SonargraphBuild User Manual

Version 9.10.0

SonargraphBuild User Manual: Version 9.10.0

Table of Contents

Chapter 1. Sonargraph's Next Generation - SonargraphBuild

SonargraphBuild integrates quality checks into the continuous integration build and can create XML and HTML reports via an Ant task or shell scripts. These reports contain all information about quality issues and calculated metrics. The XML report can be used for further downstream processing via transformations. The XML schema for the report can be found in <sonargraphBuild-inst>/doc.

SonargraphBuild additionally offers the possibility to mark the build as failed based on issues detected during the analysis. So, if you have written custom queries via Groovy scripts that check on the proper usage of an external library or detect a code smell, you can be sure that it is detected immediately.

Chapter 2. Licensing

When you start *Sonargraph* you will be asked for an activation code or a license file. For additional licensing and pricing information please contact <sales@hello2morrow.com> or <support@hello2morrow.com> and check our *web site* .

2.1. Getting an Activation Code or a License

When you have purchased a *Sonargraph* license, an activation code or a license file will be delivered to you.

There might be a program for free *Sonargraph* licenses which are time-limited and/or size-limited. Please register on our website and check the available programs.

In order to replace a valid license by a new one, choose "Help" → "Manage License..." from the user menu in the GUI-based product. *Sonargraph* licenses are bound to a named user. The usage by a different user is a violation of the license agreement.

2.2. Activation Code Based Licensing

Activation code based licensing activates *Sonargraph* licenses via Internet or a local license server by requesting a so-called ticket. Every activation code is customer specific and represents a pool of *Sonargraph* user licenses as purchased and licensed to the specific customer. Activation code based licensing technically requires that *Sonargraph* has Internet access or that a local license server is reachable. There are two types of activation code based licenses available:

1. Flexible User License (if you bought Sonargraph before version 9.0 you have flexible user licenses)

2. Floating License (new with Sonargraph 9.0)

Flexible user licenses support a feature that allows customer-driven transfer of a *Sonargraph* user license to another user after some amount of time. This works like this:

- When an activation code based license is requested, *Sonargraph* automatically requests a license ticket from the hello2morrow license server. This ticket expires after some time, for example after 30 days. During these 30 days, the use of the *Sonargraph* installation that requested the ticket is licensed (by the user who ran *Sonargraph* when the license ticket was requested). *Sonargraph* can be used during this period without any access to the Internet.

- After the ticket of a *Sonargraph* installation has expired (in our example scenario, this happens on the 31st day after the ticket has been requested), one of two things typically happen:

  1. The same *Sonargraph* installation is started again. *Sonargraph* then notices that the license ticket has expired and lets the user know about it by presenting a dialog to manually request a new ticket from the hello2morrow license server, for the same activation code or a different one if desired. The new ticket again is valid for the same time period. You can toggle the feature at ' Help → Renew License Ticket Automatically ' to have *Sonargraph* silently perform license ticket requests using the current activation code, without further user interaction.

  2. Alternatively, the user of the installation might not continue to work with *Sonargraph*; then the license is now, after the expiration of the ticket in the *Sonargraph* installation, available to some other user. The hello2morrow license server will supply a license ticket to the next user that requests one for the given activation code.

Note that the number of license tickets that can be supplied by the license server for some activation code might be more than one. For example, a company might license *Sonargraph* for 20 users. The same activation code can be used by all of them, but as soon as the 21st license ticket is requested for this activation code, this request will be denied. A new request for a ticket will only be fulfilled after one of the already supplied tickets has expired, so that at any one moment, at most 20 non-expired license tickets exist for the activation code.

It is not required that the same user requests a replacement of an expired license ticket; any user that knows the activation code can request one of the free tickets. This mechanism reduces the effort needed for license management in a changing user group.

However, in order to avoid any misuse we strongly encourage you to restrict the information about your activation code to those persons who are supposed to use *Sonargraph*.

If you have any suspicion about misuse please inform <support@hello2morrow.com> immediately. We can promptly deactivate an activation code so that any further misuse is stopped and provide a new activation code to you.

Floating licenses bind a ticket to an instance of Sonargraph while it is running. As soon as Sonargraph is terminated the license can be used by another user.

Most of our customers are using our Internet based license server, so there is no need for you to operate your own license server as long as the machines running Sonargraph have access to the Internet. If this is not the case or you want to avoid being dependent on the availability of hello2morrow's web-based license server you can request the usage of a local license server by contacting us via <sales@hello2morrow.com> or <support@hello2morrow.com>. Once your request has been approved, you can download hello2morrow's local license server and run it on your premises. If you have a *flexible user license* it is also possible to run Sonargraph with file based licenses.

2.3. Proxy Settings

If you use hello2morrow's Internet servers and Activation code based licensing, you need Internet access. If your network configuration does not allow direct Internet access, but provides access through an HTTP proxy instead, you can specify the host name and port of the proxy server. If the proxy server access is password protected, you can supply a user name and a password in order to authenticate.

For the GUI-based product, the proxy settings can be changed via "Preferences..." → "Proxy Settings" .

Check the user manual of SonargraphBuild for proxy configuration options of the build server integrations.

2.4. License Server Settings

I you use your own license server you need to configure the access to it. You must specify the host name and port of the license server.

For the GUI-based product, the proxy settings can be changed via "Preferences..." → "License Server Settings" .

Chapter 3. Getting Started

This chapter summarizes what is needed for *SonargraphBuild* to run.

3.1. Installation Requirements

The following prerequisites must be fulfilled for *SonargraphBuild* :

1. Microsoft™ Windows™ , Mac OS-X or Linux® operating system.
2. Java Runtime Environment 1.8 or higher
3. At least 2048 MB RAM (Win32: 1400 MB)

   NOTE

   Sonargraph keeps all information in main memory. For very large systems, you need to increase the memory for the JVM in case you run into out of memory exceptions.

3.2. Prerequisites

1. If you plan to run *SonargraphBuild* via ANT or the command line you need to download it from our web site: *https://www.hello2morrow.com/products/downloads* and extract the Zip file to a convenient location. If you are using Maven for your build process you only need to install *SonargraphBuild* if your build server has no Internet connectivity.
2. If the machine that executes *SonargraphBuild* has internet access, use an activation code parameter to obtain a ticket from your pool of licenses. If the machine does not have internet access, you need to obtain a license file and pass the location of this file as a parameter to your build.
3. For integration with Shell script or Ant a "software system" must have been created via Sonargraph containing a valid workspace configuration including modules and root directories.

   Integrations with Maven and Gradle allow to dynamically create a "software system" on the fly and create a report for it. Those integrations can be used to create an initial software system that is refined using Sonargraph rich-client application.

Chapter 4. Executing from the Command-line

SonargraphBuild can be executed as a standalone Java application which enables the integration in any kind of continuous integration environment. The necessary configuration is straight-forward and an example shell script is provided in the directory <inst-dir>/example/bin. The batch script starts *SonargraphBuild* and specifies an XML file for the detailed configuration.

NOTE

SonargraphBuild returns an exit code indicating the execution status:

- 0 : Successful execution
- 1 : Execution failed because of failset properties
- 2 : Execution failed because of handled exception, e.g. configuration error, license error, etc.
- 3 : Execution failed because of unexpected exception. Please check the log file for details.

NOTE

The attribute "logLevel" affects the logging after the SonargraphBuild engine has been started. Setting it to "debug" and below will generate additional debug information into the report.

WARNING

Setting the attribute "reportType" to "standard" only generates metric values for "system" and "module" levels. "full" generates values for all element levels, but results in a significant larger report!

The splitting of the HTML report into different files is controlled by the attribute "elementCountToSplitHtmlReport". The resulting detail pages contain tables of issues / resolutions of a specific issue type. The maximum number of items shown is limited by the parameter "maxElementCountForHtmlDetailsPage".

WARNING

The attribute "qualityModelFile" can be used to apply a fixed set of scripts, architecture files and analyzer configurations across several systems.

The reported issues are most likely very different from the results of analyzing the system with the Sonargraph application. If you specify a "qualityModelFile", it is advisable to specify the "Parser" virtual model, since resolutions e.g. for architecture violations probably will not match.

4.1. Report Creation

The following table lists all parameters that are available to create a report for an existing Sonargraph system:

Attribute	Mandatory	Description	Default
installationDirectory	Yes	Installation directory of SonargraphBuild	No default
activationCode	No	Sonargraph license activation code. If this parameter is not specified, you must specify a license file parameter (see below).	No default
licenseServerPort	No	Port of license server to be used. This parameter is ignored if licenseServerHost is not set.	8080
licenseServerHost	No	Host name or IP address of license server. If this parameter is not specified, the web-based	No default

Attribute	Mandatory	Description	Default
		hello2morrow license server will be used for activation code based licenses.	
licenseFile	No	Sonargraph license file location. If this parameter is not specified, you must specify the activation code parameter (see above).	No default
languages	No	The languages that should be initialized, separated by ",". The fewer languages are specified the faster the startup of SonargraphBuild. If no value is specified, all languages will be initialized.	Java, CSharp, CPlusPlus
logFile	No	Path of the log file to be used for SonargraphBuild.	${currentDir}/ sonargraph_build.log
logLevel	No	Level of logging detail. One of: off, error, warn, info, debug, trace, all	info
compilerDefinitionPath	No	The path to the active compiler definition file to be used for parsing a C/C++ system. If a built-in or automatically generated definition should be used, prefix the definition with "CPlusPlus:", e.g. CPlusPlus:GnuCpp.cdef. **NOTE:** If the standalone Sonargraph application is used on the same machine with the same user and this parameter is empty, the active definition specified with the standalone application will be used.	If empty, the default compiler definition for the build server's operating system is used: GnuCpp.cdef (Linux), CLang.cdef (Mac), VisualCpp*.cdef (Windows, generated definition for the latest Visual Studio installation)
systemDirectory	Yes	Directory of the Sonargraph System (xyz.sonargraph)	No default
virtualModel	No	The virtual model to be used when checking for issues. This parameter overrides the default virtual model that is set when the system is opened. The default virtual model is "Modifiable.vm", if virtual models are licensed, "Parser" if not licensed. **Parameter can only be used with Sonargraph Architect license.**	No default
workspaceProfile	No	The profile file name (e.g. "BuildProfile.xml") for transforming the workspace paths to match the build environment.	No default
qualityModelFile	No	The path to the quality model file (xyz.sgqm) that should be applied for the report creation. Built-in quality models are the language-independent "Sonargraph:Default.sgqm" and language-specific "Sonargraph:Java.sgqm", "Sonargraph:CSharp.sgqm" and "Sonargraph:CPlusPlus.sgqm". **NOTE: All scripts, analyzer configurations and architecture files present in the system are ignored!**	No default
snapshotDirectory	No	Target directory for the created snapshot. Only if either this parameter or snapshotFileName is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	Current directory
snapshotFileName	No	The target file name (without extension). Only if either this parameter or snapshotDirectory is	<system-name>_<timestamp>

