Using Istanbul to integrate code coverage report in Jenkins

Istanbul is a code coverage tool which can be used to generate report on JS code coverage. The generated report returns statistics on how many and which functions, branches and line of JS code are being called during code execution. How Istanbule works: Istanbul parses and injects in the JS files instrumented code; running the instrumented code Istanbul can generate an HTML report of the code coverage.

The Post-build actions of Jenkins can be utilized to display the generated code coverage reports. When Jenkins builds the test solutions using ANT, ANT should trigger also the creationg of the code coverage reports.

Install Istanbul

Istanbull is using nodejs. First step is to install node http://nodejs.org/

Once node is configured, install Instabull by running the command:

```
$ npm install istanbul
```

Make sure that node and instanbul are both set in system variable PATH of the environment variables or node and istanbul command won't run. The Server might need to be restarted to make available for all the accounts the node and istanbul paths stored in the PATH system variables.

How to instrument the workspace using Istanbul

The code should be instrumented any time the build action of Jenkins is triggered. Istanbul should instrument the code during the build process of the test solution. The instrumentation of the code should happen after the workspace has been synced with the Repositories and before the export of the Servoy test solution is created. Therefore is required to modify the behavior of the build process by customasing the build.xml ANT files.

Customize Jenkins build process

Store the Jenkins configuration files in a separate repository. Having the configurations files in a separate repository allow to share the same configuration between multiple projects. The configuration files must include the build.xml ANT file and all the other config files. Any customization to the build process should be made in the build.xml file or in any related files used during the build. Servoy BAP modules use the configuration project shared in the GitHub repository at https://github.com/Servoy/svyJenkinsConfig. In order to include such repository the Jenkins job should checkout project from multiple SCM. Add an additional behavior to checkout each repository in a separate sub-folder of the workspace and provide an unique SCM name to each repository.

Configure Ant to point to the custom build.xml file. In the Build section of Jenkins set Build File to value

JenkinsConfig/svyJenkinsConfig/CBI_config/jenkins_build.xml

click on the button **Advanced** to set other configuration properties

work.servoy.install.dir = C:/Servoy
smart_test_solutions = servoyTestSolutionName
WORKSPACE = ${WORKSPACE}
ANT_CONTRIB_JAR = ${ANT_CONTRIB_JAR}
antRunner.dir = ${WORKSPACE}/JenkinsConfig/svyJenkinsConfig/CBI_config
servoy.test.property.file = ${WORKSPACE}/JenkinsConfig/svyJenkinsConfig/CBI_config/servoy.properties
instrument.exclude.modules = JenkinsConfig, solution_test, customFile.js
instrument.include.modules = solutionName

The property **instrument.modules** can be used to exclude files or folders from being instrumented. Use this property to create a cleaner report. You might prefer to review the instrumentation report of files of separate modules in their own project space and not include them in your project space.
Can use in combination with property instrument.include.modules to specify which modules should be included in the report. Can specify in the include property only modules name, is not possible to specify file names.

The svyJenkinsConfig project contains also a standard servoy.properties file which references only to the servoy_repository database. To use different servoy settings and to references other database provide your own servoy.properties file.

The export_and_test.properties contains all the ANT properties values used for the build process.
Set **instrument_code_for_code_coverage_report = yep** to create code coverage report during build and **always_fails_if_instrumentation_fails = true** to force build to fail when an error occurs during code instrumentation.

Instrumenting the workspace

Create an ANT task to instrument all the JS files in workspace, excluding the JS files of the test solution; ANT should later create an export using the instrumented JS files instead of the original JS files of the workspace. In such a way the instrumented code will be executed when running the test solution; this is required to generate the coverage-report.

The instrumented code will be saved in a second directory; the original files in workspace will not be overwritten. To instrument the code in workspace create an ANT task which is going to execute the instrument command of istanbul

```
$ istanbul instrument <workspace-directory> --output <temp-instrumented-workspace-directory> --preserve-comments -x
<exlude-pattern>
```

The **<workspace-directory>** is the standard workspace directory used by Jenkins to generare the export of the solutions. The <instrumented-workspace-directory> is output directory where all the instrumented JS files are going to be saved.

