[file]"
```

Coverity

Description

Coverity is a static analysis tool for C, C++, Java and C#. It provides an XML output which can be imported to generate findings.

For more details, refer to http://www.coverity.com/.

Usage

Coverity has the following options:

- **XML results file (`xml`)**: Specify the path to the XML file containing Coverity results.

The full command line syntax for Coverity is:

```
-d "type=Coverity,xml=[file]"
```

ESLint

Description

ESLint is an open source tool that verifies that JavaScript applications adhere to certain coding standards. It produces an XML file which can be imported to generate findings.

For more details, refer to https://eslint.org/.

Usage

ESLint has the following options:

- **ESLint results file (`xml`)**: Point to the XML file that contains ESLint results in Checkstyle format.

The full command line syntax for ESLint is:

```
-d "type=ESLint,xml=[file]"
```

FindBugs-SpotBugs

Description

Findbugs (and its successor SpotBugs) is an open source tool that looks for bugs in Java code. It produces an XML result file which can be imported to generate findings.

For more details, refer to http://findbugs.sourceforge.net/.

Usage

FindBugs-SpotBugs has the following options:

- **XML results file (`xml`)**: Specify the location of the XML file containing Findbugs results. Note that the minimum supported version for FindBugs is 1.3.9, and 3.1.7 to 3.1.12 for SpotBugs.

The full command line syntax for FindBugs-SpotBugs is:

```
-d "type=Findbugs,xml=[file]"
```

FindBugs-SpotBugs (plugin)

Description

FindBugs is an open source tool that looks for bugs in Java code. It produces an XML result file which can be imported to generate findings. Note that the data provider requires an extra download to extract the FindBugs binary in [INSTALLDIR]/addons/tools/Findbugs/. You are free to use FindBugs 3.0 or FindBugs 2.0 depending on what your standard is. For more information, refer to the Installation and Administration Manual's "Third-Party Plugins and Applications" section.This Data Provider also supports SpotBugs (successor to FindBugs), with the same parameters. If you are using SpotBugs, its binary also has to be accessible, in [INSTALLDIR]/addons/tools/Findbugs/.

For more details, refer to http://findbugs.sourceforge.net/.

This data provider requires an extra download to extract the FindBugs or SpotBugs binary in *<SQUORE_HOME>/addons/tools/Findbugs_auto/*. For more information, refer to the Installation and Administration Guide's 'Third-Party Plugins and Applications' section.

Usage

FindBugs-SpotBugs (plugin) has the following options:

- **Classes (`class_dir`, mandatory)**: Specify the folders and/or jar files for your project in classpath format, or point to a text file that contains one folder or jar file per line.
- **Auxiliary Class path (`auxiliarypath`)**: Specify a list of folders and/or jars in classpath format, or specify the path to a text file that contains one folder or jar per line. This information will be passed to FindBugs or SpotBugs via the -auxclasspath parameter.
- **Memory Allocation (`xmx`, default: 1024m)**: Maximum amount of memory allocated to the java process launching FindBugs or SpotBugs.

The full command line syntax for FindBugs-SpotBugs (plugin) is:

```
-d
"type=Findbugs_auto,class_dir=[file_or_directory],auxiliarypath=[file_or_directory],xm
x=[text]"
```

Function Relaxer

Description

Function Relaxer provides a generic import mechanism for relaxing functions in source code.

Usage

Function Relaxer has the following options:

- **CSV File (`csv`)**:

The full command line syntax for Function Relaxer is:

```
-d "type=Function_Relaxer,csv=[file]"
```

FxCop

Description

FxCop is an application that analyzes managed code assemblies (code that targets the .NET Framework common language runtime) and reports information about the assemblies, such as possible design, localization, performance, and security improvements. FxCop generates an XML results file which can be imported to generate findings.

For more details, refer to https://msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx.

Usage

FxCop has the following options:

- **XML results file (`xml`)**: Specify the XML file containing FxCop's analysis results. Note that the minimum supported version of FxCop is 1.35.

The full command line syntax for FxCop is:

```
-d "type=FxCop,xml=[file]"
```

GCov

Description

GCov is a Code coverage program for C application. GCov generates raw text files which can be imported to generate metrics.

For more details, refer to http://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

Usage

GCov has the following options:

- **Directory containing results files (`dir`)**: Specify the path of the root directory containing the GCov results files.
- **Results files extension (`ext`, default: *.c.gcov)**: Specify the file extension of GCov results files.

The full command line syntax for GCov is:

```
-d "type=GCov,dir=[directory],ext=[text]"
```

GNATcheck

Description

GNATcheck is an extensible rule-based tool that allows developers to completely define a coding standard. The results are output to a log file or an XML file that can be imported to generate findings.

For more details, refer to http://www.adacore.com/gnatpro/toolsuite/gnatcheck/.

Usage

GNATcheck has the following options:

- **Log or XML file (`txt`)**: Specify the path to the log file or the XML file generated by the GNATcheck run.

The full command line syntax for GNATcheck is:

```
-d "type=GnatCheck,txt=[file]"
```

GNATCompiler

Description

GNATCompiler is a free-software compiler for the Ada programming language which forms part of the GNU Compiler Collection. It supports all versions of the language, i.e. Ada 2012, Ada 2005, Ada 95 and Ada 83. It creates a log file that can be imported to generate findings.

For more details, refer to http://www.adacore.com/gnatpro/toolsuite/compilation/.

Usage

GNATCompiler has the following options:

- **Log file (`log`)**: Specify the path to the log file containing the compiler warnings.

The full command line syntax for GNATCompiler is:

```
-d "type=GnatCompiler,log=[file]"
```

JSHint

Description

JSHint is an open source tool that verifies that JavaScript applications adhere to certain coding standards. It produces an XML file which can be imported to generate findings.

For more details, refer to http://jshint.com/.

Usage

JSHint has the following options:

- **JSHint results file (Checkstyle formatted): (`xml`)**: Point to the XML file that contains JSHint results Checkstyle formatted.

The full command line syntax for JSHint is:

```
-d "type=JSHint,xml=[file]"
```

JUnit Format

Description

JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks. JUnit XML result files are imported as test artefacts and links to tested classes are generated in the project.

For more details, refer to http://junit.org/.

Usage

JUnit Format has the following options:

- **Results folder (`resultDir`, mandatory)**: Specify the path to the folder containing the JUnit results (or by a tool able to produce data in this format). The data provider will parse subfolders recursively. Note that the minimum support version of JUnit is 4.10.
- **File Pattern (`filePattern`, mandatory, default: TEST-**.xml)*: Specify the pattern for files to read reports from.
- **Root Artefact (`root`, mandatory, default: tests[type=TEST_FOLDER]/junit[type=TEST_FOLDER])**: Specify the name and type of the artefact under which the test artefacts will be created.

The full command line syntax for JUnit Format is:

```
-d "type=JUnit,resultDir=[directory],filePattern=[text],root=[text]"
```

JaCoCo

Description

JaCoCo is a free code coverage library for Java. Its XML report file can be imported to generate code coverage metrics for your Java project.

For more details, refer to http://www.eclemma.org/jacoco/.

Usage

JaCoCo has the following options:

- **XML report (`xml`, mandatory)**: Specify the path to the XML report generated by JaCoCo. Note that the folder containing the XML file must also contain JaCoCo's report DTD file, available from http://www.eclemma.org/jacoco/trunk/coverage/report.dtd. XML report files are supported from version 0.6.5.

The full command line syntax for JaCoCo is:

```
-d "type=Jacoco,xml=[file]"
```

Klocwork

Description

Klocwork is a static analysis tool. Its XML result file can be imported to generate findings.

For more details, refer to http://www.klocwork.com.

Usage

Klocwork has the following options:

- **XML results file (`xml`)**: Specify the path to the XML results file exported from Klocwork. Note that Klocwork version 9.6.1 is the minimum required version.

The full command line syntax for Klocwork is:

```
-d "type=Klocwork,xml=[file]"
```

Klocwork MISRA

Description

Klocwork is a static analysis tool. Its XML result file can be imported to generate findings.

For more details, refer to http://www.klocwork.com.

Usage

Klocwork MISRA has the following options:

- **XML results file (`xml`)**: Specify the path to the XML results file exported from Klocwork. Note that Klocwork version 9.6.1 is the minimum required version.

The full command line syntax for Klocwork MISRA is:

```
-d "type=Klocwork_misra,xml=[file]"
```

Rational Logiscope

Description

The Logiscope suite allows the evaluation of source code quality in order to reduce maintenance cost, error correction or test effort. It can be applied to verify C, C++, Java and Ada languages and produces a CSV results file that can be imported to generate findings.

For more details, refer to http://www.kalimetrix.com/en/logiscope.

Usage

Rational Logiscope has the following options:

- **RuleChecker results file (`csv`)**: Specify the path to the CSV results file from Logiscope.

The full command line syntax for Rational Logiscope is:

```
-d "type=Logiscope,csv=[file]"
```

MSTest

Description

MS-Test automates the process of testing Windows applications. It combines a Windows development language, Basic, with a testing-oriented API.

For more details, refer to https://en.wikipedia.org/wiki/Visual_Test.

Usage

MSTest has the following options:

- **MSTest results directory (`resultDir`)**: Specify the path to the results directory generated by MSTest.
- **Test result file pattern (`filePattern`)**: Specify the pattern of files to extract Test data from.

The full command line syntax for MSTest is:

```
-d "type=MSTest,resultDir=[directory],filePattern=[text]"
```

MSTest Code Coverage

Description

MSTest is a code coverage library for C#. Its XML report file can be imported to generate code coverage metrics for your C# project.

Usage

MSTest Code Coverage has the following options:

- **XML report (`xml`)**: Specify the path to the XML report generated by MSTest Visual Studio 2017.

The full command line syntax for MSTest Code Coverage is:

```
-d "type=MSTest_Coverage,xml=[file]"
```

MemUsage

Description

Usage

MemUsage has the following options:

- **Memory Usage excel file (`excel`)**:

The full command line syntax for MemUsage is:

```
-d "type=MemUsage,excel=[file]"
```

NCover

Description

NCover is a Code coverage program for C# application. NCover generates an XML results file which can be imported to generate metrics.

For more details, refer to http://www.ncover.com/.

Usage

NCover has the following options:

- **XML results file (`xml`)**: Specify the location of the XML results file generated by NCover. Note that the minimum supported version is NCover 3.0.

The full command line syntax for NCover is:

```
-d "type=NCover,xml=[file]"
```

Oracle PLSQL compiler Warning checker

Description

This data provider reads an Oracle compiler log file and imports the warnings as findings. Findings extracted from the log file are filtered using a prefix parameter.

For more details, refer to http://www.oracle.com/.

Usage

Oracle PLSQL compiler Warning checker has the following options:

- **Compiler log file (`log`)**:
- **Prefixes (`prefix`)**: Prefixes and their replacements are specified as pairs using the syntax [prefix1|node1;prefix2|node2]. Leave this field empty to disable filtering.

  The parsing algorithm looks for lines fitting this pattern:

  [PATH;SCHEMA;ARTE_ID;ARTE_TYPE;LINE;COL;SEVERITY_TYPE;WARNING_ID;SEVERITY_ID; DESCR] and keeps lines where [PATH] begins with one of the input prefixes. In each kept [PATH], [prefix] is replaced by [node]. If [node] is empty, [prefix] is removed from [PATH], but not replaced. Some valid syntaxes for prefix:

  One prefix to remove: svn://aaaa:12345/valid/path/from/svn

  One prefix to replace: svn://aaaa:12345/valid/path/from/svn|node1

  Two prefixes to remove: svn://aaaa:12345/valid/path/from/svn|;svn://bbbb:12345/valid/path/from/other_svn|

  Two prefixes to remove: svn://aaaa:12345/valid/path/from/svn;svn://bbbb:12345/valid/path/from/other_svn

  Two prefixes to replace: svn://aaaa:12345/valid/path/from/svn|node1;svn://bbbb:12345/valid/path/from/other_svn|node2

The full command line syntax for Oracle PLSQL compiler Warning checker is:

```
-d "type=Oracle_PLSQLCompiler,log=[file],prefix=[text]"
```

MISRA Rule Checking using PC-lint

Description

PC-lint is a static code analyser. The PC-lint data provider reads PC-lint log file(s) and imports MISRA violations as findings.

For more details, refer to http://www.gimpel.com/html/pcl.htm.

Usage

MISRA Rule Checking using PC-lint has the following options:

- **Log file or folder (`logDir`)**: Specify the path to the folder containing the PC-lint log files, or to a single log file.
- **Extensions to exclude (`excludedExtensions`, default: .h;.H)**: Specify the file extensions to exclude from the reported violations.

The full command line syntax for MISRA Rule Checking using PC-lint is:

```
-d "type=PC_Lint_MISRA,logDir=[file_or_directory],excludedExtensions=[text]"
```

PMD

Description

PMD scans Java source code and looks for potential problems like possible bugs, dead code, sub-optimal code, overcomplicated expressions, duplicate code... The XML results file it generates is read to create findings.

For more details, refer to http://pmd.sourceforge.net.

Usage

PMD has the following options:

- **XML results file (`xml`)**: Specify the path to the PMD XML results file. Note that the minimum supported version of PMD for this data provider is 4.2.5.

The full command line syntax for PMD is:

```
-d "type=PMD,xml=[file]"
```

PMD (plugin)

Description

PMD scans Java source code and looks for potential problems like possible bugs, dead code, sub-optimal code, overcomplicated expressions, duplicate code ... The XML results file it generates is read to create findings.

For more details, refer to http://pmd.sourceforge.net.

This data provider requires an extra download to extract the PMD binary in *<SQUORE_HOME>/addons/tools/PMD_auto/*. For more information, refer to the Installation and Administration Guide's 'Third-Party Plugins and Applications' section. You may also deploy your own version of PMD and make the Data Provider use it by editing *<SQUORE_HOME>/configuration/tools/PMD_auto/config.tcl*.

Usage

PMD (plugin) has the following options:

- **Ruleset file (`configFile`)**: Specify the path to the PMD XML ruleset you want to use for this analysis. If you do not specify a ruleset, the default one from INSTALLDIR/addons/tools/PMD_auto will be used.

The full command line syntax for PMD (plugin) is:

```
-d "type=PMD_auto,configFile=[file]"
```

Polyspace

Description

Polyspace is a static analysis tool which includes a MISRA checker. It produces an XML output which can be imported to generate findings. Polyspace Verifier detects RTE (RunTime Error) such as Division by zero, Illegal Deferencement Pointer, Out of bound array index... Such information is turned into statistical measures at function level. Number of Red (justified/non-justified), Number of Grey (justified/non-justified), Number of Orange (justified/non-justified), Number of Green.

For more details, refer to http://www.mathworks.com/products/polyspace/index.html.

Usage

Polyspace has the following options:

- **DocBook results file (`xml`)**: Specify the path to the DocBook (main XML file) generated by Polyspace.
- **Ignore source file path (`ignoreSourceFilePath`, default: false)**: Removes all path elements when doing the mapping between files in Squore project and files in the Pomyspace report. Becareful this can work only if file names in Squore project are unique.

The full command line syntax for Polyspace is:

```
-d "type=Polyspace,xml=[file],ignoreSourceFilePath=[booleanChoice]"
```

MISRA Rule Checking with QAC

Description

QAC identifies problems in C source code caused by language usage that is dangerous, overly complex, non-portable, difficult to maintain, or simply diverges from coding standards. Its CSV results file can be imported to generate findings.

For more details, refer to http://www.phaedsys.com/principals/programmingresearch/pr-qac.html.

Usage

MISRA Rule Checking with QAC has the following options:

- **Code Folder (`logDir`)**: Specify the path to the folder that contains the annotated files to process.

  For the findings to be successfully linked to their corresponding artefact, several requirements have to be met:

  - The annotated file name should be [Original source file name].txt

  e.g. The annotation of file "controller.c" should be called "controller.c.txt"

  - The annotated file location in the annotated directory should match the associated source file location in the source directory.

  e.g. The annotation for source file "[SOURCE_DIR]/subDir1/subDir2/controller.c" should be located in "[ANNOTATIONS_DIR]/subDir1/subDir2/controller.c.txt"

  The previous comment suggests that the source and annotated directory are different.

  However, these directories can of course be identical, which ensures that locations of source and annotated files are the same.

- **Extension (`ext`, default: html)**: Specify the extension used by QAC to create annotated files.

- **Force import of all QAC violations (not only MISRA) (`force_all_import`, default: false)**: Force the import of all QAC findings (not only the MISRA violations)

The full command line syntax for MISRA Rule Checking with QAC is:

```
-d "type=QAC_MISRA,logDir=[directory],ext=[text],force_all_import=[booleanChoice]"
```

Rational Test RealTime

Description

Rational Test RealTime is a cross-platform solution for component testing and runtime analysis of embedded software. This Data Provider extracts coverage results, as well as tests and their status

For more details, refer to http://www-01.ibm.com/software/awdtools/test/realtime/.

Usage

Rational Test RealTime has the following options:

- **.xrd folder (`logDir`)**: Specify the path to the folder containing the .xrd files generated by RTRT.
- **Excluded file extensions (`excludedExtensions`, default: .h;.H)**:
- **Do you want to include FE (Function and Exit) in MCDC computation? (`include_fe_in_mcdc`, default: false)**:
- **Generate Test artefacts and structure from .xrd files? (`generateTests`, default: false)**:

The full command line syntax for Rational Test RealTime is:

```
-d
"type=RTRT,logDir=[directory],excludedExtensions=[text],include_fe_in_mcdc=[booleanCho
ice],generateTests=[booleanChoice]"
```

ReqIF

Description

RIF/ReqIF (Requirements Interchange Format) is an XML file format that can be used to exchange requirements, along with its associated metadata, between software tools from different vendors.

For more details, refer to http://www.omg.org/spec/ReqIF/.

Usage

ReqIF has the following options:

- **Reqif Directory (`dir`)**: Specify the directory which contains the Reqif files
- **Spec Object Type (`objType`, default: *AUTO*)**: Specify the SPEC_OBJECT_TYPE property LONG-NAME to be used to process the ReqIf file. Using the _AUTO_ value will let the Data Provider extract the value fro the ReqIf file, and assumes that there is only one such definition.

The full command line syntax for ReqIF is:

```
-d "type=ReqIf,dir=[directory],objType=[text]"
```

SQL Code Guard

Description

SQL Code Guard is a free solution for SQL Server that provides fast and comprehensive static analysis for T-Sql code, shows code complexity and objects dependencies.

For more details, refer to http://www.sqlcodeguard.com.

Usage

SQL Code Guard has the following options:

- **XML results (`xml`)**: Specify the path to the XML file containing SQL Code Guard results.

The full command line syntax for SQL Code Guard is:

```
-d "type=SQLCodeGuard,xml=[file]"
```

Squan Sources

Description

Squan Sources provides basic-level analysis of your source code.

For more details, refer to https://www.vector.com/squore.

The analyser can output info and warning messages in the build logs. Recent additions to those logs include better handling of structures in C code, which will produce these messages:

- [Analyzer] Unknown syntax declaration for function XXXXX at line yyy to indicate that we whould have found a function but, probably due to preprocessing directives, we are not able to parse it.
- [Analyzer] Unbalanced () blocks found in the file. Probably due to preprocessing directives, parenthesis in the file are not well balanced.
- [Analyzer] Unbalanced {} blocks found in the file. Probably due to preprocessing directives, curly brackets in the file are not well balanced.

You can specify the languages for your source code by passing pairs of language and extensions to the `languages` parameter. Extensions are case-sensitive and cannot be used for two different languages. For example, a project mixing php and javascript files can be analysed with:

```
--dp "type=SQuORE,languages=php:.php;javascript:.js,.JS"
```

In order to launch an analysis using all the available languages by default, do not specify the `languages` parameter in your command line.

Usage

Squan Sources has the following options:

- **Languages (`languages`, default: ada;c;cpp;csharp;cobol;java;fortran77;fortran90;php;python;swift;vbnet)**: Check the boxes for the languages used in the specified source repositories. Adjust the list of file extensions as necessary. Note that two languages cannot use the same file extension, and that the list of extensions is case-sensitive. Tip: Leave all the boxes unchecked and Squan Sources will auto-detect the language parser to use.

- **Force full analysis (`rebuild_all`, default: false)**: Analyses are incremental by default. Check this box if you want to force the source code parser to analyse all files instead of only the ones

that have changed since the previous analysis. This is useful if you added new rule files or text parsing rules and you want to re-evaluate all files based on your modifications.

- **Generate control graphs (`genCG`, default: true)**: This option allows generating a control graph for every function in your code. The control graph is visible in the dashboard of the function when the analysis completes.

- **Use qualified names (`qualified`, default: false)**: Note: This option cannot be modified in subsequent runs after you create the first version of your project.

- **Limit analysis depth (`depth`, default: false)**: Use this option to limit the depth of the analysis to file-level only. This means that Squan Sources will not create any class or function artefacts for your project.

- **Add a 'Source Code' node (`scnode`, default: false)**: Using this options groups all source nodes under a common source code node instead of directly under the APPLICATION node. This is useful if other data providers group non-code artefacts like tests or requirements together under their own top-level node. This option can only be set when you create a new project and cannot be modified when creating a new version of your project.

- **'Source Code' node label (`scnode_name`, default: Source Code)**: Specify a custom label for your main source code node. Note: this option is not modifiable. It only applies to projects where you use the "Add a 'Source Code' node" option. When left blank, it defaults to "Source Code".

- **Compact folders (`compact_folder`, default: true)**: When using this option, folders with only one son are aggregates together. This avoids creating many unnecessary levels in the artefact tree to get to the first level of files in your project. This option cannot be changed after you have created the first version of your project.

- **Content exclusion via regexp (`pattern`)**: Specify a PERL regular expression to automatically exclude files from the analysis if their contents match the regular expression. Leave this field empty to disable content-based file exclusion.

- **File Filtering (`files_choice`, default: Exclude)**: Specify a pattern and an action to take for matching file names. Leave the pattern empty to disable file filtering.

- **pattern (`pattern_files`)**: Use a shell-like wildcard e.g. '*-test.c'.

  - * Matches any sequence of characters in string, including a null string.
  - ? Matches any single character in string.
  - [chars] Matches any character in the set given by chars. If a sequence of the form x-y appears in chars, then any character between x and y, inclusive, will match. On Windows, this is used with the -nocase option, meaning that the end points of the range are converted to lower case first. Whereas [A-z] matches '_' when matching case-sensitively ('_' falls between the 'Z' and 'a'), with -nocase this is considered like [A-Za-z].
  - \x Matches the single character x. This provides a way of avoiding the special interpretation of the characters *?[] in pattern.

  Tip: Use ';' to separate multiple patterns.

  How to specify a file:

  - By providing its name, containing or not a pattern
  - By providing its name and its path, both containing or not a pattern

  e.g.

  - *D??l?g.* : will match MyDialog.java, WinDowlog.c, ... anywhere in the project
  - */[Dd]ialog/*D??l?g.* : will match src/java/Dialog/MyDialog.java, src/c/dialog/WinDowlog.c, but not src/Dlg/c/WinDowlog.c

- **Folder Filtering (`dir_choice`, default: Exclude)**: Specify a pattern and an action to take for matching folder names. Leave the pattern empty to disable folder filtering.
- **pattern (`pattern_dir`)**: Use a shell-like wildcard e.g. 'Test_*'.
  - * Matches any sequence of characters in string, including a null string.
  - ? Matches any single character in string.
  - [chars] Matches any character in the set given by chars. If a sequence of the form x-y appears in chars, then any character between x and y, inclusive, will match. On Windows, this is used with the -nocase option, meaning that the end points of the range are converted to lower case first. Whereas [A-z] matches '_' when matching case-sensitively ('_' falls between the 'Z' and 'a'), with -nocase this is considered like [A-Za-z].
  - \x Matches the single character x. This provides a way of avoiding the special interpretation of the characters *?[] in pattern.

  Tip: Use ';' to separate multiple patterns.

  A directory can be specified:

  - By providing its name, containing or not a pattern
  - By providing its name and its path, both containing or not a pattern. In that case the full path has to match.

  e.g.

  - source? : will match directories source, sources, ... anywhere in the project
  - src/tests : will not match any directory because the full path can not match
  - */src/tests : will match java/src/tests, native/c/src/tests, ...

  To get the root path of the project it is possible to use the nodes variables ($src, $Node1, ...). Refers to "Using Data Provider Input Files From Version Control" in the Getting Started to learn more.

  e.g. $src/source/tests will match only the directory source/tests if it is a root directory of the project.
- **Exclude files whose size exceeds (`size_limit`, default: 500000)**: Provide the size in bytes above which files are excluded automatically from the Squore project (Big files are usually generated files or test files). Leave this field empty to deactivate this option.
- **Detect algorithmic cloning (`clAlg`, default: true)**: When checking this box, Squan Sources launches a cloning detection tool capable of finding algorithmic cloning in your code.
- **Detect text cloning (`clTxt`, default: true)**: When checking this box, Squan Sources launches a cloning detection tool capable of finding text duplication in your code.
- **Ignore blank lines (`clIgnBlk`, default: true)**: When checking this box, blanks lines are ignored when searching for text duplication
- **Ignore comment blocks (`clIgnCmt`, default: true)**: When checking this box, blocks of comments are ignored when searching for text duplication
- **Minimum size of duplicated blocks (`clRSlen`, default: 10)**: This threshold defines the minimum size (number of lines) of blocks that can be reported as cloned.
- **Textual Cloning fault ratio (`clFR`, default: 0.1)**: This threshold defines how much cloning between two artefacts is necessary for them to be considered as clones by the text duplication tool. For example, a fault ratio of 0.1 means that two artefacts are considered clones if less than 10% of their contents differ.

- **Algorithmic cloning fault ratio (`clAlgFR`, default: 0.1)**: This threshold defines how much cloning between two artefacts is necessary for them to be considered as clones by the algorithmic cloning detection tool.

- **Compute Textual stability (`genTs`, default: true)**: This option allows keeping track of the stability of the code analysed for each version. The computed stability is available on the dashboard as a metric called and can be interpreted as 0% meaning completely changed and 100% meaning not changed at all.

- **Compute Algorithmic stability (`genAs`, default: true)**: This option allows keeping track of the stability of the code analysed for each version. The computed stability is available on the dashboard as a metric called Stability Index (SI) and can be interpreted as 0% meaning completely changed and 100% meaning not changed at all.

- **Detect artefact renaming (`clRen`, default: true)**: This option allows Squan Sources to detect artefacts that have been moved since the previous version, ensuring that the stability metrics of the previous artefact are passed to the new one. This is typically useful if you have moved a file to a different folder in your source tree and do not want to lose the previous metrics generated for this file. If you do not use this option, moved artefacts will be considered as new artefacts.

- **Mark relaxed or confirmed findings as suspicious (`susp`, default: MODIFIED_BEFORE)**: Depending on the choosen option, relaxed findings can be flagged as suspicious in case of changes in and around the finding. In all cases, the following is to be considered:

  - Only changes on effective code are considered, comments are ignored.
  - Only changes inside the scope of the artefact containing the finding are considered.

- **Accept Relaxation from source code comment (`relax`, default: true)**: **Relaxing Violations in Code**

  Squore interprets comments formatted in one of these three ways:

  - Inline Relaxation

  This syntax is used to relax violations on the current line.

  some code; /* **%RELAX<keys>** : Text to justify the relaxation */

  - Relax Next Line

  This syntax is used to relax a violation on the first following line that is not a comment. In the example the text of the justification will be: "Text to justify the relaxation the text of the justification continues while lines are made of comments only"

  /* **>RELAX<keys>** : Text to justify the relaxation */

  /* the text of the justification continues while */

  /* lines are made of comments only */

  some code;

  - Block Relaxation

  This syntax is used to relax violations in an entire block of code.

  /* **{{ RELAX<keys>** : Text to justify the relaxation */

/* like for format 2 text can be on more than one line */

int my_func() {

  /* contains many violations */

  ...

}

/* **}} RELAX<keys>** */

<keys> can be one of the following:

- <*>: relax all violations
- <MNEMO>: relax violations of the rule MNEMO
- <MNEMO1,MNEMO2,...,MNEMOn>: relax violations of rules MNEMO1 and MNEMO2 ... and MNEMOn

It is possible to relax using a status different from derogation. In that case the keyword RELAX has to be followed by :RELAXATION_STATUS

e.g. RELAX:APPROVED if the status RELAXED_APPOVED is defined in the model.

- **Additional parameters (`additional_param`)**: These additional parameters can be used to pass instructions to external processes started by this data provider. This value is generally left empty in most cases.

The full command line syntax for Squan Sources is:

