Summary

The personal build is a serial process that a software developer writes a software source code and complies it in his local PC to convert it to an executable code. The personal build is usually carried out by using a build tool.

Description

The typical open source build tools include make, ant, maven, etc. The e-government standard framework uses maven as its build tool.

Open source build tool

- GNU Make [http://www.gnu.org/software/make/]
  The make is a build tool usually used in Linux-based operating system.
- Apache Ant [http://ant.apache.org/]
  The ant is a Java-based build tool and reduces difficulties with using make in Java projects, while providing more functions.
  - Compiles Java source files
  - Creates jar, war, ear and zip files
  - Runs javadoc to create helps.
  - Set up dependencies on each task
- Maven – the standard build tool for e-government standard framework [http://maven.apache.org/]
  Maven doesn't only carry out the build function as a project management tool, but also provides project reporting and documentation.

Maven

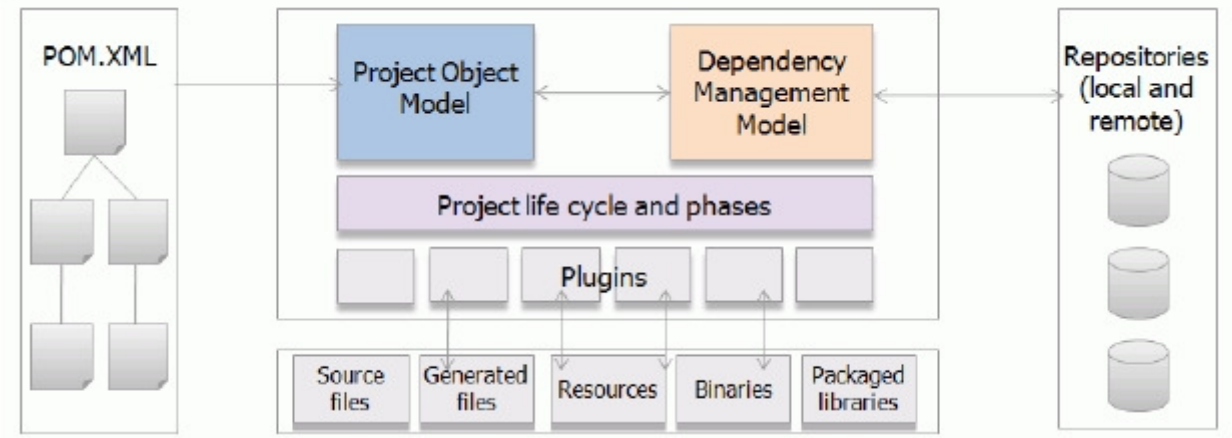
The maven is a project management tool that provides project dependency management, library management and project life cycle management based on the concept of Project Object Model (POM). It doesn't only carry out the build function that creates distributable products from source codes based on plug-ins, but also provides project reporting and documentation

Features of Maven

Advantages	Disadvantages
· Excellent dependency management - Automatic dependency update - Integrated library management through repository · Consistent usage that is easily applicable to all projects · Repository continuously expanding to store libraries and meta · Scalability through easily written plug-ins · Easy setup-based mechanism that can handle multiple projects at the same time · Distribution management through simple setup	· Inconvenient repository management - With the rapid expansion of maven projects, the libraries provided by the central repository are increasing fast, but still some libraries are not provided including the 3rd party libraries. · pom.xml file management - The pom.xml could be lengthy as all information on the maven project is contained in the file · Limitation on complicated build functions specialized for projects - The support for details and specialized build environments is weak, as the life cycle commonly used for software builds is used.

Maven Architecture

The maven is composed of POM, which is responsible for description and setup of a whole project, dependency/repository management model for libraries (artifacts in maven), life cycle that handles the build life cycle of compiling, testing and packaging, and plug-ins.



