VALUING AND IMPROVING
SOFTWARE
CAPITAL

SQUORING Technologies

Key Performance Indicator

Line Counting			
Lines of Code	481219	↗	✓
Source Lines Of Code	401326	↗	✓
Effective Lines Of Code	325196	↗	✓
Cyclomatic Complexity	12265	↗	✓
Comment Rate	16 %	↘	✗

Decision Making			
Business Value	1912	→	i
Technical Debt	3626	↗	i
Maturity Index	65 %	↘	C
Stability Index	84 %	↗	B
Reusability Index	44 %	↘	D

Squore 17.0.8

Command Line Interface

Reference : SUM_Squore_CLI
Version : 17.0.8
Date : 30/11/2017

Squoring Technologies
76, allées Jean Jaurès / 31000 Toulouse - FRANCE / Tel. +33 581 346 397
http://www.squoring.com/
APE 5829 C - SIREN 521171629 – RCS Toulouse

Command Line Interface

Abstract

This edition of the Command Line Interface applies to Squore 17.0.8 and to all subsequent releases and modifications until otherwise indicated in new editions.

Licence

Warranty

Table of Contents

Typographical Conventions ix
Acronyms and Abbreviations x
1. Introduction 1
1.1. Foreword 1
1.2. About This Document 1
1.3. Contacting Squoring Technologies Product Support 1
1.4. Responsabilities 2
1.5. Getting the Latest Version of this Manual 2
2. Getting Started With the Squore CLI 3
2.1. Installation Prerequisites 3
2.1.1. Supported Operating Systems 3
2.1.2. For All Systems 4
2.1.3. Packages for Windows 4
2.1.4. Packages for Linux 4
2.1.5. Packages for CentOS and Red Hat Enterprise Linux 5
2.1.6. Packages for Ubuntu 6
2.2. Deploying Squore CLI 7
2.2.1. On Windows 7
2.2.2. On Linux 12
2.2.3. Third-Party Plugins and Applications 13
2.2.4. Post Installation Actions 13
2.3. Upgrading Squore CLI 14
2.4. Removing Squore CLI 14
2.4.1. On Windows 14
2.4.2. On Linux 16
2.5. Saving Credentials to Disk 16
2.6. Running The Sample Scripts 17
2.7. Squore in a Continuous Integration Environment 17
3. Command Line Reference 19
3.1. Squore CLI Commands 19
3.2. Squore CLI Parameters 20
3.3. Project Parameters 20
3.4. Exit Codes 23
4. Repository Connectors 24
4.1. Folder Path 24
4.1.1. Description 24
4.1.2. Usage 24
4.2. Zip Upload 24
4.2.1. Description 24
4.2.2. Usage 24
4.3. CVS 24
4.3.1. Description 24
4.3.2. Usage 25
4.4. ClearCase 25
4.4.1. Description 25
4.4.2. Usage 25
4.5. Perforce 26
4.5.1. Description 26
4.5.2. Usage 26
4.6. Git 27

4.6.1. Description 27
4.6.2. Usage 27
4.7. PTC Integrity 27
4.7.1. Description 27
4.7.2. Usage 28
4.8. TFS 28
4.8.1. Description 28
4.8.2. Usage 29
4.9. Synergy 29
4.9.1. Description 29
4.9.2. Usage 30
4.10. SVN 30
4.10.1. Description 30
4.10.2. Usage 30
4.11. Using Multiple Nodes 31
4.12. Using Data Provider Input Files From Version Control 31
5. Data Providers 34
5.1. AntiC 34
5.1.1. Description 34
5.1.2. Usage 34
5.2. Automotive Coverage Import 34
5.2.1. Description 34
5.2.2. Usage 34
5.3. Automotive Tag Import 34
5.3.1. Description 34
5.3.2. Usage 35
5.4. BullseyeCoverage Code Coverage Analyzer 35
5.4.1. Description 35
5.4.2. Usage 35
5.5. CPD 35
5.5.1. Description 35
5.5.2. Usage 35
5.6. Cppcheck 35
5.6.1. Description 36
5.6.2. Usage 36
5.7. Cppcheck (plugin) 36
5.7.1. Description 36
5.7.2. Usage 36
5.8. CPPTest 36
5.8.1. Description 36
5.8.2. Usage 37
5.9. Cantata 37
5.9.1. Description 37
5.9.2. Usage 37
5.10. CheckStyle 37
5.10.1. Description 37
5.10.2. Usage 37
5.11. CheckStyle (plugin) 37
5.11.1. Description 38
5.11.2. Usage 38
5.12. CheckStyle for SQALE (plugin) 38
5.12.1. Description 38
5.12.2. Usage 38

5.13. Cobertura 39
5.13.1. Description 39
5.13.2. Usage 39
5.14. CodeSonar 39
5.14.1. Description 39
5.14.2. Usage 39
5.15. Compiler 39
5.15.1. Description 39
5.15.2. Usage 40
5.16. Coverity 40
5.16.1. Description 40
5.16.2. Usage 40
5.17. FindBugs 40
5.17.1. Description 40
5.17.2. Usage 40
5.18. FindBugs (plugin) 40
5.18.1. Description 41
5.18.2. Usage 41
5.19. Function Relaxer 41
5.19.1. Description 41
5.19.2. Usage 41
5.20. FxCop 41
5.20.1. Description 41
5.20.2. Usage 42
5.21. GCov 42
5.21.1. Description 42
5.21.2. Usage 42
5.22. GNATcheck 42
5.22.1. Description 42
5.22.2. Usage 42
5.23. GNATCompiler 43
5.23.1. Description 43
5.23.2. Usage 43
5.24. JUnit 43
5.24.1. Description 43
5.24.2. Usage 43
5.25. JaCoCo 43
5.25.1. Description 43
5.25.2. Usage 44
5.26. Klocwork 44
5.26.1. Description 44
5.26.2. Usage 44
5.27. Rational Logiscope 44
5.27.1. Description 44
5.27.2. Usage 44
5.28. MemUsage 44
5.28.1. Description 45
5.28.2. Usage 45
5.29. NCover 45
5.29.1. Description 45
5.29.2. Usage 45
5.30. Oracle PLSQL compiler Warning checker 45
5.30.1. Description 45

5.30.2. Usage 45
5.31. MISRA Rule Checking using PC-lint 46
5.31.1. Description 46
5.31.2. Usage 46
5.32. PMD 46
5.32.1. Description 46
5.32.2. Usage 46
5.33. PMD (plugin) 46
5.33.1. Description 47
5.33.2. Usage 47
5.34. Polyspace 47
5.34.1. Description 47
5.34.2. Usage 47
5.35. Polyspace MISRA 47
5.35.1. Description 47
5.35.2. Usage 48
5.36. Polyspace (plugin) 48
5.36.1. Description 48
5.36.2. Usage 48
5.37. MISRA Rule Checking with QAC 48
5.37.1. Description 48
5.37.2. Usage 49
5.38. Unit Test Code Coverage from Rational Test RealTime 49
5.38.1. Description 49
5.38.2. Usage 49
5.39. ReqIF 49
5.39.1. Description 49
5.39.2. Usage 49
5.40. SQL Code Guard 50
5.40.1. Description 50
5.40.2. Usage 50
5.41. Squan Sources 50
5.41.1. Description 50
5.41.2. Usage 51
5.42. Squore Import 52
5.42.1. Description 52
5.42.2. Usage 52
5.43. Squore Virtual Project 53
5.43.1. Description 53
5.43.2. Usage 53
5.44. StyleCop 53
5.44.1. Description 53
5.44.2. Usage 53
5.45. StyleCop (plugin) 53
5.45.1. Description 53
5.45.2. Usage 54
5.46. Tessy 54
5.46.1. Description 54
5.46.2. Usage 54
5.47. VectorCAST 6.3 54
5.47.1. Description 54
5.47.2. Usage 54
5.48. CodeSniffer 55

5.48.1. Description 55
5.48.2. Usage 55
5.49. Configuration Checker 55
5.49.1. Description 55
5.49.2. Usage 55
5.50. Csv Coverage Import 55
5.50.1. Description 55
5.50.2. Usage 55
5.51. CSV Findings 56
5.51.1. Description 56
5.51.2. Usage 56
5.52. Csv Tag Import 56
5.52.1. Description 56
5.52.2. Usage 56
5.53. Csv Test Results Import 56
5.53.1. Description 56
5.53.2. Usage 56
5.54. OSLC 57
5.54.1. Description 57
5.54.2. Usage 57
5.55. pep8 57
5.55.1. Description 57
5.55.2. Usage 57
5.56. pep8 (plugin) 57
5.56.1. Description 58
5.56.2. Usage 58
5.57. PHP Code Coverage 58
5.57.1. Description 58
5.57.2. Usage 58
5.58. pylint 58
5.58.1. Description 58
5.58.2. Usage 58
5.59. pylint (plugin) 59
5.59.1. Description 59
5.59.2. Usage 59
5.60. Qac_8_2 59
5.60.1. Description 59
5.60.2. Usage 59
5.61. Advanced COBOL Parsing 59
5.62. Creating your own Data Providers 60
5.62.1. Choosing the Right Data Provider Framework 60
5.62.2. Extending a Framework 62
5.62.3. Creating a Freestyle Data Provider 62
5.62.4. Data Provider Parameters 63
5.62.5. Localising your Data Provider 66
A. Reference pages 68
install 68
B. Data Provider Frameworks 69
Csv Reference 69
csv_findings Reference 72
CsvPerl Reference 73
Generic Reference 75
GenericPerl Reference 79

FindingsPerl Reference 82
ExcelMetrics Reference 86
Index 92

Typographical Conventions

The following conventions are used in this manual.