Attribute	Mandatory	Description	Default
		provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	
reportDirectory	No	Target directory for the created report	Current directory
reportFileName	No	The target file name (without extension)	<system-name>_<timestamp>
reportType	No	"standard" only creates metric information of system and module level. "full" creates metric information of all levels.	standard
reportFormat	No	"xml", "html" or "xml, html"	html
elementCountToSplitHtmlReport	No	Issue and resolution tables might contain too many items making it impossible to open the HTML report in a browser. This parameter controls the lower limit of items that will cause separate files being generated per issue type. Possible values are: -1 (never split), 0 (use default value), 1 (always split), positive number > 1 (threshold for split)	1000
maxElementCountForHtmlDetailsPage	No	If HTML report is split because of too many issues and/or resolutions, detail tables might contain too many items making it impossible to open the page in a browser. This parameter controls the upper limit of elements shown in the table. Possible values are: -1 (no limit), 0 (use default limit), positive number > 1 (maximum number of elements contained in page)	2000
splitByModule	No	If set to 'true', individual HTML reports are created per module.	false
baselineReportPath	No	Path of the baseline XML report that the current report is compared against	No default
deltaReportPath	No	Path of the output file that the delta info is written to. This log file is only generated if a baselineReportPath has been specified.	<reportDirectory>/<reportFileName>_delta.log
proxyHost	No	Proxy host	No default
proxyPort	No	Proxy port	No default
proxyUsername	No	Proxy user name	No default
proxyPassword	No	Proxy password	No default
pythonInterpreterPath	No	The path to a valid Python 3 interpreter to be used for the build. **NOTE:** If the standalone Sonargraph application is used on the same machine with the same user and this parameter is empty, the Python interpreter specified with the standalone application will be used.	PATH is searched for a valid Python 3 interpreter. If none can be found in this parameter is not set the build will fail.

Table 4.1. Configuration for Element "sonargraphBuild"

Example

This is an example configuration for creating an XML and HTML report:

```
<sonargraphBuild
    activationCode="_your activation code_"
    languages="Java"
    installationDirectory="../.."
    systemDirectory="../javaProject/AlarmClock.sonargraph"
    reportDirectory="./_temp/report"
    reportFileName=""
    reportType="full"
    reportFormat="xml,html"
    snapshotFileName="./_temp/AlarmClock.snapshot"
    proxyHost=""
    proxyPort=""
    proxyUsername=""
    proxyPassword=""
    logLevel="warn">
</sonargraphBuild>
```

4.2. Specify Conditions for Build Failure

SonargraphBuild can check for the existence of specific issues and mark the build as failed. The nested "failSet" element of the "sonargraphBuild" element can include any number of "include" and "exclude" definitions based on the issues that are either built-in (like duplicate warnings, cycle group warnings, etc.) or custom issues created via Groovy scripts.

TIP

The issue type that must be specified for include/exclude definitions can be determined via the Properties View of *Sonargraph* .

TIP

If you want the build to fail only for newly introduced issues, apply resolutions like "Ignore" or "Fix" to the already know issues in *Sonargraph* and only filter for resolution value "none".

TIP

If you want the build to fail because of certain metric values (e.g. Lines of Code per Source File), define a threshold for that metric in *Sonargraph* to create issues if files grow too large.

TIP

The include/exclude definitions are applied in the following sequence, regardless of their definition order:

1. First all include definitions are matched against the set of existing issues.
2. Then the exclude definitions are applied and the set of previously matched issues is reduced accordingly.

Element	Parameter	Mandatory	Description	Default
failSet	failOnEmptyWorkspace	No	Marks the build as failed if modules and root directories are detected but no components are found. Possible values are "true" or "false".	true
include/exclude	issueType	Yes	Name of the issue type or "any" for wildcard matching.	No default
include/exclude	severity	No	Severity of the issue. Possible values are: error, warning, info, none, any	any
include/exclude	resolution	No	The issue's resolution type. Possible values are: task, ignore, any, none	none

Table 4.2. Configuration Parameters for Build Failure

Example

This is an example failSet definition:

```
<sonargraphBuild
...
  <failSet failOnEmptyWorkspace="false">
    <include issueType="any" severity="error" resolution="none"/>
    <exclude issueType="ScriptCompilationError"/>
    <include issueType="Supertype uses subtype"/>
    <include issueType="any" severity="warning"/>
    <exclude issueType="ThresholdViolation"/>
  </failSet>
</sonargraphBuild>
```

The console output provides some basic information about the number of issues matched by either "include" and "exclude":

Failed:
Sonargraph: Start creating report...
Sonargraph: Opening system...
Sonargraph: Refreshing system...
Sonargraph: Creating report...
Sonargraph: Check if build should be marked as failed...
Include filter [issueType=any, severity=error, resolution=none] matches 0 issue(s).
Include filter [issueType=Supertype uses subtype, severity=any, resolution=none] matches 0 issue(s).
Include filter [issueType=any, severity=warning, resolution=none] matches 2 issue(s).
Exclude filter [issueType=ScriptCompilationError, severity=any, resolution=none] removes 0 previously matched issue(s).
Exclude filter [issueType=ThresholdViolation, severity=any, resolution=none] removes 0 previously matched issue(s).
Summary: Build failed as 2 issue(s) match the specified failset on virtual model 'Modifiable.vm'.
Sonargraph: Finished.

Chapter 5. Integrating with Ant

The provided SonargraphReportTask makes it easy to integrate *SonargraphBuild* into Apache Ant based builds and generate HTML or XML reports containing info about metrics and issues of a software system. Additionally, using the optional "failSet" element, the Ant build can be marked as failed if certain issues exist.

Prerequisites:

1. You need at least Ant 1.8.3 installed.

2. Set the environment variable ANT_HOME.

3. Include ANT_HOME/bin in your PATH environment variable.

The following shows the SonargraphReportTask definition:

```
<taskdef name="sonargraphBuild"
     classname="com.hello2morrow.sonargraph.build.client.ant.SonargraphReportTask">
  <classpath>
    <fileset dir="${sonargraph.build.installation}/plugins">
      <include name="org.eclipse.osgi_3.1*.jar" />
      <include name="com.hello2morrow.sonargraph.build.client*.jar"/>
    </fileset>
    <fileset dir="${sonargraph.build.installation}/client" includes="*.jar" />
  </classpath>
</taskdef>
```

An example Ant build.xml is provided in the directory <inst-dir>/example/ant. The parameters are the same as for the shell integration described in Chapter 4, *Executing from the Command-line*

Chapter 6. Integration with Maven

The *SonargraphBuild* Maven plugin makes it easy to check the quality of your projects. The plugin is able to automatically download and install *SonargraphBuild* if your build server has Internet connectivity. You can make the build fail depending on issues detected by *SonargraphBuild*.

There are two different goals available:

1. **create-report**: Creates a report for an existing system.
2. **dynamic-report**: Creates a system on-the-fly and creates a report for it. This is currently only available for Java systems.
3. **help**: Displays information about the other two goals and can be parameterized to show more details.

Prerequisites:

1. You need at least Maven 3.0.5 installed.
2. The plugin requires at least a Java 8 runtime.
3. The plugin cannot be used together with Tycho.

6.1. Maven Tips and Best Practices

TIP

Add the following repository to your Maven settings.xml, so you do not need to repeat it in your project's pom.xml:

```
<pluginRepository>
   <id>hello2morrow.maven.repository</id>
   <url>http://maven.hello2morrow.com/repository</url>
</pluginRepository>
```

TIP

The goals are **not** configured to be executed within any default Maven lifecycle phase. Typically you would run the plugin with a command-line like the following to ensure that everything is compiled from scratch before the report is created. The first command-line explicitly specifies a version, the second one uses the Maven prefix resolution (check *Maven Prefix Resolution* for details):

```
mvn clean compile com.hello2morrow:sonargraph-maven-plugin:9.10.0:create-report
```

```
mvn clean compile sonargraph:create-report
```

TIP

All parameters of the top-level goals (i.e. not the failSet) can equally be set via the command-line using system properties of the form

```
-Dsonargraph.<parameter-name>=<value>
```

TIP

To enforce certain rules, specify a failSet that lets the build fail based on detected issues. See Section 6.4, “Maven FailSet Configuration”

NOTE

The plugin cannot be used together with Tycho.

You need to use another option to execute Sonargraph. See Chapter 4, *Executing from the Command-line*, Chapter 5, *Integrating with Ant*. If the class root paths of the Sonargraph workspace do not match the Maven target

directories, check the section about "Workspace Profiles" in the user manual of the standalone application: *http://eclipse.hello2morrow.com/doc/standalone/content/workspace_profiles.html*

NOTE

The attribute "logLevel" affects the logging after the SonargraphBuild engine has been started. Setting it to "debug" and below will generate additional debug information into the XML report.

WARNING

Setting the attribute "reportType" to "standard" only generates metric values for "system" and "module" levels. "full" generates values for all element levels, but results in a significant larger report!

The splitting of the HTML report into different files is controlled by the attribute "elementCountToSplitHtmlReport". The resulting detail pages contain tables of issues / resolutions of a specific issue type. The maximum number of items shown is limited by the parameter "maxElementCountForHtmlDetailsPage".

WARNING

The attribute "qualityModelFile" can be used to apply a fixed set of scripts, architecture files and analyzer configurations across several systems.

The reported issues are most likely very different from the results of analyzing the system with the Sonargraph application. If you specify a "qualityModelFile", it is advisable to specify the "Parser" virtual model, since resolutions e.g. for architecture violations probably will not match.

6.2. Parameters of Goal "create-report"

The following table lists all parameters that are available to create a report for an existing Sonargraph system. The class root directories are replaced by the output directories known to Maven.

NOTE

If Maven generates additional output directories dynamically that must be part of the Sonargraph workspace, the Sonargraph plugin must be executed within the same process as the plugins that generate the additional directories.

The build can be marked as failed based on a failSet. See Section 6.4, “Maven FailSet Configuration”.