```
-d
"type=SQuORE,languages=[multipleChoice],rebuild_all=[booleanChoice],genCG=[booleanChoi
ce],qualified=[booleanChoice],depth=[booleanChoice],scnode=[booleanChoice],scnode_name
=[text],compact_folder=[booleanChoice],pattern=[text],files_choice=[multipleChoice],pa
ttern_files=[text],dir_choice=[multipleChoice],pattern_dir=[text],size_limit=[text],cl
Alg=[booleanChoice],clTxt=[booleanChoice],clIgnBlk=[booleanChoice],clIgnCmt=[booleanCh
oice],clRSlen=[text],clFR=[text],clAlgFR=[text],genTs=[booleanChoice],genAs=[booleanCh
oice],clRen=[booleanChoice],susp=[multipleChoice],relax=[booleanChoice],additional_par
am=[text]"
```

Squore Import

Description

Squore Import is a data provider used to import the results of another data provider analysis. It is generally only used for debugging purposes.

For more details, refer to support@vector.com.

Usage

Squore Import has the following options:

- **XML folder (`inputDir`)**: Specify the folder that contains the squore_data_*.xml files that you want to import.

The full command line syntax for Squore Import is:

```
-d "type=SQuOREImport,inputDir=[directory]"
```

Squore Virtual Project

Description

Squore Virtual Project is a data provider that can use the output of several projects to compile metrics in a meta-project composed of the import sub-projects.

For more details, refer to support@vector.com.

Usage

Squore Virtual Project has the following options:

- **Paths to output.xml files (`output`)**: Specify the paths to all the output.xml files you want to include in the virtual project. Separate paths using ';'.

The full command line syntax for Squore Virtual Project is:

```
-d "type=SQuOREVirtualProject,output=[file]"
```

StyleCop

Description

StyleCop is a C# code analysis tool. Its XML output is imported to generate findings.

For more details, refer to https://stylecop.codeplex.com/.

Usage

StyleCop has the following options:

- **XML results file (`xml`)**: Specify the path to the StyleCop XML results file. The minimum version compatible with this data provider is 4.7.

The full command line syntax for StyleCop is:

```
-d "type=StyleCop,xml=[file]"
```

StyleCop (plugin)

Description

StyleCop is a C# code analysis tool. Its XML output is imported to generate findings.

For more details, refer to https://stylecop.codeplex.com/.

Note that this data provider is not supported on Linux. On windows, this data provider requires an extra download to extract the StyleCop binary in *<SQUORE_HOME>/addons/tools/StyleCop_auto/* and .NET framework 3.5 to be installed on your machine (run *Net.SF.StyleCopCmd.Console.exe* manually once to install .NET automatically). For more information, refer to the Installation and Administration Guide's 'Third-Party Plugins and Applications' section.

Usage

StyleCop (plugin) has the following options:

- **Solution (`sln`)**: Specify the path to the .sln file to analyse. Leave empty to analyse all .sln found in the source repository.

The full command line syntax for StyleCop (plugin) is:

```
-d "type=StyleCop_auto,sln=[file]"
```

Tessy

Description

Tessy is a tool automating module/unit testing of embedded software written in dialects of C/C++. Tessy generates an XML results file which can be imported to generate metrics. This data provider supports importing files that have a xml_version="1.0" attribute in their header.

For more details, refer to https://www.hitex.com/en/tools/tessy/.

Usage

Tessy has the following options:

- **Results folder (`resultDir`)**: Specify the top folder containing XML result files from Tessy. Note that this data provider will recursively scan sub-folders looking for index.xml files to aggregate results.

The full command line syntax for Tessy is:

```
-d "type=Tessy,resultDir=[directory]"
```

VectorCAST

Description

The VectorCAST Data Provider extracts coverage results, as well as tests and their status

For more details, refer to https://www.vectorcast.com/.

Usage

VectorCAST has the following options:

- **HTML Report (`html_report`)**: Specify the path to the HTML report which contains the test results.
- **Source code file extension (`file_extension`, default: .c)**: Source code file extension
- **Create test artefacts from HTML report (`generateTests`, default: false)**:
- **Sub Folder for test results (`sub_root`)**: Sub Folder for test results.

The full command line syntax for VectorCAST is:

```
-d
"type=VectorCAST,html_report=[file_or_directory],file_extension=[text],generateTests=[
booleanChoice],sub_root=[text]"
```

VectorCAST API

Description

The VectorCAST Data Provider extracts coverage results, as well as tests and their status

For more details, refer to https://www.vectorcast.com/.

Usage

VectorCAST API has the following options:

- **Retrieve the coverage data via vectorCAST API? (`generate_report`, mandatory)**: Squore imports vectorCAST data via ".sqc" files.

  The sqc files are extracted from vectorCAST API.

  If vectorCAST is installed on the Squore server, you can select "Yes" to ask Squore to generate the sqc files.

  In that case, make sure the Squore server can access to the vectorCAST results directory.

If vectorCAST is not available on the Squore server, thus, you have to select "No" to import the test and coverage data via sqc files.

- **VectorCAST project configuration files (Path to vcm, vce or vcp) (`project_file_list`)**: Specify the path to your project configuration file.

  The path should be either:

  1) Path to your project ".vcm" file

  2) Path to the directory which contains all the vce or vcp (Squore will look for all reccursive folders)

- **Folder of vectorCAST data files (**.sqc) (`squore_report_folder`)*: Specify the folder which contains all the vectorCAST data files (*.sqc).

  The .sqc files are generated from vectorCAST API for squore.

- **Root Artefact (`sub_root`, default: Tests)**: Specify the name of the artefact under which the test artefacts will be created.

- **Don't Be "case sensitve" (`case_sensitve_option`, default: true)**: Don't Be "case sensitve"

- **Generate a testcase unique id (`create_path_unique_id`, default: false)**: Generate a testcase unique id based on "path + test name"

  This option is needed if you want to link test objects with external requirements.

The full command line syntax for VectorCAST API is:

```
-d
"type=VectorCAST_API,generate_report=[multipleChoice],project_file_list=[file_or_direc
tory],squore_report_folder=[directory],sub_root=[text],case_sensitve_option=[booleanCh
oice],create_path_unique_id=[booleanChoice]"
```

Vector Trace Items

Description

Import Trace Items in Vector generic format as Requirements in Squore

For more details, refer to https://www.vector.com/int/en/products/products-a-z/software/vTESTstudio/.

Usage

Vector Trace Items has the following options:

- **Trace Items folder (`dir`)**: Specify the folder containing Trace Items (Requirements) files

- **Trace Items file suffix (`suff`, default: .vti-tso)**: Provide the suffix of Trace Items (Requirements) files.

- **Default status (`default_status`, default: VERIFIED)**: It is possible to specify the defauls status for Requirements imported in Squore.

- **Planned Trace Items folder (`dirPlanned`)**: Specify the folder containing Planned Trace Items files.
- **Filter on Requirements (`filter`)**: The filter is a way to keep Requirements which properties match a certain pattern.

  Syntax:<PROPERTY_NAME>?regex=<REGEX>

  Examples:

  - ***No filters are provided***... If no filters are provided, all Requirements from vTESTstudio are shown in Squore (default behavior)
  - **Property 1?regex=V_.***... Only keep Requirements where 'Property 1' starts with 'V_'
  - **Property 1?regex=V_.*;Property 2?regex=.*VALID.***... Only keep Requirements where 'Property 1' starts with 'V_', **AND** 'Property 2' contains 'VALID'
- **Requirements grouping (`grouping`)**: Grouping is a way to structure Requirements in Squore by the value of given properties, in the order they are provided.

  Examples:Suppose Requirements have:

  - an 'Origin' property ('Internal', 'External')
  - and a 'Criticity' property ('A', 'B', 'C, 'D')

  Possible values for grouping:

  - **grouping is empty** ... If no grouping is provided, Requirement will be shown in Squore with the same structure as in vTESTstudio (default behavior)
  - **grouping ='Origin'** ... In addition to the original structure, Requirements will be broken down by origin ('Internal, 'External', or 'Unknown' if the 'Origin' property is absent or empty)
  - **grouping ='Origin;Criticity'** ... Same as before, but the Requirements will be broken down by Origin, **THEN** by Criticity ('A', 'B', 'C, 'D', or 'Unknown' if the 'Criticity' property is absent or empty)

The full command line syntax for Vector Trace Items is:

```
-d
"type=Vector_TraceItems,dir=[directory],suff=[text],default_status=[multipleChoice],di
rPlanned=[directory],filter=[text],grouping=[text]"
```

Axivion

Description

Import Findings from Axivion

For more details, refer to http://www.axivion.com.

Usage

Axivion has the following options:

- **CSV File (`csv`)**: Specify the CSV file which contains the findings results (MISRA, Coding Style...)

The full command line syntax for Axivion is:

```
-d "type=bauhaus,csv=[file]"
```

CodeSniffer

Description

CodeSniffer is a rulecker for PHP and Javascript

For more details, refer to http://www.squizlabs.com/php-codesniffer.

Usage

CodeSniffer has the following options:

- **CodeSniffer results file (checkstyle formatted XML) (`xml`)**: Point to the XML file that contains CodeSniffer results.

The full command line syntax for CodeSniffer is:

```
-d "type=codesniffer,xml=[file]"
```

Configuration Checker

Description

Use this tool to check for duplicated files or XML Elements between a custom configuration and the standard configuration.

Usage

Configuration Checker has the following options:

- **Standard Configuration Path (`s`)**:
- **Custom Configurations Path (`p`)**:

The full command line syntax for Configuration Checker is:

```
-d "type=conf-checker,s=[directory],p=[directory]"
```

CSV Coverage Import

Description

CSV Coverage Import provides a generic import mechanism for coverage results at function level

Usage

CSV Coverage Import has the following options:

- **CSV file (`csv`)**: Enter the path to the CSV containing the coverage data.

  The expected format of each line contained in the file is **PATH;NAME;TESTED_C1;OBJECT_C1;TESTED_MCC;OBJECT_MCC;TESTED_MCDC;OBJECT_MCDC;TCOV_MCC;TCOV_MCDC;TCOV_C1**

The full command line syntax for CSV Coverage Import is:

```
-d "type=csv_coverage,csv=[file]"
```

CSV Findings

Description

CSV Findings is a generic tool that allows importing findings into the project.

Usage

CSV Findings has the following options:

- **CSV File(s) (`csv`)**: Specify the path(s) to your CSV file(s) containing findings. To provide multiple files click on '+'. Each line in the file must use the following format and the file should include the following header:

  FILE;FUNCTION;RULE_ID;MESSAGE;LINE;COL;STATUS;STATUS_MESSAGE;TOOL

The full command line syntax for CSV Findings is:

```
-d "type=csv_findings,csv=[file]"
```

CSV Import

Description

Imports artefacts, metrics, findings, textual information and links from one or more CSV files. The expected CSV format for each of the input files is described in the user manuals in the csv_import framework reference.

Consult csv_import Reference for more details about the expected CSV format.

Usage

CSV Import has the following options:

- **CSV Separator (`separator`, default: ;)**: Specify the CSV Separator used in the CSV file.
- **CSV Delimiter (`delimiter`, default: ")**: CSV Delimiter is used when the separator is used inside a cell value. If a delimiter is used as a char in a cell it has to be doubled.

  The ' char is not allowed as a delimiter.
- **Artefact Path Separator (`pathSeparator`, default: /)**: Specify the character used as a separator in an artefact's path in the input CSV file.
- **Case-sensitive artefact lookup (`pathAreCaseSensitive`, default: true)**: When this option is turned on, artefacts in the CSV file are matched with existing source code artefacts in a case-sensitive manner.
- **Ignore source file path (`ignoreSourceFilePath`, default: false)**: When ignoring source file path it is your responsbility to ensure that file names are unique in the project.
- **Create missing files (`createMissingFile`, default: false)**: Automatically creates the artefacts declared in the CSV file if they do not exist.
- **Ignore finding if artefact not found (`ignoreIfArtefactNotFound`, default: true)**: If a finding can not be attached to any artefact then it is either ignored (checked) or it is attached to the project node instead (unchecked).
- **Unknown rule ID (`unknownRuleId`)**: For findings of a type that is not in your ruleset, set a default rule ID. The value for this parameter must be a valid rule ID from your analysis model.
- **Measure ID for orphan artifacts count (`orphanArteCountId`)**: To save the total count of orphan findings as a metric at application level, specify the ID of the measure to use in your analysis model.
- **Measure ID for unknown rules count (`orphanRulesCountId`)**: To save the total count of unknown rules as a metric at application level, Specify the ID of the measure to use in your analysis model.
- **Information ID receiving the list of unknown rules IDs (`orphanRulesListId`)**: To save the list of unknown rule IDs as textual information at application level, specify the ID of the textual information to use in your analysis model.
- **CSV File (`csv`)**: Specify the path to the input CSV file containing artefacts, metrics, findings, textual information, links and keys.
- **Metrics CSV File (`metrics`)**: Specify the path to the CSV file containing metrics.
- **Infos CSV File (`infos`)**: Specify the path to the CSV file containing textual information.
- **Findings CSV File (`findings`)**: Specify the path to the CSV file containing findings.
- **Keys CSV File (`keys`)**: Specify the path to the CSV file containing artefact keys.
- **Links CSV File (`links`)**: Specify the path to the CSV file containing links.
- **Reports artefacts mapping problem as (`level`, default: info)**: When an artefact referenced in the CSV file can not be found in the project, reports the problem as an information or as a warning.

The full command line syntax for CSV Import is:

```
-d
"type=csv_import,separator=[text],delimiter=[text],pathSeparator=[text],pathAreCaseSen
sitive=[booleanChoice],ignoreSourceFilePath=[booleanChoice],createMissingFile=[boolean
Choice],ignoreIfArtefactNotFound=[booleanChoice],unknownRuleId=[text],orphanArteCountI
d=[text],orphanRulesCountId=[text],orphanRulesListId=[text],csv=[file],metrics=[file],
infos=[file],findings=[file],keys=[file],links=[file],level=[multipleChoice]"
```

CSV Tag Import

Description

This data provider allows setting values for attributes in the project.

Usage

CSV Tag Import has the following options:

- **CSV file (`csv`)**: Specify the path to the file containing the metrics.

The full command line syntax for CSV Tag Import is:

```
-d "type=csv_tag_import,csv=[file]"
```

Generic Findings XML Import

Description

Generic Findings XML Import

Usage

Generic Findings XML Import has the following options:

- **XML File (`xml`)**: Specify the XML file which contains the findings results (MISRA, Coding Style...)
- **"Issue" mapping definition (`issue`)**:
- **"Rule Id" mapping definition (`id_rule`)**:
- **"Message" mapping definition (`message`)**:
- **"File" mapping definition (`file`)**:
- **"Line" mapping definition (`line`)**:
- **"Justification" mapping definition (`justification`)**:

The full command line syntax for Generic Findings XML Import is:

```
-d
"type=findings_xml,xml=[file],issue=[text],id_rule=[text],message=[text],file=[text],l
ine=[text],justification=[text]"
```

GNAThub

Description

Import data from GNAThub. GNAThub integrates and aggregates the results of AdaCore's various static and dynamic analysis tools (GNATmetric, GNATcheck, GNATcoverage, CodePeer). Compatible with GNAT Pro versions 7.4.2 up to 18.2.

For more details, refer to https://www.adacore.com/gnatpro/toolsuite/gnatdashboard.

This Data Provider will only be available after you configure your server or client *config.xml* with the path to your gnathub executable with a *<path name="gnatub" path="C:\tools\GNAThub\gnathub.exe" />* definition. Consult the Configuration Manual for more information about referencing external executables.

Usage

GNAThub has the following options:

- **Path of the gnathub.db file (`gnatdb`)**: Specify the absolute path of the gnathub.db file.

The full command line syntax for GNAThub is:

```
-d "type=gnathub,gnatdb=[file]"
```

CPU Data Import

Description

CPU Data Import provides a generic import mechanism for CPU data from a CSV or Excel file.

Usage

CPU Data Import has the following options:

- **Root node name (`root_node`, default: Resources)**: Specify the name of root node in the artefact tree.
- **Data File (`xls_file`)**: Specify the path to the file containing CPU information.
- **Sheet Name (`xls_sheetname`)**: Specify the name of the Excel sheet that contains the CPU list.

- **CPU Column name (`xls_key`)**: Specify the header name of the column which contains the CPU key.
- **Grouping Structure (`xls_groups`)**: Specify the headers for Grouping Structure, separated by ";".
- **Filtering (`xls_filters`)**: Specify the list of Header for filtering

  For example: **"column_name_1=regex1;column_name_2=regex2;**
- **Specify the CSV separator (`csv_separator`, default: ;)**: Specify the CSV separator
- **"CPU Loop Time" Column name (`cpu_loop_column_name`, default: Total Loop Time [ms])**: Specify the column name of the CPU Loop Time (Ex: "Total Loop Time [ms]")
- **"Average Idle Time per loop" Column name (`cpu_idle_column_name`, default: Average idle Time per loop [ms])**: Specify the column name of the Average Idle Time per loop (Ex: "Average idle Time per loop [ms]")
- **"Worst Case Idle Time per loop" Column name (`cpu_worst_column_name`, default: Worse case idle Time per loop [ms])**: Specify the column name of the Worst Case Idle Time per loop (Ex: "Worse case idle Time per loop [ms]")
- **Create an output file (`createOutput`, default: true)**: Create an output file

The full command line syntax for CPU Data Import is:

```
-d
"type=import_cpu,root_node=[text],xls_file=[file],xls_sheetname=[text],xls_key=[text],
xls_groups=[text],xls_filters=[text],csv_separator=[text],cpu_loop_column_name=[text],
cpu_idle_column_name=[text],cpu_worst_column_name=[text],createOutput=[booleanChoice]"
```

Excel Import

Description

Excel Import

Usage

Excel Import has the following options:

- **Input file (`input_file`)**: Specify the Excel input file
- **Sheetname (`sheetname`)**: Sheetname to read data from
- **Artefact Type (`artefact_type`)**: Artefact Type used by Squore Analysis model.

  Example: TEST
- **Artefact Type container (`artefact_type_container`)**: Artefact Type container used by Squore Analysis model.

  Example: TEST_FOLDER
- **Artefact unique ID (`artefact_uid`)**: **Optional unless you want to use links to these artefacts.**

This is the artefact unique ID, to be used by links, from this Data Provider, or another Data Provider.Examples:

- ${ID}
- T_${Name}
- ${Name} ${Descr}

Note:${NAME} designates the column called NAME

- **Links to this artefact (`artefact_link`)**: Specify how to create links between this artefact and other artefacts with the following format:

  *<LINK_TYPE>?direction=<IN OR OUT>&column=<COLUMN_NAME>&separator=<SEPARATOR>*Examples:

  - **TESTED_BY?column=Test**

  A 'TESTED_BY' link will be created with the UID found in column 'Test'

  - **IMPLEMENTED_BY?direction=IN&column=Implements**

  An 'IMPLEMENTED_BY' link will be created with the UID found in column 'Implements'. Since the optional 'direction' attribute is provided, it will be set as 'IN' (default value is 'OUT')

  - **TESTED_BY?column=Tests&separator=','**

  'TESTED_BY' links will be created with all UIDs found in column 'Tests', separated by a comma

  - **TESTED_BY?column=Tests&separator=',';REFINED_BY?column=DownLinks&separator=','**

  'TESTED_BY' and 'REFINED_BY' links will be created with UIDs found in columns 'Tests' and 'DownLinks' respectively

- **Artefact name (`artefact_name`)**: Artefact name as displayed in Squore. Examples:

  - ${ID}
  - T_${Name}
  - ${Name} ${Descr}

  Note:${NAME} designates the column called NAME

- **Path to the artefact (`path_list`)**: **Optional. If not used, artefacts extracted from the Excel file will be directly added to the Squore root.**

  To specify the path in Squore of artefacts exracted from the Excel file, using the following format:

  *<COLUMN_NAME>?map=[<REGEX_1>:<GROUP_NAME_1>,... ,<REGEX_N>:<GROUP_NAME_N>]&groupByDate=<YES>&format=<dd-mm-YYYY>*Examples:

  - **Area**

  Artefacts will be regrouped by the value found in the 'Area' column

  - **Area?map=[A*:Area A,B*:Area B]**

  Artefacts will be regrouped into two groups:'Area A', for all values of 'Area' column starting with letter 'A', and 'Area B' for letter 'B'.

  - **Started on?groupByDate=Yes&format=YYYY/mm/dd**

Artefacts will be regrouped by the date found in column 'Started on', using the format 'YYYY/mm/dd'

Note:Date patterns are based on SimpleDateFormat Java class specifications.

- **Textual data to extract (`info_list`): Optional.**

  To specify the list of textual data to extract from the Excel file, using the following format:

  *<METRIC_ID>?column=<COLUMN_NAME>&map=[<REGEX_1>:<TEXT_1>,...
  ,<REGEX_N>:<TEXT_N>]*Examples:

  - **ZONE_ID?column=Zone**

  Textual data found in column 'Zone' will be associated to metric ZONE_ID

  - **ZONE_ID?column=Zone;OWNER?column=Belongs to**

  Textual data found in columns 'Zone' and 'Belongs to' will be associated to metric ZONE_ID and OWNER respectively

  - **ORIGIN?column=Comes from,map=[Cust*:External,Sub-contractor*:External,Support:Internal,Dev:Internal]**

  _Textual data found in column 'Comes from' will be associated to metric ORIGIN:

  - With value 'External' if the column starts with 'Cust' or 'Sub-contractor'
  - With value 'Internal' if the column equals 'Support' or 'Dev'

  _

  - **Started on?groupByDate=Yes&format=YYYY/mm/dd**

  Artefacts will be regrouped by the date found in column 'Started on', using the format 'YYYY/mm/dd'

- **Numerical metrics to extract (`metric_list`): Optional.**

  To specify the list of numerical data to extract from the Excel file, using the following format:

  *<METRIC_ID>?column=<COLUMN_NAME>&extract=<REGEX_EXRACT>&map=[<REGEX_1>:<
  VALUE_1>,...,<REGEX_N>:<VALUE_N>]*Examples:

  - **PRIORITY?column=Priority level**

  Numerical values found in column 'Priority level' will be associated to metric PRIORITY

  - **SEVERITY?column=Severity level,extract=S_**

  Numerical values found in column 'Severity level' will be associated to metric SEVERITY, after having extracted (removed) the string 'S_', because in this example, column 'Severity level' contains for example 'S_1', 'S_4', etc., and we want to obtain '1', '4', etc.

  - **STATUS?column=State&map=[passed:0,Passed:0,Pass:0,*nconclusive*:1,failed:2,Failed:2,FAIL:2]**

  _Textual values found in column 'State' will be mapped to numerical values using these rules:

  - For values containing 'passed', 'Passed', 'Pass'
  - For values containing 'nconclusive'
  - For values containing 'failed', 'Failed, 'FAIL'

–

- **Date metrics to extract (`date_list`): Optional.**

  To specify the list of date data to extract from the Excel file, using the following format:

  *<METRIC_ID>?column=<COLUMN_NAME>&format=<DATE_FORMAT>*Examples:

  - **CREATION_DATE?column=Created on**

  Date values found in column 'Created on' will be associated to metric CREATION_DATE, using the default dd-MMM-yyyy format

  - **LAST_UPDATE?column=Updated on&format=yyyy/mm/dd**

  Date values found in column 'Created on' will be associated to metric CREATION_DATE, using the yyyy/mm/dd format

  Note:Date patterns are based on SimpleDateFormat Java class specifications.

- **Filters to set the list of artefacts to keep (`filter_list`): Optional.**

  If specified only artefacts complying with the provided filters are kept. Use the following format:

  *<COLUMN_NAME>?regex=<REGEX>*Examples:

  - **Name?regex=^ST***

  Only create artefacts for which column 'Name' starts with 'ST'

  - **Name?regex=^ST*;Region?regex=Europe**

  Same as before, but restrict to artefacts where column 'Region' is 'Europe'

- **Import data via UID only (`import_data_via_uid_only`, default: 0)**: Specify this option if you want to add metric/info to an artefact which created in another Data Provider

- **Header row index (`initial_row`, default: 0)**: Specify the line number where headers are defined.Note:Indexes start at value '0', e.g. the 4th line has index 3.

The full command line syntax for Excel Import is:

