**Context:**

Most of the teams are taking code coverage for unit tests with the plugins available in integration with Jenkins as part of CI.

Solution provided below is to calculate code coverage for Functional tests for explicitly Services.

This is a strategy for achieving reliable quality through identifying untested areas of application for quick benefits.

**Who all can Use:**

Rest Services based QA Automation frameworks, Manual functional Test Teams can use to calculate code coverage for the components (.jar's,.war's and .ear's).

**Details:**

Code Coverage analysis can be generated for required builds by following execution process:

            i. Integrate Instrumentation with Build

            ii. Deploy instrumented application

            iii. Collect coverage data during/post testing

            iv. Final Report Generation

**What it supports?**

Apart from common coverage tool support, POC is capable of:

* This POC Supports communication with a JVM running instrumented classes via a TCP socket, that is a added advantage.
* Analyze coverage for N number of binaries.
* Extendable to merge coverage data for consolidated reports
* Extendable for CI

**Code Coverage using EMMA:**

Pre-Requisites

•          Ant installed: <http://ant.apache.org/>

•          Emma installed: <http://emma.sourceforge.net/>

**Usage**:

•          Create emma directory and copy the contents (lib,build.xml).

Ex:  emma/build.xml

•          Under emma/ dir, execute “ant -f build.xml”

•          Restart Jboss

•          Run Tests

•          Under emma/ dir, execute “ant –f build.xml report”

•          View coverage.html to view the result.

**Explanation:**

Step 1: Set classpath to include Emma and ANT binaries

Step 2: Settings to use ANT tasks

Step 3: Initialize directories

Step 4: Instrument the .jar's/.classes/.war's/.ear's

Step 5: Collect the data from JVM Port runtime

Step 6: Generate Coverage Report

**Tool Evaluation:**

|  |  |
| --- | --- |
| **Parameters** | **Comments** |
| **Coverage Levels** |  |
| Package |  |
| Class |  |
| Method |  |
| Block |  |
| Line |  |
| File |  |
| **Report Clarity** |  |
| Hit Count | No. of times the statement/code block is hit |
| Source Linking | Ability to link coverage Report and Source Code |
| **Exclusion Management** |  |
| Source File Exclusion |  |
| Exclusion Patterns Support | Ability to exclude certain areas of code from reporting |
| **Advanced Reporting** |  |
| HTML Reports |  |
| Incremental Reporting |  |
| Base lining & Versioning | Only for newly added code |
| **Platform Support** |  |
| Command Line |  |
| **Licensing** |  |
| Well Documented |  |
| Open Source |  |
| Community based Support |  |
| **Technical Aspects** |  |
| Source Level Instrumentation |  |
| Merging |  |
| Performance |  |

**Tool References:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Technology** | **Tools** |
| 1 | Java | Emma, Cobertura |
| 2 | .NET | NCover, PartCover |
| 3 | C/C++ | BullsEye, CoverageMeter |

**Other Details:**

**Command line support for coverage useful for CI Integration:**

ant -f build.xml -Demma.dir=. -Ddeploy.dir=/<jboss deploy dir> -Ddeploy.lib=<jboss default lib>

**For report Generation:**

ant -f build.xml -Demma.dir=. report

**Sample script snippet for coverage collection from JVM :**

<emma>

<ctl connect="${jvm.host.port}">

<command name="coverage.get" args="${coverage.dir}/${coverage.file},true"/>

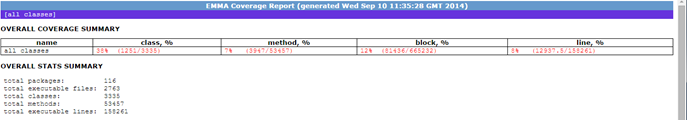
<command name="coverage.reset"/>

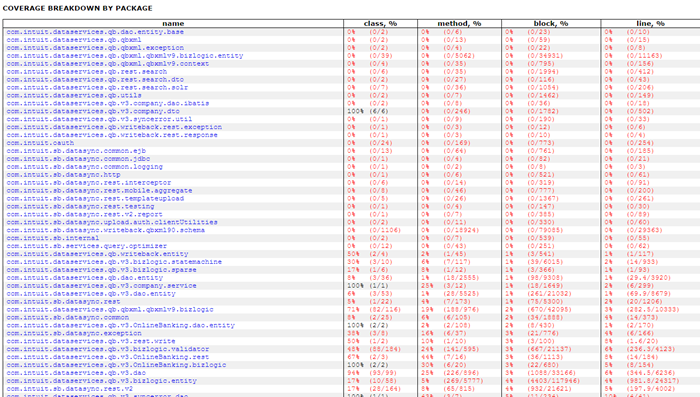
</ctl>

</emma>

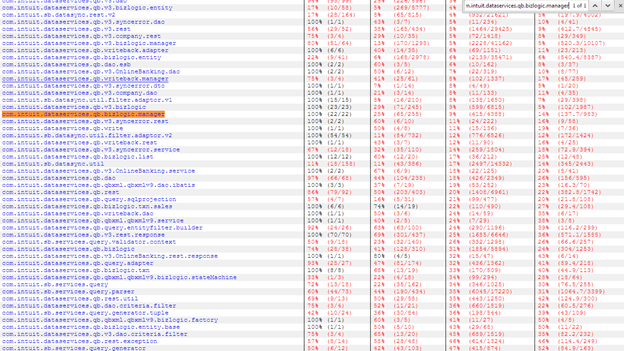
**Sample Reports:**

Overall Coverage

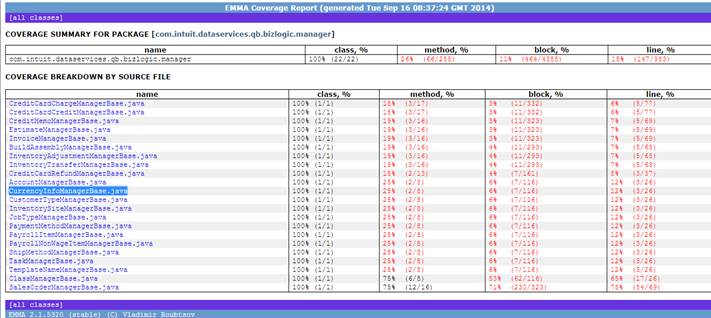




Package Level



Class Level



Method Level