Typeface or Symbol	Meaning
Bold	Book titles, important items, or items that can be selected including buttons and menu choices. For example: Click the **Next** button to continue
Italic	A name of a user defined textual element. For example: **`Username`** : *admin*
`Courier New`	Files and directories; file extensions, computer output. For example: Edit the `config.xml` file
`Courier Bold`	Commands, screen messages requiring user action. For example: **`Username`** : *admin*
>	Menu choices. For example: Select **`File > Open`**. This means select the File menu, then select the Open command from it.
<...>	Generic terms. For example: <SQUORE_HOME> refers to the Squore installation directory.

Notes

Screenshots displayed in this manual may differ slightly from the ones in the actual product.

Acronyms and Abbreviations

The following acronyms and abbreviations are used in this manual.

CI	Continuous Integration
CLI	Command Line Interface
DP	Data Provider, a Squore module capable of handling input from various other systems and import information into Squore
RC	Repository Connector, a Squore module capable of extracting source code from source code management systems.

1. Introduction

1.1. Foreword

This document was released by Squoring Technologies.

It is part of the user documentation of the Squore software product edited and distributed by Squoring Technologies.

1.2. About This Document

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:

- Chapter 2, *Getting Started With the Squore CLI* provides a basic introduction to Squore CLI and the examples provided with your Squore installation.
- Chapter 3, *Command Line Reference* provides a complete reference of all the command line options and parameters for creating projects.
- Chapter 4, *Repository Connectors* covers the default Repository Connectors and the parameters to pass to Squore to use them.
- Chapter 5, *Data Providers* is a reference guide to all the Data Providers shipped with Squore.

If you are already familiar with Squore, you can navigate this manual by looking for what has changed since the previous version. New functionality is tagged with **(new in 17.0)** throughout this manual. A summary of the new features described in this manual is available in the entry *** What's New in Squore 17.0?** of this manual's Index.

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

- Squore Installation Checklist
- Squore Installation and Administration Guide
- Squore Getting Started Guide
- Squore Command Line Interface
- Squore Configuration Guide
- Squore Eclipse Plugin Guide
- Squore Reference Manual

1.3. Contacting Squoring Technologies Product Support

If the information provided in this manual is erroneous or inaccurate, or if you encounter problems during your installation, contact Squoring Technologies Product Support: http://support.squoring.com/

You will need a valid Squore 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@squoring.com

Squoring Technologies Product Support

76, allées Jean Jaurès / 31000 Toulouse - FRANCE

1.4. Responsabilities

Approval of this version of the document and any further updates are the responsibility of Squoring Technologies.

1.5. 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 Squoring Technologies documentation site to consult or download the latest Squore manuals at http://support.squoring.com/documentation/17.0.8. Manuals are constantly updated and published as soon as they are available.

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.

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

2.1. Installation Prerequisites

2.1.1. Supported Operating Systems

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

- CentOS 6
- CentOS 7
- Fedora 19
- Ubuntu Server 14.04
- Windows 7
- Windows 8
- Windows 10

Note
On Linux, a 64-bit version of the OS is required

On Windows, a 64-bit version of the OS is required if you want Squore to run as a Windows service, but the installation is also supported on a 32-bit system (and is started and stopped using .bat files instead of a Windows Service)

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

- RedHat EL 6
- RedHat EL 7
- SuSe Linux 11.1
- Ubuntu Server 10.04
- Ubuntu Server 16.04
- Windows Server 2008 R2
- Windows Server 2012 R2

2.1.2. 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 http://support.squoring.com/download_area.php
- The Oracle Java Runtime Environment version 8 (other versions are not supported)
- At least 2 GB of space available on the disk for a full installation
- 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.

2.1.3. 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.

2.1.4. Packages for Linux

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:
    - **Algorithm::Diff** [module details] [http://search.cpan.org/~nedkonz/Algorithm-Diff/lib/Algorithm/Diff.pm]
    - **Archive::Zip** [module details] [http://search.cpan.org/~phred/Archive-Zip/lib/Archive/Zip.pm]
    - **Date::Calc** [module details] [http://search.cpan.org/~stbey/Date-Calc/lib/Date/Calc.pod]
    - **Digest::SHA** [module details] [http://search.cpan.org/~mshelor/Digest-SHA/lib/Digest/SHA.pm]
    - **HTTP::Request** [module details] [http://search.cpan.org/~gaas/HTTP-Message/lib/HTTP/Request.pm]
    - **JSON** [module details] [http://search.cpan.org/~makamaka/JSON/lib/JSON.pm]
    - **LWP** [module details] [http://search.cpan.org/~ether/libwww-perl/lib/LWP.pm]
    - **LWP::UserAgent** [module details] [http://search.cpan.org/~gaas/libwww-perl/lib/LWP/UserAgent.pm]
    - **Time::HiRes** [module details] [http://search.cpan.org/~zefram/Time-HiRes/HiRes.pm]
    - **XML::Parser** [module details] [http://search.cpan.org/~toddr/XML-Parser/Parser.pm]
  - Optional packages for working with Microsoft Excel:
    - **HTML::Entities** [module details] [http://search.cpan.org/dist/HTML-Parser/lib/HTML/Entities.pm]
    - **Spreadsheet::BasicRead** [module details] [http://search.cpan.org/~gng/Spreadsheet-BasicRead/BasicRead.pm]
  - Optional packages for working with OSLC systems:
    - **Date::Parse** [module details] [http://search.cpan.org/~gbarr/TimeDate/lib/Date/Parse.pm]
    - **WWW::Mechanize** [module details] [http://search.cpan.org/~ether/WWW-Mechanize/lib/WWW/Mechanize.pm]
    - **XML::LibXML** [module details] [http://search.cpan.org/~shlomif/XML-LibXML/LibXML.pod]

- Optional packages for working with GitHub systems:
  - **Date::Parse** [module details] [http://search.cpan.org/~gbarr/TimeDate/lib/Date/Parse.pm]
  - **Mail::Box::Manager** [module details] [http://search.cpan.org/~markov/Mail-Box/lib/Mail/Box/Manager.pod]
  - **Mail::Message::Body::Lines** [module details] [http://search.cpan.org/~markov/Mail-Box/lib/Mail/Message/Body/Lines.pod]
  - **Mail::Message::Construct** [module details] [http://search.cpan.org/~markov/Mail-Box/lib/Mail/Message/Construct.pod]
  - **Mail::Mbox::MessageParser** [module details] [http://search.cpan.org/~dcoppit/Mail-Mbox-MessageParser/lib/Mail/Mbox/MessageParser.pm]
  - **Net::GitHub** [module details] [http://search.cpan.org/~fayland/Net-GitHub/lib/Net/GitHub.pm]
- Optional packages for working with Semios/Prometil systems:
  - **File::Slurp** [module details] [http://search.cpan.org/~uri/File-Slurp/lib/File/Slurp.pm]
- Optional packages for Advanced CSV Export Management:
  - **Text::CSV** [module details] [http://search.cpan.org/~makamaka/Text-CSV-1.33/lib/Text/CSV.pm]

Tip

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,

2.1.5. Packages for CentOS and Red Hat Enterprise Linux

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

Mandatory packages:

- **java-1.8.0-openjdk**
- **perl**
- **perl-Algorithm-Diff**
- **perl-Archive-Zip**
- **perl-Date-Calc**
- **perl-Digest-SHA**
- **perl-JSON**
- **perl-libwww-perl**
- **perl-Time-HiRes**
- **perl-XML-Parser**
- **tcl**

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)

Warning
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 working with GitHub systems:

- **perl-TimeDate**
- **perl-Mail-Box** (available in the EPEL repository)
- **perl-Mail-Mbox-MessageParser** (available in the EPEL repository)
- **perl-Net-GitHub** (available in the EPEL repository)

Optional packages for working with Semios/Prometil systems:

- **perl-File-Slurp**

Optional packages for Advanced CSV Export Management:

- **perl-Text-CSV** (available in the EPEL repository)

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

2.1.6. Packages for Ubuntu

On Ubuntu 14.04.3 LTS, the dependencies are satisfied by the following packages:

Mandatory packages:

- **libalgorithm-diff-perl**
- **libarchive-zip-perl**
- **libdate-calc-perl**
- **libhttp-message-perl**
- **libjson-perl**
- **libwww-perl**
- **libxml-parser-perl**
- **openjdk-8-jre**
- **perl**
- **tcl**

Optional packages for working with Microsoft Excel:

- **make** (CPAN utility requirement)
- **libhtml-parser-perl**

- **libspreadsheet-parseexcel-perl**
- **libspreadsheet-xlsx-perl**

> **Warning**
> 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 working with GitHub systems:

- **libtimedate-perl**
- **libmail-box-perl**
- **libmail-mbox-messageparser-perl**
- **libnet-github-perl**

Optional packages for working with Semios/Prometil systems:

- **libfile-slurp-perl**

Optional packages for Advanced CSV Export Management:

- **libtext-csv-perl**

2.2. Deploying Squore CLI

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

> **Warning**
> There is currently no Squore CLI installation package for Squore 17.0. If you need to install Squore CLI, download the latest version of the previous release and perform an upgrade after installing following the steps in Section 2.3, “Upgrading Squore CLI”.

2.2.1. On Windows

After verifying that you meet the prerequisites detailed in Section 2.1, “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.

