**Cruise Control** is the tool used for triggering the build scripts at a user defined intervals.

The Cruise Control contains the following configuration,

1. **cc.properties**: Main configuration file for Cruise Control

* cc.home – Cruise Control tool bin path. Eg: E:/CIT with Maven triger from CC/cruisecontrol-bin-2.8.4
* maven2home – Maven 2 bin path. Eg: E:/CIT with Maven triger from CC/cruisecontrol-bin-2.8.4
* starteam.home – StarTeam SDK Home. Eg: C:/Program Files/Borland/StarTeam SDK 11.0
* stserver – StarTeam server
* stport – StarTeam port number
* stprojec t – StarTeam project
* stview – StarTeam view
* rootserverfolder – StarTeam root server folder
* stuser – StarTeam Username
* stpassword – StarTeam Password
* connection.url – SCM socnnection URL

Eg: scm:svn:username:password@serverhost:portnumber:/project/view/root/projectname

* developerconnection.url – SCM socnnection URL

Eg: scm:svn:username:password@serverhost:portnumber:/project/view/root/projectname

* connection.url – SCM socnnection URL

Eg: scm:svn:username:password@serverhost:portnumber:/project/view/root/projectname

1. **config.xml**: Contains the configuration to be made to trigger Maven Script

* <property file="cc.properties"/> - to read external property file
* <project name="${projectname}"> - To read project name from cc.properties
* <starteambootstrapper files="pom.xml cc.properties"> - Specifies what files are to be downloaded via Cruise Control
* <schedule interval="10"> - Specifies the time interval for Maven trigger
* <maven2 mvnscript="${maven2home}" pomfile="projects/${project.name}/pom.xml"

goal="initialize scm:update clean install checkstyle:checkstyle">

</maven2>

* 1. mvnscript – Specifies the path to the Maven bin(Read from cc.properties)
  2. pomfile – Location of the project’s pom.xml
  3. goal – Specifies the tasks to be executed
* initialize – Invoke initialize phase to read cc.properties
* scm:update – Maven SCM command for StarTeam Checkout
* clean – Maven command to clean previous builds
* install – Maven compile command
* checkstyle: checkstyle – Used to check coding standards for quality control (HexCodeVerifier)

**Maven Configuration part:**

The code check out, compilation and coding standards check are done using the Maven script (pom.xml). The following are the configurations,

To make maven not to hit the internet and to take our local server repo as default add the following mirror configuration in the settings.xml found in conf folder of maven which is set as maven home,

<mirrors>

<mirror>

<mirrorOf>\*</mirrorOf>

<name>repo</name>

<url>http://172.25.108.50:8081/artifactory/repo</url>

<id>repo</id>

</mirror>

</mirrors>

The Essential Maven Plugin apart from project specific dependencies are

1. maven-scm-plugin
2. maven-checkstyle-plugin
3. properties-maven-plugin

**1.** **Maven SCM Plugin Configuration:**

The SCM Plugin offers vendor independent access to common scm commands by offering a set of command mappings for the configured scm. Each command is implemented as a goal.

**Step 1:** Adding a Maven SCM plugin dependency

<dependency>

<groupId>org.apache.maven.scm</groupId>

<artifactId>maven-scm-provider-starteam</artifactId>

<version>1.8</version>

</dependency>

**Step 2:** Configuring the maven scm plugin

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-scm-plugin</artifactId>

<version>1.8</version>

<configuration>

<checkoutDirectory>${basedir}</checkoutDirectory>

</configuration>

<executions>

<execution>

<id>perform-checkout</id>

<configuration>

</configuration>

</execution>

</executions>

</plugin>

* <checkoutDirectory> - Specifies the local directory to check out. ${basedir} checks out the project to base directory. Default- ${basedir}/target/checkout

**Step 3:** Provide connection URL (read from cc.properties) to the required SVN server

<scm>

<connection>${connection.url} </connection>

<developerConnection>${developerconnection.url} </developerConnection>

<url> >${url}</url>

</scm>

**2. Maven CheckStyle Plugin Configuration**

The Checkstyle Plugin generates a report regarding the code style used by the developers. The plugin can be configured in the project's POM. Predefined rulesets are included with the plugin, these are: sun\_checks.xml, turbine\_checks.xml, avalon\_checks.xml and maven\_checks.xml. We used HexCodeVerifier on top of checkstyle to define custom coding standards. The following are the configurations,

**Step 1:** Add the following dependencies in the maven-checkstyle-plugin.pom

<dependency>

<groupId>com.hex.HexCodeVerifier</groupId>

<artifactId>HexCodeVerifier</artifactId>

<version>1.0</version>

</dependency>

<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc14</artifactId>

<version>1.0</version>

</dependency>

<dependency>

<groupId>jakarta-regexp</groupId>

<artifactId>jakarta-regexp</artifactId>

<version>1.3</version>

</dependency>

<dependency>

<groupId>eclipse</groupId>

<artifactId>jdtcore</artifactId>

<version>1.0</version>

</dependency>

**Step 2:** Add the following entries into the project’s pom.xml under build tag

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-checkstyle-plugin</artifactId>

<version>2.6</version>

<executions>

<execution>

<id>checkstyle</id>

<goals>

<goal>checkstyle</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</pluginManagement>

**Step 3:** Add the plugin lifecycle configuration under plugins tag along with other plugins,

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-checkstyle-plugin</artifactId>

<version>2.6</version>

<executions>

<execution>

<id>checkstyle</id>

<phase>install</phase>

<goals>

<goal>checkstyle</goal>

</goals>

</execution>

</executions>

</plugin>

**Step 4:** Add this after the closing of build tag,

<reporting>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-checkstyle-plugin</artifactId>

<configuration>

<configLocation>CodingStandards\_Java.xml</configLocation>

<sourceDirectory>./</sourceDirectory>

<consoleOutput>true</consoleOutput>

<useFile>Report.txt</useFile>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jxr-plugin</artifactId>

</plugin>

</plugins>

</reporting>

* configLocation – Specifies where the config file is located in the project (Configured as Base Directory)
* sourceDirectory – Specifies where the source file are in the project
* consoleOutput – Specifies whether to display logs in the console
* useFile – Specifies a text file to which the log is written

**3. Maven properties plugin configuration:**

The Properties Maven Plugin provides goals to read and write properties from and to files, and also to set system properties. The following are the configuration,

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>properties-maven-plugin</artifactId>

<version>1.0-alpha-2</version>

<executions>

<execution>

<phase>initialize</phase>

<goals>

<goal>read-project-properties</goal>

</goals>

<configuration>

<files>

<file>${basedir}/cc.properties</file>

</files>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

* <phase>initialize</phase> – Specifies that the properties are read at initialize phase
* <goal>read-project-properties</goal>– Goal used to read a property file
* <file>${basedir}/cc.properties</file> – Specifies the path of the property file to be read