```
-d
"type=import_excel,input_file=[file],sheetname=[text],artefact_type=[text],artefact_ty
pe_container=[text],artefact_uid=[text],artefact_link=[text],artefact_name=[text],path
_list=[text],info_list=[text],metric_list=[text],date_list=[text],filter_list=[text],i
mport_data_via_uid_only=[text],initial_row=[text]"
```

Memory Data Import

Description

Memory Data Import provides a generic import mechanism for memory data from a CSV or Excel file.

Usage

Memory Data Import has the following options:

- **Root node name (`root_node`, default: Resources)**: Specify the name of root node in the artefact tree.
- **Data File (`xls_file`)**: Specify the path to the file containing Memory information.
- **Sheet Name (`xls_sheetname`)**: Specify the name of the Excel sheet that contains the Memory list.
- **Memory Column name (`xls_key`)**: Specify the header name of the column which contains the Memory key.
- **Grouping Structure (`xls_groups`)**: Specify the headers for Grouping Structure, separated by ";".
- **Filtering (`xls_filters`)**: Specify the list of Header for filtering

  For example: **"column_name_1=regex1;column_name_2=regex2;**
- **Specify the CSV separator (`csv_separator`, default: ;)**: Specify the CSV separator
- **Memory size column name (`memory_size_column_name`, default: Total)**: Specify the header name of the column which contains the memory size.
- **Used memory column name (`memory_used_column_name`, default: Used)**: Specify the header name of the column which contains the used memory.
- **Memory type column name (`memory_type_column_name`, default: Type)**: Specify the header name of the column which contains the memory type.
- **ROM memory type name (`memory_type_rom_name`, default: ROM)**: Specify the name used for ROM memory.
- **RAM memory type name (`memory_type_ram_name`, default: RAM)**: Specify the name used for RAM memory.
- **NVM memory type name (`memory_type_nvm_name`, default: NVM)**: Specify the name used for NVM memory.
- **Create an output file (`createOutput`, default: true)**: Create an output file

The full command line syntax for Memory Data Import is:

```
-d
"type=import_memory,root_node=[text],xls_file=[file],xls_sheetname=[text],xls_key=[tex
t],xls_groups=[text],xls_filters=[text],csv_separator=[text],memory_size_column_name=[
text],memory_used_column_name=[text],memory_type_column_name=[text],memory_type_rom_na
me=[text],memory_type_ram_name=[text],memory_type_nvm_name=[text],createOutput=[boolea
nChoice]"
```

Requirement Data Import

Description

Requirement Data Import provides a generic import mechanism for requirements from a CSV.

Requirement Data Import provides fields so you can map all your requirements and spread them over the following statuses: Proposed, Analyzed, Approved, Implemented, Verified, Postponed, Deleted, Rejected. Overlapping statuses will cause an error, but if a requirement's status is not declared in the definition, the requirement will still be imported, and a finding will be created.

Usage

Requirement Data Import has the following options:

- **Root Node (`root_node`, default: Requirements)**: Specify the name of the node to attach requirements to.
- **Data File (`input_file`)**: Specify the path to the CSV file containing requirements.
- **Sheet Name (`xls_sheetname`)**: Specify the sheet name that contains the requirement list.
- **Requirement ID (`artefact_id`)**: Specify the header name of the column which contains the requirement ID.
- **Requirement version (`version`)**: Specify the header name of the column which contains the requirement version.
- **Linked Requirements IDs which satisfy this requirement (`link_satisfied_by`)**: Specify the header name of the column which contains the requirements IDs which satisfy this requirement.
- **Linked Test ID verifying this requirement (`link_tested_by`)**: Specify the header name of the column which contains the linked test ID verifying this requirement.
- **Linked Ticket ID associated to this requirement (`link_ticket`)**: Specify the header name of the column which contains the linked Ticket ID corresponding to an issue or enhancement request.
- **Requirement Name (`artefact_name`)**: Specify the pattern used to build the name of the requirement. The name can use any information collected from the CSV file as a parameter.

  Example: **${ID} : ${Summary}**
- **Requirement UID (`artefact_uid`)**: Specify the pattern used to build the requirement Unique ID. The UID can use any information collected from the CSV file as a parameter.

  Example: **TK#${ID}**
- **Grouping Structure (`artefact_groups`)**: Specify the headers for Grouping Structure, separated by ";".

  For example: **"column_name_1=regex1;column_name_2=regex2;**
- **Filtering (`artefact_filters`)**: Specify the list of Header for filtering

  For example: **"column_name_1=regex1;column_name_2=regex2;**
- **Applicable Requirement Pattern (`definition_applicable`)**: Specify the pattern applied to define requirements as Applicable. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Applicable=Yes**

- **Proposed Requirement Pattern (`definition_proposed`)**: Specify the pattern applied to define requirements as proposed. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Proposed**

- **Analyzed Requirement Pattern (`definition_analyzed`)**: Specify the pattern applied to define requirements as analyzed. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Examples:

  - **Status=Analyzed**
  - **Status=[Analyzed|Introduced]**
  - **Status=Analyzed;Decision=[final;revised]**

- **Approved Requirement Pattern (`definition_approved`)**: Specify the pattern applied to define requirements as approved. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Proposed**

- **Implemented Pattern (`definition_implemented`)**: Specify the pattern applied to define requirements as Implemented. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Implemented**

- **Verified Requirement Pattern (`definition_verified`)**: Specify the pattern applied to define requirements as Verified. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Verified**

- **Postponed Requirement Pattern (`definition_postponed`)**: Specify the pattern applied to define requirements as Postponed. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=postponed**

- **Deleted Requirement Pattern (`definition_deleted`)**: Specify the pattern applied to define requirements as deleted. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Deleted**

- **Rejected Requirement Pattern (`definition_rejected`)**: Specify the pattern applied to define requirements as rejected. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Rejected**

- **'Very high' Requirement priority Pattern (`definition_priority_very_high`)**: Specify the pattern applied to define requirements priority as 'Very High' (usually associated to value '1'). This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Priority=1**

- **'High' Requirement priority Pattern (`definition_priority_high`)**: Specify the pattern applied to define requirements priority as 'High' (usually associated to value '2'). This field accepts a

regular expression to match one or more column headers with a list of possible values.

Example: **Priority=2**

- **'Medium' Requirement priority Pattern (`definition_priority_medium`)**: Specify the pattern applied to define requirements priority as 'Medium' (usually associated to value '3'). This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Priority=3**

- **'Low' Requirement priority Pattern (`definition_priority_low`)**: Specify the pattern applied to define requirements priority as 'Low' (usually associated to value '4'). This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Priority=4**

- **'Met' Compliance Pattern (`definition_met`)**: Specify the pattern applied to define requirement Compliance as 'Met'. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Compliance=Met**

- **'Partially Met' Compliance Pattern (`definition_partially_met`)**: Specify the pattern applied to define requirement Compliance as 'Partially Met'. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Compliance=Partially Met**

- **'Not Met' Compliance Pattern (`definition_not_met`)**: Specify the pattern applied to define requirement Compliance as 'Not Met'. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Compliance=Not Met**

- **'Inspection' Test Method Pattern (`definition_inspection`)**: Specify the pattern applied to define requirement Test method as 'Inspection'. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **IADT Method=Inspection**

- **'Analysis' Test Method Pattern (`definition_analysis`)**: Specify the pattern applied to define requirement Test method as 'Analysis'. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **IADT Method=Analysis**

- **'Demonstration' Test Method Pattern (`definition_demonstration`)**: Specify the pattern applied to define requirement Test method as 'Demonstration'. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **IADT Method=Demonstration**

- **'Test' Test Method Pattern (`definition_test`)**: Specify the pattern applied to define requirement Test method as 'Test'. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **IADT Method=Test**

- **Creation Date Column (`creation_date`)**: Enter the name of the column containing the creation date of the requirement.

Accepted formats are detailed here.

- **Last Update Column (`last_updated`)**: Enter the name of the column containing the last modification date of the requirement.

  Accepted formats are detailed here.

- **URL (`url`)**: Specify the pattern used to build the requirement URL. The URL can use any information collected from the CSV file as a parameter.

  Example: **https://example.com/bugs/${ID}**

- **Description Column (`description`)**: Specify the header of the column containing the description of the requirement.

- **Priority Column (`priority`)**: Specify the header of the column containing priority data.

- **'A' critical factor Pattern (`definition_crit_factor_A`)**: Specify the pattern applied to define requirement critical factor as 'A' (low). This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Criticity=A**.

- **'B' critical factor Pattern (`definition_crit_factor_B`)**: Specify the pattern applied to define requirement critical factor as 'B' (medium). This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Criticity=B**.

- **'C' critical factor Pattern (`definition_crit_factor_C`)**: Specify the pattern applied to define requirement critical factor as 'C' (high). This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Criticity=C**.

- **'D' critical factor Pattern (`definition_crit_factor_D`)**: Specify the pattern applied to define requirement critical factor as 'D' (highest). This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Criticity=D**.

- **CSV Separator (`csv_separator`)**: Specify the character used in the CSV file to separate columns.

- **Information Fields (`informations`)**: Specify the list of extra textual information to import from the CSV file. This parameter expects a list of headers separated by ";" characters.

  For example: **Company;Country;Resolution**

- **Save Output (`createOutput`)**:

The full command line syntax for Requirement Data Import is:

```
-d
"type=import_req,root_node=[text],input_file=[file],xls_sheetname=[text],artefact_id=[
text],version=[text],link_satisfied_by=[text],link_tested_by=[text],link_ticket=[text]
,artefact_name=[text],artefact_uid=[text],artefact_groups=[text],artefact_filters=[tex
t],definition_applicable=[text],definition_proposed=[text],definition_analyzed=[text],
definition_approved=[text],definition_implemented=[text],definition_verified=[text],de
finition_postponed=[text],definition_deleted=[text],definition_rejected=[text],definit
ion_priority_very_high=[text],definition_priority_high=[text],definition_priority_medi
um=[text],definition_priority_low=[text],definition_met=[text],definition_partially_me
t=[text],definition_not_met=[text],definition_inspection=[text],definition_analysis=[t
ext],definition_demonstration=[text],definition_test=[text],creation_date=[text],last_
updated=[text],url=[text],description=[text],priority=[text],definition_crit_factor_A=
[text],definition_crit_factor_B=[text],definition_crit_factor_C=[text],definition_crit
_factor_D=[text],csv_separator=[text],informations=[text],createOutput=[booleanChoice]
"
```

Requirement ASIL via Excel Import

Description

Requirement ASIL via Excel Import

Usage

Requirement ASIL via Excel Import has the following options:

- **Input file (`input_file`)**: Specify the Excel input file
- **Sheetname (`sheetname`)**: Sheetname to read data from
- **Artefact name (`artefact_name`)**: Artefact name as displayed in Squore. Examples:
  - ${ID}
  - T_${Name}
  - ${Name} ${Descr}

  Note:${NAME} designates the column called NAME
- **Path to the artefact (`path_list`)**: **Optional. If not used, artefacts extracted from the Excel file will be directly added to the Squore root.**

  To specify the path in Squore of artefacts exracted from the Excel file, using the following format:

  *<COLUMN_NAME>?map=[<REGEX_1>:<GROUP_NAME_1>,...
  ,<REGEX_N>:<GROUP_NAME_N>]&groupByDate=<YES>&format=<dd-mm-YYYY>*Examples:
  - **Area**

  Artefacts will be regrouped by the value found in the 'Area' column
  - **Area?map=[A*:Area A,B*:Area B]**

Artefacts will be regrouped into two groups:'Area A', for all values of 'Area' column starting with letter 'A', and 'Area B' for letter 'B'.

- **Started on?groupByDate=Yes&format=YYYY/mm/dd**

Artefacts will be regrouped by the date found in column 'Started on', using the format 'YYYY/mm/dd'

Note:Date patterns are based on SimpleDateFormat Java class specifications.

- **Textual data to extract (`info_list`): Optional.**

To specify the list of textual data to extract from the Excel file, using the following format:

*<METRIC_ID>?column=<COLUMN_NAME>&map=[<REGEX_1>:<TEXT_1>,...
,<REGEX_N>:<TEXT_N>]*Examples:

- **ZONE_ID?column=Zone**

Textual data found in column 'Zone' will be associated to metric ZONE_ID

- **ZONE_ID?column=Zone;OWNER?column=Belongs to**

Textual data found in columns 'Zone' and 'Belongs to' will be associated to metric ZONE_ID and OWNER respectively

- **ORIGIN?column=Comes from,map=[Cust*:External,Sub-contractor*:External,Support:Internal,Dev:Internal]**

_Textual data found in column 'Comes from' will be associated to metric ORIGIN:

- With value 'External' if the column starts with 'Cust' or 'Sub-contractor'
- With value 'Internal' if the column equals 'Support' or 'Dev'

_

- **Started on?groupByDate=Yes&format=YYYY/mm/dd**

Artefacts will be regrouped by the date found in column 'Started on', using the format 'YYYY/mm/dd'

- **Numerical metrics to extract (`metric_list`): Optional.**

To specify the list of numerical data to extract from the Excel file, using the following format:

*<METRIC_ID>?column=<COLUMN_NAME>&extract=<REGEX_EXRACT>&map=[<REGEX_1>:<
VALUE_1>,...,<REGEX_N>:<VALUE_N>]*Examples:

- **PRIORITY?column=Priority level**

Numerical values found in column 'Priority level' will be associated to metric PRIORITY

- **SEVERITY?column=Severity level,extract=S_**

Numerical values found in column 'Severity level' will be associated to metric SEVERITY, after having extracted (removed) the string 'S_', because in this example, column 'Severity level' contains for example 'S_1', 'S_4', etc., and we want to obtain '1', '4', etc.

- **STATUS?column=State&map=[passed:0,Passed:0,Pass:0,*nconclusive*:1,failed:2,Failed:2,FAIL:2]**

_Textual values found in column 'State' will be mapped to numerical values using these rules:

- For values containing 'passed', 'Passed', 'Pass'
- For values containing 'nconclusive'
- For values containing 'failed', 'Failed, 'FAIL'

–

- **Artefact unique ID (`artefact_uid`): Optional unless you want to use links to these artefacts.**

  This is the artefact unique ID, to be used by links, from this Data Provider, or another Data Provider.Examples:

  - ${ID}
  - T_${Name}
  - ${Name} ${Descr}

  Note:${NAME} designates the column called NAME

The full command line syntax for Requirement ASIL via Excel Import is:

```
-d
"type=import_req_asil,input_file=[file],sheetname=[text],artefact_name=[text],path_lis
t=[text],info_list=[text],metric_list=[text],artefact_uid=[text]"
```

Stack Data Import

Description

Stack Data Import provides a generic import mechanism for stack data from a CSV or Excel file.

Usage

Stack Data Import has the following options:

- **Root node name (`root_node`, default: Resources)**: Specify the name of root node in the artefact tree.
- **Data File (`xls_file`)**: Specify the path to the file containing Stack information.
- **Sheet Name (`xls_sheetname`)**: Specify the sheetname that contains the Stack list.
- **Stack Column name (`xls_key`)**: Specify the header name of the column which contains the Stack key.
- **Grouping Structure (`xls_groups`)**: Specify the headers for Grouping Structure, separated by ";".
- **Filtering (`xls_filters`)**: Specify the list of Header for filtering

  For example: **"column_name_1=regex1;column_name_2=regex2;**
- **Specify the CSV separator (`csv_separator`, default: ;)**: Specify the CSV separator
- **Stack size column (`stack_size_column_name`, default: Stack Size [Bytes])**: Specify the name of the column of Stack Size

- **Stack Average column (`stack_average_column_name`, default: Average Stack Size used [Bytes])**: Specify the name of the column of Stack Average
- **Stack Worst column (`stack_worst_column_name`, default: Worse Case Stack Size used [Bytes])**: Specify the name of the column of Stack Worst
- **Create an output file (`createOutput`, default: true)**: Create an output file

The full command line syntax for Stack Data Import is:

```
-d
"type=import_stack,root_node=[text],xls_file=[file],xls_sheetname=[text],xls_key=[text
],xls_groups=[text],xls_filters=[text],csv_separator=[text],stack_size_column_name=[te
xt],stack_average_column_name=[text],stack_worst_column_name=[text],createOutput=[bool
eanChoice]"
```

Test Data Import

Description

Test Data Import provides a generic import mechanism for tests from a CSV, Excel or JSON file. Additionnally, it generates findings when the imported tests have an unknown status or type.

This Data Provider provides fields so you can map all your tests and spread them over the following statuses: Failed, Inconclusive, Passd. Overlapping statuses and types will cause an error, but if a test status is not declared in the definition, the test will still be imported, and a finding will be created.

Usage

Test Data Import has the following options:

- **Root Node (`root_node`, default: Tests)**: Specify the name of the node to attach tests to.
- **Data File (`input_file`)**: Specify the path to the CSV, Excel or JSON file containing tests.
- **Excel Sheet Name (`xls_sheetname`)**: Specify the sheet name that contains the test list if your import file is in Excel format.
- **TestID (`artefact_id`)**: Specify the header name of the column which contains the test ID.
- **Linear Index Column (`linear_idx`)**: Specify the column name of the Linear Index (=Linear Index is used to order unit or integration tests in matrix graph).
- **Test Name (`artefact_name`)**: Specify the pattern used to build the name of the test. The name can use any information collected from the CSV file as a parameter.

  Example: **${ID} : ${Summary}**
- **Test UID (`artefact_uid`)**: Specify the pattern used to build the test Unique ID. The UID can use any information collected from the CSV file as a parameter.

  Example: **TST#${ID}**

- **Grouping Structure (`artefact_groups`)**: Specify the headers for Grouping Structure, separated by ";".

  For example: **"column_name_1=regex1;column_name_2=regex2;**

- **Filtering (`artefact_filters`)**: Specify the list of Header for filtering

  For example: **"column_name_1=regex1;column_name_2=regex2;**

- **Failed Test Pattern (`definition_failed`)**: Specify the pattern applied to define tests as failed. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Failed**

- **Inconcusive Test Pattern (`definition_inconclusive`)**: Specify the pattern applied to define tests as inconclusive. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=[Inconclusive|Unfinished]**

- **Passed Test Pattern (`definition_passed`)**: Specify the pattern applied to define tests as passed. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Passed**

- **Date when the test was executed (`execution_date`)**: Enter the name of the column containing the execution date of the test.

  Accepted formats are detailed here.

- **Unit of test duration (`execution_duration_unit`, default: ms)**: Enter the unit used for the test duration. Possible values are 's' (seconds) or 'ms' (milliseconds), default is 'ms')

- **Duration of the test (`execution_duration`)**: Enter duration of the test, in milliseconds.

- **TODO Pattern (`in_todo_list`)**: Specify the pattern applied to include tests in the TODO list. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Active=Yes**

- **Creation Date Column (`creation_date`)**: Enter the name of the column containing the creation date of the test.

  Accepted formats are detailed here.

- **Last Updated Date Column (`last_updated_date`)**: Enter the name of the column containing the last updated date of the test.

  Accepted formats are detailed here.

- **URL (`url`)**: Specify the pattern used to build the test URL. The URL can use any information collected from the CSV file as a parameter.

  Example: **https://example.com/tests/${ID}**

- **Description Column (`description`)**: Specify the header of the column containing the description of the test.

- **Category Column (`category`)**: Specify the header of the column containing the category of the

test.

- **Priority Column (`priority`)**: Specify the header of the column containing priority data.
- **CSV Separator (`csv_separator`)**: Specify the character used in the CSV file to separate columns.
- **Information Fields (`informations`)**: Specify the list of extra textual information to import from the CSV file. This parameter expects a list of headers separated by ";" characters.

  For example: **Architecture;Responsible;Target**
- **Save Output (`createOutput`)**:

The full command line syntax for Test Data Import is:

```
-d
"type=import_test,root_node=[text],input_file=[file],xls_sheetname=[text],artefact_id=
[text],linear_idx=[text],artefact_name=[text],artefact_uid=[text],artefact_groups=[tex
t],artefact_filters=[text],definition_failed=[text],definition_inconclusive=[text],def
inition_passed=[text],execution_date=[text],execution_duration_unit=[multipleChoice],e
xecution_duration=[text],in_todo_list=[text],creation_date=[text],last_updated_date=[t
ext],url=[text],description=[text],category=[text],priority=[text],csv_separator=[text
],informations=[text],createOutput=[booleanChoice]"
```

Test Excel Import

Description

Test Excel Import

Usage

Test Excel Import has the following options:

- **Input file (`input_file`)**: Specify the Excel input file
- **Sheetname (`sheetname`)**: Sheetname to read data from
- **Artefact name (`artefact_name`)**: Artefact name as displayed in Squore. Examples:
  - ${ID}
  - T_${Name}
  - ${Name} ${Descr}

  Note:${NAME} designates the column called NAME
- **Path to the artefact (`path_list`)**: **Optional. If not used, artefacts extracted from the Excel file will be directly added to the Squore root.**

  To specify the path in Squore of artefacts exracted from the Excel file, using the following format:

  *<COLUMN_NAME>?map=[<REGEX_1>:<GROUP_NAME_1>,...

,<REGEX_N>:<GROUP_NAME_N>]&groupByDate=<YES>&format=<dd-mm-YYYY>*Examples:

- **Area**

Artefacts will be regrouped by the value found in the 'Area' column

- **Area?map=[A*:Area A,B*:Area B]**

Artefacts will be regrouped into two groups:'Area A', for all values of 'Area' column starting with letter 'A', and 'Area B' for letter 'B'.

- **Started on?groupByDate=Yes&format=YYYY/mm/dd**

Artefacts will be regrouped by the date found in column 'Started on', using the format 'YYYY/mm/dd'

Note:Date patterns are based on SimpleDateFormat Java class specifications.

- **Textual data to extract (`info_list`): Optional.**

To specify the list of textual data to extract from the Excel file, using the following format:

*<METRIC_ID>?column=<COLUMN_NAME>&map=[<REGEX_1>:<TEXT_1>,...
,<REGEX_N>:<TEXT_N>]*Examples:

- **ZONE_ID?column=Zone**

Textual data found in column 'Zone' will be associated to metric ZONE_ID

- **ZONE_ID?column=Zone;OWNER?column=Belongs to**

Textual data found in columns 'Zone' and 'Belongs to' will be associated to metric ZONE_ID and OWNER respectively

- **ORIGIN?column=Comes from,map=[Cust*:External,Sub-contractor*:External,Support:Internal,Dev:Internal]**

_Textual data found in column 'Comes from' will be associated to metric ORIGIN:

- With value 'External' if the column starts with 'Cust' or 'Sub-contractor'
- With value 'Internal' if the column equals 'Support' or 'Dev'

_

- **Started on?groupByDate=Yes&format=YYYY/mm/dd**

Artefacts will be regrouped by the date found in column 'Started on', using the format 'YYYY/mm/dd'

- **Numerical metrics to extract (`metric_list`): Optional.**

To specify the list of numerical data to extract from the Excel file, using the following format:

*<METRIC_ID>?column=<COLUMN_NAME>&extract=<REGEX_EXRACT>&map=[<REGEX_1>:<VALUE_1>,...,<REGEX_N>:<VALUE_N>]*Examples:

- **PRIORITY?column=Priority level**

Numerical values found in column 'Priority level' will be associated to metric PRIORITY

- **SEVERITY?column=Severity level,extract=S_**

Numerical values found in column 'Severity level' will be associated to metric SEVERITY, after having extracted (removed) the string 'S_', because in this example, column 'Severity level' contains for example 'S_1', 'S_4', etc., and we want to obtain '1', '4', etc.

- **STATUS?column=State&map=[passed:0,Passed:0,Pass:0,*nconclusive*:1,failed:2,Failed:2,FAIL:2]**

_Textual values found in column 'State' will be mapped to numerical values using these rules:

- For values containing 'passed', 'Passed', 'Pass'
- For values containing 'nconclusive'
- For values containing 'failed', 'Failed, 'FAIL'

_

- **Date metrics to extract (`date_list`): Optional.**

  To specify the list of date data to extract from the Excel file, using the following format:

  *<METRIC_ID>?column=<COLUMN_NAME>&format=<DATE_FORMAT>*Examples:

  - **CREATION_DATE?column=Created on**

  Date values found in column 'Created on' will be associated to metric CREATION_DATE, using the default dd-MMM-yyyy format

  - **LAST_UPDATE?column=Updated on&format=yyyy/mm/dd**

  Date values found in column 'Created on' will be associated to metric CREATION_DATE, using the yyyy/mm/dd format

  Note:Date patterns are based on SimpleDateFormat Java class specifications.

- **Filters to set the list of artefacts to keep (`filter_list`): Optional.**

  If specified only artefacts complying with the provided filters are kept. Use the following format:

  *<COLUMN_NAME>?regex=<REGEX>*Examples:

  - **Name?regex=^ST***

  Only create artefacts for which column 'Name' starts with 'ST'

  - **Name?regex=^ST*;Region?regex=Europe**

  Same as before, but restrict to artefacts where column 'Region' is 'Europe'

- **Artefact unique ID (`artefact_uid`): Optional unless you want to use links to these artefacts.**

  This is the artefact unique ID, to be used by links, from this Data Provider, or another Data Provider.Examples:

  - ${ID}
  - T_${Name}
  - ${Name} ${Descr}

  Note:${NAME} designates the column called NAME

- **Links to this artefact (`artefact_link`)**: Specify how to create links between this artefact and other artefacts with the following format:

*<LINK_TYPE>?direction=<IN OR OUT>&column=<COLUMN_NAME>&separator=<SEPARATOR>*Examples:

- **TESTED_BY?column=Test**

A 'TESTED_BY' link will be created with the UID found in column 'Test'

- **IMPLEMENTED_BY?direction=IN&column=Implements**

An 'IMPLEMENTED_BY' link will be created with the UID found in column 'Implements'. Since the optional 'direction' attribute is provided, it will be set as 'IN' (default value is 'OUT')

- **TESTED_BY?column=Tests&separator=','**

'TESTED_BY' links will be created with all UIDs found in column 'Tests', separated by a comma

- **TESTED_BY?column=Tests&separator=',';REFINED_BY?column=DownLinks&separator=','**

'TESTED_BY' and 'REFINED_BY' links will be created with UIDs found in columns 'Tests' and 'DownLinks' respectively

The full command line syntax for Test Excel Import is:

```
-d
"type=import_test_excel,input_file=[file],sheetname=[text],artefact_name=[text],path_l
ist=[text],info_list=[text],metric_list=[text],date_list=[text],filter_list=[text],art
efact_uid=[text],artefact_link=[text]"
```

Ticket Data Import

Description

Ticket Data Import provides a generic import mechanism for tickets from a CSV, Excel or JSON file. Additionnally, it generates findings when the imported tickets have an unknown status or type.

This Data Provider provides fields so you can map all your tickets as Enhancements and defects and spread them over the following statuses: Open, In Implementation, In Verification, Closed. Overlapping statuses and types will cause an error, but if a ticket's type or status is not declared in the definition, the ticket will still be imported, and a finding will be created.

Usage

Ticket Data Import has the following options:

- **Root Node (`root_node`, default: Tickets)**: Specify the name of the node to attach tickets to.
- **Data File (`input_file`)**: Specify the path to the CSV, Excel or JSON file containing tickets.
- **Excel Sheet Name (`xls_sheetname`)**: Specify the sheet name that contains the ticket list if your import file is in Excel format.
- **Ticket ID (`artefact_id`)**: Specify the header name of the column which contains the ticket ID.

- **Ticket Name (`artefact_name`)**: Specify the pattern used to build the name of the ticket. The name can use any information collected from the CSV file as a parameter.

  Example: **${ID} : ${Summary}**

- **Ticket UID (`artefact_uid`)**: Specify the pattern used to build the ticket Unique ID. The UID can use any information collected from the CSV file as a parameter.

  Example: **TK#${ID}**

- **Grouping Structure (`artefact_groups`)**: Specify the headers for Grouping Structure, separated by ";".

  For example: **"column_name_1=regex1;column_name_2=regex2;**

- **Filtering (`artefact_filters`)**: Specify the list of Header for filtering

  For example: **"column_name_1=regex1;column_name_2=regex2;**

- **Open Ticket Pattern (`definition_open`)**: Specify the pattern applied to define tickets as open. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=[Open|New]**

- **In Development Ticket Pattern (`definition_rd_progress`)**: Specify the pattern applied to define tickets as in development. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Implementing**

- **Fixed Ticket Pattern (`definition_vv_progress`)**: Specify the pattern applied to define tickets as fixed. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Verifying;Resolution=[fixed;removed]**

- **Closed Ticket Pattern (`definition_close`)**: Specify the pattern applied to define tickets as closed. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Closed**

- **Defect Pattern (`definition_defect`)**: Specify the pattern applied to define tickets as defects. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Type=Bug**

- **Enhancement Pattern (`definition_enhancement`)**: Specify the pattern applied to define tickets as enhancements. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Type=Enhancement**

- **TODO Pattern (`in_todo_list`)**: Specify the pattern applied to include tickets in the TODO list. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Sprint=2018-23**

- **Creation Date Column (`creation_date`)**: Enter the name of the column containing the creation

date of the ticket.

Accepted formats are detailed here.

- **Due Date Column (`due_date`)**: Enter the name of the column containing the due date of the ticket.

  Accepted formats are detailed here.

- **Last Updated Date Column (`last_updated_date`)**: Enter the name of the column containing the last updated date of the ticket.

  Accepted formats are detailed here.

- **Closure Date Column (`closure_date`)**: Enter the name of the column containing the closure date of the ticket.

  Accepted formats are detailed here.

- **URL (`url`)**: Specify the pattern used to build the ticket URL. The URL can use any information collected from the CSV file as a parameter.

  Example: **https://example.com/bugs/${ID}**

- **Description Column (`description`)**: Specify the header of the column containing the description of the ticket.

- **Category Column (`category`)**: Specify the header of the column containing the category of the ticket.

- **Reporter Column (`reporter`)**: Specify the header of the column containing the reporter of the ticket.

- **Handler Column (`handler`)**: Specify the header of the column containing the handler of the ticket.

- **Priority Column (`priority`)**: Specify the header of the column containing priority data.

- **Severity Column (`severity`)**: Specify the header of the column containing severity data.

- **CSV Separator (`csv_separator`)**: Specify the character used in the CSV file to separate columns.

- **Information Fields (`informations`)**: Specify the list of extra textual information to import from the CSV file. This parameter expects a list of headers separated by ";" characters.

  For example: **Company;Country;Resolution**

- **Save Output (`createOutput`)**:

The full command line syntax for Ticket Data Import is:

```
-d
"type=import_ticket,root_node=[text],input_file=[file],xls_sheetname=[text],artefact_i
d=[text],artefact_name=[text],artefact_uid=[text],artefact_groups=[text],artefact_filt
ers=[text],definition_open=[text],definition_rd_progress=[text],definition_vv_progress
=[text],definition_close=[text],definition_defect=[text],definition_enhancement=[text]
,in_todo_list=[text],creation_date=[text],due_date=[text],last_updated_date=[text],clo
sure_date=[text],url=[text],description=[text],category=[text],reporter=[text],handler
=[text],priority=[text],severity=[text],csv_separator=[text],informations=[text],creat
eOutput=[booleanChoice]"
```

Jira

Description

This Data Provider extracts tickets and their attributes from a Jira instance to create ticket artefacts in your project.

For more details, refer to https://www.atlassian.com/software/jira.

The extracted JSON from Jira is then passed to the Ticket Data Import Data Provider (described in Ticket Data Import). Finer configuration of the data passed from this Data Provider to Ticket Data Import is available by editing (or overriding) *<SQUORE_HOME>/addons/tools/jira/jira_config.xml*.

Usage

Jira has the following options:

- **Jira REST API URL (`url`, mandatory)**: The URL used to connect to yout Jira instance's REST API URL (e.g: https://jira.domain.com/rest/api/2)
- **Jira User login (`login`, mandatory)**: Specyfy your Jira User login.
- **Jira User password (`pwd`, mandatory)**: Specify your Jira User password.
- **Number of queried tickets (`max_results`, mandatory, default: -1)**: Maximum number of queried tickets returned by the query (default is -1, meaning 'retrieve all tickets').
- **Additional Fields (`additional_fields`, default: environment,votes)**: List additional fields to be exported from Jira.

  This field accepts a comma-separated list of field names that are added to the export request URL, for example **fixVersions,versions**
- **Grouping Structure (`artefact_groups`, default: fields/components[0]/name)**: Specify the headers for Grouping Structure, separated by ";".

  For example: **"column_name_1=regex1;column_name_2=regex2;**
- **Creation Date Field (`creation_date`, default: fields/created)**: Enter the name of the column containing the creation date of the ticket.

  For example: **column_name{format="dd/mm/yyyy"})**.

If format is not specified, the following is used by default: **dd/mm/yyyy**.

- **Closure Date Field (`closure_date`, default: fields/resolutiondate)**: Enter the name of the column containing the closure date of the ticket.

  For example: **column_name{format="dd/mm/yyyy"})**.

  If format is not specified, the following is used by default: **dd/mm/yyyy**.

- **Due Date Field (`due_date`, default: fields/duedate)**: Enter the name of the column containing the due date of the ticket.

  For example: **column_name{format="dd/mm/yyyy"})**.

  If format is not specified, the following is used by default: **dd/mm/yyyy**.

- **Last Updated Date Field (`last_updated_date`, default: fields/updated)**: Enter the name of the column containing the last updated date of the ticket.

  For example: **column_name{format="dd/mm/yyyy"})**.

  If format is not specified, the following is used by default: **dd/mm/yyyy**.

- **Category (`category`, default: fields/components[0]/name)**: Specify the path to the field that will contain the ticket category.

- **Priority (`priority`, default: fields/priority/name)**: Specify the path to the field that will contain the ticket priority.

- **JQL Request (`jql_request`)**: Specify a JQL request (see JIRA documentation) in order to limit the number of elements sent by the JIRA server.

  For example: **project=MyProject**.This parameter is optional.

- **Filtering (`artefact_filters`, default: fields/issuetype/name=(Task|Bug|Improvement|New Feature))**: Specify the list of Header for filtering

  For example: **"column_name_1=regex1;column_name_2=regex2;**

- **Information Fields (`informations`, default: fields/environment;fields/votes/votes)**: Specify a semicolon-separated list of paths to fields you want to extract from the Jira JSON export to be added as textual information for the ticket artefacts.

  For example: **fields/fixVersions[0]/name;fields/versions[0]/name**

- **Open Ticket Pattern (`definition_open`, default: fields/status/name=[To Do|Open|Reopened])**: Specify the pattern applied to define tickets as open. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=[Open|New]**

- **In Development Ticket Pattern (`definition_rd_progress`, default: fields/status/name=[In Progress|In Review])**: Specify the pattern applied to define tickets as in development. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Implementing**

- **Fixed Ticket Pattern (`definition_vv_progress`, default: fields/status/name=[Verified])**: Specify the pattern applied to define tickets as fixed. This field accepts a regular expression to match one or more column headers with a list of possible values.

Example: **Status=Verifying;Resolution=[fixed;removed]**

- **Closed Ticket Pattern (`definition_close`, default: fields/status/name=[Resolved|Closed|Done])**: Specify the pattern applied to define tickets as closed. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Status=Closed**

- **Defect Pattern (`definition_defect`, default: fields/issuetype/name=[Bug])**: Specify the pattern applied to define tickets as defects. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Type=Bug**

- **Enhancement Pattern (`definition_enhancement`, default: fields/issuetype/name=[Improvement|New Feature])**: Specify the pattern applied to define tickets as enhancements. This field accepts a regular expression to match one or more column headers with a list of possible values.

  Example: **Type=Enhancement**

- **Todo list regex (`in_todo_list`, default: fields/status/name=.)***: Todo list regex (ticket which fit the regex will be considered as part of the TODO list for the analysis)

The full command line syntax for Jira is:

```
-d
"type=jira,url=[text],login=[text],pwd=[password],max_results=[text],additional_fields
=[text],artefact_groups=[text],creation_date=[text],closure_date=[text],due_date=[text
],last_updated_date=[text],category=[text],priority=[text],jql_request=[text],artefact
_filters=[text],informations=[text],definition_open=[text],definition_rd_progress=[tex
t],definition_vv_progress=[text],definition_close=[text],definition_defect=[text],defi
nition_enhancement=[text],in_todo_list=[text]"
```

Mantis

Description

The Mantis Data Provider extracts tickets and their attributes from a Mantis installation and creates ticket artefacts.

Prerequisites:

- This Data Provider queries Mantis tickets using the Mantis BT REST API. An **API token** is required to access this API.
- The Mantis server should be configured to avoid filtering 'Authorization' headers.

See http://docs.php.net/manual/en/features.http-auth.php#114877 for further details.

For more details, refer to https://www.mantisbt.com.

The extracted JSON from Mantis BT is then passed to the Ticket Data Import Data Provider (described in Ticket Data Import). Finer configuration of the data passed from this Data Provider to Ticket Data Import is available by editing (or overriding) *<SQUORE_HOME>/addons/tools/mantis/mantis_config.xml*.

Usage

Mantis has the following options:

- **Mantis URL (`url`, mandatory)**: Specify the URL of the Mantis instance (e.g: https://www.mantisbt.org/bugs/api/rest)
- **Mantis API Token (`api_token`, mandatory)**: Copy the Mantis API Token generated from your Account Settings in Mantis.
- **Number of queried tickets (`max_results`, mandatory, default: 50)**: Maximum number of queried tickets returned by the query (default is 50. value=-1 means 'retrieve all tickets').

The full command line syntax for Mantis is:

```
-d "type=mantis,url=[text],api_token=[text],max_results=[text]"
```

OSLC

Description

OSLC-CM allows retrieving information from Change Management systems following the OSLC standard. Metrics and artefacts are created by connecting to the OSLC system and retrieving issues with the specified query.

For more details, refer to http://open-services.net/.

Usage

OSLC has the following options:

- **Change Server (`server`)**: Specify the URL of the project you want to query on the OSLC server. Typically the URL will look like this: http://myserver:8600/change/oslc/db/3454a67f-656ddd4348e5/role/User/
- **Query (`query`)**: Specify the query to send to the OSLC server (e.g.: release="9TDE/TDE_00_01_00_00"). It is passed to the request URL via the ?oslc_cm.query= parameter.
- **Query Properties (`properties`, default: request_type,problem_number,crstatus,severity,submission_area,functionality...)**: Specify the properties to add to the query. They are passed to the OSLC query URL using the ?oslc_cm.properties= parameter.
- **Login (`login`)**:
- **Password (`password`)**:

The full command line syntax for OSLC is:

```
-d
"type=oslc_cm,server=[text],query=[text],properties=[text],login=[text],password=[pass
word]"
```

pep8

Description

pep8 is a tool to check your Python code against some of the style conventions in PEP 88. Its CSV report file is imported to generate findings.

For more details, refer to https://pypi.python.org/pypi/pep8.

Usage

pep8 has the following options:

- **CSV results file (`csv`)**: Specify the path to the CSV report file created by pep8.

The full command line syntax for pep8 is:

```
-d "type=pep8,csv=[file]"
```

pycodestyle / pep8 (plugin)

Description

Style Guide for Python Code. Pep8 results are imported to produce findings on Python code. This data provider requires having pycodestyle or pep8 installed on the machine running the analysis and the pycodestyle or pep8 command to be available in the path. It is compatible with pycodestyle 2.4 or pep8 1.7 and may also work with older versions.

For more details, refer to https://pypi.org/project/pycodestyle.

Usage

pycodestyle / pep8 (plugin) has the following options:

- **Source code directory to analyse (`dir`)**: Leave this field empty to analyse all sources.

The full command line syntax for pycodestyle / pep8 (plugin) is:

```
-d "type=pep8_auto,dir=[directory]"
```

PHP Code Coverage

Description

Library that provides collection, processing, and rendering functionality for PHP code coverage information.

For more details, refer to https://github.com/sebastianbergmann/php-code-coverage.

Usage

PHP Code Coverage has the following options:

- **Report file or folder (`html_report`)**: Specify the path to the HTML report folder or file which contains the coverage results.

The full command line syntax for PHP Code Coverage is:

```
-d "type=phpcodecoverage,html_report=[file_or_directory]"
```

pylint

Description

Pylint is a Python source code analyzer which looks for programming errors, helps enforcing a coding standard and sniffs for some code smells (as defined in Martin Fowler's Refactoring book). Pylint results are imported to generate findings for Python code.

For more details, refer to http://www.pylint.org/.

Usage

pylint has the following options:

- **CSV results file (`csv`)**: Specify the path to the CSV file containing pylint results. Note that the minimum version supported is 1.1.0.

The full command line syntax for pylint is:

```
-d "type=pylint,csv=[file]"
```

pylint (plugin)

Description

Coding Guide for Python Code. Pylint results are imported to produce findings on Python code. This data provider requires having pylint installed on the machine running the analysis and the pylint command to be available in the path. It is known to work with pylint 1.7.0 and may also work with older versions.

Usage

pylint (plugin) has the following options:

- **Source code directory to analyse (`dir`)**: Leave this field empty to analyse all sources.

The full command line syntax for pylint (plugin) is:

```
-d "type=pylint_auto,dir=[directory]"
```

QAC 8.2

Description

QA-C is a static analysis tool for MISRA checking.

For more details, refer to http://www.programmingresearch.com/static-analysis-software/qac-qacpp-static-analyzers/.

Usage

QAC 8.2 has the following options:

- **QAC output file(s) (`txt`, mandatory)**: Specify the path(s) to the .tab file(s) to extract findings from. To provide multiple files click on '+'
- **Eliminate duplicated findings (`eliminate_duplicate`, default: false)**: When 2 occurences of the same finding (same rule, same file, same line, same description) is found, only one is reported.

The full command line syntax for QAC 8.2 is:

```
-d "type=qac,txt=[file],eliminate_duplicate=[booleanChoice]"
```

QAC 8.2 CERT Import

Description

QA-C is a static analysis tool for MISRA and CERT checking.

For more details, refer to http://www.programmingresearch.com/static-analysis-software/qac-qacpp-static-analyzers/.

Usage

QAC 8.2 CERT Import has the following options:

- **QAC CERT output file(s) (`txt`, mandatory)**: Specify the path(s) to the .tab file(s) to extract findings from. To provide multiple files click on '+'
- **Eliminate duplicated findings (`eliminate_duplicate`, default: false)**: When 2 occurences of the

same finding (same rule, same file, same line, same description) is found, only one is reported.

The full command line syntax for QAC 8.2 CERT Import is:

```
-d "type=qac_cert,txt=[file],eliminate_duplicate=[booleanChoice]"
```

SonarQube

Description

This data provider imports findings from SonarQube. Note that versions prior to 6.2 may not be supported.

For more details, refer to https://www.sonarqube.org/.

Usage

SonarQube has the following options:

- **SonarQube Location (`sonar`, default: `http://127.0.0.1:9000`)**: Specify the URL of the SonarQube installation to work with (for example: http://localhost:9000)
- **SonarQube Component Key (`key`)**:
- **Version Name (`version`)**:
- **Login (`login`)**:
- **Password (`password`)**:

The full command line syntax for SonarQube is:

```
-d
"type=sonarqube,sonar=[text],key=[text],version=[text],login=[text],password=[password
]"
```

Testwell CTC++

Description

Import data from Testwell CTC++ XML results

For more details, refer to http://www.testwell.fi/ctcdesc.html.

Usage

Testwell CTC++ has the following options:

- **Results folder (`dir`)**: Specify the folder containing XML test results files from Testwell CTC++.

- **Instrumented files extension (`extension`, default: .test.runner.c)**: Instrumented files extension (Extension of the instrumented files generated by ctc++)

The full command line syntax for Testwell CTC++ is:

```
-d "type=testwell_ctc,dir=[directory],extension=[text]"
```

vTESTstudio Traceability

Description

Import vTESTstudio traeability Matrix information

For more details, refer to https://www.vector.com/int/en/products/products-a-z/software/vTESTstudio/.

Usage

vTESTstudio Traceability has the following options:

- **Traceability matrix file path (`file`)**: Specify the absolute path to the vTESTstudio traceability matrix file (.vti-tso format)
- **Test path (`testPath`, default: Tests)**: Define test path (for example Test/HIL Test), by default the value is Tests.

The full command line syntax for vTESTstudio Traceability is:

```
-d "type=vTestStudio_Traceability,file=[file],testPath=[text]"
```

PC Lint MISRA 2012

Description

PC Lint MISRA 2012 (via XML import)

Usage

PC Lint MISRA 2012 has the following options:

- **XML File (`xml`)**: Specify the XML file which contains the findings results (MISRA, Coding Style...)

The full command line syntax for PC Lint MISRA 2012 is:

```
-d "type=vectorCAST_Lint,xml=[file]"
```

Adding More Languages to Squan Sources

Squan Sources can handle files written in languages that are not officially supported with a bit of extra configuration. In this mode, only a basic analysis of the file is carried out so that an artefact is created in the project and findings can be attached to it. A subset of the base metrics from Squan Sources is optionally recorded for the artefact so that line counting, stability and text duplication metrics are available at file level for the new language.

The example below shows how you can add TypeScript files to your analysis:

1. Copy *<SQUORE_HOME>/configuration/tools/SQuORE/form.xml* and its *.properties* files into your own configuration
2. Edit *form.xml* to add a new language key and associated file extensions:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <tags baseName="SQuORE" ...>
       <tag type="multipleChoice" key="languages" ... defaultValue="...;typescript">
           ...
           <value key="typescript" option=".ts,.TS" />
       </tag>
   </tags>
   ```

   Files with extensions matching the **typescript** language will be added to your project as TYPESCRIPT_FILE artefacts

3. Edit the `defaultValue` of the `additional_param` field to specify how Squan Sources should count source code lines and comment lines in the new language, based on another language officially supported by Squore. This step is optional, and is only needed if you want the to record basic line counting metrics for the artefacts.

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <tags baseName="SQuORE" ...>
       ...
       <tag type="text" key="additional_param" defaultValue="typescript=javascript" />
       ...
   </tags>
   ```

   Lines in TypeScript files will be counted as they would for Javascript code.

4. Add translations for the new language key to show in the web UI in Squan Sources's *form_en.properties*

   ```
   OPT.typescript.NAME=TypeScript
   ```

5. Add translations for the new artefact type and new LANGUAGE information value in one of the properties files imported by your Description Bundle:

```
T.TYPESCRIPT_FILE.NAME=TypeScript File

INFO_VALUE.LANGUAGE.TYPESCRIPT.NAME=Typescript
INFO_VALUE.LANGUAGE.TYPESCRIPT.COLOR=#2b7489
```

6. The new artefact type should also be declared as a type in your model. The easiest way to do this is to add it to the **GENERIC_FILE** alias in your analysis model, which is pre-configured to record the line counting metrics for new artefacts. You should also define a root indicator for you new artefact type. The following snippet shows a minimal configuration using a dummy indicator:

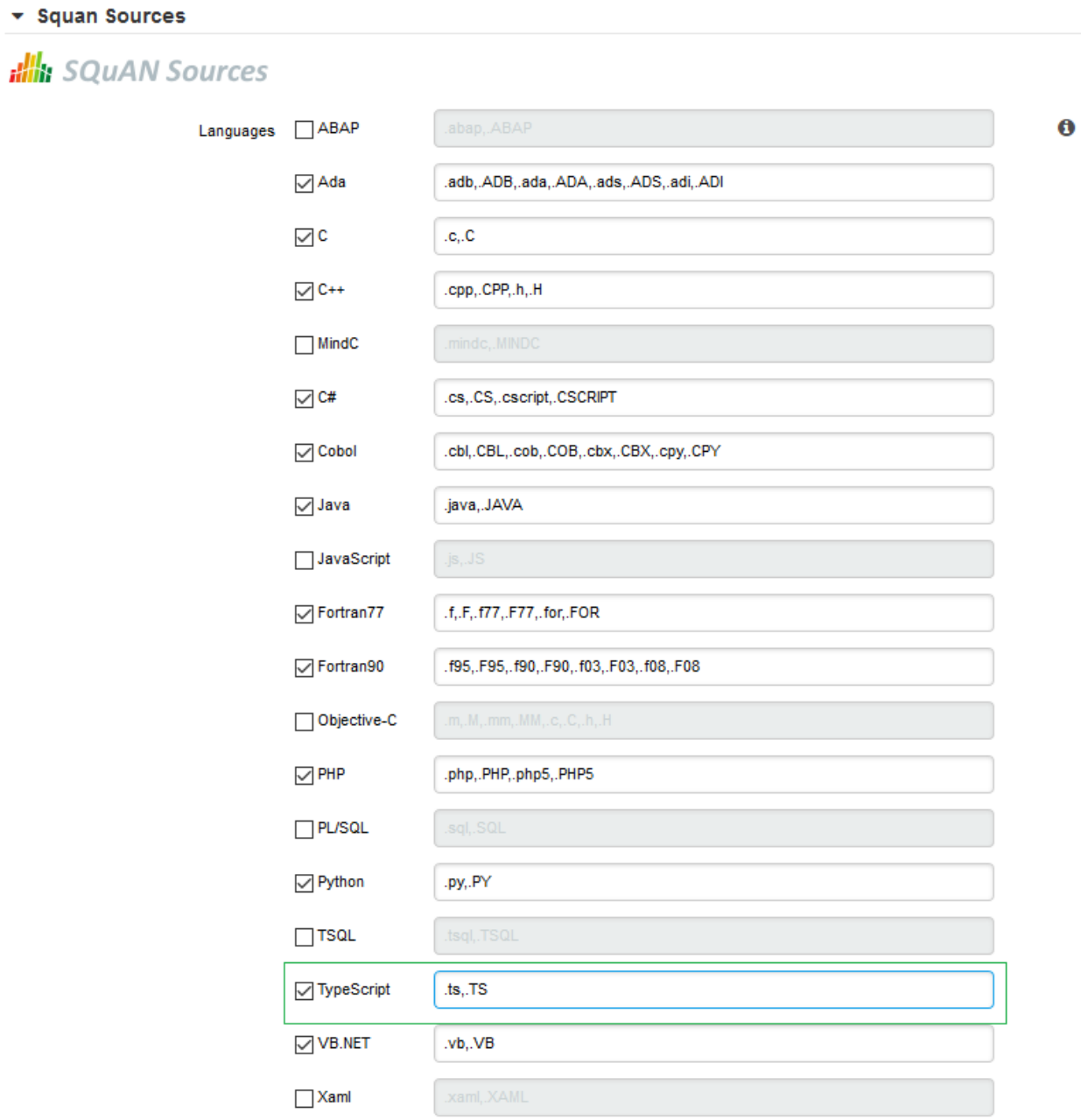
```
<!-- <configuration>/MyModel/Analysis/Bundle.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<Bundle>
...
    <ArtefactType id="GENERIC_FILE" heirs="TYPESCRIPT_FILE" />