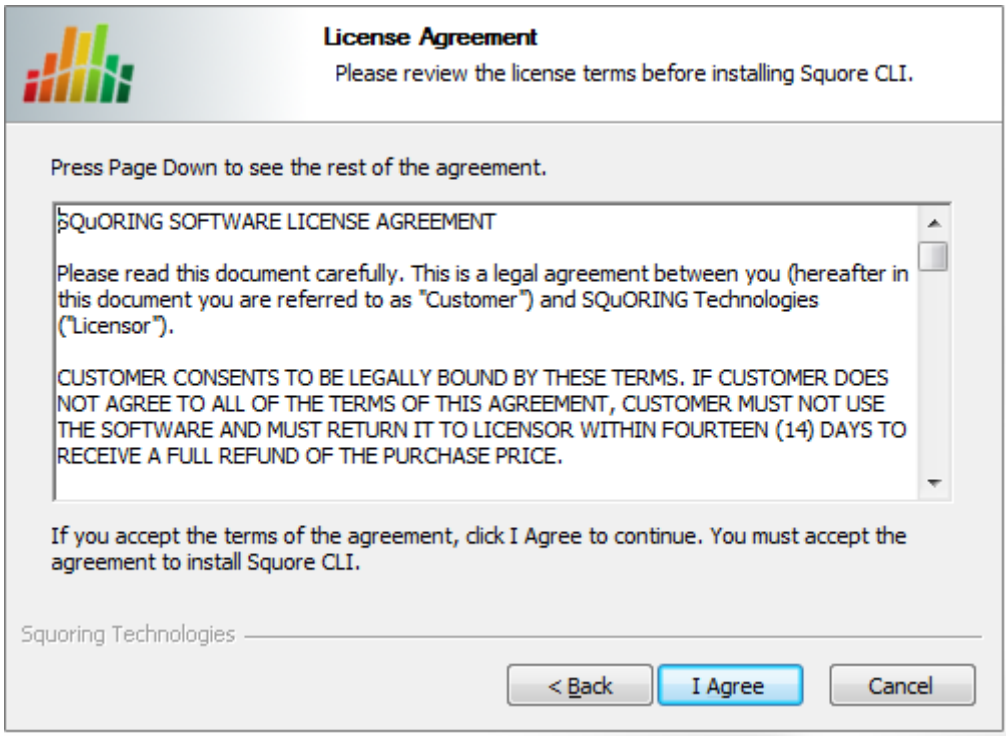
> **Warning**
> The data and temporary folders must be excluded from the scope of virus scanners, malware protectors and search indexers to avoid any errors during an analysis.

1. Squore CLI installer Welcome screen

Squore CLI installer Welcome screen

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

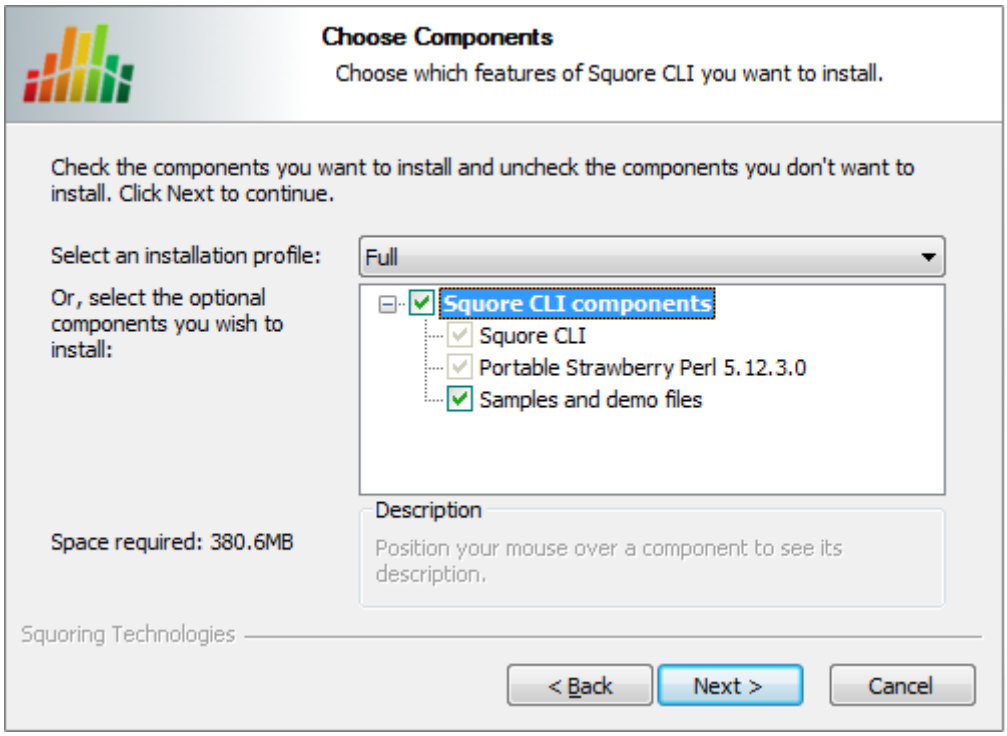
2. Squore CLI licence agreement screen



Squore CLI licence agreement screen

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

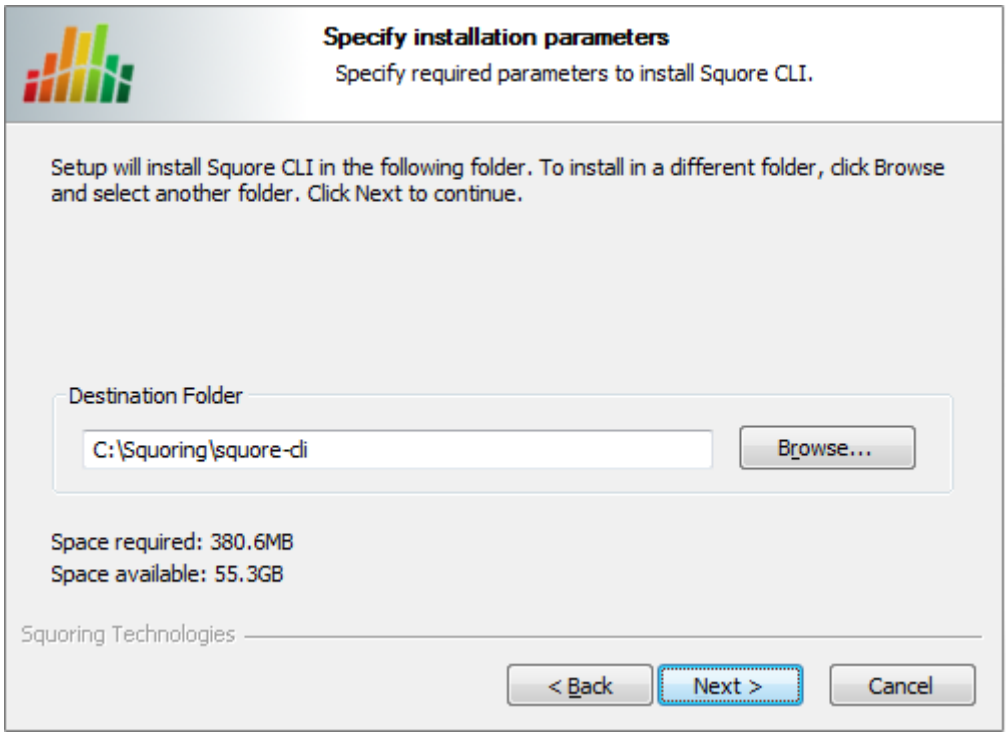
3. Squore CLI components screen



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.

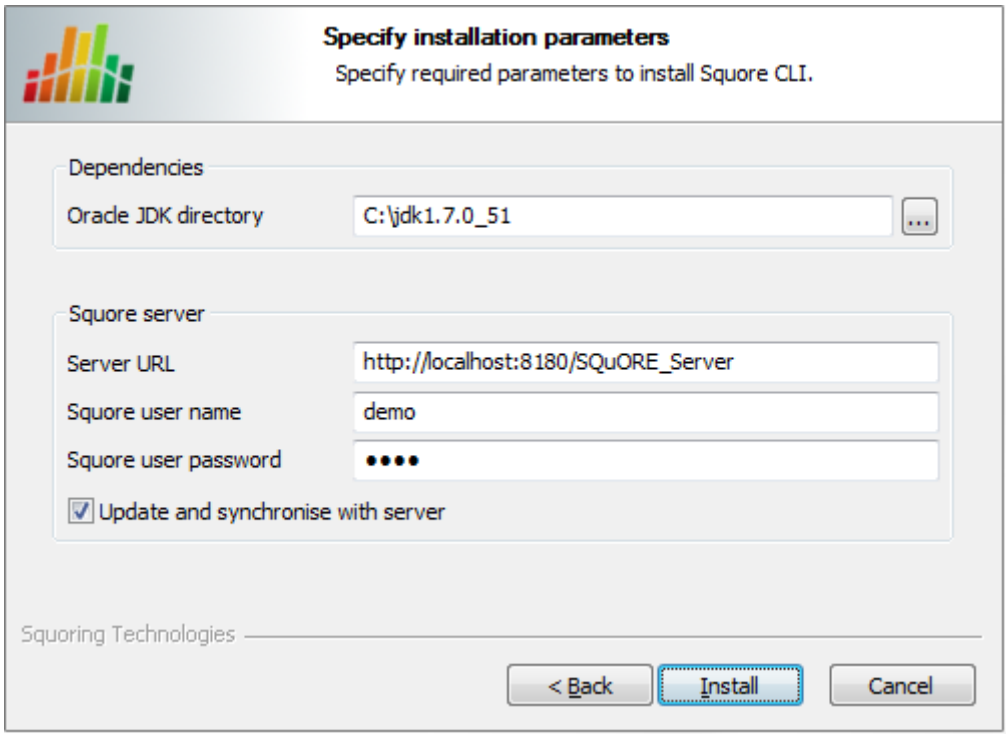
4. Squore CLI destination folder screen



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.