Attribute	Mandatory	Description	Default
sonargraphBuildVersion	No	Allows you to use a specific or restricted version of SonargraphBuild. Can be used in combination with 'autoUpdate'. As an example, if you specify '8.7' the newest available version starting with '8.7' will be used. If you specify '8.7.0.361' you are locked on that specific version of SonargraphBuild. There are two special values: "same" and "newest". If "same" is defined the version of SonargraphBuild must be the same as the version of the Maven plugin. If "newest" is defined the plugin will always try to use the newest version of SonargraphBuild.	same
skip	No	Skip SonargraphBuild.	false
autoUpdate	No	If the plugin is configured to download SonargraphBuild automatically, this parameter decides if it also should be updated automatically if a new version becomes available.	false
useHttpProxyHost	No	The id of a proxy entry in the Maven settings. If defined the plugin will use this proxy for all HTTP communication.	No default

Attribute	Mandatory	Description	Default
repository	No	URL of Json file containing information about available Sonargraph Build versions with their download locations.	
installationDirectory	No	Installation directory of SonargraphBuild. If unspecified the plugin will automatically download SonargraphBuild. The version to be downloaded can be controlled by the parameter 'sonargraphBuildVersion'.	No default
activationCode	No	Sonargraph license activation code. If this parameter is not specified, you must specify a license file parameter (see below).	No default
licenseServerPort	No	Port of license server to be used. This parameter is ignored if licenseServerHost is not set.	8080
licenseServerHost	No	Host name or IP address of license server. If this parameter is not specified, the web-based hello2morrow license server will be used for activation code based licenses.	No default
licenseFile	No	Sonargraph license file location. If this parameter is not specified, you must specify the activation code parameter (see above).	No default
languages	No	The languages that should be initialized, separated by ",". The fewer languages are specified the faster the startup of SonargraphBuild. Possible values: Java, CSharp, CPlusPlus.	Java
logFile	No	Path of the log file to be used for SonargraphBuild.	${baseDir}/${target}/sonargraph_build.log
logLevel	No	Level of logging detail. One of: off, error, warn, info, debug, trace, all	info
compilerDefinitionPath	No	The path to the active compiler definition file to be used for parsing a C/C++ system. If a built-in or automatically generated definition should be used, prefix the definition with "CPlusPlus:", e.g. CPlusPlus:GnuCpp.cdef. **NOTE:** If the standalone Sonargraph application is used on the same machine with the same user and this parameter is empty, the active definition specified with the standalone application will be used.	If empty, the default compiler definition for the build server's operating system is used: GnuCpp.cdef (Linux), CLang.cdef (Mac), VisualCpp*.cdef (Windows, generated definition for the latest Visual Studio installation)
systemDirectory	No	Directory of the Sonargraph System (xyz.sonargraph)	${baseDir}/${artifact.id}.sonargraph
overrideSonargraphWorkspace	No	If true the output directories defined in the Sonargraph system will be overridden by the ones provided by the client.	true
includeTestCode	No	If true the workspace will also contain the test source and test class file directories.	false
includeEmptyModules	No	If true the workspace will also contain empty modules (without any source and class file directories).	false
virtualModel	No	The virtual model to be used when checking for issues. This parameter overrides the default virtual	No default

Attribute	Mandatory	Description	Default
		model that is set when the system is opened. The default virtual model is "Modifiable.vm", if virtual models are licensed, "Parser" if not licensed. **Parameter can only be used with Sonargraph Architect license.**	
qualityModelFile	No	The path to the quality model file (xyz.sgqm) that should be applied for the report creation. Built-in quality models are the language-independent "Sonargraph:Default.sgqm" and language-specific "Sonargraph:Java.sgqm", "Sonargraph:CSharp.sgqm" and "Sonargraph:CPlusPlus.sgqm". **NOTE: All scripts, analyzer configurations and architecture files present in the system are ignored!**	No default
snapshotDirectory	No	Target directory for the created snapshot. Only if either this parameter or snapshotFileName is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	${basedir}/${target}
snapshotFileName	No	The target file name (without extension). Only if either this parameter or snapshotDirectory is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	<system-name>_<timestamp>
reportDirectory	No	Target directory for the created report	${basedir}/${target}/sonargraph
reportFileName	No	The target file name (without extension)	<system-name>_<timestamp>
reportType	No	"standard" only creates metric information of system and module level. "full" creates metric information of all levels.	standard
reportFormat	No	"xml", "html" or "xml, html"	html
elementCountToSplitHtmlReport	No	Issue and resolution tables might contain too many items making it impossible to open the HTML report in a browser. This parameter controls the lower limit of items that will cause separate files being generated per issue type. Possible values are: -1 (never split), 0 (use default value), 1 (always split), positive number > 1 (threshold for split)	1000
maxElementCountForHtmlDetailsPage	No	If HTML report is split because of too many issues and/or resolutions, detail tables might contain too many items making it impossible to open the page in a browser. This parameter controls the upper limit of elements shown in the table. Possible values are: -1 (no limit), 0 (use default limit), positive number > 1 (maximum number of elements contained in page)	2000
splitByModule	No	If set to 'true', individual HTML reports are created per module.	false
baselineReportPath	No	Path of the baseline XML report that the current report is compared against	No default
deltaReportPath	No	Path of the output file that the delta info is written to. This log file is only generated if a baselineReportPath has been specified.	<reportDirectory>/<reportFileName>_delta.log

Attribute	Mandatory	Description	Default
prepareForSonarQube	No	Creates an XML report and stores it at ${basedir}/${target}/sonargraph/sonargraph-sonarqube-report.xml, where the SonarQube Sonargraph plugin expects it.	false
pythonInterpreterPath	No	The path to a valid Python 3 interpreter to be used for the build. **NOTE:** If the standalone Sonargraph application is used on the same machine with the same user and this parameter is empty, the Python interpreter specified with the standalone application will be used.	PATH is searched for a valid Python 3 interpreter. If none can be found in this parameter is not set the build will fail.

Table 6.1. Configuration for goal "create-report"

Related topics:

- Section 6.1, “Maven Tips and Best Practices”
- Section 6.4, “Maven FailSet Configuration”
- Section 6.5, “Example POM”

6.3. Configuration for goal "dynamic-report"

The following table lists all parameters that are available to create a report for a Java project where no Sonargraph system has been defined. A Sonargraph system is created on the fly based on the workspace information contained in the Maven project setup.

> ### NOTE
>
> If your Maven build generates source and class roots dynamically, the "dynamic-report" goal should be called in the same process as those plugins that generate the additional roots. The following Maven execution also makes the dynamic roots available to Sonargraph:
>
> ```
> mvn compile sonargraph:dynamic-report
> ```
>
> Whereas using the following two separate Maven invocations, the dynamic roots will **NOT** be visible to Sonargraph:
>
> ```
> mvn compile
> mvn sonargraph:dynamic-report
> ```

The build can be marked as failed based on a failSet. See Section 6.4, “Maven FailSet Configuration”.

Attribute	Mandatory	Description	Default
sonargraphBuildVersion	No	Allows you to use a specific or restricted version of SonargraphBuild. Can be used in combination with 'autoUpdate'. As an example, if you specify '8.7' the newest available version starting with '8.7' will be used. If you specify '8.7.0.361' you are locked on that specific version of SonargraphBuild. There are two special values: "same" and "newest". If "same" is defined the version of SonargraphBuild must be the same as the version of the Maven plugin. If "newest" is defined the plugin will always try to use the newest version of SonargraphBuild.	same
skip	No	Skip SonargraphBuild.	false
autoUpdate	No	If the plugin is configured to download SonargraphBuild automatically, this parameter	false

Attribute	Mandatory	Description	Default
		decides if it also should be updated automatically if a new version becomes available.	
useHttpProxyHost	No	The id of a proxy entry in the Maven settings. If defined the plugin will use this proxy for all HTTP communication.	No default
repository	No	URL of Json file containing information about available Sonargraph Build versions with their download locations.	
installationDirectory	No	Installation directory of SonargraphBuild. If unspecified the plugin will automatically download SonargraphBuild. The version to be downloaded can be controlled by the parameter 'sonargraphBuildVersion'.	No default
activationCode	No	Sonargraph license activation code. If this parameter is not specified, you must specify a license file parameter (see below).	No default
licenseServerPort	No	Port of license server to be used. This parameter is ignored if licenseServerHost is not set.	8080
licenseServerHost	No	Host name or IP address of license server. If this parameter is not specified, the web-based hello2morrow license server will be used for activation code based licenses.	No default
licenseFile	No	Sonargraph license file location. If this parameter is not specified, you must specify the activation code parameter (see above).	No default
languages	No	The languages that should be initialized, separated by ",". The fewer languages are specified the faster the startup of SonargraphBuild. Possible values: Java, CSharp, CPlusPlus.	Java
logFile	No	Path of the log file to be used for SonargraphBuild.	${baseDir}/${target}/ sonargraph_build.log
logLevel	No	Level of logging detail. One of: off, error, warn, info, debug, trace, all	info
systemBaseDirectory	No	The directory where the Sonargraph System (${artifactId}.sonargraph) is created.	${baseDir}/${target}
systemId	No	A system id should stay constant over the lifetime of a software system and should be also unique with respect to other systems. If you anticipate that the the group id or artifact id of the root pom might change you should assign a value to this parameter.	${groupdId}_ ${artifactId}
useGroupIdInModuleName	No	If true the module names will use group id and artifact id as their name, separated by an underscore. By default only the artifact id is used as the module name.	false
includeTestCode	No	If true the workspace will also contain the test source and test class file directories.	false
includeEmptyModules	No	If true the workspace will also contain empty modules (without any source and class file directories).	false