    <RootIndicator artefactTypes="TYPESCRIPT_FILE" indicatorId="DUMMY" />
    <Indicator indicatorId="DUMMY" scaleId="SCALE_INFO" targetArtefactTypes=
"TYPESCRIPT_FILE" displayTypes="IMAGE" />

    <Measure measureId="DUMMY">
        <Computation targetArtefactTypes="TYPESCRIPT_FILE" result="0" />
    </Measure>
...
</Bundle>
```

Make sure that this declaration appears in your analysis model before the inclusion of *import.xml* so it overrides the default analysis model.

Don't forget to add translations for your dummy indicator to avoid warnings in the Model Validator:

```
DUMMY.NAME= Generic Indicator
DUMMY.DESCR= This is an indicator for additional languages in Squan Sources. It
does not rate files in any way.
```

7. Reload your configuration and analyse a project, checking the box for TypeScript in Squan Sources's options to get Typescrypt artefacts in your project.



The new option for TypeScript files in Squan Sources

If you are launchin an analysis from the command line, use the language key defined in step 2 to analyse TypeScript files:

```
-d
"type=SQuORE,languages=typescript,additional_param=typescript=javascript"
```

8. After the analysis finishes and you can see your artefacts in the tree, use the Dashboard Editor to build a dashboard for your new artefact type.
9. Finally, create a handler for the source code viewer to display your new file type into your configuration folder, by copying *<SQUORE_HOME>/configuration/sources/javascript_file.properties* into your own configuration as *<SQUORE_HOME>/configuration/sources/typescript_file.properties*.

Advanced COBOL Parsing

By default, Squan Sources generates artefacts for all PROGRAMs in COBOL source files. It is possible to configure the parser to also generate artefacts for all SECTIONs and PARAGRAPHs in your source code. This feature can be enabled with the following steps:

1. Open *<SQUORE_HOME>/configuration/tools/SQuORE/Analyzer/artifacts/cobol/ArtifactsList.txt*
2. Edit the list of artefacts to generate and add the section and paragraph types:

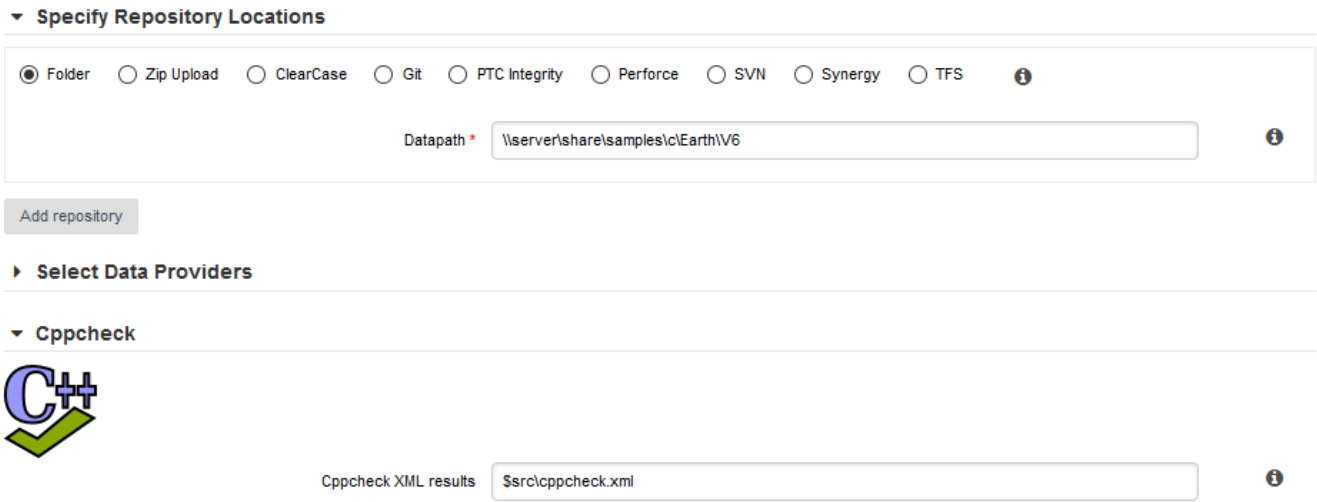
```
program
section
paragraph
```

3. Save your changes

If you create a new project, you will see the new artefacts straight away. For already-existing projects, make sure to launch a new analysis and check Squan Sources's **Force full analysis** option to parse the entire code again and generate the new artefacts.

Using Data Provider Input Files From Version Control

Input files for Squore's Data Providers, like source code, can be located in your version control system. When this is the case, you need to specify a variable in the input field for the Data Provider instead of an absolute path to the input file.



A Data Provider using an input file extracted from a remote repository

The variable to use varies depending on your scenario:

- **You have only one node of source code in your project**

  In this case, the variable to use is **$src**.

- **You have more than one node of source code in your project**

  In this case, you need to tell Squore in which node the input file is located. This is done using a variable that has the same name as the alias you defined for the source code node in the previous step of the wizard. For example, if your nodes are labelled *Node1* and *Node2* (the default names), then you can refer to them using the **$Node1** and **$Node2** variables.

When using these variables from the command line on a linux system, the *$* symbol must be escaped:

```
-d "type=PMD,configFile=\$src/pmd_data.xml"
```

Providing a catalog file to a Data Provider for Offline XSL Transformations

When transforming an XML results file with an XSL stylesheet, the XML parser used by Squore will try to validate the XML file against the DTD declared in the XML header. In cases where the XSL transformation is running on a machine with no internet access, this can result in the execution of the Data Provider failing with a *No route to host* error message.

You can fix this issue by modifying the data provider to use a catalog file that will provide an alternate location for the DTD used to validate the XML. This feature can be used by all Data Providers that include an XSL transformation [1: The list includes:] .

The following example adds this functionality to the Cobertura Data Provider:

1. Add a catalog.xml file in the Data Provider's configuration folder:

   ```
   <configuration>/tools/cobertura/catalog.xml:
   <?xml version="1.0"?>
   <catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
       <rewriteSystem systemIdStartString="http://cobertura.sourceforge.net/xml"
   rewritePrefix="./DTD"/>
   </catalog>
   ```

2. Copy the dtd that the XML needs to validate again inside a *DTD* folder in *<configuration>/tools/cobertura/*.

The catalog file will be used the next time the Data Provider is executed and the DTD declaration will dynamically be changed from:

```
<!DOCTYPE coverage SYSTEM "http://cobertura.sourceforge.net/xml/coverage-04.dtd">
```

to:

```
<!DOCTYPE coverage SYSTEM "<configuration>/tools/cobertura/DTD/coverage-04.dtd">">
```

For more information about how to write your catalog file, refer to https://xerces.apache.org/xerces2-j/faq-xcatalogs.html.

Creating a *form.xml* for your own Data Providers, Repository Connectors and Export Definitions

All Data Providers are utilities that run during an analysis. They usually take an input file to parse or parameters specified by the user to generate output files containing violations or metrics to add to your project. Here is a non-exhaustive list of what some of them do:

- Use XSLT files to transform XML files
- Read information from Microsoft Excel files
- Parse HTML test results
- Query web services
- Export data from OSLC systems
- Launch external processes

Repository Connectors are based on the same model and are used to specifically retrieve source code and other data from source code management systems.

Export Definitions use the same *form.xml* specification to offer custom export formats to users from the web interface, dumping data from highlight definitions into a specified, custom format.

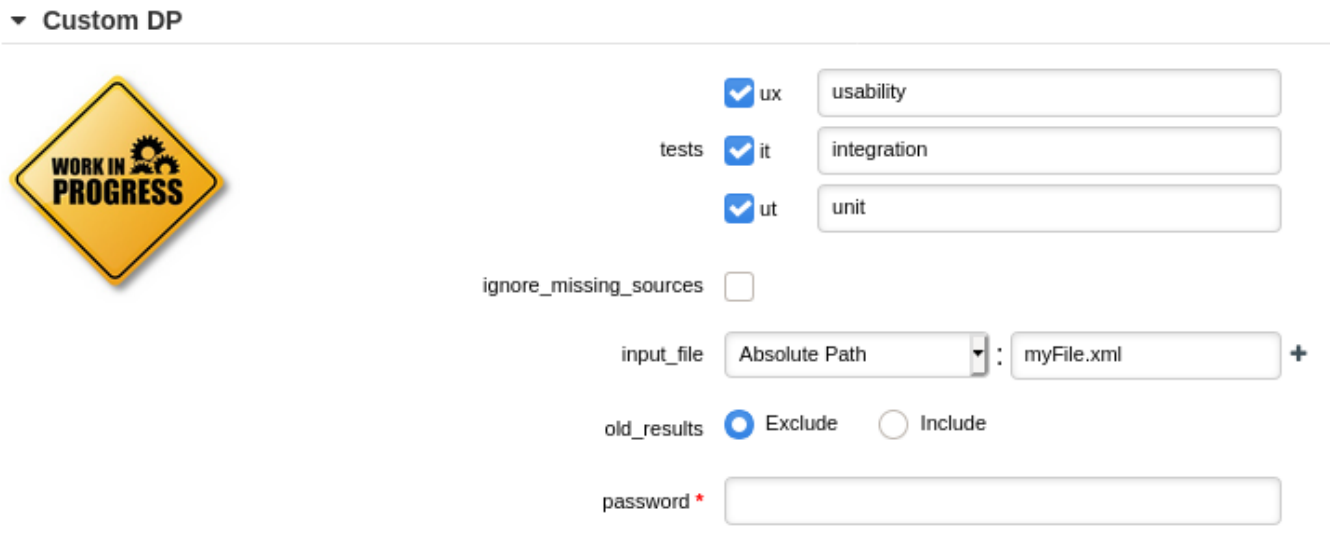
Read on to learn about how to configure your Data Provider, make it available in the web interface, and then understand how to implement the scripted part of a Data Provider that is executed during an analysis.

After you understand how to build a a Data Provider using a *form.xml* file, you can apply this knowledge to building Repository Connectors and Export Definitions, as described in Creating Repository Connectors and Creating Export Definitions.

You can find the XML schema for *form.xml* in form.xsd.

Defining Data Provider Parameters

A Data Provider's parameters are defined in a file called *form.xml*. The following is an example of *form.xml* for a Data Provider extending the GenericPerl framework:



CustomDP parameters

```
<?xml version="1.0" encoding="UTF-8"?>
<tags baseName="GenericPerl" needSources="true" image="CustomDP.png"
projectStatusOnFailure="ERROR">
    <tag type="multipleChoice" displayType="checkbox" key="tests" optionTitle=" ">
        <value key="ux" option="usability"/>
        <value key="it" option="integration"/>
        <value key="ut" option="unit"/>
    </tag>
    <tag type="booleanChoice" key="ignore_missing_sources" defaultValue="false" />
    <tag type="file" key="input_file" defaultValue="myFile.xml" multi="true"/>
    <tag type="multipleChoice" key="old_results" style="margin-left:10px" displayType
="radioButton" defaultValue="Exclude">
        <value key="Exclude" />
        <value key="Include" />
    </tag>
    <tag type="text" key="java_path" defaultValue="/usr/bin/java" hide="true" />
    <tag type="password" required="true" key="password" />
</tags>
```

The **tags** element accepts the following attributes:

- **baseName (mandatory if you are not using an exec-phase)** indicates on which framework you are basing this Data Provider. The value of this attribute must match a folder from the *addons* folder of your installation.
- **needSources (optional, default: false)** allows specifying whether the Data Provider requires sources or not. When set to true, an error will be displayed if you try to select this Data Provider without adding any Repository Connector location to your project.
- **image (optional, default: none)** allows displaying a logo in the web UI for the Data Provider
- **projectStatusOnFailure (optional, default: ERROR)** defines what status the project ends in when this Data Provider produces an error. The following values are allowed:
  - **IGNORE**
  - **WARNING**
  - **ERROR**
- **projectStatusOnWarning (optional, default: WARNING)** defines what status the project ends in when this Data Provider produces a warning. The following values are allowed:
  - **IGNORE**
  - **WARNING**
  - **ERROR**

Each **tag** element is a Data Provider option and allows the following attributes:

- **key (mandatory)** is the option's key that will be passed to the perl script, or can be used to specify the parameter's value from the command line
- **type (mandatory)** defines the type of the parameter. The following values are accepted:
  - **text** for free text entry
  - **file** for file path with native permission and validity checks
  - **directory** for directory path with native permission and validity checks
  - **file_or_directory** for file or directory path with native permission and validity checks

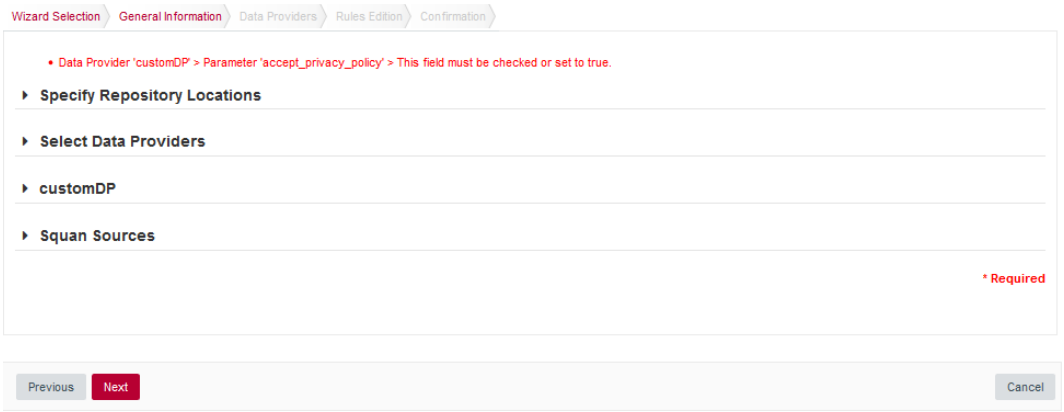
- **password** for password fields
- **booleanChoice** for a boolean
- **multipleChoice** for offering a selection of predefined values

Predefined values are specified with a `value` element with a mandatory `key` attribute and an optional `option` attribute that allows modifying the value of the option from the UI. The input field for each `option` attribute is only displayed if the parent `tag` contains an `optionTitle` attribute.

- `displayType` **(optional)** allows specifying how to display a `multipleChoice` parameter by using one of:
  - **comboBox**
  - **radioButton**
  - **checkbox**
- `multi` **(optional, default: false)** allows for dynamic addition/removal of multiple files or directories path
- `defaultValue` **(optional, default: empty)** is the value used for the parameter when not specified
- `hide` **(optional, default: false)** allows hiding a parameter from the web UI, which is useful when combining it with a default value
- `changeable` **(optional, default: true)** allows making a parameter configurable only when creating the project but read-only for following analyses when set to true
- `style` **(optional, default: empty)** allows setting basic css for the attribute in the web UI
- `required` **(optional, default: false)** allows showing a red asterisk next to the field in the web UI to make it visibly required

You can use a required `tag` of type **booleanchoice** to ensure that users must check a box in the web UI or set its value to *true* when building from the command line in order to proceed with the analysis.

```
<tag type="booleanChoice" required="true" key="
accept_privacy_policy" />
```



Clicking the **Next** *button without checking a required checkbox displays an error*

Hiding your Data Provider elements in the web UI

You can associate to your tag element the displayIf element:

The `displayIf` (optional)* This element allows the user to define conditions on the tagged field to

make it visible in the web UI.

The `displayIf` element accepts logical conditions. These conditions are designed as containers that can contains the following elements:

- `and` **(optional, applied by default)** all conditions defined in the "and" container must be true in order to display tagged items of the data-provider
- `or` **(optional)** one condition defined in the "or" container must be true in order to hide tagged items of the data-provider

The `displayIf` elements and conditionnal containers can accepts the following elements:

- `equals` **(optional)** this element is associated to a tag element defined in the "key" and "value" attributes. The tagged element must contains the value specified in the value attribute in order to be displayed in the web UI
- `notEmpty` **(optional)** the tag element defined in the "key" attribute has to be filed in order to display the data-provider element in the web UI

You can use the displayIf condition in a `tag` element in order to display the tagged field following conditions you have defined.

Syntax example:

```
    <tag type="text" key="config_file" hide="true"/>
    <tag type="text" key="url" required="true" >
        <displayIf>

            <notEmpty key="max_results" />

        </displayIf>
    </tag>
    <tag type="text" key="api_token" required="true" >
        <displayIf>
            <or> <!-- Conditionnal containers can be stacked  -->
                <equals key="max_results" value="100"/>
                <notEmpty key="url" />
            </or>
        </displayIf>
    </tag>
    <tag type="text" key="max_results" required="true" defaultValue="
50" />
```

Localising your Data Provider

In order to display your Data Provider parameters in different languages in the web UI, yout Data Provider's *form.xml* does not contain any hard-coded strings. Instead, Squore uses each parameter's `key` attribute to dynamically retrieve a translation from a *form_xx.properties* file located next to *form.xml*.

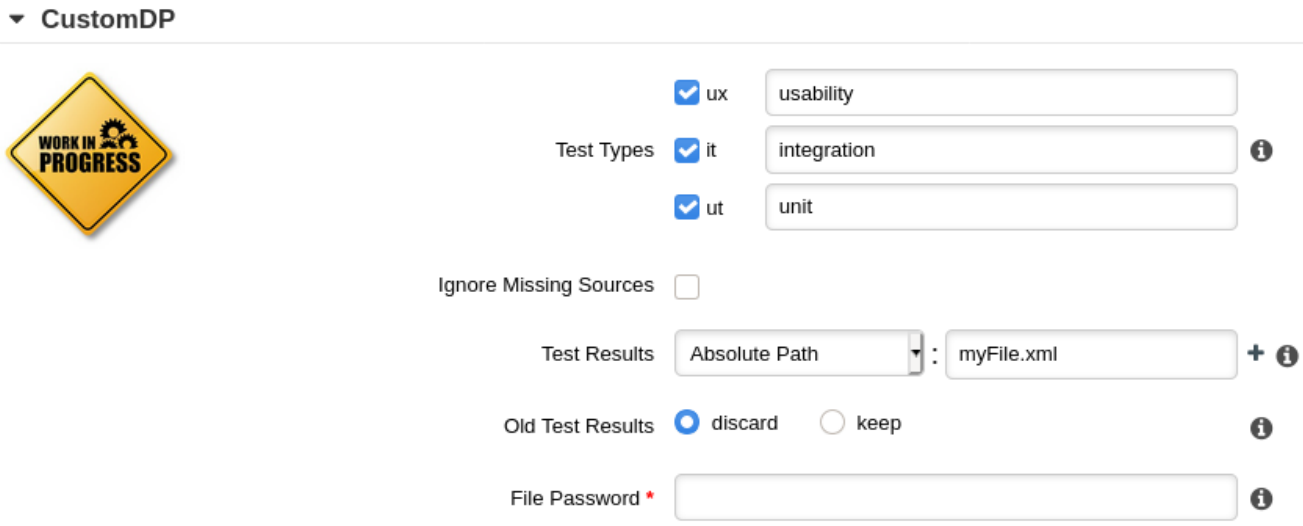
When you create a Data Provider, it is mandatory to include at least an English version of the strings in a file called *form_en.properties*. You are free to add other languages as needed. Here is a sample *.properties* for for the CustomDP you created in the previous section:

```
FORM.GENERAL.NAME = CustomDP
FORM.DASHBOARD.NAME = Test Status
FORM.GENERAL.DESCR = CustomDP imports test results for my project
FORM.GENERAL.URL  = http://example.com/CustomDP

TAG.tests.NAME = Test Types
TAG.tests.DESCR = Check the boxes next to the types of test results contained in the
results

TAG.ignore_missing_sources.NAME = Ignore Missing Sources

TAG.input_file.NAME = Test Results
TAG.input_file.DESCR = Specify the absolute path to the file containing the test
results

TAG.old_results.NAME = Old Test Results
TAG.old_results.DESCR = If the previous analysis contained results that are not in
this results file, what do you want to do with the old results?
OPT.Exclude.NAME = discard
OPT.Include.NAME = keep

TAG.password.NAME = File Password
TAG.password.DESCR = Specify the password to decrypt the test results file
```

The syntax for the *.properties* file is as follows:

- **FORM.GENERAL.NAME** is the display name of the Data Provider in the project wizard
- **FORM.DASHBOARD.NAME** is the display name of the Data Provider in the Explorer
- **FORM.GENERAL.DESCR** is the description displayed in the Data Provider's tooltip in the web UI
- **FORM.GENERAL.URL** is a reference URL for the Data Provider. Note that it is not displayed in ther web UI yet.
- **TAG.tag_name.NAME** allows setting the display name of a parameter
- **TAG.tag_name.DESCR** is a help text displayed in a tooltip next to the Data Provider option in the web UI
- **OPT.option_name.NAME** allows setting the display name of an option

Using the *form_en.properties* above for CustomDP results in the following being displayed in the web UI when launching an analysis:



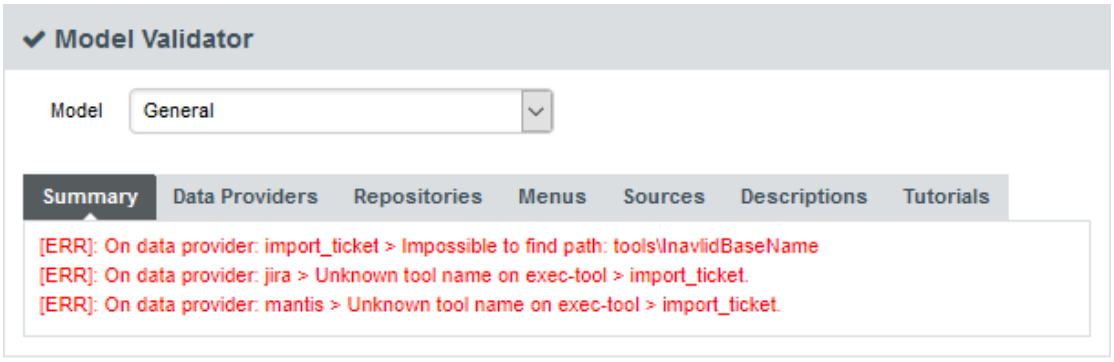
CustomDP pulling translations from a .properties file

Not all wizards display all Data Providers by default. If your Data Provider does not appear after refreshing your configuration, make sure that your wizard bundle allows displaying all Data Providers by reviewing the `tools` element of *Bundle.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>
<Bundle>
    <Wizard ... >
        ...
        <tools all="true">
            ...
        </tools>
        ...
    </Wizard>
</Bundle>
```

For more information about the wizard bundle, consult the the chapter called "Project Wizards" in the Configuration Guide.

If you have made this change and your Data Provider still does not appear in your wizard, consult the Validator to find out if it was disabled because of an error in its configuration.



The General section of the Validator shows errors in your Data Providers

Running your Data Provider

Now that you have a new Data Provider available in the web interface (and the command line), this section will show you how to use these parameters and pass them to one or more scripts or executables in order to eventually write data in the format that Squore expects to import during the analysis.

At the end of a Data Provider execution, Squore expects a file named *input-data.xml* to be written in a specific location. The syntax of the XML file to generate is as follows:

```
<!-- input-data.xml syntax -->
<bundle version="2">
    <artifact [local-key=""] [local-parent=""|parent=""] >
        <artifact [id="<guid-stable-in-time-also-used-as-a-key>"] name="Component"
type="REQ" [location=""] >
            <info name|n="DESCR" value="The description of the object"/>
            <key value="3452-e89b-ff82"/>
            <metric name="TEST_KO" value="2"/>
            <finding name="AR120" loc="xxx" p0="The message" />
            <link name="TEST" local-src=""|src=""|local-dst=""|dst="" />
                <artifact id="" name="SubComponent" type="REQ">
                ...
                </artifact>
        </artifact>
    </artifact>

    <artifact id="" local-key="" name="" type="" local-parent=""|parent="" [location=
""] />
    ...

    <link name="" local-src=""|src="" local-dst=""|dst="" />
    ...

    <info local-ref=""|ref="" name="" value=""/>
    ...

    <metric local-ref=""|ref="" name="" value=""/>
    ...

    <finding local-ref=""|ref="" [location=""] p0="" />
    <finding local-ref=""|ref="" [location=""] p0="">
        <location local-ref=""|ref="" [location=""] />
        ...
        <relax status="RELAXED_DEROGATION|RELAXED_LEGACY|RELAXED_FALSE_POSITIVE"
><![CDATA[My Comment]]></relax>
    </finding>
    ...
</bundle>
```

You can find the XML schema for *input-data.xml* in input-data-2.xsd.

Your Data Provider is configured by adding an `exec-phase` element with a mandatory `id="add-data"` attribute in *form.xml*.

The basic syntax of an `exec-phase` can be seen below:

```
<exec-phase id="add-data">
    <exec name="tcl|perl|java" | executable="/path/to/bin" | executable=
"executable_name" failOnError="true|false" failOnStdErr="true|false" warn="[WARN]"
error="[ERROR|ERR]" fatal="[FATAL]">
        <arg value="${<function>(<args>)}"/>
        <arg value="-freeText" />
        <arg value="${<predefinedVars>}" />
        <arg value="versions" />
        <arg value="-myTag"/>
        <arg tag="myTag"/>
        <env key="MY_VAR" value="SOME_VALUE"/>
    </exec>
    <exec ... />
    <exec-tool name="another_data_provider">
        <param key="<tagName>" value="<value>" />
        <param key="<tagName>" tag="<tag>" />
        <param ... />
    </exec-tool>
    <exec-tool ... >
        ...
    </exec-tool>
</exec-phase>
```

You can also use Groovy in order to configure your Data Provider.