2.2.2. On Linux

Before installing Squore CLI on a Linux platform, verify that all prerequisites are met, as described in Section 2.1, “Installation Prerequisites”

1. Copy the installation package (a compressed tar.bz2 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:
   `<SQUORE_HOME>/bin/install -v -s` *`http://localhost:8180/SQuORE_Server`* `-u` *`user`* `-p` *`password`*

   For more details on **install** options, refer to install(1).

Tip

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.

2.2.3. 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.

> **Tip**
> 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 <SQUORE_HOME>/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.

2.2.4. 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.

> **Tip**
> 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="1.1">
 <paths>
  <path name="perl.dir" path="path/to/perl"/>
  <path name="tclsh.dir" path="path/to/tclsh"/>
 </paths>
 <configuration>
  <path directory="<SQUORE_HOME>/configuration"/>
 </configuration>
 <addons>
  <path directory="<SQUORE_HOME>/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="1.1>
 <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>
```

Tip

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

2.3. Upgrading Squore CLI

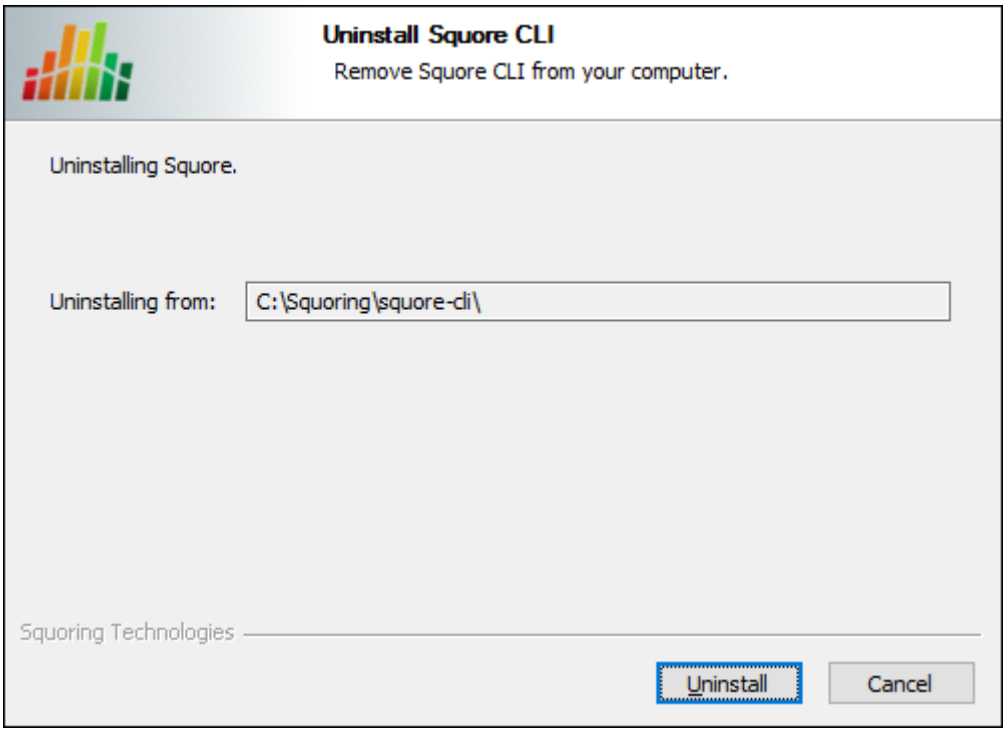
In order to upgrade Squore CLI to a new version, simply run <SQUORE_HOME>`\bin\synchronise.bat` (on Windows) or <SQUORE_HOME>`/bin/synchronise` (on Linux) script to retrieve the latest version of the binaries from Squore Server.

2.4. Removing Squore CLI

2.4.1. 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:

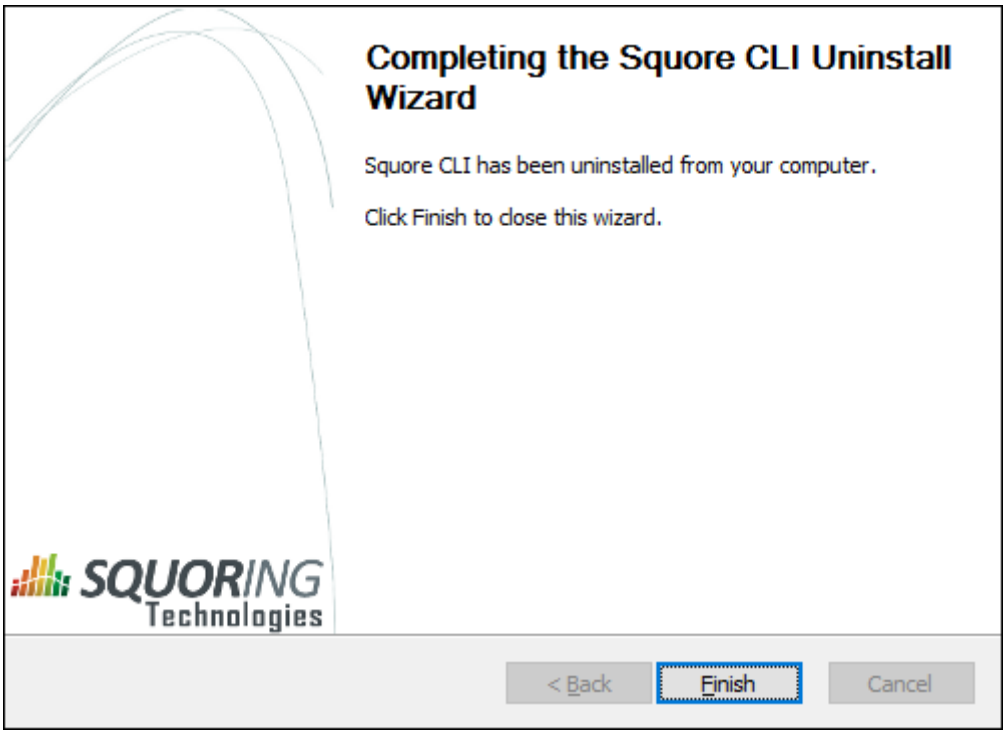


The Squore CLI uninstallation wizard

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

> **Warning**
> 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.

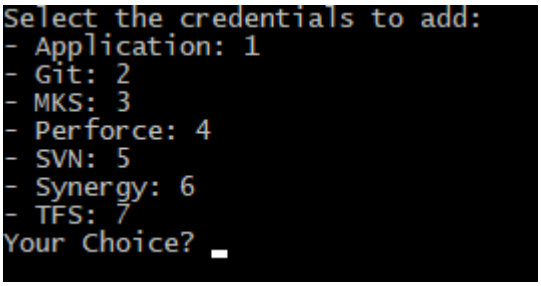
2.4.2. On Linux

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.

2.5. 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.

Note

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.

Tip

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
```

2.6. 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 Chapter 3, *Command Line Reference*

2.7. Squore in a Continuous Integration Environment

Squore can be used in a continuous integration environment using the commands detailed in Chapter 3, *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>
```

3. Command Line Reference

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

Tip

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**.

3.1. Squore CLI Commands

The following is a list of commands you can use with Squore CLI and their meaning:

- **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, it is shortcut for **PROCESS_TOOLS;GENERATE_TOOLS_DATA_ZIP;SEND_TOOLS_DATA**.
- **PROCESS_TOOLS** Generates data for the Data Providers specified in a project. It should always be called before any other generation command is called.
- **GENERATE_TOOLS_DATA_ZIP** Creates a zip archive of the data generated by the PROCESS_TOOLS command. It should be called after the PROCESS_TOOLS command.
- **SEND_TOOLS_DATA** Sends the zip archive generated by the GENERATE_TOOLS_DATA_ZIP command and the project settings to the server, to request a project creation (analysis model computation and database update). It should be called after the GENERATE_TOOLS_DATA_ZIP command.
- **DELEGATE_CREATION** Sends the project settings to the server to request a project creation.
- **ANALYZE_AND_DECIDE_DATA** Performs the analysis model and the decision model computation on the data generated by the PROCESS_TOOLS command. It should be called after the PROCESS_TOOLS command.
- **GENERATE_OUTPUT** Generates output data and statistics of the project's creation. It should always be called after all other commands.
- **DELETE_PROJECT --name=project_name** Deletes the project `project_name`. This operation cannot be undone and must be called separately from any other command.
- **DELETE_VERSIONS --name=project_name --version=version_to_delete_from** Deletes the versions `project_name` from `version_to_delete_from` until the latest version. This operation cannot be undone and must be called separately from any other command.

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'** .

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

3.2. Squore CLI Parameters

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

- **--commands or -c (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 Section 3.1, “Squore CLI Commands”.
- **--url or -s (optional, default='http://localhost:8180/SQuORE_Server'):** The URL of Squore Server.
- **--outputFile or -o (optional, default='null'):** The absolute path to the output file generated by the engine.
- **--outputCheckModelsFile or -m (optional, default='null'):**Define the absolute path to the output check models file generated by the engine.
- **--printOutput or -print (optional, default='false'):** Redirect the engine's output to the standard output.
- **--help or -? (optional, default='false'):** Display help.
- **--subFoldersAsVersions or -sub (optional, default='false'):** Loop on the repository path to create a version for each sub-folder using the sub-folder name as the version name.
- **--help:commands or -?cmd (optional, default='false'):** Display help about the available commands.
- **--projectConfFile or -x (optional, default='null'):** The XML file defining the project settings.
- **--updateModelFile or -uf (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>
```

Note

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 Section 3.3, “Project Parameters”.

Tip

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.