POM	Maven engine included. Provided declaratively in POM.XML file. – refer to POM
Dependency management model	Uses local and remote repositories – refer to Dependency Management
Project life cycle module	Maven engine uses plug-ins to carry out all tasks that handles files – refer to build lifecycle

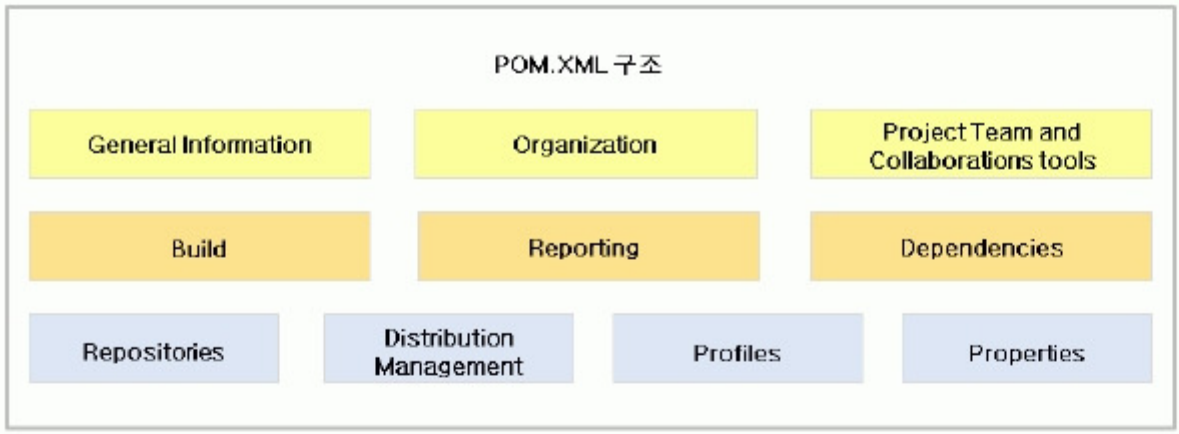
POM

POM contains project information including environment setup and dependency management and this information is written in the pom.xml file which is created as a basic setup file when a maven project is created.

Description

The pom.xml file is largely composed of 10 parts including detailed meta data of a project.

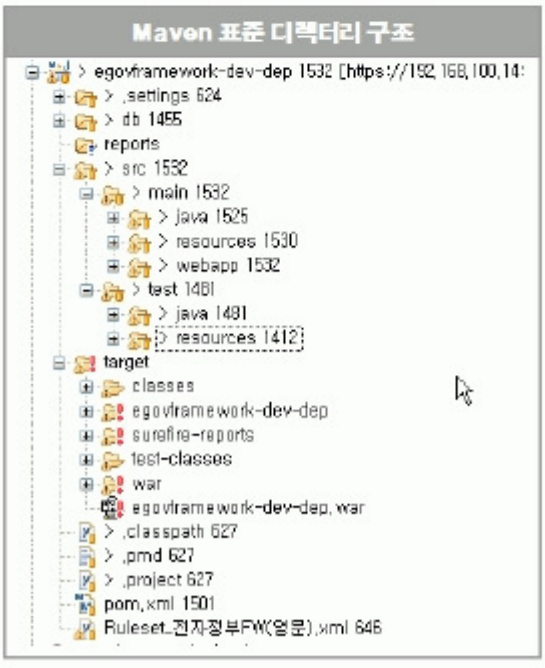
It contains general project information, version and setup management, build environment, library repository and dependency.



- General Information
  Project name, description and version information. – refer to project information creation
- Organization: project organization information: name, website URL
- Project Team and Collaborations tools
  Configuration management server, issue tracker, integrated build server information, etc.
- Build
  Build life cycle environment setup (e.g. encoding information)
- Reporting
  Report creation function setup.
- Dependencies
  Declares the libraries used in a project and includes in the build path. – refer to Dependency setup.
- Repositories
  Sets up the library repository location
- Distribution Management
  Sets up the distribution environment
- Profiles
  Sets up the build to improve portability in a disparate environment.
  Maven build profile introduction [http://maven.apache.org/guides/introduction/introduction-to-profiles.html]
- Properties – sets up project properties.

Structure of the Maven directory

Maven provides a directory structure fully qualified based on the best practices and all source files are placed in /src directory and built outputs are placed in /target directory.
A maven project provides a mechanism accessing the project resources based on fully qualified directory structure without setting up of the source/resource path by the developer.
Maven doesn't require path setup for sources or resources.



디렉터리/파일	설명
/pom.xml	프로젝트 객체 모델. 해당 프로젝트에 대한 전반적인 정보를 갖는다.
/src/main/java	Java 소스 파일 위치
/src/main/resources	배포할 리소스, XML, properties, ...
/src/main/webapp	웹 어플리케이션 관련 파일 위치(WEB-INF, css 등)
/src/test/java	테스트 케이스 java 소스
/src/test/resources	테스트 케이스 리소스
/target	빌드 된 output이 위치하는 디렉터리

Dependency Management

Maven sets up declarative dependency in POM to manage the libraries required for build and distribution and downloads the libraries declared in local and remote repositories.