The basic syntax of a Groovy `exec name` is indicated below:

```
<exec name="java">
            <arg value="${javaClasspath(poi,groovy,jackson)}"/>
            <arg value="groovy.lang.GroovyShell" />
            <arg value="${getConfigFile(to_excel.groovy)}"/>
            <arg value="${getSharedAddonsFile(GroovyScriptUtils.groovy)}"/>
...
```

Only the `exec name` section is different. The syntax of the others sections of your Data Provider is still the same.

Executables

The `exec-phase` element accepts one or more launches of scripts or executables specified in an `exec` child element, that can receive arguments and environment variables specified via `arg` and `env` elements.

There are four built-in languages for executables:

- **tcl**

- **perl**
- **java**
- **Groovy**

The scripts are launched using the tcl, perl, or java runtimes defined in your Squore installation. This is also the case for Groovy, which is handled by Java engine.

The following attributes of the `exec` element allow you to control error handling:

- `failOnError` **(optional, default: true)** marks the Data Provider execution as failed if the executable returns an error code
- `failOnStdErr` **(optional, default: true)** marks the Data Provider execution as failed if the executable prints something to stdErr during the execution
- `warn` , `error` **and** `fatal` **(optional, default: see code block above**) allow you to define patterns to look for in the executable's standard output to fine-tune the result of the execution.

Other executables can be called, as long as they are available on the system's PATH, or configured in *config.xml*

Given the following *config.xml*:

```
<!-- config.xml (server or cli) -->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<squore type="server" version="1.3">
    <paths>
        <path name="python" path="C:\Python\python.exe" />
        <path name="git" path="C:\Git\bin\git.exe" />
    </paths>
...
</squore>
```

git and python can be called in your Data Provider as follows:

```
<exec-phase id="add-data">
    <exec name="git">
        ...
    </exec>
    <exec name="python">
        ...
    </exec>
</exec-phase>
```

Arguments

Argument values can be:

1. Free text passed in a `value` tag, useful to specify a parameter for your script

```
<exec name="perl">
    <arg value="-V" />
</exec>
```

2. A tag key declared in *form.xml* passed as a `tag ` attribute to retrieve the input specified by the user. If no input was specified, you can define a `defaultValue`:

```
<arg tag="maxValue" defaultValue="50" />
<arg tag="configFile" defaultValue="${getConfigFile(default.xml)}" />
```

3. One of the predefined functions
   - **${getOutputFile(<relative/path/to/file>,<abortIfMissing>)}** returns the absolute path of an *input-data.xml* file output by an `exec-phase` . *failIfMissing* is an optional boolean which aborts the execution when set to **true** if the file is missing.
   - **${getTemporaryFile(<relative/path/to/file>)}** returns the absolute path of a temporary file created by an `exec` (only for *add-data* and *repo-add-data* phases)
   - **${getAddonsFile(<relative/path/to/file>)}** returns the absolute path of a file in the Data Provider's addons folder
   - **${getConfigFile(<relative/path/to/file>)}** returns the absolute path of a file in the Data Provider's configuration folder
   - **${getSharedAddonsFile(<relative/path/to/file>)}** returns the absolute path of a file in Data Provider's addons/shared folder, if not returns the absolute path of a file in addons/shared folder
   - **${path(<executable_name>)}** returns the absolute path of an executable configured in *config.xml*, or just the executable name if the executable is available from the system's PATH.

     ```
     <exec name="...">
         <arg value="-git_path" />
         <arg value="${path(git)}" />
     ```

   - **${javaClasspath(poi,groovy,jackson,abc.jar,xyz.jar)}** adds the specified list of jars to the classpath for java execution.

     Squore will look for the jars in the *addons/lib* folder of your configuration and return a classpath parameter for the desired runtime environment (*-cp="..." for java*)

poi is a shortcut for *poi-ooxml-3.17.jar,poi-3.17.jar,poi-ooxml-schemas-3.17.jar,xmlbeans-2.6.0.jar,commons-collections4-4.1.jar* and configures the environment necessary to use Apache POI when creating custom Export Definitions, as described in Creating Export Definitions.

groovy is a shortcut for groovy-3.0.1.jar, groovy-json-3.0.1.jar and groovy-xml-3.0.1.jar libraries needed to run Groovy scripts

jackson is a shortcut for jackson-core-2.6.3.jar, jackson-databind-2.6.3.jar and jackson-annotations-2.6.0.jar libraries needed to parse Json file

4. One of the predefined variables
   - **${tmpDirectory}** to get an absolute path to a temp folder to create files

- **${sourcesList}** to get a list of the aliases and locations containing the data extracted by the repository connectors used in the analysis
- **${outputDirectory}** to get the absolute path of folder where the Data Provider needs to write the final *input-data.xml*

Conditions

You can use condition statements in the `exec` and `exec-tool` elements in order to parametrize the execution of your Data Providers. The `execute-if` element is used as follow :

```
<exec-phase id="add-data">
    <exec name="java">
        <executeIf>
            <equals key="outputFile" value="" />   <!-- Execute this Java process only
if the output file is not provided. -->
        </executeIf>
        ...
    </exec>
</exec-phase>
```

The `execute-if` element uses the same syntax as the `displayIf` element : Hiding your Data Provider elements in the web UI.

Calling Other Data Providers

You can call and pass parameters to other Data Providers after your `exec-phase` using an `exec-tool` element. The `exec-tool` element uses a mandatory `name` which is the name of the folder containing the other Data Provider to launch in your configuration folder and supports passing the parameters expected by the other Data Provider via one or more `param` elements where:

- `key` is the name of the parameter expected by the other Data Provider (as defined in its *form.xml*)
- `value` allows passing free text
- `tag` allows passing the value of your own Data Provider's tag value to the other Data Provider and can be combined with a `defaultValue` attribute in case no value was specified by the user for the tag

As an example, the following Data Provider generates a CSV file that is then passed to the pep8 Data Provider:

```
<exec-phase id="add-data">
    <exec name="python">
        <arg value="consolidate-reports-recursive.py" />
        <arg value="-folders" />
        <arg tag="root_folder" />
        <arg value="-outputFile" />
        <arg value="output.csv" />
    </exec>
    <exec-tool name="pep8">
        <param key="csv" value="${getOutputFile(output.csv)}" />
        <param key="separator" tag="separator" defaultValue=";" />
    </exec-tool>
</exec-phase>
```

In this other example, a perl script is launched to retrieves issues from a ticketing system and the export data is passed to the **import_ticket** Data Provider:

```
<exec-phase id="add-data">
    <exec name="perl">
        <arg value="${getConfigFile(export_ticket.pl)}" />
        <arg value="-url" />
        <arg tag="url" />
        <arg value="-login" />
        <arg tag="login" />
        <arg value="-pwd" />
        <arg tag="pwd" />
        <arg value="-outputFile" />
        <arg value="${getOutputFile(exportdata.csv,false)}" />
    </exec>
    <exec-tool name="import_ticket">
        <param key="input_file" value="${getOutputFile(exportdata.csv)}" />
        <param key="csv_separator" value=";" />
    </exec-tool>
</exec-phase>
```

If your Data Provider uses a perl script, Squore provides a small library that makes it easy to retrieve script arguments called **SQuORE::Args**. Using it as part of your script, you can retrieve arguments using the **get_tag_value()** function, as shown below:

```
# name: export_ticket.pl
# description: exports issues to a CSV file
use SQuORE::Args;
# ...
# ...
my $url = get_tag_value("url");
my $login = get_tag_value("login");
my $pwd = get_tag_value("pwd");
my $outputFile = get_tag_value("outputFile");
# ...
exit 0;
```

Using the Squore toolkit

If you want your Data Provider to use the Squore toolkit to retrieve references to artefacts, the following variables are available (in the *add-data* and *repo-add-data* phases only):

- **${tclToolkitDirectory}:** the directory of the toolkit tcl code to execute
- **${squanOutputDirectory}:** the directory of containing the results of the execution of Squan Sources

In order to use the toolkit, your `exec` must use the tcl language. As an example, here is a sample `exec-phase` and associated tcl file to get you started:

```
 <!-- form.xml -->
<exec-phase id="repo-add-data">
    <exec name="tcl">
        <arg value="${getAddonsFile(repo-add-data.tcl)}" />
        <arg value="${tclToolkitFile}" />
        <arg value="${squanOutputDirectory}" />
        <arg value="${outputDirectory}" />
        <arg tag="xxx" />
    </exec>
</exec-phase>
```

```
#repo-add-data.tcl:
set toolkitFile [lindex $argv 0]
set sqOutputDir [lindex $argv 1]
set outputDir [lindex $argv 2]
set xxx [lindex $argv 3]

# Initialise the toolkit
puts "Initializing toolkit"
source $toolkitFile
toolkit::initialize $sqOutputDir $outputDir

# Execute your code
puts "Main execution"
# yout code here
# ...

# Generate xml files (artefacts)
puts "Generating xml files"
toolkit::generate $outputDir {artefacts}
```

Finding More Examples

If you want to find more examples of working Data Providers that use this syntax, check the following Data Providers in Squore's default configuration folder:

- **conf-checker** calls a jar file to write an XML file in Squore's exchange format
- **import_ticket** parses a file to translate it into a format that can then be passed to **csv_import** to import the tickets into Squore
- **jira** retrieves data from Jira and passes it to **import_ticket**

Built-in Data Provider Frameworks

In order to help you import data into Squore, the following Data Provider frameworks are provided and can write a valid *input-data.xml* file for you:

1. **csv_import**

   The csv_import framework allows you to write Data Providers that produce CSV files and then pass them on to the framework to be converted to an XML format that Squore understands. This framework allows you to import metrics, findings, textual information and links as well as generate your own artefacts. It is fully linked to the source code parser and therefore allows to locate existing source code artefacts generated by the source code parser. Refer to the full csv_import Reference for more information.

2. **xml**

   The xml framework is a sample implementation of a Data Provider that allows you to directly import an XML file or run it through an XSL transformation to that it matches the input format expected by Squore (input-data.xml). This framework therefore allows you to import metrics, findings, textual information and links as well as generate your own artefacts. Refer to the full xml Reference for more information.

If you are looking for the legacy Data Provider frameworks from previous versions of Squore, consult Legacy Frameworks.

The legacy Data Provider frameworks are still supported, however using the new frameworks is recommended for developping new Data Providers, as they are more flexible and provide more functionality to interact with source code artefacts.

Creating Repository Connectors

The same syntax used to create Data Providers can be used to create Repository Connectors, and therefore instruct Squore to get source code from SCMs. Instead of using an `exec-phase` with the `id="add-data"`, your Repository Connector should define the following phases:

- `id="import"` defines how you extract source code and make it available to Squan Sources so it can be analysed. This phase is expected to return a path to a folder containing the sources to analyse or a *data.properties* file listing the path to the folder containing source and various other properties to be used in other executions:

  ```
  directory=/path/to/sources-to-analyse
  data.<key1>=<value1>
  data.<key2>=<value2>
  ```

  This phase is executed once per source code node in the project and allows you to use the following additional variables: **${outputSourceDirectory} is the folder containing the sources to analyse** **${alias}** is the alias used for the source code node (empty if there is only one source code node)

- `id="repo-add-data"` is similar to the *add-data* phase described for Data Providers in Running your Data Provider and is expected to produce an *input-data.xml*. The only difference in the case of a Repository Connector is that this phase is executed once per source code node in the analysis.
- `id="display"` is the phase that is called when users request to view the source code for an artefact from the web UI. This phase is expected to return a *data.properties* file with the following keys:

  ```
  filePath=/path/to/source/file
  displayPath=<Artefact Display Path (optional)>
  ```

  The contents of *filePath* will be loaded in the source code viewer, while the value of *displayPath* will be used as the file path displayed in the header of the source code viewer.

  This phase allows you to use the following additional variables:

- **${scaInfo}** is text to display in the title bar of the source code viewer in the web interface
- **${artefactName}** is the name of the file to display
- **${artefactPath}** is the path (without the alias) of the file to display

  During the **display** phase, you can retrieve any data set during the **import** phase for the repository using the **${getImportData(<key1>)}** function

Additional attributes are available for the **`tags`** element of a Repository Connector:

- **`deleteTmpSrc`** **(optional, default: false)** indicates whether or not the content of **sources** folder

coming from this Repository Connector will be deleted upon Squore Server restart.

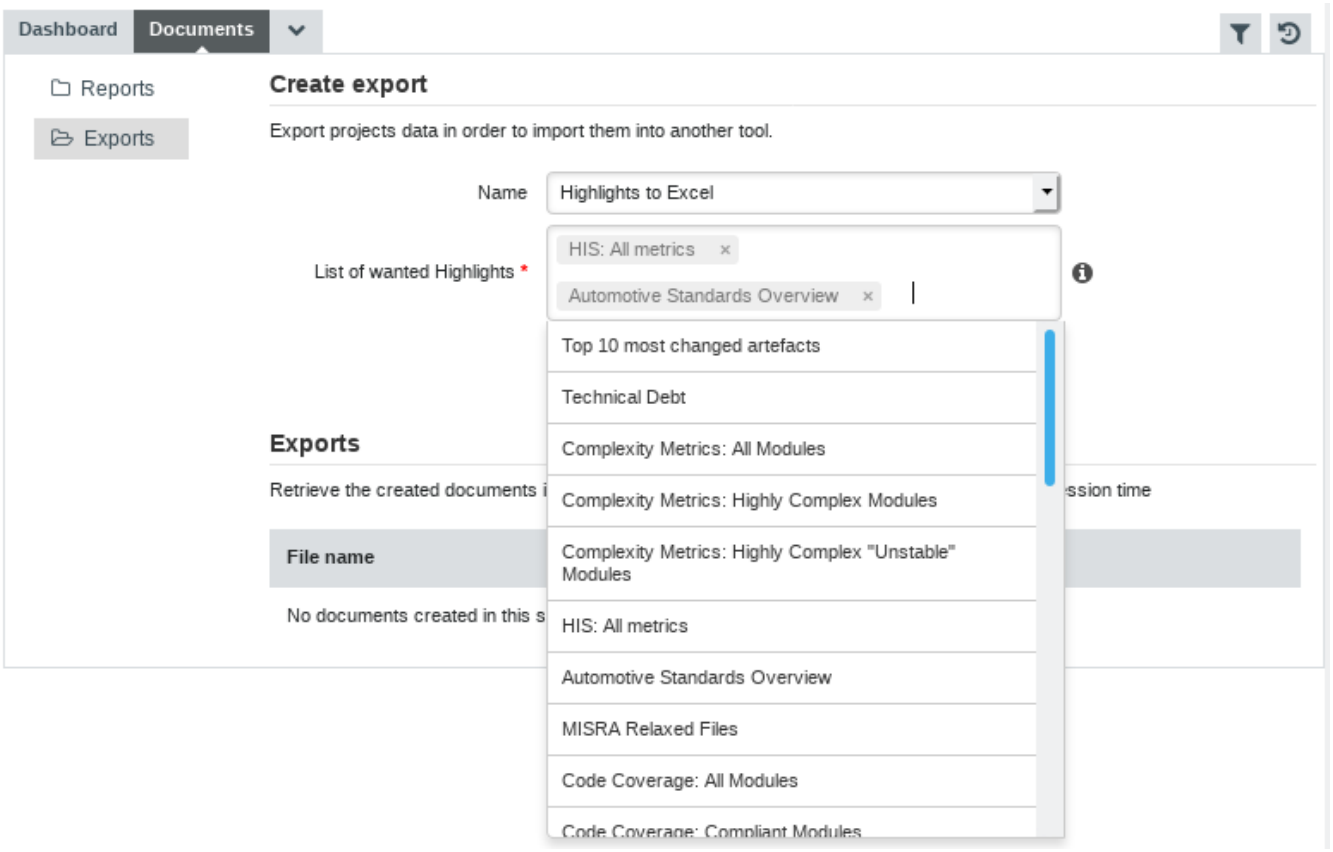
- `useCredentialsForSCA` **(optional, default: true)** allows specifying whether credentials dialog will be prompted in View Source Code or not.

Consult SVN's *form.xml* in *<SQUORE_HOME>/configuration/repositoryConnectors/SVN* for a working example of a Repository Connector that uses all the phases described above.

Please note, as data-provider, you can use the <exec-tool> parameter in order to call other elements while processing, like Data Provider or Scripts. For more informations about <exec-tool> parameter, please refer to Running your Data Provider.

Creating Export Definitions

The *form.xml* specification can also be used to create Export Definitions that allow users to export data based on one or more highlight categories from the web interface.



The **Highlights to Excel** *Export Definition*

The **Highlights to Excel** Export Definition uses the following *form.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>
<tags>
    <tag type="multipleChoice" displayType="multi-autocomplete" required="true" key=
"highlights">
        <values type="highlights" />
    </tag>

    <exec-phase id="export">
        <exec name="java">
```

```
            <arg value="${javaClasspath(poi,groovy,jackson)}"/>
            <arg value="groovy.lang.GroovyShell" />
            <arg value="${getConfigFile(to_excel.groovy)}"/>
            <arg value="${getSharedAddonsFile(GroovyScriptUtils.groovy)}"/>

            <arg value="-importScript"/>
            <arg value="${getSharedAddonsFile(exports_utils.groovy)}" />

            <arg value="-squoreApiUtils"/>
            <arg value="${getSharedAddonsFile(SquoreApiUtils.groovy)}" />

            <arg value="-excelUtilsScript"/>
            <arg value="${getSharedAddonsFile(ExcelUtils.groovy)}" />

            <arg value="-highlights"/>
            <arg tag="highlights" />

            <arg value="-outputDirectory" />
            <arg value="${outputDirectory}" />

            <arg value="-idArtefact" />
            <arg value="${idArtefact}"/>

            <arg value="-idVersion"/>
            <arg value="${idVersion}"/>

            <arg value="-idModel"/>
            <arg value="${idModel}"/>

            <arg value="-group"/>
            <arg value="${group}"/>

            <arg value="-serverUrl"/>
            <arg value="${localUrl}"/>

            <arg value="-token"/>
            <arg value="${token}"/>

            <arg value="-template"/>
            <arg value="${getConfigFile(template.xlsx)}" />
        </exec>
    </exec-phase>
</tags>
```

Data is exported from the server as a JSON file, which your Export Definition can modify as needed before sending it to the end-user who launched the export. You can consult the format of the JSON file in the **Data Exchange Formats** appendix for more information.

In order to create an Export Definition, the syntax described in Defining Data Provider Parameters and Running your Data Provider is augmented to include the extra additional capabilities:

1. A **multi-autocompletedisplayType** for multipleChoice `tag` elements.

   The `tag` element accepts a `values` sub-element with a mandatory `type` attributes. When set to **highlights**, the widget automatically displays all the available highlight definitions for the currently selected artefact.

2. A mandatory `exec-phase` with `id="export"` that contains one or more `exec`s.

   This `exec-phase` is expected to return a data.properties file with the following keys:

   ```
   filename=/path/to/export/file
   ```

3. Variables that can be used in the `exec-phase` to pass the context of the currently selected artefact to the Export Definition:
   - **${idUser}** is the ID of the user generating the export
   - **${token}** is the auto-generated token for on the fly authentification to the API REST
   - **${idArtefact}** is the ID of the currently selected artefact
   - **${idVersion}** is the ID of the version of the project that is currently selected
   - **${idApplication}** is the ID of the project that currently selected
   - **${idModel}** is the ID of the analysis model used for the project that is currently selected
   - **${group}** is the path of the current selected group portfolio
   - **${serverUrl}** is the Squore Server URL, as defined in **Administration > System**
   - **${localUrl}** is the Squore Local URL

You can add your own Export Definition by following these steps:

1. Create a folder in *configuration/exports* called *my_export_definition*.
2. Create a *form.xml* and *form_en.properties* in *my_export_definition*
3. Define the `exec-phase` that your Export Definition will run
4. Add your Export Definition to your model's Export Bundle for the desired project role and artefact type, using the folder name (*my_export_definition*) as the `ExportDef`'s `name` attribute:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <Bundle>
   <Role name="DEFAULT">
       <Export type="...">
           <ExportDef name="my_export_definition" />
           ...
       </Export>
       ...
   </Role>
   </Bundle>
   ```

5. Reload the Squore configuration and your Export Definition should appear in the Documents tab of the Explorer.

For more examples of custom Export Definitions, consult the *configuration/exports* and *addons/exports* folders of the default Squore configuration.

Please note, as data-provider, you can use the <exec-tool> parameter in order to call other elements while processing, like Data Provider or Scripts. For more informations about <exec-tool> parameter, please refer to Running your Data Provider.

Appendix A: Man Pages

install(2)

support@vector.com Vector Informatik GmbH

NAME

install - installation script

SYNOPSIS

install [-v] [-s server_url] [-u user] [-p password] [options ...]

DESCRIPTION

Installs and configures Squore CLI.

The most common options when installing Squore CLI are -s, -u and -p, to configure the server URL, user and password used to connect to the server. These details will be stored on the machine so that the password does not have to be passed again on the command line for this user account. The -N option disables the automatic synchronisation of the configuration folders with the server at the end of the installation. This can also be launched manually later on if needed.

OPTIONS

-s server_url

The URL of Squore Server that Squore CLI will connect to after installation. (default: *http://localhost:8180/SQuORE_Server*)

-u user

The username to use to connect to Squore Server. (default: *demo*)

-p password

The username to use to connect to Squore Server. (default: *demo*)

-N

Do not synchronise client with server

-v

Turn on verbose mode

BUGS

Contact support at support@vector.com or report bugs at https://portal.vector.com/.

RESOURCES

Full documentation is available at https://support.squoring.com/documentation/latest.

COPYRIGHT

Appendix B: Data Provider Frameworks

Current Frameworks

The following Data Provider frameworks support importing all kinds of data into Squore. Whether you choose one or the other depends on the ability of your script or executable to produce CSV or XML data. Note that these frameworks are recommended over the legacy frameworks described in Legacy Frameworks, which are deprecated as of Squore 18.0.