3.3. 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:

- **--name or -n=MyProject** defines the name of the project that will be created
- **--group=MyGroup** 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)** 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 or -b=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** instructs Squore to keep or discard analysis files from old versions or only for the latest baseline. Note that this behaviour only affects disk space on the server, not the analysis results.
- **--version or -v=v1** defines the label used for the version of this project.
- **--versionDate="YYYY-MM-DDTHH:MM:SS" (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.
- **--login or u=demo** is the ID of the user requesting the project creation.
- **--password -k=demo** is the password of the user requesting the project creation. If you do not want to specify a password in your command line, refer to Section 2.5, “Saving Credentials to Disk”.
- **--teamUser or q="mike,DEVELOPER;john,TESTER;peter,PROJECT_MANAGER"** is a semicolon-separated list of `login,roleID` pairs 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 roleIDs in Squore by clicking **Administration > Roles**. This option can be combined with the **teamGroup** parameter if needed.
- **--teamGroup or g="devUsers,DEVELOPER;management,GUEST"** is a semicolon-separated list of `group,roleID` 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 roleIDs in Squore by clicking **Administration > Roles**. This option can be combined with the **teamUser** parameter if needed.
- **--WizardId or -w=ANALYTICS** The id of the wizard used to create the project. If this parameter is not specified, it will be read from the wizard settings.
- **--rulesetTemplate="my template"** 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.
- **--versionPattern=V#.N#** defines the pattern used to label the version automatically if no version parameter was passed.
- **--tag or -t TAGNAME="tagValue"** 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 or -r "type=REPOTYPE,opt1=value1,opt2=value2"** is how you specify repository for sources. For more information about repositories syntax, refer to Chapter 4, *Repository Connectors*.

- **--dp or -d "type=DPName,dp_opt=dp_opt_value"** is how you specify information for Data Providers. For more information about individual Data Provider syntax, refer to Chapter 5, *Data Providers*.
- **--filter or -f "FILTER_OPTS"** is a semicolon-separated string of triplets {artefactType,filterType,filterValue}. In order to export the measure LC at application level and the indicator MAIN at application level, pass **-f "APPLICATION,MEASURE,LC;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** (new in 17.0), **ALL_BASE_LINKS** (new in 17.0) and **ALL_COMPUTED_LINKS** (new in 17.0) 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,;"**.
- **-M "id=BETA_RELEASE,date=2015/05/31,PROGRESS=95"** 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.

Tip

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 Chapter 4, *Repository Connectors* and Chapter 5, *Data Providers*.

Tip

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**

3.4. 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.

4. Repository Connectors

4.1. Folder Path

4.1.1. Description

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

> **Note**
> 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.

4.1.2. Usage

Folder Path has the following options:

- **Datapath (path, mandatory)** Specify the absolute path to the files you want to include in the analysis. The path specified must be accessible from the server.

The full command line syntax for Folder Path is:

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

4.2. Zip Upload

4.2.1. 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.

> **Note**
> 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.

4.2.2. Usage

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

4.3. CVS

4.3.1. 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.

Note
The following is a list of commands used by the CSV to retrieve sources:

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

4.3.2. 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]"
```

4.4. ClearCase

4.4.1. 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.

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

4.4.2. 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=[te
```

4.5. Perforce

4.5.1. 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/.

Note

The Perforce repository connector assumes that the specified depot exists on the specified Perforce server, that 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 . 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 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/...
```

4.5.2. 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=[multipleChoice],
```

4.6. Git

4.6.1. 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/.

Note

The following is a list of commands used by the Git 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`

4.6.2. 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)**
- **Username (`username`)**
- **Password (`password`)**

The full command line syntax for Git is:

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

4.7. PTC Integrity

4.7.1. 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/.

> **Note**
> 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.

4.7.2. 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],useAccountCr
```

4.8. TFS

4.8.1. 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.

> **Note**
> The TFS repository connector (Team Foundation Server - Team Foundation Version Control) assumes that a TFS command-line client (Visual Studio Client or Team Explorer Everywhere) is installed on the server and fully functional. The configuration of this client must be set up in the 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 to retrieve sources:

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

Note

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

4.8.2. 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],usernar
```

4.9. Synergy

4.9.1. 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.

Note

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, as stated in IBM's documentation on 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, using credentials is only supported on Windows, so use the NO_CREDENTIALS option when Synergy runs on a Linux host. The following is a list of commands used by the Synergy 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`

4.9.2. 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.
- **Authentication: (`useAccountCredentials` , default: NO_CREDENTIALS)** Note that, as stated in IBM's documentation, using credentials is only supported on Windows. The "No Credentials" must be used option 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.
- **(`name`)**
- **Password (`password`)**

The full command line syntax for Synergy is:

```
-r
 "type=Synergy,server=[text],db=[text],projectSpec=[text],subFolder=[text],useAccountCredentia
```

4.10. SVN

4.10.1. 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/.

> **Note**
> The following is a list of commands used by the SVN to retrieve sources (you can edit the common command base or the path to the executable in `/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`

4.10.2. 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.
- **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],useAccountCredentials=[multipleCho
```

4.11. 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"
```

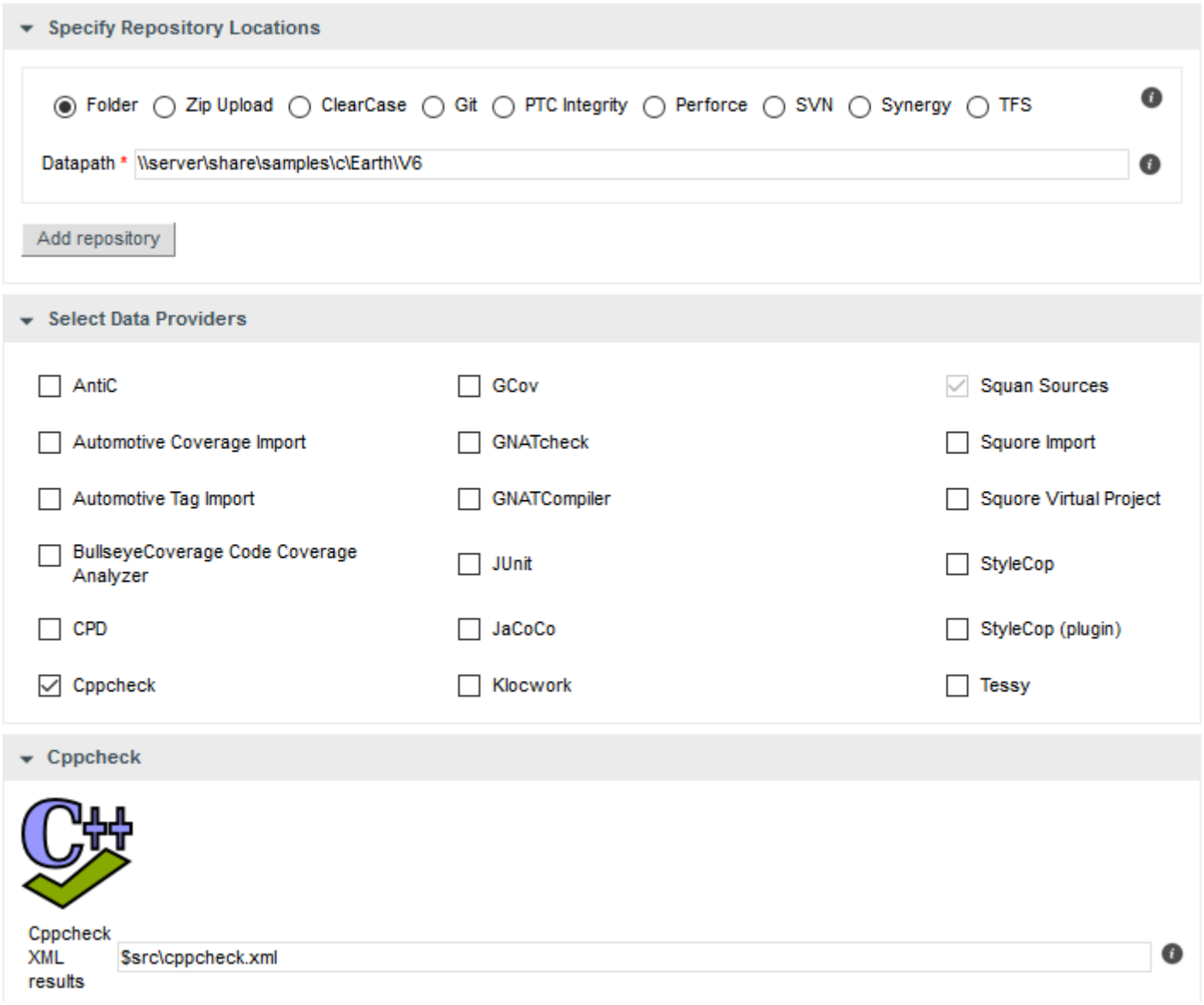
Note that only alpha-numeric characters are allowed to be used as labels. In the artefact tree, each node will appear as a separate top-level folder with the label provided at project creation.

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"
```

4.12. 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.

Tip

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"
```

5. Data Providers

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

5.1. AntiC

5.1.1. 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/.

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

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

5.1.2. 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=[text]"
```

5.2. Automotive Coverage Import

5.2.1. Description

Automotive Coverage Import: generic import mechanism for coverage results at FUNCTION level

5.2.2. Usage

Automotive Coverage Import has the following options:

- **Enter the CSV file for coverage measures (`csv`)** CSV File shall contain the following (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=[text]"
```

5.3. Automotive Tag Import

5.3.1. Description

5.3.2. Usage

Automotive Tag Import has the following options:

- **Enter the CSV file for measures (`csv`)**

The full command line syntax for Automotive Tag Import is:

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

5.4. BullseyeCoverage Code Coverage Analyzer

5.4.1. 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/.