Attribute	Mandatory	Description	Default
virtualModel	No	The virtual model to be used when checking for issues. This parameter overrides the default virtual model that is set when the system is opened. The default virtual model is "Modifiable.vm", if virtual models are licensed, "Parser" if not licensed. **Parameter can only be used with Sonargraph Architect license.**	No default
qualityModelFile	No	The path to the quality model file (xyz.sgqm) that should be applied for the report creation. Built-in quality models are the language-independent "Sonargraph:Default.sgqm" and language-specific "Sonargraph:Java.sgqm", "Sonargraph:CSharp.sgqm" and "Sonargraph:CPlusPlus.sgqm". **NOTE: All scripts, analyzer configurations and architecture files present in the system are ignored!**	No default
snapshotDirectory	No	Target directory for the created snapshot. Only if either this parameter or snapshotFileName is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	${basedir}/${target}
snapshotFileName	No	The target file name (without extension). Only if either this parameter or snapshotDirectory is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	<system-name>_<timestamp>
reportDirectory	No	Target directory for the created report	${basedir}/${target}/sonargraph
reportFileName	No	The target file name (without extension)	<system-name>_<timestamp>
reportType	No	"standard" only creates metric information of system and module level. "full" creates metric information of all levels.	standard
reportFormat	No	"xml", "html" or "xml, html"	html
elementCountToSplitHtmlReport	No	Issue and resolution tables might contain too many items making it impossible to open the HTML report in a browser. This parameter controls the lower limit of items that will cause separate files being generated per issue type. Possible values are: -1 (never split), 0 (use default value), 1 (always split), positive number > 1 (threshold for split)	1000
maxElementCountForHtmlDetailsPage	No	If HTML report is split because of too many issues and/or resolutions, detail tables might contain too many items making it impossible to open the page in a browser. This parameter controls the upper limit of elements shown in the table. Possible values are: -1 (no limit), 0 (use default limit), positive number > 1 (maximum number of elements contained in page)	2000
splitByModule	No	If set to 'true', individual HTML reports are created per module.	false
baselineReportPath	No	Path of the baseline XML report that the current report is compared against	No default

Attribute	Mandatory	Description	Default
deltaReportPath	No	Path of the output file that the delta info is written to. This log file is only generated if a baselineReportPath has been specified.	<reportDirectory>/<reportFileName>_delta.log
prepareForSonarQube	No	Creates an XML report and stores it at ${basedir}/${target}/sonargraph/sonargraph-sonarqube-report.xml, where the SonarQube Sonargraph plugin expects it.	false

Table 6.2. Configuration for goal "dynamic-report"

Related topics:

- Section 6.1, “Maven Tips and Best Practices”
- Section 6.4, “Maven FailSet Configuration”
- Section 6.5, “Example POM”

6.4. Maven FailSet Configuration

The following elements allow to mark a build as failed. An example is shown in the next section Section 6.5, “Example POM”.

TIP

The issue type that must be specified for include/exclude definitions can be determined via the Properties View of *Sonargraph* .

TIP

If you want the build to fail only for newly introduced issues, apply resolutions like "Ignore" or "Fix" to the already know issues in *Sonargraph* and only filter for resolution value "none".

TIP

If you want the build to fail because of certain metric values (e.g. Lines of Code per Source File), define a threshold for that metric in *Sonargraph* to create issues if files grow too large.

TIP

The include/exclude definitions are applied in the following sequence, regardless of their definition order:

1. First all include definitions are matched against the set of existing issues.
2. Then the exclude definitions are applied and the set of previously matched issues is reduced accordingly.

Element	Parameter	Mandatory	Description	Default
failSet	failOnEmptyWorkspace	No	Marks the build as failed if modules and root directories are detected but no components are found. Possible values are "true" or "false".	true
include/exclude	issueType	Yes	Name of the issue type or "any" for wildcard matching.	No default
include/exclude	severity	No	Severity of the issue. Possible values are: error, warning, info, none, any	any

Element	Parameter	Mandatory	Description	Default
include/exclude	resolution	No	The issue's resolution type. Possible values are: task, ignore, any, none	none

Table 6.3. Configuration Parameters for Build Failure

6.5. Example POM

The following example shows how to integrate the Sonargraph Maven plugin into your project specific pom file. For multi-module projects it is sufficient to only add the plugin to the pom of the root project. It runs as an aggregator after all modules have been compiled. The example project in the installation contains a complete pom.xml. Typically you would run the plugin with a command-line like the following to ensure that everything is compiled from scratch before the report is created. The first command-line explicitly specifies a version, the second one uses the Maven prefix resolution (check *Maven Prefix Resolution* for details):

```
mvn clean compile com.hello2morrow:sonargraph-maven-plugin:9.10.0:create-report
```

```
mvn clean compile sonargraph:create-report
```