csv_import Reference

```
==============
= csv_import =
==============

The csv_import framework allows you to create Data Providers that produce CSV files
that the framework will translate into XML files that can be imported in your analysts
results. This framework is useful if writing XML files directly from your script is
not practical.

Using csv_import, you can import metrics, findings (including relaxed findings),
textual information, and links between artefacts (including to and from source code
artefacts).
This framework replaces all the legacy frameworks that wrote CSV files in previous
versions.

Note that this framework can be called by your Data Provider simply by creating an
exec-tool phase that calls the part of the framework located in the configuration
folder:
<exec-tool name="csv_import">
    <param key="csv" value="${getOutputFile(output.csv)}" />
    <param key="separator" value=";" />
    <param key="delimiter" value="&amp;quot;" />
</exec-tool>

For a full description of all the parameters that can be used, consult the section
called "CSV Import" in the "Data Providers" chapter of this manual.


==========================================
= CSV format expected by the data provider =
==========================================

- Line to define an artefact (like a parent artefact for instance):
Artefact

- Line to add n metrics to an artefact:
Artefact;(MetricId;Value)*
```

```
- Line to add n infos to an artefact:
Artefact;(InfoId;Value)*

- Line to add a key to an artefact:
Artefact;Value

- Line to add a finding to an artefact:
Artefact;RuleId;Message;Location

- Line to add a relaxed finding to an artefact:
Artefact;RuleId;Message;Location;RelaxStatus;RelaxMessage

- Line to add a link between artefacts:
Artefact;LinkId;Artefact

where:
- MetricId is the id of the metric as declared in the Analysis Model
- InfoId is the id of the information to import
- Value is the value of the metric or the information or the key to import (a key is a
UUID used to reference an artefact)
- RuleId is the id of the rule violated as declared in the Analysis Model
- Message is the message of the finding, which is displayed after the rule description
- Location is the location of the finding (a line number for findings attached source
code artefacts, a url for findings attached to any other kind of artefact)
- RelaxStatus is one of DEROGATION, FALSE_POSITIVE or LEGACY and defines the
relaxation stat of the imported finding
- RelaxMessage is the justification message for the relaxation state of the finding
- LinkId is the id of the link to create between artefacts, as declared in the
Analysis Model

==========================
= Manipulating Artefacts =
==========================

The following functions are available to locate and manipulate source code artefacts
in the project:
- ${artefact(type,path)} ==> Identify an artefact by its type and full path
- ${artefact(type,path,uid)} ==> Identify an artefact by its type and full path and
assign it the unique identifier uid
- ${uid(value)} ==> Identify an artefact by its unique identifier (value)
- ${file(path)} ==> Tries to find a source code file matching the "path" in the
project
- ${function(fpath,line)} ==> Tries to find a source code function at line "line" in
file matching the "fpath" in the project
- ${function(fpath,name)} ==> Tries to find a source code function whose name matches
"name" in the file matching the "fpath" in the project
- ${class(fpath,line)} ==> Tries to find a source code class at line "line" in the
file matching the "fpath" in the project
- ${class(fpath,name)} ==> Tries to find a source code class whose name matches "name"
in the file matching the "fpath" in the project
```

```
Note: In the above definitions if "name" contains either '(' or ',' characters then
the full name has to be between simple quote
e.g. ${function(main.c,'main(int argc, char* argv)')}

===============
= Input Files =
===============

The data provider accepts the following files:
Metrics file accepts:
    Artefact definition line
    Metrics line

Findings file accepts:
    Artefact definition line
    Findings line

Keys file accepts:
    Artefact definition line
    Keys line

Information file accepts:
    Artefact definition line
    Information line

Links file accepts:
    Artefact definition line
    Links line

It is also possible to mix every kind of line in a single csv file, as long as each
line is prefixed with the kind of data it contains.
In this case, the first column must contain one of:
DEFINE (or D): when the line is used to define an artefact
METRIC (or M): to add a metric
INFO (or I): to add an information
KEY (or K): to add a key
FINDING (or F): to add a finding, relaxed or not
LINK (or L): to add link between artefacts

The following is an example of a csv file containing mixed lines:
D;${artefact(CR_FOLDER,/CRsCl)}
M;${artefact(CR,/CRsCl/cr2727,2727)};NB;2
M;${artefact(CR,/CRsCl/cr1010,1010)};NB;4
I;${uid(1010)};NBI;Bad weather
K;${artefact(CR,/CRsCl/cr2727,2727)};#CR2727
I;${artefact(CR,/CRsCl/cr2727,2727)};NBI;Nice Weather
F;${artefact(CR,/CRsCl/cr2727,2727)};BAD;Malformed
M;${uid(2727)};NB_EXT;3
I;${uid(2727)};NBI_EXT;Another Info
F;${uid(2727)};BAD_EXT;Badlyformed
F;${uid(2727)};BAD_EXT1;Badlyformed1;;FALSE_POSITIVE;Everything is in the title]]>
```

```
F;${function(machine.c,41)};R_GOTO;"No goto; neither togo;";41
F;${function(machine.c,42)};R_GOTO;No Goto;42;LEGACY;Was done a long time ago
F;${function(main.c,'main(int argc, char* argv)')};R_GOTO;No Goto;42
L;${uid(1010)};CR2CR;${uid(2727)}
L;${uid(2727)};CR2CR;${uid(1010)}
```

xml Reference

```
=======
= xml =
=======

The xml framework is an implementation of a data provider that allows to import an xml
file, potentially after an xsl transformation. The transformed XML file is expected to
follow the syntax expected by other data providers (see input-data.xml specification).

This framework can be extended like the other frameworks, by creating a folder for
your data provider in your configuration/tools folder and creating a form.xml.
Following are three examples of the possible uses of this framework.

Example 1 - User enters an xml path and an xsl path, the xml is transformed using the
xsl and then imported
=========
<?xml version="1.0" encoding="UTF-8"?>
<tags baseName="xml">
  <tag type="text" key="xml" />
    <tag type="text" key="xslt" />

    <exec-phase id="add-data">
      <exec name="java" failOnError="true" failOnStdErr="true">
        <arg value="${javaClasspath(groovy,xml-resolver-1.2.jar)}"/>
        <arg value="groovy.lang.GroovyShell" />
        <arg value="xml.groovy" />
        <arg value="${outputDirectory}" />
        <arg tag="xml"/>
        <arg tag="xsl" />
      </exec>
    </exec-phase>
</tags>

Example 2 - The user enter an xml path, the xsl file is predefined (input-data.xsl)
and present in the same directory as form.xml
=========
<?xml version="1.0" encoding="UTF-8"?>
<tags baseName="xml">
  <tag type="text" key="xml" />

  <exec-phase id="add-data">
```

```
    <exec name="java" failOnError="true" failOnStdErr="true">
      <arg value="${javaClasspath(groovy,xml-resolver-1.2.jar)}"/>
      <arg value="groovy.lang.GroovyShell" />
      <arg value="xml.groovy" />
      <arg value="${outputDirectory}" />
      <arg tag="xml" />
      <arg value="${getToolConfigDir(input-data.xsl)}" />
    </exec>
  </exec-phase>
</tags>

Example 3 - The user enter an xml path of a file already in the expected format
=========
<?xml version="1.0" encoding="UTF-8"?>
<tags baseName="xml">
  <tag type="text" key="xml" />

  <exec-phase id="add-data">
    <exec name="java" failOnError="true" failOnStdErr="true">
      <arg value="${javaClasspath(groovy,xml-resolver-1.2.jar)}"/>
      <arg value="groovy.lang.GroovyShell" />
      <arg value="xml.groovy" />
      <arg value="${outputDirectory}" />
      <arg tag="xml" />
    </exec>
  </exec-phase>
</tags>
```

Legacy Frameworks

	Import Metrics	Import Textual Information	Import Findings	Import Links	Create Artefacts	Parse Subfolders
CSV	✓	✓	✗	✗	✓	✓
csv_findings	✗	✗	✓	✗	✗	✗
CSVPerl	✓	✓	✗	✗	✓	✓
Generic	✓	✓	✓	✓	✓	✗
GenericPerl	✓	✓	✓	✓	✓	✓
FindingsPerl	✗	✗	✓	✗	✗	✓
ExcelMetrics	✓	✓	✓	✗	✓	✓

✓ Supported

✓ Your Perl script needs to handle subfolder parsing

✗ Not Supported

Legacy Data Provider frameworks and their capabilities

1. **Csv**

   The Csv framework is used to import metrics or textual information and attach them to artefacts of type Application or File. While parsing one or more input CSV files, if it finds the same metric for the same artefact several times, it will only use the last occurrence of the metric and ignore the previous ones. Note that the type of artefacts you can attach metrics to is limited to Application and File artefacts. If you are working with File artefacts, you can let

the Data Provider create the artefacts by itself if they do not exist already. Refer to the full Csv Reference for more information.

2. **csv_findings**

   The csv_findings framework is used to import findings in a project and attach them to artefacts of type Application, File or Function. It takes a single CSV file as input and is the only framework that allows you to import relaxed findings directly. Refer to the full csv_findings Reference for more information.

3. **CsvPerl**

   The CsvPerl framework offers the same functionality as Csv, but instead of dealing with the raw input files directly, it allows you to run a perl script to modify them and produce a CSV file with the expected input format for the Csv framework. Refer to the full CsvPerl Reference for more information.

4. **FindingsPerl**

   The FindingsPerl framework is used to import findings and attach them to existing artefacts. Optionally, if an artefact cannot be found in your project, the finding can be attached to the root node of the project instead. When launching a Data Provider based on the FindingsPerl framework, a perl script is run first. This perl script is used to generate a CSV file with the expected format which will then be parsed by the framework. Refer to the full FindingsPerl Reference for more information.

5. **Generic**

   The Generic framework is the most flexible Data Provider framework, since it allows attaching metrics, findings, textual information and links to artefacts. If the artefacts do not exist in your project, they will be created automatically. It takes one or more CSV files as input (one per type of information you want to import) and works with any type of artefact. Refer to the full Generic Reference for more information.

6. **GenericPerl**

   The GenericPerl framework is an extension of the Generic framework that starts by running a perl script in order to generate the metrics, findings, information and links files. It is useful if you have an input file whose format needs to be converted to match the one expected by the Generic framework, or if you need to retrieve and modify information exported from a web service on your network. Refer to the full GenericPerl Reference for more information.

7. **ExcelMetrics**

   The ExcelMetrics framework is used to extract information from one or more Microsoft Excel files (.xls or .xslx). A detailed configuration file allows defining how the Excel document should be read and what information should be extracted. This framework allows importing metrics, findings and textual information to existing artefacts or artefacts that will be created by the Data Provider. Refer to the full ExcelMetrics Reference for more information.

After you choose the framework to extend, you should follow these steps to make your custom Data Provider known to Squore:

1. Create a new configuration *tools* folder to save your work in your custom configuration folder: *MyConfiguration/configuration/tools*.
2. Create a new folder for your data provider inside the new *tools* folder: **CustomDP**. This folder needs to contain the following files:
   - **form.xml** defines the input parameters for the Data Provider, and the base framework to use, as described in Defining Data Provider Parameters
   - **form_en.properties** contains the strings displayed in the web interface for this Data Provider, as described in Localising your Data Provider

- **config.tcl** contains the parameters for your custom Data Provider that are specific to the selected framework
- **CustomDP.pl** is the perl script that is executed automatically if your custom Data Provider uses one of the *Perl frameworks.

3. Edit Squore Server's configuration file to register your new configuration path, as described in the Installation and Administration Guide.
4. Log into the web interface as a Squore administrator and reload the configuration.

Your new Data Provider is now known to Squore and can be triggered in analyses. Note that you may have to modify your Squore configuration to make your wizard aware of the new Data Provider and your model aware of the new metrics it provides. Refer to the relevant sections of the Configuration Guide for more information.

Csv Reference

```
=======
= Csv =
=======

The Csv framework is used to import metrics or textual information and attach them to
artefacts of type Application, File or Function. While parsing one or more input CSV
files, if it finds the same metric for the same artefact several times, it will only
use the last occurrence of the metric and ignore the previous ones. Note that the type
of artefacts you can attach metrics to is limited to Application, File and Function
artefacts. If you are working with File artefacts, you can let the Data Provider
create the artefacts by itself if they do not exist already.

============
= form.xml =
============

You can customise form.xml to either:
- specify the path to a single CSV file to import
- specify a pattern to import all csv files matching this pattern in a directory

In order to import a single CSV file:
=====================================
<?xml version="1.0" encoding="UTF-8"?>
<tags baseName="Csv" needSources="true">
    <tag type="text" key="csv" defaultValue="/path/to/mydata.csv" />
</tags>

Notes:
- The csv key is mandatory.
- Since Csv-based data providers commonly rely on artefacts created by Squan Sources,
you can set the needSources attribute to force users to specify at least one
repository connector when creating a project.

In order to import all files matching a pattern in a folder:
============================================================
```

```
<?xml version="1.0" encoding="UTF-8"?>
<tags baseName="Csv" needSources="true">
    <!-- Root directory containing Csv files to import-->
    <tag type="text" key="dir" defaultValue="/path/to/mydata" />
    <!-- Pattern that needs to be matched by a file name in order to import it-->
    <tag type="text" key="ext" defaultValue="*.csv" />
    <!-- search for files in sub-folders -->
    <tag type="booleanChoice" defaultValue="true" key="sub" />
</tags>

Notes:
- The dir and ext keys are mandatory
- The sub key is optional (and its value set to false if not specified)


==============
= config.tcl =
==============

Sample config.tcl file:
=======================
# The separator used in the input CSV file
# Usually \t or ;
set Separator "\t"

# The delimiter used in the input CSV file
# This is normally left empty, except when you know that some of the values in the CSV
file
# contain the separator itself, for example:
# "A text containing ; the separator";no problem;end
# In this case, you need to set the delimiter to \" in order for the data provider to
find 3 values instead of 4.
# To include the delimiter itself in a value, you need to escape it by duplicating it,
for example:
# "A text containing "" the delimiter";no problemo;end
# Default: none
set Delimiter \"

# ArtefactLevel is one of:
#      Application: to import data at application level
#      File: to import data at file level. In this case ArtefactKey has to be set
#             to the value of the header (key) of the column containing the file path
#             in the input CSV file.
#      Function : to import data at function level, in this case:
#                 ArtefactKey has to be set to the value of the header (key) of the
column containing the path of the file
#                 FunctionKey has to be set to the value of the header (key) of the
column containing the name and signature of the function
# Note that the values are case-sensitive.
set ArtefactLevel File
set ArtefactKey File
```

```
# Should the File paths be case-insensitive?
# true or false (default)
# This is used when searching for a matching artefact in already-existing artefacts.
set PathsAreCaseInsensitive "false"

# Should file artefacts declared in the input CSV file be created automatically?
# true (default) or false
set CreateMissingFile "true"

# FileOrganisation defines the layout of the input CSV file and is one of:
#     header::column: values are referenced from the column header
#     header::line: NOT AVAILABLE
#     alternate::line: lines are a sequence of {Key Value}
#     alternate::column: columns are a sequence of {Key Value}
# There are more examples of possible CSV layouts later in this document
set FileOrganisation header::column

# Metric2Key contains a case-sensitive list of paired metric IDs:
#     {MeasureID KeyName [Format]}
# where:
#   - MeasureID is the id of the measure as defined in your analysis model
#   - KeyName, depending on the FileOrganisation, is either the name of the column or
the name
#      in the cell preceding the value to import as found in the input CSV file
#   - Format is the optional format of the data, the only accepted format
#      is "text" to attach textual information to an artefact, for normal metrics omit
this field
set Metric2Key {
    {BRANCHES Branchs}
    {VERSIONS Versions}
    {CREATED Created}
    {IDENTICAL Identical}
    {ADDED Added}
    {REMOV Removed}
    {MODIF Modified}
    {COMMENT Comment text}
}


==========================
= Sample CSV Input Files =
==========================

Example 1:
==========
 FileOrganisation : header::column
 ArtefactLevel : File
 ArtefactKey   : Path

Path    Branchs Versions
```

```
./foo.c 15      105
./bar.c 12      58

Example 2:
==========
 FileOrganisation : alternate::line
 ArtefactLevel : File
 ArtefactKey   : Path

Path    ./foo.c Branchs 15  Versions    105
Path    ./bar.c Branchs 12  Versions    58

Example 3:
==========
 FileOrganisation : header::column
 ArtefactLevel : Application

ChangeRequest   Corrected   Open
27              15          11

Example 4:
==========
 FileOrganisation : alternate::column
 ArtefactLevel : Application

ChangeRequest   15
Corrected       11

Example 5:
==========
 FileOrganisation : alternate::column
 ArtefactLevel : File
 ArtefactKey   : Path

Path    ./foo.c
Branchs 15
Versions    105
Path    ./bar.c
Branchs 12
Versions    58

Example 6:
==========
 FileOrganisation : header::column
 ArtefactLevel : Function
 ArtefactKey   : Path
 FunctionKey   : Name

Path    Name    Decisions Tested
./foo.c end_game(int*,int*) 15      3
./bar.c bar(char)   12      6
```

```
Working With Paths:
===================

- Path seperators are unified: you do not need to worry about handling differences
between Windows and Linux
- With the option PathsAreCaseInsensitive, case is ignored when searching for files in
the Squore internal data
- Paths known by Squore are relative paths starting at the root of what was specified
in the repository connector durign the analysis. This relative path is the one used to
match with a path in a csv file.

Here is a valid example of file matching:
  1. You provide C:\A\B\C\D as the root folder in a repository connector
  2. C:\A\B\C\D contains E\e.c then Squore will know E/e.c as a file

  3. You provide a csv file produced on linux and containing
    /tmp/X/Y/E/e.c as path, then Squore will be able to match it with the known file.

Squore uses the longest possible match.
In case of conflict, no file is found and a message is sent to the log.
```

csv_findings Reference

```
================
= csv_findings =
================

The csv_findings data provider is used to import findings (rule violations) and attach
them to artefacts of type Application, File or Function.
The format of the csv file given as parameter has to be:

FILE;FUNCTION;RULE_ID;MESSAGE;LINE;COL;STATUS;STATUS_MESSAGE;TOOL

where:
=====
FILE : is the full path of the file where the finding is located
FUNCTION : is the name of the function where the finding is located
RULE_ID : is the Squore ID of the rule which is violated
MESSAGE : is the specific message of the violation
LINE: is the line number where the violation occurs
COL: (optional, leave empty if not provided) is the column number where the violation
occurs
STATUS: (optional, leave empty if not provided) is the staus of the relaxation if the
violation has to be relaxed (DEROGATION, FALSE_POSITIVE, LEGACY)
STATUS_MSG: (optional, leave empty if not provided) is the message for the relaxation
when relaxed
TOOL: is the tool providing the violation
```

```
The header line is read and ignored (it has to be there)
The separator (semicolon by default) can be changed in the config.tcl file (see below)
The delimiter (no delimiter by default) can be changed in the config.tcl (see below)

==============
= config.tcl =
==============

Sample config.tcl file:
=======================
# The separator used in the input CSV file
# Usually ; or \t
set Separator \;

# The delimiter used in the CSV input file
# This is normally left empty, except when you know that some of the values in the CSV
file
# contain the separator itself, for example:
# "A text containing ; the separator";no problem;end
# In this case, you need to set the delimiter to \" in order for the data provider to
find 3 values instead of 4.
# To include the delimiter itself in a value, you need to escape it by duplicating it,
for example:
# "A text containing "" the delimiter";no problemo;end
# Default: none
set Delimiter \"

# You can add some patterns to avoid new findings when some strings in the finding
message changes
# i.e. Unreachable code Default switch clause is unreachable. switch-expression at
line 608 (column 12).
# In this case we do not want the line number to be part of the signagture of the
finding,
# to achieve this user will add a pattern as shown below (patterns are TCL regex
patterns):
lappend InconstantFindingsPatterns {at line [0-9]+}
```

CsvPerl Reference

```
===========
= CsvPerl =
===========

The CsvPerl framework offers the same functionality as Csv, but instead of dealing
with the raw input files directly, it allows you to run a perl script to modify them
and produce a CSV file with the expected input format for the Csv framework.
```

```
============
= form.xml =
============

In your form.xml, specify the input parameters you need for your Data Provider.
Our example will use two parameters: a path to a CSV file and another text parameter:

<?xml version="1.0" encoding="UTF-8"?>
<tags baseName="CsvPerl" needSources="true">
    <tag type="text" key="csv" defaultValue="/path/to/csv" />
    <tag type="text" key="param" defaultValue="MyValue" />
</tags>

- Since Csv-based data providers commonly rely on artefacts created by Squan Sources,
you can set the needSources attribute to force users to specify at least one
repository connector when creating a project.


==============
= config.tcl =
==============

Refer to the description of config.tcl for the Csv framework.

For CsvPerl one more option is possible:

# The variable NeedSources is used to request the perl script to be executed once for
each
# repository node of the project. In that case an additional parameter is sent to the
# perl script (see below for its position)
#set ::NeedSources 1


==========================
= Sample CSV Input Files =
==========================

Refer to the examples for the Csv framework.


===============
= Perl Script =
===============

The perl scipt will receive as arguments:
 - all parameters defined in form.xml (as -${key} $value)
 - the input directory to process (only if ::NeedSources is set to 1 in the config.tcl
file)
 - the location of the output directory where temporary files can be generated
 - the full path of the csv file to be generated
```

```
For the form.xml we created earlier in this document, the command line will be:
 perl <configuration_folder>/tools/CustomDP/CustomDP.pl -csv /path/to/csv -param
MyValue <output_folder> <output_folder>/CustomDP.csv

Example of perl script:
======================
#!/usr/bin/perl
use strict;
use warnings;
$|=1 ;

($csvKey, $csvValue, $paramKey, $paramValue, $output_folder, $output_csv) = @ARGV;

    # Parse input CSV file
    # ...

    # Write results to CSV
    open(CSVFILE, ">" . ${output_csv}) || die "perl: can not write: $!\n";
    binmode(CSVFILE, ":utf8");
    print CSVFILE  "ChangeRequest;15";
    close CSVFILE;

exit 0;
```

Generic Reference

```
===========
= Generic =
===========

The Generic framework is the most flexible Data Provider framework, since it allows
attaching metrics, findings, textual information and links to artefacts. If the
artefacts do not exist in your project, they will be created automatically. It takes
one or more CSV files as input (one per type of information you want to import) and
works with any type of artefact.


============
= form.xml =
============

In form.xml, allow users to specify the path to a CSV file for each type of data you
want to import.
You can set needSources to true or false, depending on whether or not you want to
require the use of a repository connector when your custom Data Provider is used.

Example of form.xml file:
```

```
=========================
<?xml version="1.0" encoding="UTF-8"?>
<tags baseName="Generic" needSources="false">
    <!-- Path to CSV file containing Metrics data -->
    <tag type="text" key="csv" defaultValue="mydata.csv" />
    <!-- Path to CSV file containing Findings data: -->
    <tag type="text" key="fdg" defaultValue="mydata_fdg.csv" />
    <!-- Path to CSV file containing Information data: -->
    <tag type="text" key="inf" defaultValue="mydata_inf.csv" />
    <!-- Path to CSV file containing Links data: -->
    <tag type="text" key="lnk" defaultValue="mydata_lnk.csv" />
</tags>

Note: All tags are optional. You only need to specify the tag element for the type of
data you want to import with your custom Data Provider.


==============
= config.tcl =
==============

Sample config.tcl file:
=======================
# The separator used in the input csv files
# Usually \t or ; or ,
# In our example below, a space is used.
set Separator " "

# The delimiter used in the input CSV file
# This is normally left empty, except when you know that some of the values in the CSV
file
# contain the separator itself, for example:
# "A text containing ; the separator";no problem;end
# In this case, you need to set the delimiter to \" in order for the data provider to
find 3 values instead of 4.
# To include the delimiter itself in a value, you need to escape it by duplicating it,
for example:
# "A text containing "" the delimiter";no problemo;end
# Default: none
set Delimiter \"

# The path separator in an artefact's path
# in the input CSV file.
# Note that artefact is spellt with an "i"
# and not an "e" in this option.
set ArtifactPathSeparator "/"

# If the data provider needs to specify a different toolName (optional)
set SpecifyToolName 1

# Metric2Key contains a case-sensitive list of paired metric IDs:
```

```
#     {MeasureID KeyName [Format]}
# where:
#   - MeasureID is the id of the measure as defined in your analysis model
#   - KeyName is the name in the cell preceding the value to import as found in the
input CSV file
#   - Format is the optional format of the data, the only accepted format
#      is "text" to attach textual information to an artefact. Note that the same
result can also
#       be achieved with Info2Key (see below). For normal metrics omit this field.
set Metric2Key {
    {CHANGES Changed}
}

# Finding2Key contains a case-sensitive list of paired rule IDs:
#     {FindingID KeyName}
# where:
#   - FindingID is the id of the rule as defined in your analysis model
#   - KeyName is the name in the finding name in the input CSV file
set Finding2Key {
    {R_NOTLINKED NotLinked}
}

# Info2Key contains a case-sensitive list of paired info IDs:
#     {InfoID KeyName}
# where:
#   - InfoID is the id of the textual information as defiend in your analysis model
#   - KeyName is the name of the information name in the input CSV file
set Info2Key
    {SPECIAL_LABEL Label}
}

# Ignore findings for artefacts that are not part of the project (orphan findings)
# When set to 1, the findings are ignored
# When set to 0, the findings are imported and attached to the APPLICATION node
# (default: 1)
set IgnoreIfArtefactNotFound 1

# If data in csv concerns source code artefacts (File, Class or Function), the way to
# match file paths can be case-insensitive
# true or false (default)
# This is used when searching for a matching artefact in already-existing artefacts.
set PathsAreCaseInsensitive "false"

# For findings of a type that is not in your ruleset, set a default rule ID.
# The value for this parameter must be a valid rule ID from your analysys model.
# (default: empty)
set UnknownRuleId UNKNOWN_RULE

# Save the total count of orphan findings as a metric at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
```

```
# (default: empty)
set OrphanArteCountId NB_ORPHANS

# Save the total count of unknown rules as a metric at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
# (default: empty)
set OrphanRulesCountId NB_UNKNOWN_RULES

# Save the list of unknown rule IDs as textual information at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
# (default: empty)
set OrphanRulesListId UNKNOWN_RULES_INFO


====================
=  CSV File Format =
====================

All the examples listed below assume the use of the following config.tcl:

set Separator ","
set ArtifactPathSeparator "/"
set Metric2Key {
    {CHANGES Changed}
}
set Finding2Key {
    {R_NOTLINKED NotLinked}
}
set Info2Key
    {SPECIAL_LABEL Label}
}

How to reference an artefact:
============================
 ==> artefact_type artefact_path
 Example:
 REQ_MODULES,Requirements
 REQ_MODULE,Requirements/Module
 REQUIREMENT,Requirements/Module/My_Req

 References the following artefact
 Application
      Requirements (type: REQ_MODULES)
            Module (type: REQ_MODULE)
                My_Req  (type: REQUIREMENT)

Note: For source code artefacts there are 3 special artefact kinds:
 ==> FILE file_path
 ==> CLASS file_path (Name|Line)
```

```
 ==> FUNCTION file_path (Name|Line)

 Examples:
 FUNCTION src/file.c 23
 references the function which contains line 23 in the source file src/file.c, if no
 function found the line whole line of the csv file is ignored.

 FUNCTION src/file.c foo()
 references a function named foo in source file src/file.c. If more than one function
foo
 is defined in this file, then the signature of the function (which is optional) is
used
 to find the best match.

Layout for Metrics File:
========================
 ==> artefact_type artefact_path (Key Value)*

 When the parent artefact type is not given it defaults to <artefact_type>_FOLDER.
 Example:
 REQ_MODULE,Requirements/Module
 REQUIREMENT,Requirements/Module/My_Req,Changed,1

 will produce the following artefact tree:
 Application
      Requirements (type: REQ_MODULE_FOLDER)
          Module (type: REQ_MODULE)
              My_Req : (type: REQUIREMENT) with 1 metric CHANGES = 1

Note: the key "Changed" is mapped to the metric "CHANGES", as specified by the
Metric2Key parameter, so that it matches what is expected by the model.


Layout for Findings File:
=========================
 ==> artefact_type artefact_path key message

 When the parent artefact type is not given it defaults to <artefact_type>_FOLDER.
 Example:
 REQ_MODULE,Requirements/Module
 REQUIREMENT,Requirements/Module/My_Req,NotLinked,A Requirememement should always been
linked

 will produce the following artefact tree:
 Application
      Requirements (type: REQ_MODULE_FOLDER)
          Module (type: REQ_MODULE)
              My_Req (type: REQUIREMENT) with 1 finding R_NOTLINKED whose description
is "A Requirememement should always been linked"

Note: the key "NotLinked" is mapped to the finding "R_NOTLINKED", as specified by the
```

```
Finding2Key parameter, so that it matches what is expected by the model.

Layout for Textual Information File:
====================================
 ==> artefact_type artefact_path label value

 When the parent artefact type is not given it defaults to <artefact_type>_FOLDER.
 Example:
 REQ_MODULE,Requirements/Module
 REQUIREMENT,Requirements/Module/My_Req,Label,This is the label of the req

 will produce the following artefact tree:
 Application
      Requirements (type: REQ_MODULE_FOLDER)
          Module (type: REQ_MODULE)
              My_Req (type: REQUIREMENT) with 1 information of type SPECIAL_LABEL
whose content is "This is the label of the req"

Note: the label "Label" is mapped to the finding "SPECIAL_LABEL", as specified by the
Info2Key parameter, so that it matches what is expected by the model.

Layout for Links File:
======================
 ==> artefact_type artefact_path dest_artefact_type dest_artefact_path link_type

 When the parent artefact type is not given it defaults to <artefact_type>_FOLDER
 Example:
 REQ_MODULE Requirements/Module
 TEST_MODULE Tests/Module
 REQUIREMENT Requirements/Module/My_Req TEST Tests/Module/My_test TESTED_BY

 will produce the following artefact tree:
 Application
    Requirements (type: REQ_MODULE_FOLDER)
        Module (type: REQ_MODULE)
            My_Req (type: REQUIREMENT) ------>
    Tests (type: TEST_MODULE_FOLDER)           |
        Module (type: TEST_MODULE)             |
            My_Test (type: TEST) <------------+ link (type: TESTED_BY)

The TESTED_BY relationship is created with My_Req as source of the link and My_test as
the destination

CSV file organisation when SpecifyToolName is set to 1
======================================================
When the variable SpecifyToolName is set to 1 (or true) a column has to be added
at the beginning of each line in each csv file. This column can be empty or filled
with a different toolName.

 Example:
 ,REQ_MODULE,Requirements/Module
```

```
 MyReqChecker,REQUIREMENT,Requirements/Module/My_Req Label,This is the label of the
req

The finding of type Label will be set as reported by the tool "MyReqChecker".
```