The option **--preserve-comments** is necessary to preserve all the JSDoc

Note that by default only the instrumented files are going to be saved in the **<temp-instrumented-workspace-directory>**. To copy all the non-javascipt files as-is include the option **--complete-copy** in the instrument command.

Generate JSDoc for the instrumented code

The instrumented code will add a variable and a block of code which should be self-executed on top of every JS file. In order to create an export of the Servoy solution which does not contains error all the instrumented JS files should be modied by

- adding JSDoc on top of the added variable with the UUID property definition of the variable.
- wrapping the block of code added by Istanbul in a self executing function.

To modify the instrumented files use a nodejs script. The modified files can be saved in a third directory **<intrumented-workspace-directory>**; this will be the workspace directory from where Jenkins is going to generate the export of the test solutions; the directory should contains also all the non-javascript file of the workspace togheter with the instrumented and modified JS files.

Save the code coverage results

The above steps will ensure that all the instrumented code is executed while running the test solution. As final step the coverage result should be written on a file before the test solution execution is terminated. When the solution is running, the instrumented code stores the coverage result in the top level variable **__coverage__**; __coverage__ is a JSON object. The value of this variable should be saved on file before closing the solution. At the event onSolutionClose of the testSolution save the value of the variable in the file **coverage.json** in the directory **<report-directory>**.

How to display the code coverage report using Jenkins Post-build actions.

The coverage.json file can be used to generate lcov report or Cobertura report. To generate the report use the istanbul commands

```
$ istanbul report --root <report-directory> --dir <report-result-directory>
```

and/or

```
$ istanbul report cobertura --root <report-directory> --dir <report-result-directory>
```

The option **--root <report-directory>** specifies to istanbul which is the root directory containing the coverage.json file. The option **--dir** is used instead to specify on which directory istanbul should create the report.

The first command creates a lcov report. Open the index.html page to view the code coverage report. The second commands generates instead the cobertura.xml file.

Installing the Plugins

To display the coverage report results on jenkins install the Jenkins plugins HTML Publisher, https://wiki.jenkins-ci.org/display/JENKINS/HTML+Publisher+Plugin, and/or the plugin Cobertura Plugin, https://wiki.jenkins-ci.org/display/JENKINS/Cobertura+Plugin.

Setup the HTML Publisher Plugin

Create the Post-build action in Jenkins **Publish HTML reports**. Specify the relative path from the Jenkins workspace to the index.html page created by the lcov istanbul report command. The index.html page will be then accessible from Jenkins.

Setup the Cobertura Plugin

Create the Post-build action in Jenkins **Publish Cobertura Coverage report**. Specify the relative path from the Jenkins workspace to the cobertura-coverage.xml file created by the cobertura istanbul report command. The Cobertura plugin will not show any report if the build fails.

HTML Publisher Plugin vs Cobertura Plugin

The reports of the HTML Publisher plugin and the Cobertura Plugin are both showing the same results. The first one provides better annotated source but the Cobertura Plugins provides trend information on how many functions, methods, branches has been found in code.

Gotcha's

- Servoy JS files are lazy-loading. If files are never referenced on code code coverage will ignore them. Solutions: 1) move the variable and the self-executing block of code of each instrumented JS file in a single scope and force the loading of the scope at the solution open. 2) modify the coverage.json file by adding the sources to the ignored JS files.

- TODO after the second build Cobertura loss data in the source files. Won't be able to display the lines of code until the workspace is deleted.
- Servoy Javascript uses the Rhino Javascript. Istanbul does not recognize all Rhino commands, won't be able to instrument the file when is not able to parse the code and will copy the file as-is.
  Known commands not recognized are:
    plain XML code
    try {} catch (e if e instanceof ....) {}
    for each loops
- TODO Servoy variables and Prototype functions are always reported as never executed.
- NodeJs is asynchronous, read-write operations on files are not immediate.
- Use Ant Patched to use formatted junit report result (.xm1 extension)
- in the Jenkins job config is not possible to use custom properties (example servoy.test.property.file = ${antRunner.dir}/servoy.properties) since ANT wont translate them to full path.