5.4.2. 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=[text]"
```

5.5. CPD

5.5.1. 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.

5.5.2. 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=[text]"
```

5.6. Cppcheck

5.6.1. 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/.

5.6.2. 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=[text]"
```

5.7. Cppcheck (plugin)

5.7.1. 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/.

> **Note**
> On Windows, this data provider requires an extra download to extract the Cppcheck binary in `/addons/tools/CPPCheck_auto/`. 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.

5.7.2. 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.

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

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

5.8. CPPTest

5.8.1. Description

Parasoft C/C++test 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/.

5.8.2. Usage

CPPTest has the following options:

- **XML results file (`xml`)** Specify the path to the CPPTest results file. This data provider is compatible with files exported from CPPTest version 7.2.10.34 and up.

The full command line syntax for CPPTest is:

```
-d "type=CPPTest,xml=[text]"
```

5.9. Cantata

5.9.1. Description

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

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

5.9.2. 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=[text]"
```

5.10. CheckStyle

5.10.1. 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/.

5.10.2. 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=[text]"
```

5.11. CheckStyle (plugin)

5.11.1. 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/.

> **Note**
> This data provider requires an extra download to extract the CheckStyle binary in `/addons/tools/CheckStyle_auto/`.

5.11.2. 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]"
```

5.12. CheckStyle for SQALE (plugin)

5.12.1. 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/.

> **Note**
> This data provider requires an extra download to extract the CheckStyle binary in `/addons/tools/CheckStyle_auto_for_SQALE/`.

5.12.2. 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]"
```

5.13. Cobertura

5.13.1. 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/.

5.13.2. Usage

Cobertura has the following options:

- **XML report (`xml`)** Specify the path to the XML report generated by Cobertura.

The full command line syntax for Cobertura is:

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

5.14. CodeSonar

5.14.1. 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.

5.14.2. 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=[text]"
```

5.15. Compiler

5.15.1. Description

Compiler Warning impor allows to import information from compiler

For more details, refer to Compiler.

5.15.2. Usage

Compiler has the following options:

- **Compiler output csv file (Path;Line;Rule;Descr - with: Rule = COMP_ERR|COMPILER_WARN|COMPILER_INFO) (`txt` , mandatory)**

The full command line syntax for Compiler is:

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

5.16. Coverity

5.16.1. 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/.

5.16.2. 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=[text]"
```

5.17. FindBugs

5.17.1. 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.

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

5.17.2. Usage

FindBugs has the following options:

- **XML results file (`xml`)** Specify the location of the XML file containing Findbugs results. Note that the minimum supported version of FindBugs is 1.3.9.

The full command line syntax for FindBugs is:

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

5.18. FindBugs (plugin)

5.18.1. 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_auto/. 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.

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

> **Note**
> This data provider requires an extra download to extract the Findbugs binary in `/addons/tools/Findbugs_auto/`.

5.18.2. Usage

FindBugs (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 via the -auxclasspath parameter.
- **Memory Allocation (`xmx` , default: 1024m)** Maximum amount of memory allocated to the java process launching FindBugs.

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

```
-d "type=Findbugs_auto,class_dir=[text],auxiliarypath=[text],xmx=[text]"
```

5.19. Function Relaxer

5.19.1. Description

5.19.2. Usage

Function Relaxer has the following options:

- **Enter the CSV file for measures (`csv`)**

The full command line syntax for Function Relaxer is:

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

5.20. FxCop

5.20.1. 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.

5.20.2. 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=[text]"
```

5.21. GCov

5.21.1. 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.

5.21.2. 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=[text],ext=[text]"
```

5.22. GNATcheck

5.22.1. 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 that can be imported to generate findings.

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

5.22.2. Usage

GNATcheck has the following options:

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

The full command line syntax for GNATcheck is:

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

5.23. GNATCompiler

5.23.1. 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/.

5.23.2. 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=[text]"
```

5.24. JUnit

5.24.1. 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 to generate findings and the total number of tests is made available as a measure.

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

5.24.2. Usage

JUnit has the following options:

- **Results folder (`resultDir` , mandatory)** Specify the path to the folder containing the JUnit results. The data provider will parse all available XML files. Note that the minimum support version of JUnit is 4.10.

The full command line syntax for JUnit is:

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

5.25. JaCoCo

5.25.1. 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/.

5.25.2. 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=[text]"
```

5.26. Klocwork

5.26.1. 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.

5.26.2. 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=[text]"
```

5.27. Rational Logiscope

5.27.1. 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.

5.27.2. 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=[text]"
```

5.28. MemUsage

5.28.1. Description

5.28.2. Usage

MemUsage has the following options:

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

The full command line syntax for MemUsage is:

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

5.29. NCover

5.29.1. 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/.

5.29.2. 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=[text]"
```

5.30. Oracle PLSQL compiler Warning checker

5.30.1. 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/.

5.30.2. 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=[text],prefix=[text]"
```

5.31. MISRA Rule Checking using PC-lint

5.31.1. Description

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

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

5.31.2. Usage

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

- **Log file folder (`logDir`)** Specify the path to the folder containing the PC-lint log files.
- **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=[text],excludedExtensions=[text]"
```

5.32. PMD

5.32.1. 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.

5.32.2. 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=[text]"
```

5.33. PMD (plugin)

5.33.1. 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.

> **Note**
> This data provider requires an extra download to extract the PMD binary in `/addons/tools/PMD_auto/`.

5.33.2. 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_autow ill be used.

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

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

5.34. Polyspace

5.34.1. 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.

5.34.2. Usage

Polyspace has the following options:

- **XML results file (`xml`)** Specify the path to the XML results file generated by Polyspace.

The full command line syntax for Polyspace is:

```
-d "type=Polyspace,xml=[text]"
```

5.35. Polyspace MISRA

5.35.1. 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.

5.35.2. Usage

Polyspace MISRA has the following options:

- **Results folder (`resultDir`)** Specify the folder containing the Polyspace results. The data provider will parse all sub-folders searching for XML result files called "MISRA-CPP-report.xml" or "MISRA-C-report.xml" located in a "Polyspace-Doc" folder and aggregate results.
- **Unit by Unit (`unitByUnit` , default: true)** Check this box if the Polyspace verification was run unit by unit.

The full command line syntax for Polyspace MISRA is:

```
-d "type=Polyspace_MISRA,resultDir=[text],unitByUnit=[booleanChoice]"
```

5.36. Polyspace (plugin)

5.36.1. Description

Polyspace is a static analysis tool which includes a MISRA checker. It produces an binary output format 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. Note that this data provider requires an extra download to extract the Polyspace Export binary in [INSTALLDIR]/addons/tools/Polyspace_RTE/. For more information, refer to the Installation and Administration Manual's "Third-Party Plugins and Applications" section.

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

> **Note**
> This data provider requires an extra download to extract the Polyspace Export binary in `/addons/tools/Polyspace_RTE`.

5.36.2. Usage

Polyspace (plugin) has the following options:

- **Results folder (`resultDir`)** Specify the folder containing the Polyspace results. The data provider will run the polyspace-export binary on all sub-folders to export results to XML and aggregate them.
- **Unit by Unit (`unitByUnit` , default: true)** Check this box if the Polyspace verification was run unit by unit.

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

```
-d "type=Polyspace_RTE,resultDir=[text],unitByUnit=[booleanChoice]"
```

5.37. MISRA Rule Checking with QAC

5.37.1. 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.

5.37.2. 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.

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

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

5.38. Unit Test Code Coverage from Rational Test RealTime

5.38.1. Description

Rational Test RealTime is a cross-platform solution for component testing and runtime analysis of embedded software. Metrics are generated from its CSV results file.

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

5.38.2. Usage

Unit Test Code Coverage from 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)**

The full command line syntax for Unit Test Code Coverage from Rational Test RealTime is:

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

5.39. ReqIF

5.39.1. 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/.

5.39.2. Usage

ReqIF has the following options:

- **(`dir`)**
- **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=[text],objType=[text]"
```

5.40. SQL Code Guard

5.40.1. 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.

5.40.2. Usage

SQL Code Guard has the following options:

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

The full command line syntax for SQL Code Guard is:

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

5.41. Squan Sources

5.41.1. Description

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

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

Note

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.

Tip

You can specify the languages for your source code by passing pairs of language and extensions to the **languages** paramater. For example, a project mixing php and javascript files can be analysed with:

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

5.41.2. Usage

Squan Sources has the following options:

- **Languages (`languages` , default: abap;ada;c;cpp;mindc;csharp;cobol;java;javascript;fortran77;fortran90;php;s...)** 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.
- **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.

- **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.
- **Backwards-compatible cloning (`clBw` , default: false)** When checking this box, the cloning detection tool is run in a way that produces metrics that are backwards-compatible with earlier versions of this product (2014-A): exact matching is used for algorithmic cloning and a 5% margin is used for text duplication. This legacy behaviour should only be used if you are using an old configuration that was developed before 2014-B.
- **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 cloning detection tool. For example, a fault ratio of 0.1 means that two artefacts are considered clones if less than 10% of their contents differ. Note that this option is ignored if you are using backwards-compatible cloning.
- **Detect Open Source cloning (deprecated) (`clOS` , default: false)** This option is no longer supported and should not be used anymore.
- **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.
- **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=[booleanChoice],qua
```

5.42. Squore Import

5.42.1. 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 http://www.squoring.com.

5.42.2. 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=[text]"
```

5.43. Squore Virtual Project

5.43.1. 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 http://www.squoring.com.