GenericPerl Reference

```
===============
= GenericPerl =
===============

The GenericPerl framework is an extension of the Generic framework that starts by
running a perl script in order to generate the metrics, findings, information and
links files. It is useful if you have an input file whose format needs to be converted
to match the one expected by the Generic framework, or if you need to retrieve and
modify information exported from a web service on your network.


============
= form.xml =
============

In your form.xml, specify the input parameters you need for your Data Provider.
Our example will use two parameters: a path to a CSV file and another text parameter:

<?xml version="1.0" encoding="UTF-8"?>
<tags baseName="CsvPerl" needSources="false">
    <tag type="text" key="csv" defaultValue="/path/to/csv" />
    <tag type="text" key="param" defaultValue="MyValue" />
</tags>


==============
= config.tcl =
==============

Refer to the description of config.tcl for the Generic framework for the basic
options.
Additionally, the following options are available for the GenericPerl framework, in
order to know which type of information your custom Data Provider should try to
import.

# If the data provider needs to specify a different toolName (optional)
#set SpecifyToolName 1

# Set to 1 to import metrics csv file, 0 otherwise
```

```
# ImportMetrics
# When set to 1, your custom Data Provider (CustomDP) will try to import
# metrics from a file called CustomDP.mtr.csv that your perl script
# should generate according to the expected format described in the
# documentation of the Generic framework.
set ImportMetrics 1

# ImportInfos
# When set to 1, your custom Data Provider (CustomDP) will try to import
# textual information from a file called CustomDP.inf.csv that your perl script
# should generate according to the expected format described in the
# documentation of the Generic framework.
set ImportInfos 0

# ImportFindings
# When set to 1, your custom Data Provider (CustomDP) will try to import
# findings from a file called CustomDP.fdg.csv that your perl script
# should generate according to the expected format described in the
# documentation of the Generic framework.
set ImportFindings 1

# ImportLinks
# When set to 1, your custom Data Provider (CustomDP) will try to import
# artefact links from a file called CustomDP.lnk.csv that your perl script
# should generate according to the expected format described in the
# documentation of the Generic framework.
set ImportLinks 0

# Ignore findings for artefacts that are not part of the project (orphan findings)
# When set to 1, the findings are ignored
# When set to 0, the findings are imported and attached to the APPLICATION node
# (default: 1)
set IgnoreIfArtefactNotFound 1

# For findings of a type that is not in your ruleset, set a default rule ID.
# The value for this parameter must be a valid rule ID from your analysys model.
# (default: empty)
set UnknownRuleId UNKNOWN_RULE

# Save the total count of orphan findings as a metric at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
# (default: empty)
set OrphanArteCountId NB_ORPHANS

# Save the total count of unknown rules as a metric at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
# (default: empty)
set OrphanRulesCountId NB_UNKNOWN_RULES
```

```
# Save the list of unknown rule IDs as textual information at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
# (default: empty)
set OrphanRulesListId UNKNOWN_RULES_INFO

====================
=  CSV File Format =
====================

Refer to the examples in the Generic framework.


===============
= Perl Script =
===============

The perl scipt will receive as arguments:
- all parameters defined in form.xml (as -${key} $value)
- the location of the output directory where temporary files can be generated
- the full path of the metric csv file to be generated (if ImportMetrics is set to 1
in config.tcl)
- the full path of the findings csv file to be generated (if ImportFindings is set to
1 in config.tcl)
- the full path of the textual information csv file to be generated (if ImportInfos is
set to 1 in config.tcl)
- the full path of the links csv file to be generated (if ImportLinks is set to 1 in
config.tcl)
- the full path to the output directory used by this data provider in the previous
analysis

For the form.xml and config.tcl we created earlier in this document, the command line
will be:
 perl <configuration_folder>/tools/CustomDP/CustomDP.pl -csv /path/to/csv -param
MyValue <output_folder> <output_folder>/CustomDP.mtr.csv
<output_folder>/CustomDP.fdg.csv <previous_output_folder>

The following perl functions are made available in the perl environment so you can use
them in your script:
- get_tag_value(key) (returns the value for $key parameter from your form.xml)
- get_output_metric()
- get_output_finding()
- get_output_info()
- get_output_link()
- get_output_dir()
- get_input_dir() (returns the folder containing sources if needSources is set to 1)
- get_previous_dir()


Example of perl script:
=======================
```

```
#!/usr/bin/perl
use strict;
use warnings;
$|=1 ;

    # Parse input CSV file
    my $csvFile = get_tag_value("csv");
    my $param = get_tag_value("param");
    # ...

    # Write metrics to CSV
    open(METRICS_FILE, ">" . get_output_metric()) || die "perl: can not write: $!\n";
    binmode(METRICS_FILE, ":utf8");
    print METRICS_FILE  "REQUIREMENTS;Requirements/All_Requirements;NB_REQ;15";
    close METRICS_FILE;

    # Write findings to CSV
    open(FINDINGS_FILE, ">" . get_output_findings()) || die "perl: can not write:
$!\n";
    binmode(FINDINGS_FILE, ":utf8");
    print FINDINGS_FILE  "REQUIREMENTS;Requirements/All_Requirements;R_LOW_REQS;\"The
minimum number of requirement should be at least 25.\"";
    close FINDINGS_FILE;

exit 0;
```

FindingsPerl Reference

```
================
= FindingsPerl =
================

The FindingsPerl framework is used to import findings and attach them to existing
artefacts. Optionally, if an artefact cannot be found in your project, the finding can
be attached to the root node of the project instead. When launching a Data Provider
based on the FindingsPerl framework, a perl script is run first. This perl script is
used to generate a CSV file with the expected format which will then be parsed by the
framework.


============
= form.xml =
============

In your form.xml, specify the input parameters you need for your Data Provider.
Our example will use two parameters: a path to a CSV file and another text parameter:

<?xml version="1.0" encoding="UTF-8"?>
```

```
<tags baseName="CsvPerl" needSources="true">
    <tag type="text" key="csv" defaultValue="/path/to/csv" />
    <tag type="text" key="param" defaultValue="MyValue" />
</tags>

- Since FindingsPerl-based data providers commonly rely on artefacts created by Squan
Sources, you can set the needSources attribute to force users to specify at least one
repository connector when creating a project.


==============
= config.tcl =
==============


Sample config.tcl file:
=======================
# The separator to be used in the generated CSV file
# Usually \t or ;
set Separator ";"

# The delimiter used in the input CSV file
# This is normally left empty, except when you know that some of the values in the CSV
file
# contain the separator itself, for example:
# "A text containing ; the separator";no problem;end
# In this case, you need to set the delimiter to \" in order for the data provider to
find 3 values instead of 4.
# To include the delimiter itself in a value, you need to escape it by duplicating it,
for example:
# "A text containing "" the delimiter";no problemo;end
# Default: none
set Delimiter \"

# Should the perl script execcuted once for each repository node of the project ?
# 1 or 0 (default)
# If true an additional parameter is sent to the
# perl script (see below for its position)
set ::NeedSources 0

# Should the violated rules definitions be generated?
# true or false (default)
# This creates a ruleset file with rules that are not already
# part of your analysis model so you can review it and add
# the rules manually if needed.
set generateRulesDefinitions false

# Should the File paths be case-insensitive?
# true or false (default)
# This is used when searching for a matching artefact in already-existing artefacts.
set PathsAreCaseInsensitive false
```

```
# Should file artefacts declared in the input CSV file be created automatically?
# true (default) or false
set CreateMissingFile true

# Ignore findings for artefacts that are not part of the project (orphan findings)
# When set to 0, the findings are imported and attached to the APPLICATION node
instead of the real artefact
# When set to 1, the findings are not imported at all
# (default: 0)
set IgnoreIfArtefactNotFound 0

# For findings of a type that is not in your ruleset, set a default rule ID.
# The value for this parameter must be a valid rule ID from your analysis model.
# (default: empty)
set UnknownRuleId UNKNOWN_RULE

# Save the total count of orphan findings as a metric at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
# (default: empty)
set OrphanArteCountId NB_ORPHANS

# Save the total count of unknown rules as a metric at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
# (default: empty)
set OrphanRulesCountId NB_UNKNOWN_RULES

# Save the list of unknown rule IDs as textual information at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
# (default: empty)
set OrphanRulesListId UNKNOWN_RULES_INFO

# The tool version to specify in the generated rules definitions
# The default value is ""
# Note that the toolName is the name of the folder you created
# for your custom Data Provider
set ToolVersion ""

# FileOrganisation defines the layout of the CSV file that is produced by your perl
script:
#     header::column: values are referenced from the column header
#     header::line: NOT AVAILABLE
#     alternate::line: NOT AVAILABLE
#     alternate::column: NOT AVAILABLE
set FileOrganisation header::column

# In order to attach a finding to an artefact of type FILE:
#   - Tool (optional) if present it overrides the name of the tool providing the
```

```
finding
#   - Path has to be the path of the file
#   - Type has to be set to FILE
#   - Line can be either empty or the line in the file where the finding is located
#   Rule is the rule identifier, can be used as is or translated using Rule2Key
#   Descr is the description message, which can be empty
#
# In order to attach a finding to an artefact of type FUNCTION:
#   - Tool (optional) if present it overrides the name of the tool providing the
finding
#   - Path has to be the path of the file containing the function
#   - Type has to be FUNCTION
#   - If line is an integer, the system will try to find an artefact function
#       at the given line of the file
#   - If no Line or Line is not an integer, Name is used to find an artefact in
#       the given file having name and signature as found in this column.
# (Line and Name are optional columns)

# Rule2Key contains a case-sensitive list of paired rule IDs:
#     {RuleID KeyName}
# where:
#   - RuleID is the id of the rule as defined in your analysis model
#   - KeyName is the rule ID as written by your perl script in the produced CSV file
# Note: Rules that are not mapped keep their original name. The list of unmapped rules
is in the log file generated by your Data Provider.
set Rule2Key {
    {   ExtractedRuleID_1   MappedRuleId_1   }
    {   ExtractedRuleID_2   MappedRuleId_2   }
}



====================
=  CSV File Format =
====================

According to the options defined earlier in config.tcl, a valid csv file would be:

Path;Type;Line;Name;Rule;Descr
/src/project/module1/f1.c;FILE;12;;R1;Rule R1 is violated because variable v1
/src/project/module1/f1.c;FUNCTION;202;;R4;Rule R4 is violated because function f1
/src/project/module2/f2.c;FUNCTION;42;;R1;Rule R1 is violated because variable v2
/src/project/module2/f2.c;FUNCTION;;skip_line(int);R1;Rule R1 is violated because
variable v2

Working With Paths:
===================

- Path seperators are unified: you do not need to worry about handling differences
between Windows and Linux
- With the option PathsAreCaseInsensitive, case is ignored when searching for files in
the Squore internal data
```

```
- Paths known by Squore are relative paths starting at the root of what was specified
in the repository connector durign the analysis. This relative path is the one used to
match with a path in a csv file.

Here is a valid example of file matching:
  1. You provide C:\A\B\C\D as the root folder in a repository connector
  2. C:\A\B\C\D contains E\e.c then Squore will know E/e.c as a file

  3. You provide a csv file produced on linux and containing
     /tmp/X/Y/E/e.c as path, then Squore will be able to match it with the known file.

Squore uses the longest possible match.
In case of conflict, no file is found and a message is sent to the log.


===============
= Perl Script =
===============

The perl scipt will receive as arguments:
    - all parameters defined in form.xml (as -${key} $value)
    - the input directory to process (only if ::NeedSources is set to 1)
    - the location of the output directory where temporary files can be generated
    - the full path of the findings csv file to be generated

For the form.xml and config.tcl we created earlier in this document, the command line
will be:
 perl <configuration_folder>/tools/CustomDP/CustomDP.pl -csv /path/to/csv -param
MyValue <output_folder> <output_folder>/CustomDP.fdg.csv
<output_folder>/CustomDP.fdg.csv

Example of perl script:
======================
#!/usr/bin/perl
use strict;
use warnings;
$|=1 ;

($csvKey, $csvValue, $paramKey, $paramValue, $output_folder, $output_csv) = @ARGV;

    # Parse input CSV file
    # ...

    # Write results to CSV
    open(CSVFILE, ">" . ${output_csv}) || die "perl: can not write: $!\n";
    binmode(CSVFILE, ":utf8");
    print CSVFILE "Path;Type;Line;Name;Rule;Descr";
    print CSVFILE "/src/project/module1/f1.c;FILE;12;;R1;Rule R1 is violated because
variable v1";
    close CSVFILE;
```

```
exit 0;
```

ExcelMetrics Reference

```
================
= ExcelMetrics =
================

The ExcelMetrics framework is used to extract information from one or more Microsoft
Excel files (.xls or .xslx). A detailed configuration file allows defining how the
Excel document should be read and what information should be extracted. This framework
allows importing metrics, findings and textual information to existing artefacts or
artefacts that will be created by the Data Provider.


============
= form.xml =
============

You can customise form.xml to either:
- specify the path to a single Excel file to import
- specify a pattern to import all Excel files matching this pattern in a directory

In order to import a single Excel file:
=====================================
<?xml version="1.0" encoding="UTF-8"?>
<tags baseName="ExcelMetrics" needSources="false">
    <tag type="text" key="excel" defaultValue="/path/to/mydata.xslx" />
</tags>

Notes:
- The excel key is mandatory.

In order to import all files matching a patter in a folder:
============================================================
<?xml version="1.0" encoding="UTF-8"?>
<tags baseName="ExcelMetrics" needSources="false">
    <!-- Root directory containing Excel files to import-->
    <tag type="text" key="dir" defaultValue="/path/to/mydata" />
    <!-- Pattern that needs to be matched by a file name in order to import it-->
    <tag type="text" key="ext" defaultValue="*.xlsx" />
    <!-- search for files in sub-folders -->
    <tag type="booleanChoice" defaultValue="true" key="sub" />
</tags>


Notes:
- The dir and ext keys are mandatory
```

```
- The sub key is optional (and its value set to false if not specified)


==============
= config.tcl =
==============

Sample config.tcl file:
=======================
# The separator to be used in the generated csv file
# Usually \t or ; or ,
set Separator  ";"

# The delimiter used in the input CSV file
# This is normally left empty, except when you know that some of the values in the CSV
file
# contain the separator itself, for example:
# "A text containing ; the separator";no problem;end
# In this case, you need to set the delimiter to \" in order for the data provider to
find 3 values instead of 4.
# To include the delimiter itself in a value, you need to escape it by duplicating it,
for example:
# "A text containing "" the delimiter";no problemo;end
# Default: none
set Delimiter \"

# The path separator in an artefact's path
# in the generated CSV file.
set ArtefactPathSeparator "/"

# Ignore findings for artefacts that are not part of the project (orphan findings)
# When set to 1, the findings are ignored
# When set to 0, the findings are imported and attached to the APPLICATION node
# (default: 1)
set IgnoreIfArtefactNotFound 1

# For findings of a type that is not in your ruleset, set a default rule ID.
# The value for this parameter must be a valid rule ID from your analysys model.
# (default: empty)
set UnknownRuleId UNKNOWN_RULE

# Save the total count of orphan findings as a metric at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
# (default: empty)
set OrphanArteCountId NB_ORPHANS

# Save the total count of unknown rules as a metric at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
# (default: empty)
```

```
set OrphanRulesCountId NB_UNKNOWN_RULES

# Save the list of unknown rule IDs as textual information at application level
# Specify the ID of the metric to use in your analysys model
# to store the information
# (default: empty)
set OrphanRulesListId UNKNOWN_RULES_INFO

# The list of the Excel sheets to read, each sheet has the number of the first line to
read
# A Perl regexp pattern can be used instead of the name of the sheet (the first sheet
matching
# the pattern will be considered)
set Sheets {{Baselines 5} {ChangeNotes 5}}

# ######################
# # COMMON DEFINITIONS #
# ######################
#
# - <value> is a list of column specifications whose values will be concatened. When
no column name is present, the
#         text is taken as it appears. Optional sheet name can be added (with ! char
to separate from the column name)
#       Examples:
#            - {C:} the value will be the value in column C on the current row
#            - {C: B:} the value will be the concatenation of values found in column C
and B of the current row
#            - {Deliveries} the value will be Deliveries
#            - {BJ: " - " BL:} the value will be the concatenation of value found in
column BJ,
#                string " - " and the value found in column BL fo the current row
#            - {OtherSheet!C:} the value will be the value in column C from the sheet
OtherSheet on the current row
#
#  - <condition> is a list of conditions. An empty condition is always true. A
condition is a column name followed by colon,
#              optionally followed by a perl regexp. Optional sheet name can be added
(with ! char to separate from the column name)
#       Examples:
#       - {B:} the value in column B must be empty on the current row
#       - {B:.+} the value in column B can not be empty on the current row
#       - {B:R_.+} the value in column B is a word starting by R_ on the current row
#       - {A: B:.+ C:R_.+} the value in column A must be empty and the value in column
B must contain something and
#           the column C contains a word starting with R_  on the current row
#       - {OtherSheet!B:.+} the value in column B from sheet OtherSheet on the current
row can not be empty.

# #############
# # ARTEFACTS #
# #############
```

```
# The variable is a list of artefact hierarchy specification:
# {ArtefactHierarchySpec1 ArtefactHierarchySpec2 ... ArtefactHierarchySpecN}
# where each ArtefactHierarchySpecx is a list of ArtefactSpec
#
# An ArtefactSpec is a list of items, each item being:
# {<(sheetName!)?artefactType> <conditions> <name> <parentType>? <parentName>?}
# where:
#    - <(sheetName!)?artefactType>: allows specifying the type. Optional sheetName can
be added (with ! char to separate from the type) to limit
#                                   the artefact search in one specific sheet. When
Sheets are given with regexp, the same regexp has to be used
#                                   for the sheetName.
#                                   If the type is followed by a question mark (?),
this level of artefact is optional.
#                                   If the type is followed by a plus char (+), this
level is repeatable on the next row
#    - <condition>: see COMMON DEFINITIONS
#    - <value>: the name of the artefact to build, see COMMON DEFINITIONS
#
#   - <parentType>: This element is optional. When present, it means that the current
element will be attached to a parent having this type
#   - <parentValue>: This is a list like <value> to build the name of the artefact of
type <parentType>. If such artefact is not found,
#                  the current artefact does not match
#
# Note: to add metrics at application level, specify an APPLICATION artefact which
will match only one line:
#       e.g. {APPLICATION {A:.+} {}} will recognize as application the line having
column A not empty.
set ArtefactsSpecs {
    {
        {DELIVERY {} {Deliveries}}
        {RELEASE {E:.+} {E:}}
        {SPRINT {O:SW_Software} {Q:}}
    }
    {
        {DELIVERY {} {Deliveries}}
        {RELEASE {O:SY_System} {Q:}}
    }
    {
        {WP {BL:.+ AF:.+} {BJ: " - " BL:} SPRINT {AF:}}
        {ChangeNotes!TASK {D:(added|changed|unchanged) T:imes} {W: AD:}}
    }
    {
        {WP {} {{Unplanned imes}} SPRINT {AF:}}
        {TASK {BL: D:(added|changed|unchanged) T:imes W:.+} {W: AD:}}
    }
}

# ############
# # METRICS #
```

```
# ############
# Specification of metrics to be retreived
# This is a list where each element is:
#  {<artefactTypeList> <metricId> <condition> <value> <format>}
# Where:
#     - <artefactTypeList>: the list of artefact types for which the metric has to be
used
#                       each element of the list is (sheetName!)?artefactType where
sheetName is used
#                       to restrict search to only one sheet. sheetName is optional.
#     - <metricId>: the name of the MeasureId to be injected into Squore, as defined
in your analysis model
#     - <confition>: see COMMON DEFINITIONS above. This is the condition for the
metric to be generated.
#     - <value> : see COMMON DEFINITIONS above. This is the value for the metric (can
be built from multi column)
#     - <format> : optional, defaults to NUMBER
#                 Possible format are:
#                           * DATE_FR, DATE_EN for date stored as string
#                           * DATE for cell formatted as date
#                           * NUMBER_FR, NUMBER_EN for number stored as string
#                           * NUMBER for cell formatted as number
#                           * LINES for counting the number of text lines in a cell
#     - <formatPattern> : optional
#                 Only used by the LINES format.
#                 This is a pattern (can contain perl regexp) used to filter lines to
count
set MetricsSpecs {
    {{RELEASE SPRINT} TIMESTAMP {} {A:} DATE_EN}
    {{RELEASE SPRINT} DATE_ACTUAL_RELEASE {} {S:} DATE_EN}
    {{RELEASE SPRINT} DATE_FINISH {} {T:} DATE_EN}
    {{RELEASE SPRINT} DELIVERY_STATUS {} {U:}}
    {{WP} WP_STATUS {} {BO:}}
    {{ChangeNotes!TASK} IS_UNPLAN {} {BL:}}
    {{TASK WP} DATE_LABEL {} {BP:} DATE_EN}
    {{TASK WP} DATE_INTEG_PLAN {} {BD:} DATE_EN}
    {{TASK} TASK_STATUS {} {AE:}}
    {{TASK} TASK_TYPE {} {AB:}}
}

# ############
# # FINDINGS #
# ############
# This is a list where each element is:
#  {<artefactTypeList> <findingId> <condition> <value> <localisation>}
# Where:
#     - <artefactTypeList>: the list of artefact type for which the metric has to be
used
#                       each element of the list is (sheetName!)?artefactType where
sheetName is used
#                       to restrict search to only one sheet. sheetName is optional.
```

```
#     - <findingId>: the name of the FindingId to be injected into Squore, as defined
in your analysis model
#     - <confition>: see COMMON DEFINITIONS above. This is the condition for the
finding to be triggered.
#     - <value>: see COMMON DEFINITIONS above. This is the value for the message of
the finding (can be built from multi column)
#     - <localisation>: this a <value> representing the localisation of the finding
(free text)
set FindingsSpecs {
    {{WP} {BAD_WP} {BL:.+ AF:.+} {{This WP is not in a correct state } AF:.+} {A:}}
}

# ########################
# # TEXTUAL INFORMATION #
# ########################
# This is a list where each element is:
#  {<artefactTypeList> <infoId> <condition> <value>}
# Where:
#     - <artefactTypeList> the list of artefact types for which the info has to be
used
#                         each element of the list is (sheetName!)?artefactType where
sheetName is used
#                         to restrict search to only one sheet. sheetName is optional.
#     - <infoId> : is the name of the Information to be attached to the artefact, as
defined in your analysis model
#     - <confition> : see COMMON DEFINITIONS above. This is the condition for the info
to be generated.
#     - <value> : see COMMON DEFINITIONS above. This is the value for the info (can be
built from multi column)
set InfosSpecs {
    {{TASK} ASSIGN_TO {} {XB:}}
}

# ########################
# # LABEL TRANSFORMATION #
# ########################
# This is a list value specification for MeasureId or InfoId:
#    <MeasureId|InfoId> { {<LABEL1> <value1>} ... {<LABELn> <valuen>}}
# Where:
#    - <MeasureId|InfoId> : is either a MeasureId, an InfoId, or * if it is available
for every measureid/infoid
#    - <LABELx> : is the label to macth (can contain perl regexp)
#    - <valuex> : is the value to replace the label by, it has to match the correct
format for the metrics (no format for infoid)
#
# Note: only metrics which are labels in the excel file or information which need to
be rewriten, need to be described here.
set Label2ValueSpec {
    {
        STATUS {
            {OPENED 0}
```

```
            {ANALYZED 1}
            {CLOSED 2}
            {.* -1}
        }
    }
    {
        * {
            {FATAL 0}
            {ERROR 1}
            {WARNING 2}
            {{LEVEL:\s*0} 1}
            {{LEVEL:\s*1} 2}
            {{LEVEL:\s*[2-9]+} 3}
        }
    }
}

Note that a sample Excel file with its associated config.tcl is available in
$SQUORE_HOME/addons/tools/ExcelMetrics in order to further explain available
configuration options.
```

Appendix C: Squore XML Schemas

input-data-2.xsd

Download input-data-2.xsd

form.xsd

Download form.xsd

properties-1.2.xsd

Download properties-1.2.xsd

config-1.3.xsd

Download config-1.3.xsd

analysis.xsd

Download analysis.xsd

decision.xsd

Download decision.xsd

description.xsd

Download description.xsd

exports.xsd

Download exports.xsd

highlights.xsd

Download highlights.xsd

properties.xsd

Download properties.xsd

tutorials.xsd

Download tutorials.xsd

wizards.xsd

Download wizards.xsd

Index

C

D