The following shows the relevant section of a pom.xml file that demonstrates the configuration of the Sonargraph functionality:

```
  <pluginRepositories>
    <pluginRepository>
        <id>hello2morrow.maven.repository</id>
        <url>http://maven.hello2morrow.com/repository</url>
    </pluginRepository>
  </pluginRepositories>
  <build>
    <plugins>
        <plugin>
            <groupId>com.hello2morrow</groupId>
            <artifactId>sonargraph-maven-plugin</artifactId>
            <version>9.10.0</version>
            <configuration>
                <systemDirectory>${basedir}/crm-domain-example.sonargraph</systemDirectory>
                <activationCode>...</activationCode>
                <autoUpdate>true</autoUpdate>
                <failSet>
                    <failOnEmptyWorkspace>true</failOnEmptyWorkspace>
                    <includes>
                        <include>
                            <issueType>ArchitectureViolation</issueType>
                        </include>
                        <include>
                            <issueType>any</issueType>
                            <severity>error</severity>
                        </include>
                    </includes>
                    <excludes>
                        <exclude>
                            <issueType>ScriptCompilationError</issueType>
                            <resolution>none</resolution>
                        </exclude>
                    </excludes>
                </failSet>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>create-report</goal>
                        <goal>dynamic-report</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

In this example the build will fail if the project contains a package cycle or an architecture violation without a resolution. Since the parameter 'installationDirectory' is not defined, the Maven plugin will automatically download the newest release of *SonargraphBuild* and also will keep it updated automatically. Of course this requires that the build server has access to the Internet.

Chapter 7. Integration with Gradle

The *SonargraphBuild* Gradle plugin makes it easy to check the quality of your projects. The plugin is able to automatically download and install *SonargraphBuild* if your build server has Internet connectivity. You can make the build fail depending on issues detected by *SonargraphBuild*.

There are two different Gradle "tasks" available for which parameters can be defined in Gradle "extensions" with the same names:

1. **sonargraphReport**: Creates a report for an existing system.
2. **sonargraphDynamicReport**: Creates a system on-the-fly and creates a report for it. This is currently only available for Java systems.

Prerequisites:

1. You need at least Gradle 2.9 installed.
2. The plugin requires at least a Java 8 runtime.

7.1. Gradle Tips and Best Practices

TIP

To enforce certain rules, specify a failSet that lets the build fail based on detected issues. See Section 7.4, “Gradle FailSet Configuration”

NOTE

The attribute "logLevel" affects the logging after the SonargraphBuild engine has been started. Setting it to "debug" and below will generate additional debug information into the XML report.

WARNING

Setting the attribute "reportType" to "standard" only generates metric values for "system" and "module" levels. "full" generates values for all element levels, but results in a significant larger report!

The splitting of the HTML report into different files is controlled by the attribute "elementCountToSplitHtmlReport". The resulting detail pages contain tables of issues / resolutions of a specific issue type. The maximum number of items shown is limited by the parameter "maxElementCountForHtmlDetailsPage".

WARNING

The attribute "qualityModelFile" can be used to apply a fixed set of scripts, architecture files and analyzer configurations across several systems.

The reported issues are most likely very different from the results of analyzing the system with the Sonargraph application. If you specify a "qualityModelFile", it is advisable to specify the "Parser" virtual model, since resolutions e.g. for architecture violations probably will not match.

7.2. Parameters of Task "sonargraphReport"

The following table lists all parameters that are available to create a report for an existing Sonargraph system. The build can be marked as failed based on a failSet. See Section 7.4, “Gradle FailSet Configuration”.

Attribute	Mandatory	Description	Default
sonargraphBuildVersion	No	Allows you to use a specific or restricted version of SonargraphBuild. Can be used in combination with 'autoUpdate'. As an example, if you specify '8.7' the newest available version starting with '8.7' will be	same

Attribute	Mandatory	Description	Default
		used. If you specify '8.7.0.361' you are locked on that specific version of SonargraphBuild. There are two special values: "same" and "newest". If "same" is defined the version of SonargraphBuild must be the same as the version of the Maven plugin. If "newest" is defined the plugin will always try to use the newest version of SonargraphBuild.	
skip	No	Skip SonargraphBuild.	false
autoUpdate	No	If the plugin is configured to download SonargraphBuild automatically, this parameter decides if it also should be updated automatically if a new version becomes available.	false
useHttpProxyHost	No	If true, the proxy configuration of the Gradle settings is used. The plugin will use this proxy for all HTTP communication. Check the *Gradle online documentation* for details.	false
repository	No	URL of Json file containing information about available Sonargraph Build versions with their download locations.	
installationDirectory	No	Installation directory of SonargraphBuild. If unspecified the plugin will automatically download SonargraphBuild. The version to be downloaded can be controlled by the parameter 'sonargraphBuildVersion'.	No default
activationCode	No	Sonargraph license activation code. If this parameter is not specified, you must specify a license file parameter (see below).	No default
licenseServerPort	No	Port of license server to be used. This parameter is ignored if licenseServerHost is not set.	8080
licenseServerHost	No	Host name or IP address of license server. If this parameter is not specified, the web-based hello2morrow license server will be used for activation code based licenses.	No default
licenseFile	No	Sonargraph license file location. If this parameter is not specified, you must specify the activation code parameter (see above).	No default
languages	No	The languages that should be initialized, separated by ",". The fewer languages are specified the faster the startup of SonargraphBuild. Possible values: Java, CSharp, CPlusPlus.	Java
logFile	No	Path of the log file to be used for SonargraphBuild.	${project.buildDir}/ sonargraph_build.log
logLevel	No	Level of logging detail. One of: off, error, warn, info, debug, trace, all	info
compilerDefinitionPath	No	The path to the active compiler definition file to be used for parsing a C/C++ system. If a built-in or automatically generated definition should be used, prefix the definition with "CPlusPlus:", e.g. CPlusPlus:GnuCpp.cdef. **NOTE:** If the standalone Sonargraph application is used on the same machine with the same user	If empty, the default compiler definition for the build server's operating system is used: GnuCpp.cdef (Linux), CLang.cdef (Mac), VisualCpp*.cdef

Attribute	Mandatory	Description	Default
		and this parameter is empty, the active definition specified with the standalone application will be used.	(Windows, generated definition for the latest Visual Studio installation)
systemDirectory	No	Directory of the Sonargraph System (xyz.sonargraph)	${project.buildDir}/ ${project.group} .sonargraph
overrideSonargraphWorkspace	No	If true the output directories defined in the Sonargraph system will be overridden by the ones provided by the client.	true
includeTestCode	No	If true the workspace will also contain the test source and test class file directories.	false
includeEmptyModules	No	If true the workspace will also contain empty modules (without any source and class file directories).	false
productionSourceSets	No	Comma separated list of source set names that contain production code. This parameter is only needed when you are not using the gradle default "main".	main
testSourceSets	No	Comma separated list of source set names that contain test code. This parameter is only needed when you are not using the gradle default "test".	test
virtualModel	No	The virtual model to be used when checking for issues. This parameter overrides the default virtual model that is set when the system is opened. The default virtual model is "Modifiable.vm", if virtual models are licensed, "Parser" if not licensed. **Parameter can only be used with Sonargraph Architect license.**	No default
qualityModelFile	No	The path to the quality model file (xyz.sgqm) that should be applied for the report creation. Built-in quality models are the language-independent "Sonargraph:Default.sgqm" and language-specific "Sonargraph:Java.sgqm", "Sonargraph:CSharp.sgqm" and "Sonargraph:CPlusPlus.sgqm". **NOTE: All scripts, analyzer configurations and architecture files present in the system are ignored!**	No default
snapshotDirectory	No	Target directory for the created snapshot. Only if either this parameter or snapshotFileName is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	${project.buildDir}
snapshotFileName	No	The target file name (without extension). Only if either this parameter or snapshotDirectory is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	<system-name>_<timestamp>
reportDirectory	No	Target directory for the created report	${project.buildDir}/ sonargraph
reportFileName	No	The target file name (without extension)	<system-name>_<timestamp>

Attribute	Mandatory	Description	Default
reportType	No	"standard" only creates metric information of system and module level. "full" creates metric information of all levels.	standard
reportFormat	No	"xml", "html" or "xml, html"	html
elementCountToSplitHtmlReport	No	Issue and resolution tables might contain too many items making it impossible to open the HTML report in a browser. This parameter controls the lower limit of items that will cause separate files being generated per issue type. Possible values are: -1 (never split), 0 (use default value), 1 (always split), positive number > 1 (threshold for split)	1000
maxElementCountForHtmlDetailsPage	No	If HTML report is split because of too many issues and/or resolutions, detail tables might contain too many items making it impossible to open the page in a browser. This parameter controls the upper limit of elements shown in the table. Possible values are: -1 (no limit), 0 (use default limit), positive number > 1 (maximum number of elements contained in page)	2000
splitByModule	No	If set to 'true', individual HTML reports are created per module.	false
baselineReportPath	No	Path of the baseline XML report that the current report is compared against	No default
deltaReportPath	No	Path of the output file that the delta info is written to. This log file is only generated if a baselineReportPath has been specified.	<reportDirectory>/<reportFileName>_delta.log
prepareForSonarQube	No	Creates an XML report and stores it at ${basedir}/${target}/sonargraph/sonargraph-sonarqube-report.xml, where the SonarQube Sonargraph plugin expects it.	false
pythonInterpreterPath	No	The path to a valid Python 3 interpreter to be used for the build. **NOTE:** If the standalone Sonargraph application is used on the same machine with the same user and this parameter is empty, the Python interpreter specified with the standalone application will be used.	PATH is searched for a valid Python 3 interpreter. If none can be found in this parameter is not set the build will fail.

Table 7.1. Configuration for Task/Extension "sonargraphReport"

Related topics:

- Section 7.1, “Gradle Tips and Best Practices”
- Section 7.4, “Gradle FailSet Configuration”
- Section 7.5, “Example Gradle Build File”

7.3. Configuration for Task "sonargraphDynamicReport"

The following table lists all parameters that are available to create a report for a Java project where no Sonargraph system has been defined. A Sonargraph system is created on the fly based on the workspace information contained in the Gradle project setup. The build can be marked as failed based on a failSet. See Section 7.4, “Gradle FailSet Configuration”.

Attribute	**Mandatory**	**Description**	**Default**
sonargraphBuildVersion	No	Allows you to use a specific or restricted version of SonargraphBuild. Can be used in combination with 'autoUpdate'. As an example, if you specify '8.7' the newest available version starting with '8.7' will be used. If you specify '8.7.0.361' you are locked on that specific version of SonargraphBuild. There are two special values: "same" and "newest". If "same" is defined the version of SonargraphBuild must be the same as the version of the Maven plugin. If "newest" is defined the plugin will always try to use the newest version of SonargraphBuild.	same
skip	No	Skip SonargraphBuild.	false
autoUpdate	No	If the plugin is configured to download SonargraphBuild automatically, this parameter decides if it also should be updated automatically if a new version becomes available.	false
useHttpProxyHost	No	If true, the proxy configuration of the Gradle settings is used. The plugin will use this proxy for all HTTP communication. Check the *Gradle online documentation* for details.	false
repository	No	URL of Json file containing information about available Sonargraph Build versions with their download locations.	
installationDirectory	No	Installation directory of SonargraphBuild. If unspecified the plugin will automatically download SonargraphBuild. The version to be downloaded can be controlled by the parameter 'sonargraphBuildVersion'.	No default
activationCode	No	Sonargraph license activation code. If this parameter is not specified, you must specify a license file parameter (see below).	No default
licenseServerPort	No	Port of license server to be used. This parameter is ignored if licenseServerHost is not set.	8080
licenseServerHost	No	Host name or IP address of license server. If this parameter is not specified, the web-based hello2morrow license server will be used for activation code based licenses.	No default
licenseFile	No	Sonargraph license file location. If this parameter is not specified, you must specify the activation code parameter (see above).	No default
languages	No	The languages that should be initialized, separated by ",". The fewer languages are specified the faster the startup of SonargraphBuild. Possible values: Java, CSharp, CPlusPlus.	Java
logFile	No	Path of the log file to be used for SonargraphBuild.	${project.buildDir}/ sonargraph_build.log
logLevel	No	Level of logging detail. One of: off, error, warn, info, debug, trace, all	info

Attribute	Mandatory	Description	Default
systemBaseDirectory	No	The directory where the Sonargraph System (${artifactId}.sonargraph) is created.	${project.buildDir}
systemId	No	A system id should stay constant over the lifetime of a software system and should be also unique with respect to other systems. If you anticipate that the the group id or artifact id of the root pom might change you should assign a value to this parameter.	${project.group}_ ${project.name}
useGroupIdInModuleName	No	If true the module names will use group id and artifact id as their name, separated by an underscore. By default only the artifact id is used as the module name.	false
includeTestCode	No	If true the workspace will also contain the test source and test class file directories.	false
includeEmptyModules	No	If true the workspace will also contain empty modules (without any source and class file directories).	false
productionSourceSets	No	Comma separated list of source set names that contain production code. This parameter is only needed when you are not using the gradle default "main".	main
testSourceSets	No	Comma separated list of source set names that contain test code. This parameter is only needed when you are not using the gradle default "test".	test
virtualModel	No	The virtual model to be used when checking for issues. This parameter overrides the default virtual model that is set when the system is opened. The default virtual model is "Modifiable.vm", if virtual models are licensed, "Parser" if not licensed. **Parameter can only be used with Sonargraph Architect license.**	No default
qualityModelFile	No	The path to the quality model file (xyz.sgqm) that should be applied for the report creation. Built-in quality models are the language-independent "Sonargraph:Default.sgqm" and language-specific "Sonargraph:Java.sgqm", "Sonargraph:CSharp.sgqm" and "Sonargraph:CPlusPlus.sgqm". **NOTE: All scripts, analyzer configurations and architecture files present in the system are ignored!**	No default
snapshotDirectory	No	Target directory for the created snapshot. Only if either this parameter or snapshotFileName is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	${project.buildDir}
snapshotFileName	No	The target file name (without extension). Only if either this parameter or snapshotDirectory is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	<system-name>_<timestamp>
reportDirectory	No	Target directory for the created report	${project.buildDir}/ sonargraph
reportFileName	No	The target file name (without extension)	<system-name>_<timestamp>

Attribute	Mandatory	Description	Default
reportType	No	"standard" only creates metric information of system and module level. "full" creates metric information of all levels.	standard
reportFormat	No	"xml", "html" or "xml, html"	html
elementCountToSplit HtmlReport	No	Issue and resolution tables might contain too many items making it impossible to open the HTML report in a browser. This parameter controls the lower limit of items that will cause separate files being generated per issue type. Possible values are: -1 (never split), 0 (use default value), 1 (always split), positive number > 1 (threshold for split)	1000
maxElementCountFor HtmlDetailsPage	No	If HTML report is split because of too many issues and/or resolutions, detail tables might contain too many items making it impossible to open the page in a browser. This parameter controls the upper limit of elements shown in the table. Possible values are: -1 (no limit), 0 (use default limit), positive number > 1 (maximum number of elements contained in page)	2000
splitByModule	No	If set to 'true', individual HTML reports are created per module.	false
baselineReportPath	No	Path of the baseline XML report that the current report is compared against	No default
deltaReportPath	No	Path of the output file that the delta info is written to. This log file is only generated if a baselineReportPath has been specified.	<reportDirectory>/ <reportFileName> _delta.log
prepareForSonarQube	No	Creates an XML report and stores it at ${basedir}/ ${target}/sonargraph/sonargraph-sonarqube-report.xml, where the SonarQube Sonargraph plugin expects it.	false

Table 7.2. Configuration for Task/Extension "sonargraphDynamicReport"

Related topics:

- Section 7.1, “Gradle Tips and Best Practices”
- Section 7.4, “Gradle FailSet Configuration”
- Section 7.5, “Example Gradle Build File”

7.4. Gradle FailSet Configuration

The following elements allow to mark a build as failed. An example is shown in the next section Section 7.5, “Example Gradle Build File”.

TIP

The issue type that must be specified for include/exclude definitions can be determined via the Properties View of *Sonargraph* .

TIP

If you want the build to fail only for newly introduced issues, apply resolutions like "Ignore" or "Fix" to the already know issues in *Sonargraph* and only filter for resolution value "none".

TIP

If you want the build to fail because of certain metric values (e.g. Lines of Code per Source File), define a threshold for that metric in *Sonargraph* to create issues if files grow too large.

TIP

The include/exclude definitions are applied in the following sequence, regardless of their definition order:

1. First all include definitions are matched against the set of existing issues.

2. Then the exclude definitions are applied and the set of previously matched issues is reduced accordingly.

Element	Parameter	Mandatory	Description	Default
failSet	failOnEmptyWorkspace	No	Marks the build as failed if modules and root directories are detected but no components are found. Possible values are "true" or "false".	true
include/exclude	issueType	Yes	Name of the issue type or "any" for wildcard matching.	No default
include/exclude	severity	No	Severity of the issue. Possible values are: error, warning, info, none, any	any
include/exclude	resolution	No	The issue's resolution type. Possible values are: task, ignore, any, none	none

Table 7.3. Configuration Parameters for Build Failure

7.5. Example Gradle Build File

The following example shows how to integrate the SonargraphBuild Gradle plugin into your project. For multi-project builds it is sufficient to only add the plugin to the root project. It runs as an aggregator after all modules have been compiled. The example project in the installation contains a complete build.gradle file. Typically you would run the plugin with a command-line like the following to ensure that everything is compiled from scratch before the report is created:

```
gradlew clean build sonargraphReport
```

The following shows the relevant section of a build.gradle file that demonstrates the configuration of the Sonargraph functionality:

```
apply plugin: 'com.hello2morrow.sonargraph'

task wrapper(type: Wrapper)
{
    gradleVersion = '2.11'
}

buildscript
{
    repositories
    {
        mavenLocal()
        mavenCentral()
        maven
        {
            url 'http://maven.hello2morrow.com/repository'
        }
        maven
        {
            url 'http://maven.hello2morrow.com/snapshots'
        }
    }

    dependencies
    {
        classpath('com.hello2morrow:sonargraph-gradle-plugin:9.10.0')
    }
}

sonargraphReport
{
    // This is a activation code for Sonargraph-Explorer Build which you can use for testing.
    // Replace with your own if you have one.
    activationCode = "36E2-0F3E-643F-B4F2"
    failSet
    {
        failOnEmptyWorkspace = true
        include(issueType: "any", severity: "error", resolution: "none")
        include(issueType: "ArchitectureViolation")
        include(issueType: "any", severity: "warning")
        exclude(issueType: "ScriptCompilationError", resolution: "none")
        exclude(issueType: "ThresholdViolation")
    }
}

sonargraphDynamicReport
{
    activationCode = "36E2-0F3E-643F-B4F2"
    qualityModelFile = "Sonargraph:Java.sgqm" //default Java quality model
    failSet
    {
        failOnEmptyWorkspace = true
        include(issueType: "any", severity: "error", resolution: "none")
        include(issueType: "ArchitectureViolation")
        include(issueType: "any", severity: "warning")
        exclude(issueType: "ScriptCompilationError", resolution: "none")
        exclude(issueType: "ThresholdViolation")
    }
}
```

In this example the build will fail if the project contains a package cycle or an architecture violation without a resolution. Since the parameter 'installationDirectory' is not defined, the Gradle plugin will automatically download the newest release of *SonargraphBuild* and also will keep it updated automatically. Of course this requires that the build server has access to the Internet.

NOTE

The boolean parameters must be set without any quotes.

NOTE

Variable substitution in parameters does not work with single quotes, use double quotes instead.

Chapter 8. Reporting Changes

Reports for large systems provide an overwhelming amount of information. Most of the times a report containing the changes compared to a baseline is enough - similar to a newspaper versus a whole encyclopedia. This functionality is available in *SonargraphBuild* version 9.4.2 and newer. To use the delta feature, you need to configure the parameter "baselineReportPath" and optionally the parameter "deltaReportPath" when creating a report.

The delta computation of two XML reports is implemented in our Open Source project "Sonargraph Integration Access" that is hosted on GitHub at *https://github.com/sonargraph/sonargraph-integration-access* . Thus, you can embed the report delta computation easily within your own custom build pipeline if the integration with SonargraphBuild does not satisfy your requirements. The functionality is available in Integration Access 3.0.0 and newer. It can be called with the following command-line and two mandatory arguments: The first being the baseline report and the second the XML report that is compared against the baseline. If no output file is specified as the third parameter, the info is printed to the console.

```
java -cp ../../target/sonargraph-integration-access-3.0.0.jar
com.hello2morrow.sonargraph.integration.access.ReportDiff
<path-to-baseline-report-xml> <path-to-report-xml> <optional: output-file-path>
```

The delta report is currently plain text. An example report is shown below (lines have been truncated) that shows differences in issues:

```
Delta of System Reports:
   Report1 (baseline): D:\00_repos\sonargraph-integration-access\src\test\diff\AlarmClockMain_01.xml
   Report2           : D:\00_repos\sonargraph-integration-access\src\test\diff\AlarmClockMain_02.xml

System Info:
   Name: AlarmClockMain
   ID: 6db0a52dfa66892be8a4bc2bb7cf1720
   Path: D:\00_repos\sonar-sonargraph-integration\src\test\AlarmClockMain\AlarmClockMain.sonargraph

Delta of Systems
   System 1 (Baseline): AlarmClockMain from Nov 30, 2016 5:01:13 PM
   System 2           : AlarmClockMain from Dec 30, 2016 5:01:13 PM

- Issue delta:
    Removed (13):
        EmptyArchitectureElement, generated by Core: Artifact 'Foundation', line 1, resolved 'false'
        Potentially dead method, generated by ./Java/BadSmells/FindDeadCode.scr: Method has ...
        Potentially dead type, generated by ./Java/BadSmells/FindDeadCode.scr: Type has no ...
        Duplicate Code Block with 2 occurrences, block size '52', resolved 'false'
         Occurrence in ./com/h2m/alarm/model/AlarmClock.java, start '52', block size '52', ...
         Occurrence in ./com/h2m/alarm/presentation/Main.java, start '34', block size '52', ...
        JavaFileClassFileMissing, generated by JavaLanguageProvider: Missing class file for ...
    Improved (1):
        Previous: ThresholdViolation, generated by ./Java/BadSmells/FindDeadCode.scr: Potentially ...
    Worsened (1):
        Previous: ThresholdViolation, generated by Core: Total Lines = 106 (allowed range: 0 to ...
    Added (6):
        Supertype uses subtype, generated by ./Core/SuperTypeUsesSubType.scr: Reference to ...
        ArchitectureViolation, generated by ./Layers.arc: [Local Variable] 'Model' cannot access ...
```

If present, the report also shows differences in the core system configuration (i.e. licensed features, active analyzers, metric provider, metric ids, etc.), workspace configuration and resolutions.

Current Limitations

The following changes only indirectly affect the Sonargraph issues, but will be treated as changes by the delta detector. The issues in the baseline report will be reported as removed and the issues from the new report as added, despite the fact that the issues are logically the same:

1. Cycle groups issues and duplicate code block issues consist of several parts that contribute to their unique IDs. If one of these parts changes (for example a source file has been renamed) then the issue's ID is changed.

2. If a script or an architecture file is renamed, the origin of the issues generated by those resources is changed.

3. For some issues the originating line within a source file is stored and used for comparison. Changing unrelated lines in the source file before the issue's origin therefore will cause the issue to be treated as changed.

NOTE

As with every modification: Frequent and small changes are easier to review than big-bang refactorings.

Ideas for feature improvements are to include the baseline report as filter in Sonargraph Architect/Explorer to let the user focus on changed issues.

Chapter 9. Integration with SonarQube

For Java projects the findings of Sonargraph can be stored and visualized in *SonarQube* using the *Sonargraph Integration* plugin.

The plugin is compatible with SonarQube versions 6.7.3 and higher.

The plugin is available here:

1. The SonarQube Marketplace accessible from within the SonarQube server's web interface.
2. GitHub *https://github.com/sonargraph/sonar-sonargraph-integration/releases.*
3. hello2morrow's web site *https://www.hello2morrow.com/products/downloads.*

9.1. Overall Process of Integration

We assume you have already a SonarQube server running and see the project of interest in the server's web interface. To add Sonargraph's analysis results you need to:

1. Install the *Sonargraph Integration* plugin in your SonarQube server.
2. Use the built-in Sonargraph quality profile or add individual *Sonargraph Integration* rules to the profile you want to use. Assign your project to this profile.
3. Define and analyze the project with Sonargraph, either using the Explorer or Architect version. You need the system definition. Alternatively the system definition could be obtained dynamically with our support for dynamic system creation.
4. Create an XML report with Sonargraph Build of that project using either Maven, Gradle, Ant or the Shell support prior to the SonarQube analysis with one of the scanners. Make sure the that the XML report is in the right spot so the *Sonargraph Integration* plugin can find it .

9.2. SonarQube Configuration

Localizing the Sonargraph XML Report

The default location of the xml report file is 'target/sonargraph/sonargraph-sonarqube-report.xml' relative to every module.

In a multi-module system the xml report file must be stored in every module and the top-level project.

Sonargraph calculates metrics and provides issues on module and system level. The system level is equivalent to SonarQube's Project in a multi module system. In a single-module system the module/project will contain both classes of information.

NOTE

Using Maven or Gradle with the prepareForSonarQube flag will copy the produced xml report automatically into all modules.

NOTE

If you want to avoid having a copy of the xml report file in all modules you can alternatively use one absolute location.

Sonargraph Script Metrics and Issues

Issues created from an automated script are activated (or deactivated) with the single rule 'Sonargraph Integration: Script Issue'.

Metrics created from an automated script are now stored in a properties file and are automatically considered after a restart of the SonarQube server. The properties file is stored at '.sonargraphintegration/metrics.properties'.

NOTE

When introducing script metrics for the first time a warning message is created in the console of the SonarQube server when a restart is required because of a modified metrics.properties file.

Related topics:

- See the section about "Workspace Profiles" in the user manual of the standalone application, if the root directories on your build server do not match the workspace definition.
- Chapter 6, *Integration with Maven*
- Chapter 7, *Integration with Gradle*
- Chapter 5, *Integrating with Ant*

9.3. SonarQube Maven Configuration

If you use the SonarQube Maven plugin, you must set the following parameter in the configuration of the SonargraphBuild Maven plugin in your project's pom.xml:

```
<configuration>
    <prepareForSonarQube>true</prepareForSonarQube>
    ...
</configuration>
```

The SonargraphBuild Maven plugin will automatically create an XML report (if not already configured) and will copy the report to ${target}/sonargraph/sonargraph-sonarqube-report.xml for the root project and all modules (excluding those with packaging "pom").

The example project contains an example pom.xml and also a batch file that demonstrates how the check can be called from the command-line.

Related topics:

- Chapter 6, *Integration with Maven*
- Section 6.5, “Example POM”

NOTE

An example command-line using only one xml report location (added line-breaks for readability):

```
mvn clean package
   sonargraph:create-report -Dsonargraph.reportFormat=xml
    -Dsonargraph.reportDirectory=D:/temp/report -Dsonargraph.reportFileName=MyReport
   sonar:sonar -Dsonar.sonargraph.integration:report.path=D:/temp/report/MyReport.xml
```

9.4. SonarQube Gradle Configuration

If you use the SonarQube Gradle plugin, you must set the following parameter in the configuration of the SonargraphBuild tasks in your project's build.gradle:

```
sonargraphReport
{
    activationCode = "36E2-0F3E-643F-B4F2"
    prepareForSonarQube = "true"
}
```

The SonargraphBuild Gradle plugin will automatically create an XML report (if not already configured) and will copy the report to ${target}/sonargraph/sonargraph-sonarqube-report.xml for the root project and all modules.

Related topics:

- Chapter 7, *Integration with Gradle*
- Section 6.5, “Example POM”

9.5. SonarQube Ant Runner Configuration

If you use the SonarQube Ant Runner the Sonargraph XML report must have been created and this report must be configured for the Sonargraph SonarQube plugin using the following parameter:

```
<property name="sonar.sonargraph.integration:report.path" value="${path.target.report}" />
```

Related topics:

- Chapter 5, *Integrating with Ant*

Chapter 10. Integration with SonarQube using Sonargraph Integration plugin 2.1.4 or lower

For Java projects the findings of Sonargraph can be stored and visualized in *SonarQube* using the *SonarQube Sonargraph Integration Plugin*. The plugin is available via the SonarQube update center and on the plugin's GitHub page at *https://github.com/sonargraph/sonar-sonargraph-integration/releases*. The plugin is compatible with SonarQube versions 5.3 and higher.

NOTE

The plugin reads the information of the XML report that has been generated using *SonargraphBuild*. You need to configure your build pipeline accordingly.

NOTE

The number of reported Sonargraph issues might be different in SonarQube for the following reasons: As far as we know, SonarQube requires a physical resource to attach an issue. There is no equivalent SonarQube resource for "logical" Sonargraph elements like "logical namespaces", so there are no SonarQube issues created for package cycles, for example. If you want to track package cycles, configure relevant metrics like "Number of cyclic packages", "Biggest Package Cycle Group", etc. to be shown in a dashboard widget or make them part of your Quality Gate.

On the other hand an individual SonarQube issue is attached to the source file of each duplicate code block occurrence of a Sonargraph duplicate code issue. The same applies to all source files involved in Sonargraph component cycle groups.

Our recommendation: Use SonarQube only as a reporting dashboard and use Sonargraph Architect/Explorer for detailed analysis. The usability and interactions for Sonargraph issues is much better in the rich-client application!

NOTE

The plugin is currently only available for Java systems.

10.1. SonarQube Configuration

The following list describes the necessary steps to get Sonargraph issues and metrics integrated in SonarQube. Additional details are given below.

1. Download the latest SonarQube LTS version. SonarQube's API changes fast, so we don't guarantee that everything works flawless with the latest and greatest SonarQube version. If you spot a problem, please let us know!
2. Download the latest Sonargraph plugin and copy it into <sonarqube-inst>/extensions/plugins (or use the Update Center, once it is available there).
3. Start the SonarQube server.
4. Change the current quality profile or create a new one that include at least one of the "Sonargraph Integration" rules. Assign your project to this profile. Details about SonarQube Quality Profiles can be found here: *https://docs.sonarqube.org/display/SONAR/Quality+Profiles*

   If no Sonargraph rules are activated, the plugin will skip this project. You can either search for rules using the term "Sonargraph Integration", or the tag "sonargraph-integration".
5. Change the dashboard configuration to include the "Sonargraph Integration" widgets (for details, see below). NOTE: Project dashboards have been dropped since SonarQube version 6.1.
6. For the full functionality of Sonargraph, you need an "Architect" license. If you don't have one, just register on our hello2morrow web site and request a trial license. Alternatively, use a free Sonargraph Explorer license with reduced feature set (no architecture checks, no scripts execution, etc.)

7. Configure your build to run SonargraphBuild prior to the SonarQube scanner. Check the previous chapters for details and don't forget to configure the "prepareForSonarQube" flag!

   The Sonargraph SonarQube Plugin repository at *https://github.com/sonargraph/sonar-sonargraph-integration* contains an example multi-module Maven project in src/test/AlarmClockMain. There are various build files and batch files available that demonstrate how the analysis can be executed.

8. Execute the build and check in the console log that the Sonargraph Integration plugin has been executed. In SonarQube the Sonargraph Integration widgets should now display metrics determined by Sonargraph and if your projects contains architecture violations or cyclic dependencies, these should be visible as issues.

   Configure your dashboard widgets to show relevant Sonargraph metrics. The Quality Gate can be adjusted to contain those metrics as well.

9. If you have difficulties setting up the integration, check the console log first for any errors reported by the SonargraphBuild execution or the Sonargraph SonarQube Plugin.

   If your system is really big and contains a lot of modules, check the info below about how to "Handling Large Systems".

Configuration of Dashboard Widgets

The following screenshot shows the available Sonargraph widgets that can be included in your SonarQube dashboard.

NOTE

The Sonargraph widgets are no longer available for SonarQube versions 6.1 and newer, since SonarQube project dashboards have been dropped.

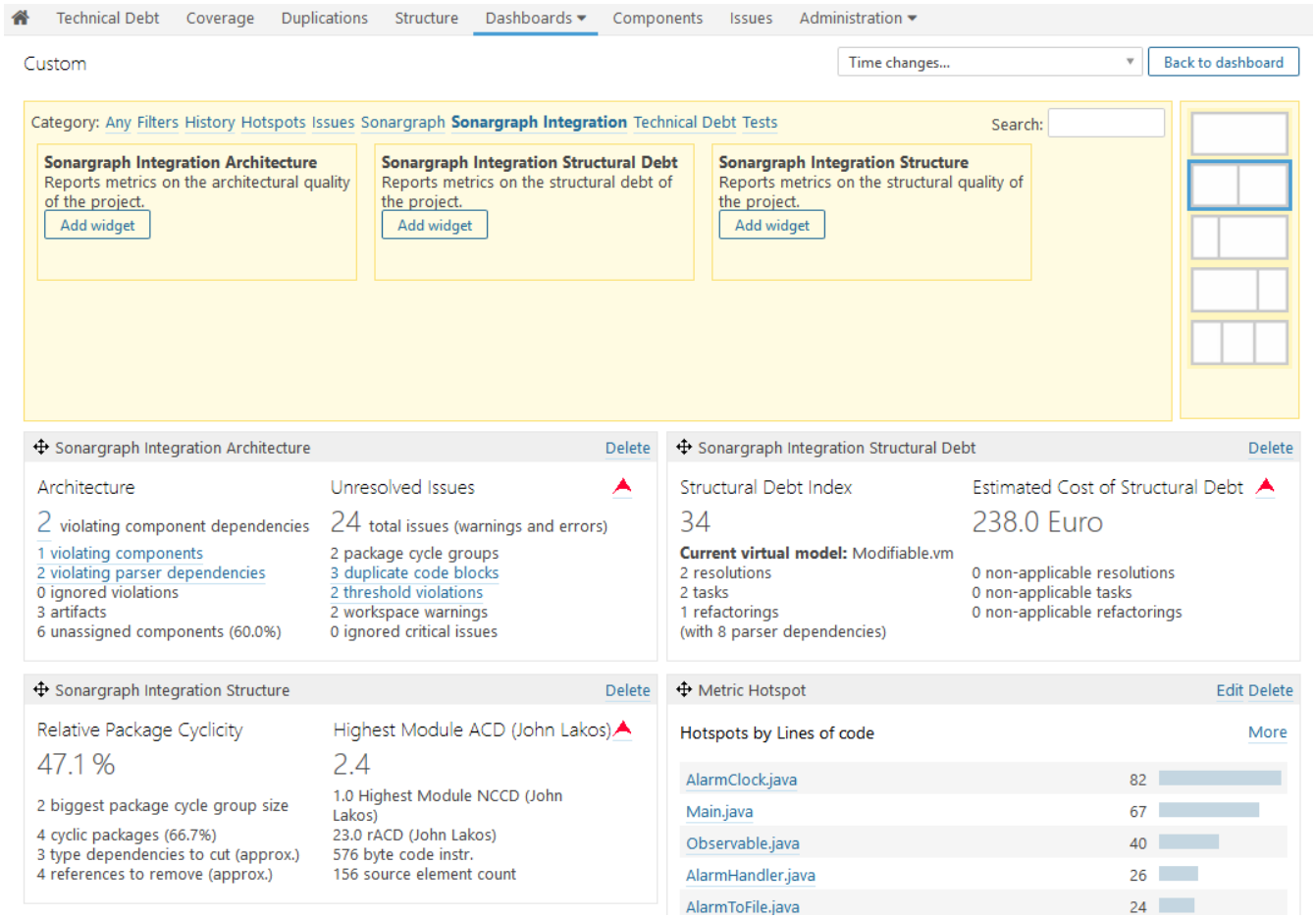

Figure 10.1. SonarQube Dashboard Configuration

Include Custom Sonargraph Metrics and Issues

Core metrics and rules of Sonargraph are pre-defined in the plugin. If you want to track custom metrics that are generated via scripts, you first need to export the report meta-data via the standalone application's menu "File" → "Export Meta-Data...". The directory of this meta-data file needs to be specified in the plugin's configuration page. The additional metrics will be available after a restart of the SonarQube server and an additional execution of the SonarQube checks.

NOTE

This configuration is affecting all projects that use the Sonargraph plugin. If you have several projects with different metrics, store the separate meta-data files in the same directory. The plugin will merge the info of the different configuration files.

Related topics:

- See the section about "Workspace Profiles" in the user manual of the standalone application, if the root directories on your build server do not match the workspace definition.
- Chapter 6, *Integration with Maven*
- Chapter 7, *Integration with Gradle*
- Chapter 5, *Integrating with Ant*

10.2. SonarQube Maven Configuration

If you use the SonarQube Maven plugin, you must set the following parameter in the configuration of the SonargraphBuild Maven plugin in your project's pom.xml:

```
<configuration>
    <prepareForSonarQube>true</prepareForSonarQube>
    ...
</configuration>
```

The SonargraphBuild Maven plugin will automatically create an XML report (if not already configured) and will copy the report to ${target}/sonargraph/sonargraph-sonarqube-report.xml for the root project and all modules (excluding those with packaging "pom").

The example project contains an example pom.xml and also a batch file that demonstrates how the check can be called from the command-line.

Related topics:

- Chapter 6, *Integration with Maven*
- Section 6.5, “Example POM”

NOTE

For very large systems with a high number of modules, do not use the prepareForSonarQube flag. This causes the generated report to be copied into each project's target folder.

Instead, use the parameters to specify the report format ("xml"), the report's target directory and file name and use the parameter "sonar.sonargraph_integration.report.path" as explained in Section 9.5, “SonarQube Ant Runner Configuration”. This causes the same report instance to be re-used for every module being analyzed by SonarQube.

An example command-line with the aforementioned parameters (added line-breaks for readability):