5.43.2. 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=[text]"
```

5.44. StyleCop

5.44.1. 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/.

5.44.2. 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=[text]"
```

5.45. StyleCop (plugin)

5.45.1. 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
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 `/addons/tools/StyleCop_auto/`.

5.45.2. 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=[text]"
```

5.46. Tessy

5.46.1. 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/.

5.46.2. 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=[text]"
```

5.47. VectorCAST 6.3

5.47.1. Description

VectorCAST 6.3

For more details, refer to VectorCAST 6.3.

5.47.2. Usage

VectorCAST 6.3 has the following options:

- **HTML Report (`html_report`)** Enter the path to the HTML report which contains the Coverage results

The full command line syntax for VectorCAST 6.3 is:

```
-d "type=VectorCAST,html_report=[text]"
```

5.48. CodeSniffer

5.48.1. Description

CodeSniffer is a rulecker for PHP and Javascript

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

5.48.2. Usage

CodeSniffer has the following options:

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

The full command line syntax for CodeSniffer is:

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

5.49. Configuration Checker

5.49.1. Description

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

5.49.2. 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=[text],p=[text]"
```

5.50. Csv Coverage Import

5.50.1. Description

Csv Coverage Import: generic import mechanism for coverage results at FUNCTION level

5.50.2. Usage

Csv Coverage Import has the following options:

- **Enter the CSV file for coverage measures (`csv`)** CSV File shall contain the following (PATH;NAME;TESTED_C1;OBJECT_C1;TESTED_MCC;OBJECT_MCC;TESTED_MCDC;OBJECT_MCDC;TCOV_MCC;TCOV_MCDC;T

The full command line syntax for Csv Coverage Import is:

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

5.51. CSV Findings

5.51.1. Description

CSV Findings (Generic Import of findings)

5.51.2. Usage

CSV Findings has the following options:

- **CSV File (FILE;FUNCTION;RULE_ID;MESSAGE;LINE;COL;STATUS;STATUS_MESSAGE;TOOL) (`csv`)** Your CSV file shoul use include the following as a header: FILE;FUNCTION;RULE_ID;MESSAGE;LINE;COL;STATUS;STATUS_MESSAGE;TOOL. CSV files in other formats are not supported.

The full command line syntax for CSV Findings is:

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

5.52. Csv Tag Import

5.52.1. Description

Csv Tag Import

5.52.2. Usage

Csv Tag Import has the following options:

- **Enter the CSV file for measures (`csv`)**

The full command line syntax for Csv Tag Import is:

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

5.53. Csv Test Results Import

5.53.1. Description

Csv Test Results Import: generic import mechanism for Test results at FILES level

5.53.2. Usage

Csv Test Results Import has the following options:

- **Enter the CSV file for Test Results measures at FILES level (`csv`)** CSV File shall contain the following (PATH;NB_TEST;NB_ERROR;NB_FAILURE;NB_PASS)

The full command line syntax for Csv Test Results Import is:

```
-d "type=csv_test,csv=[text]"
```

5.54. OSLC

5.54.1. 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/.

5.54.2. 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=[password]"
```

5.55. pep8

5.55.1. 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.

5.55.2. 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=[text]"
```

5.56. pep8 (plugin)

5.56.1. Description

Style Guide for Python Code. Pep8 results are imported to produce findings on Python code. This data provider requires having pep8 installed on the machine running the analysis and the pep8 command to be available in the path. It is compatible with pep8 1.4.6 and may also work with older versions.

5.56.2. Usage

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 pep8 (plugin) is:

```
-d "type=pep8_auto,dir=[text]"
```

5.57. PHP Code Coverage

5.57.1. Description

PHP Code Coverage

For more details, refer to https://github.com/sebastianbergmann/php-code-coverage.

5.57.2. Usage

PHP Code Coverage has the following options:

- **HTML Report Folder (`html_report`)** Enter the path to the HTML report folder which contains the Coverage results

The full command line syntax for PHP Code Coverage is:

```
-d "type=phpcodecoverage,html_report=[text]"
```

5.58. pylint

5.58.1. 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/.

5.58.2. 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=[text]"
```

5.59. pylint (plugin)

5.59.1. 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.

5.59.2. 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=[text]"
```

5.60. Qac_8_2

5.60.1. 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/.

5.60.2. Usage

Qac_8_2 has the following options:

- **QAC output file (.tab file) (`txt` , mandatory)**

The full command line syntax for Qac_8_2 is:

```
-d "type=qac,txt=[text]"
```

5.61. 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.

5.62. Creating your own Data Providers

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 data other 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 Word Files
- Parse HTML test results
- Query web services
- Export data from OSLC systems
- Launch external processes

This section describes two ways to add your own Data Providers to Squore:

1. Using one of the generic, built-in Data Provider frameworks. Each solution uses a different approach, but the overall goal is to produce one or more CSV files that your Data Provider will send to Squore to associate metrics, findings, textual information or links to artefacts in your project.
2. Writing your own utility to generate the XML files expected by Squore so they can be imported as part of the analysis result (new in 17.0).

Tip
If you are interested in developping Data Providers that go beyond the scope of what is described in this manual, consult Squoring Technologies to learn more about the available training courses in writing Data Providers.

5.62.1. Choosing the Right Data Provider Framework

The following is a list of the available Data Provider 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

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.

5.62.2. Extending a Framework

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 Section 5.62.4, “Data Provider Parameters”
   - **form_en.properties** contains the strings displayed in the web interface for this Data Provider, as described in Section 5.62.5, “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.

5.62.3. Creating a Freestyle Data Provider

Instead of using one of the Data Provider frameworks, you can directly produce your results in an XML format that Squore can read and import (new in 17.0). The syntax of the XML file to generate is as follows:

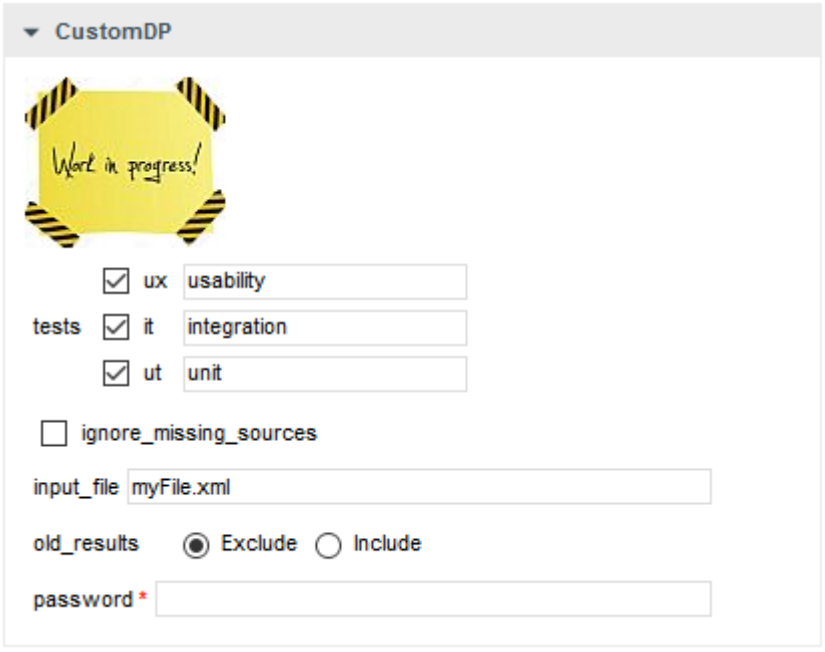
```
input-data.xml:
<bundle version="2.0">
 <artifact [local-key=""] [local-parent=""|parent=""]>
        <artifact [id="<guid-stable-in-time-also-a-key>"] name="Component"
 type="REQ" [location=""]>
   <info name|n="DESCR" value="The description of the object"/>
   <key value="3452-e89b-ff82"/>
   <metric name="TEST_KO" value="2"/>
   <finding name="AR120" loc="xxx" p0="The message" />
   <link name="TEST" local-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>
```

`input-data.xml` must be written in a specific location by an executable or a script declared in the Data Provider's `form.xml` in an `exec-phase` element, as described in Section 5.62.4, “Data Provider Parameters”.

5.62.4. 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" optionTitle=" " key="tests">
  <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="text" key="input_file" defaultValue="myFile.xml" changeable="false" /
>
 <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)** indicates which framework you are basing this Data Provider on
- **`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 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
  - **password** for password fields
  - **booleanChoice** for a boolean
  - **multipleChoice** for offering a selection of predefined values

> **Note**
> 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.

- **`displayType` (optional)** allows specifying how to display a `multipleChoice` parameter by using one of:
  - **comboBox**
  - **radioButton**
  - **checkbox**
- **`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. Note that this is only a visual aid at the moment and cannot be used to force users to enter a value for the parameter.

In order to create a freestyle Data Provider, your `form.xml` must contain an `exec-phase` element with a mandatory `id="add-data"` attribute (new in 17.0):