```
mvn clean package
   sonargraph:create-report -Dsonargraph.reportFormat=xml
    -Dsonargraph.reportDirectory=D:/temp/report -Dsonargraph.reportFileName=MyReport
   sonar:sonar -Dsonar.sonargraph_integration.report.path=D:/temp/report/MyReport.xml
```

10.3. SonarQube Gradle Configuration

If you use the SonarQube Gradle plugin, you must set the following parameter in the configuration of the SonargraphBuild tasks in your project's build.gradle:

```
sonargraphReport
{
    activationCode = "36E2-0F3E-643F-B4F2"
    prepareForSonarQube = "true"
}
```

The SonargraphBuild Gradle plugin will automatically create an XML report (if not already configured) and will copy the report to ${target}/sonargraph/sonargraph-sonarqube-report.xml for the root project and all modules.

Related topics:

- Chapter 7, *Integration with Gradle*
- Section 6.5, “Example POM”

10.4. SonarQube Ant Runner Configuration

If you use the SonarQube Ant Runner the Sonargraph XML report must have been created and this report must be configured for the Sonargraph SonarQube plugin using the following parameter:

```
<property name="sonar.sonargraph_integration.report.path" value="${path.target.report}" />
```

The example project contains this configuration in the Ant build file.

Related topics:

- Chapter 5, *Integrating with Ant*

Chapter 11. Integration with Jenkins

With *Jenkins Sonargraph Integration Plugin* for *Jenkins* jobs the findings of *Sonargraph* can be used to let builds fail, or mark them unstable. Additionally *Sonargraph* metric values are stored for every build and can be visualized as charts.

11.1. Jenkins Server Configuration

The first step is to configure one or more versions of Sonargraph Build in "Manage Jenkins" → "Configure System". Click "Sonargraph Build installations..."

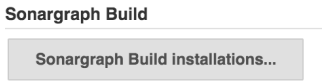

Figure 11.1. Jenkins - Sonargraph Build Configuration

and select a name, a version and an installer.

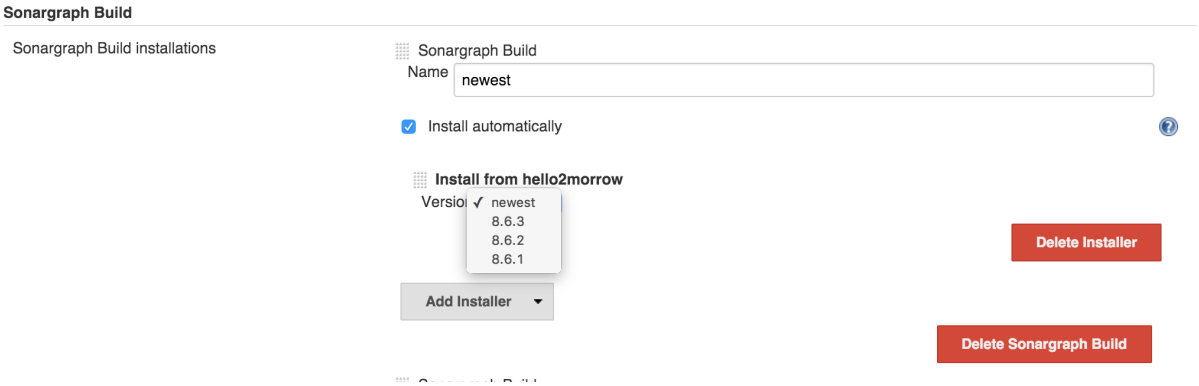

Figure 11.2. Jenkins - New Sonargraph Build

11.2. Jenkins Job Configuration

Add post build action "Sonargraph Integration Report Generation & Analysis" to your job.

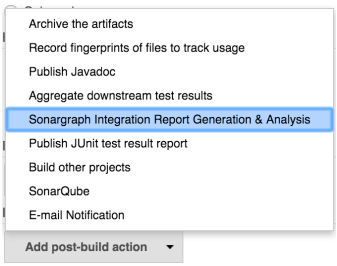

Figure 11.3. Job - Post Build Action

First decide if *Sonargraph Build* is used to create the report,

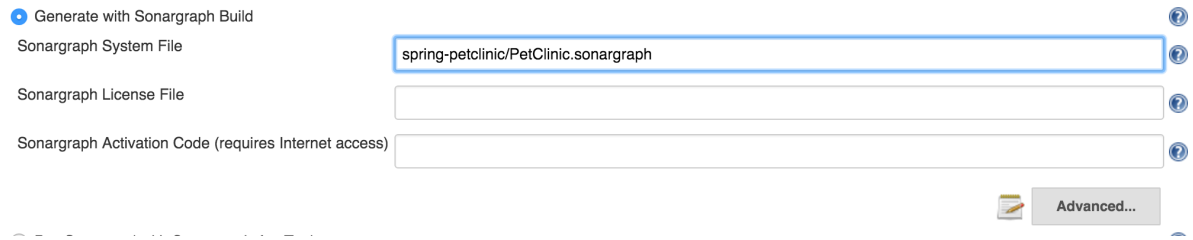

Figure 11.4. Report - Generate With Sonargraph Build

or there already exists a report generated by an upstream build action.

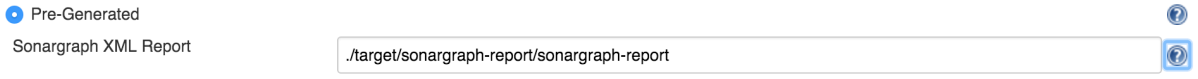

Figure 11.5. Report - Pre Generated

When *Sonargraph Build* is used to create the report fill out all required information:

Figure 11.6. Report - Standard Options

By pressing "Advanced..." some more options pop up:

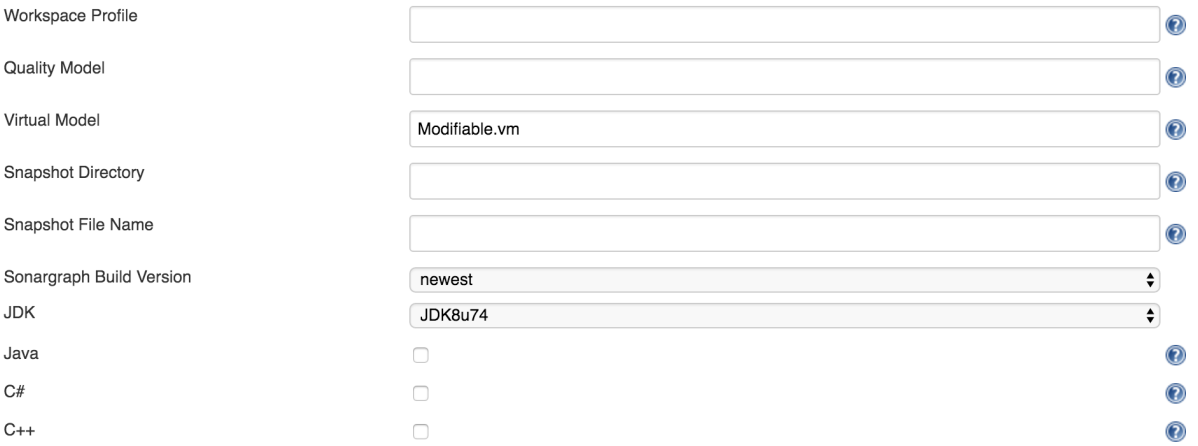

Figure 11.7. Report - Advanced Options

11.3. Charts Configuration

To see some charts, a meta-data file must be configured, and either all contained charts/metrics are shown, or a list of charts/metrics to be shown can be given. If you want to track custom metrics that are generated via scripts, you first need to export the report meta-data via the standalone application's menu "File" → "Export Meta-Data...".

Figure 11.8. Job - Chart Configuration

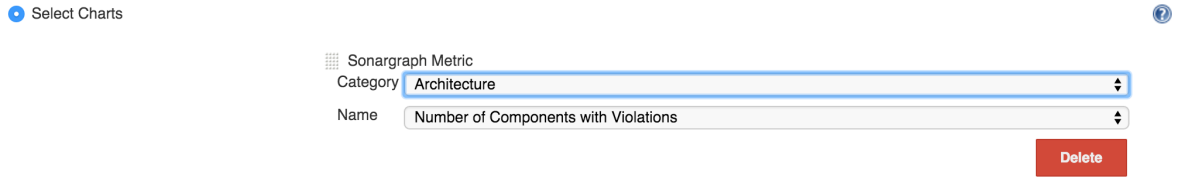

Figure 11.9. Job - Select Charts

11.4. Build Configuration

Finally the reasons for marking the build as failed or unstable can be set:

Figure 11.10. Mark build failed or instable

Related topics:

- See the section about "Workspace Profiles" in the user manual of the standalone application, if the root directories on your build server do not match the workspace definition.
- Chapter 6, *Integration with Maven*

Chapter 12. FAQ

This section summarizes common problems and their solutions.

Different Results in Sonargraph and SonargraphBuild

If you notice differences in the number of issues or metrics reported by SonargraphBuild, this might be due to the following reasons:

1. The SonargraphBuild integrations for Maven and Gradle use as default the workspace information about root directories as provided by Maven or Gradle. Thus the number of root directories might be different, if the Sonargraph workspace does not contain all available root directories. If you know that all root directories contained in the Sonargraph workspace are present at build-time, deactivate this dynamic workspace configuration by setting the parameter "overrideSonargraphWorkspace" to "false".

2. Check if test code should be part of the workspace. As default it is excluded in SonargraphBuild, because the default value of the parameter "includeTestCode" is "false".

3. If the above points did not provide an answer, check chapter Chapter 8, *Reporting Changes* on how to create a detailed report about differences.

Chapter 13. Trademark Attributions, Library License Texts, and Source Code

Eclipse is a trademark of Eclipse Foundation, Inc.

IntelliJ is a trademark of JetBrains s.r.o.

Java and all Java-based trademarks are trademarks of Oracle Corporation in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft, and Windows are trademarks of Microsoft Corporation in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Chapter 14. Legal Notice

Appendix A. SonargraphBuild API Documentation

SonargraphBuild API is documented via JavaDoc that is available within the installation of the product.

Link to JavaDoc of SonargraphBuild API.

Index

R

S