```
<exec-phase id="add-data">
  <exec name="tcl|perl|java|javascript or nashorn" | executable="/path/to/bin">
    <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-phase>
```

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**
- **javascript or nashorn**

The scripts are launched using the tcl, perl, or java runtimes defined in your Squore installation. This is also the case for javascript, which is handled by Java's Nashorn engine. Alternatively, you can call an executable directly by specifying its absolute path using the `executable` attribute.

Argument values can be:

1. Free text, useful to specify a parameter for your script
2. A tag `key` declared in `form.xml` to retrieve the input specified by the user
3. One of the predefined functions
   - **getToolConfigDir(<relative/path>)** to get the absolute path of the Data Provider's configuration folder
   - **getToolAddonsDir(<relative/path>)** to get the absolute path of the Data Provider's addons folder

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`

5.62.5. 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`.

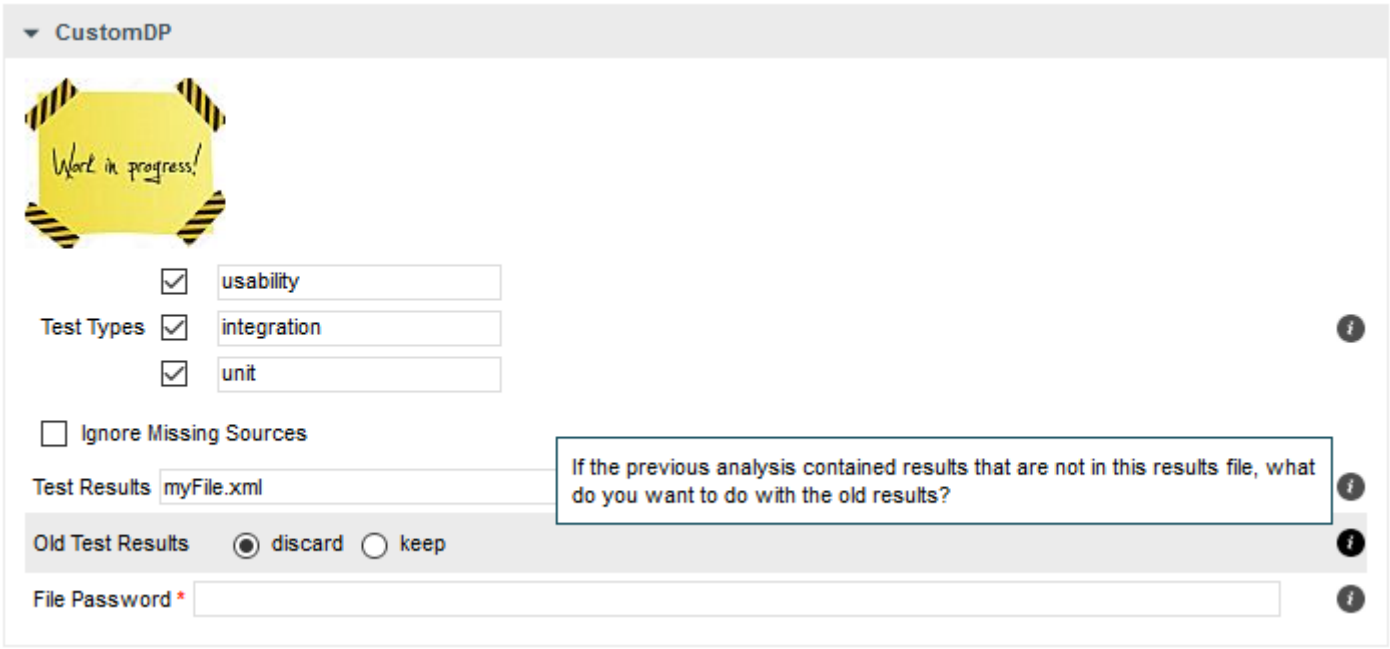
When you create a Data Provider, it is mandatory to include at least an English version of the strings in a file called `form_en.properties`. You are free to add other languages as needed. Here is a sample `.properties` for for the CustomDP you created in the previous section:

```
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`

Appendix A. Reference pages

Name

install — Squore CLI install 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` 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`	(default: *`http://localhost:8180/SQuORE_Server`*) The URL of Squore Server that Squore CLI will connect to after installation.
`-u user`	(default: *`demo`*) The username to use to connect to Squore Server.
`-p password`	(default: *`demo`*) The username to use to connect to Squore Server.
`-N`	Do not synchronise client with server
`-v`	Turn on verbose mode

Appendix B. Data Provider Frameworks

```
=======
= 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 =
================

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 \"
```

```
===========
= 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 =
===========

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
```

```
# 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}
}


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 =
===============

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 =
================

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 =
================

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.
```

Index

Symbols

* What's New in Squore 17.0?
    New filter parameters allow including findings and links in output XML files, 21
    Write your own Data Provider to produce XML that can be directly read by Squore, 62, 65

A

add-credentials.sh, 16
additional_param, 52
Advanced config.xml Configuration, 13
Analysis Model Editor
    Override the ruleset from the command line at project creation, 20
arg, 65
auxiliarypath, 41

B

baseName, 64
branch, 25

C

changeable, 65
clAlg, 52
class_dir, 41
clBw, 52
clFR, 52
clOS, 52
clRen, 52
clTxt, 52
COBOL, 59
commit, 27
compact_folder, 51
configFile, 38, 38, 47
Configuration
    Configuring the client's work folders, 13
    Using one Squore CLI installation for multiple users, 14
Continuous Integration, 17
csv, 34, 35, 41, 44, 55, 56, 56, 56, 57, 58

D

Data Providers
    AntiC, 34
    Automotive Coverage Import, 34
    Automotive Tag Import, 34
    BullseyeCoverage Code Coverage Analyzer, 35
    Cantata, 37
    CheckStyle, 37
    CheckStyle (plugin), 37
CheckStyle for SQALE (plugin), 38
Cobertura, 39
CodeSniffer, 55
CodeSonar, 39
Compiler, 39
Configuration Checker, 55
Coverity, 40
CPD, 35
Cppcheck, 35
Cppcheck (plugin), 36
CPPTest, 36
Csv, 60, 69
csv_findings, 60, 72
Csv Coverage Import, 55
CSV Findings, 56
CsvPerl, 60, 73
Csv Tag Import, 56
Csv Test Results Import, 56
ExcelMetrics, 60, 86
FindBugs, 40
FindBugs (plugin), 40
FindingsPerl, 60, 82
Frameworks, 60
Function Relaxer, 41
FxCop, 41
GCov, 42
Generic, 60, 75
GenericPerl, 60, 79
GNATcheck, 42
GNATCompiler, 43
JaCoCo, 43
JUnit, 43
Klocwork, 44
MemUsage, 44
MISRA Rule Checking using PC-lint, 46
MISRA Rule Checking with QAC, 48
NCover, 45
Oracle PLSQL compiler Warning checker, 45
OSLC, 57
pep8, 57
pep8 (plugin), 57
PHP Code Coverage, 58
PMD, 46
PMD (plugin), 46
Polyspace, 47
Polyspace (plugin), 48
Polyspace MISRA, 47
pylint, 58
pylint (plugin), 59
Qac_8_2, 59
Rational Logiscope, 44

ReqIF, 49
SQL Code Guard, 50
Squan Sources, 50
Advanced COBOL parsing, 59
Squore Import, 52
Squore Virtual Project, 53
StyleCop, 53
StyleCop (plugin), 53
Tessy, 54
Unit Test Code Coverage from Rational Test RealTime, 49
VectorCAST 6.3, 54
db, 30
defaultValue, 65
depot, 26
depth, 51
dir, 34, 36, 42, 50, 58, 59
dir_choice, 51
Disk Space, 4
displayType, 65

E

env, 65
excel, 45
excludedDirectoryPattern, 38
excludedExtensions, 46, 49
exec, 65
exec-phase, 63, 65, 65
executable, 65
ext, 42, 49
externals, 31

F

files_choice, 51
Filters
ALL_BASE_FINDINGS, 21
ALL_BASE_LINKS, 21
ALL_COMPUTED_LINKS, 21
ALL_DEFECT_REPORTS, 21
ALL_INDICATORS_LEVELS, 21
ALL_INDICATORS_RANKS, 21
ALL_MEASURES, 21
ALL_TYPES, 21

G

genAs, 52
genCG, 51
genTs, 52

H

hide, 65
hostname, 28
html, 35
html_report, 54, 58

I

id="add-data", 65
image, 64
inputDir, 53
Installer
Linux, 12
Windows, 7

J

Java, 4

K

key, 64, 64, 65, 66

L

label, 26
languages, 51
Languages
COBOL, 59
log, 43, 45
logDir, 46, 49, 49
login, 57

M

Memory, 4, 4
Milestone Management, 21
Milestones, 22
multipleChoice, 65
Multiple Users, 13

N

name, 30
needSources, 64

O

objType, 50
option, 64
output, 53

P

p, 55
p4port, 26
password, 26, 27, 28, 29, 30, 31, 57
path, 29
pattern, 51
pattern_dir, 51
pattern_files, 51
port, 28
prefix, 45
Prerequisites, 4

project, 25, 28
projectSpec, 30
projectStatusOnFailure, 64
projectStatusOnWarning, 64
properties, 57

Q

qualified, 51
query, 57

R

rebuild_all, 51
repository, 25
Repository Connectors
  ClearCase, 25
  CVS, 24
  Folder Path, 24
  Git, 27
  Multiple Source Nodes, 31
  Perforce, 26
  PTC Integrity, 27
  SVN, 30
  Synergy, 29
  TFS, 28
  Zip Upload, 24
required, 65
resultDir, 43, 48, 48, 54
rev, 31
revision, 28
Ruleset Templates, 21

S

s, 55
scnode, 51
scnode_name, 51
scope, 28
server, 30, 57
server_display_view, 25
sln, 54
style, 65
sub_path, 25
subDir, 27
subFolder, 30

T

tag, 64
tags, 64
txt, 40, 42, 59
type, 64

U

unitByUnit, 48, 48
url, 27, 31
URL, 29
useAccountCredentials, 26, 27, 28, 29, 30, 31
username, 26, 27, 28, 29, 31

V

value, 64
version, 29
Version Date, 21
versionPattern, 22
view, 25
view_root_path, 25
vob_root_path, 25

W

Wizard, 22

X

xml, 35, 36, 37, 37, 37, 39, 39, 40, 40, 42, 44, 44, 45, 46, 47, 50, 53, 55
xmx, 38, 38, 41