Description

One of main advantages of maven is the library dependency management. In general, to set up libraries, developers download libraries into the project directories directly and set up the path to use them. In this case, there is inconvenience to find out and set up libraries, and there could be a problem with controlling the types and versions of the libraries. Especially in the integrated build process, library-related problems can occur frequently. To handle these problems, maven provides the dependency management mechanism and maven developers can download and use easily by declaring dependency in the project pom.xml. Project administrators can control the libraries and versions effectively and provide developers with verified and unified versions of libraries.

Example) You can declare as in the following to use the libraries for junit test.

```
<project>
  ......
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.4</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project
```

Dependency analysis sequence

1. Check dependency in the local repository
2. Check dependency in the remote repository list
3. If above 1 and 2 fail, report dependency error

Items requiring dependency declaration

1. <groupId>: Provided to identify library sets in projects or organizations.
2. <artifactId>: Actual project name, combined with groupId to identify projects.
3. <version>: Declared dependency artifact version, allowing consistency of artifacts used in projects.

Dependency scope

1. compile: the default value, usable in all class paths.
2. provided: similar to compile, but not included in the package. It is provided by the container or JDK. Example) Servlet API for web apps
3. runtime: Used during runtime, not compiling. Example) JDBC drivers
4. test: Used only in the test phase. Example)Junit
5. system: Similar to provided, but the developer should provide the JAR file himself and it does not look for the dependency designated by the repository.

The dependency scope is not declared and, by default, the compile is applied. Since a conflict may occur between the libraries provided by JDK or WAS and the libraries declared in dependency, be sure to specify the scope of the libraries you want to use.

Manual

In pom.xml write the project name, packaging method, version information, project description, etc. and describe the project environment setup, build method and library information to create the build and project management environment.

Create project information

Writes project groupId, artifactId, packaging type, version information and project description.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>egovframework.guideprogram</groupId>
  <artifactId>egovframework.guideprogram.basicsample</artifactId>
  <packaging>war</packaging>
  <version>1.0.0-SNAPSHOT</version>
  <name>egovframework.guideprogram.basicsample</name>
  <url>http://maven.apache.org</url>

  <properties>
    <!-- Spring version -->
    <spring.maven.artifact.version>2.5.6</spring.maven.artifact.version>
    <compileSource>1.5</compileSource>
    <encoding>UTF-8</encoding>
  </properties>
</project>
```

Repository configuration

Set up the repository to use. If using a local repository such as Nexus in a project, you can set up the connection information to connect to Nexus and to download the artifacts

```
  <repositories>
    <repository>
      <id>central</id>
      <releases>
        <enabled>true</enabled>
      </releases>
      <url>http://192.168.100.15:8081/nexus/content/groups/public</url>
    </repository>
    <repository>
      <id>snapshots</id>
      <snapshots>
        <enabled>true</enabled>
```

```
        </snapshots>
        <url>http://192.168.100.15:8081/nexus/content/groups/public-snapshots</url>
      </repository>
    </repositories>
```

Dependency configuration

Declare the libraries (artifacts in Maven) to use in the project.

```
    <dependencies>
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring</artifactId>
        <version>${spring.maven.artifact.version}</version>
      </dependency>
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>2.5</version>
        <type>jar</type>
      </dependency>
    </dependencies>
```

System dependency configuration

The dependency that has system scope should be always usable as an artifact that the repository doesn't look for. It provide the dependencies provided by JDK or VM. JDBC standard extension or Java Authentication and Authorization Service(JAAS) is a typical example.

.

jdbc standard extension - example

```
<project>
  ...
  <dependencies>
    <dependency>
      <groupId>javax.sql</groupId>
      <artifactId>jdbc-stdext</artifactId>
      <version>2.0</version>
      <scope>system</scope>
      <systemPath>${java.home}/lib/rt.jar</systemPath>
    </dependency>
  </dependencies>
  ...
</project>
```

JDK's tools.jar declaration example

```
<project>
  ...
  <dependencies>
    <dependency>
      <groupId>sun.jdk</groupId>
      <artifactId>tools</artifactId>
      <version>1.5.0</version>
      <scope>system</scope>
      <systemPath>${java.home}/../lib/tools.jar</systemPath>
    </dependency>
  </dependencies>
  ...
</project>
```

Reporting configuration

Maven doesn't only provide the basic report for the project information, but also a function to compose documents such as emma, JavaDoc by setting up plug-ins.

```
  <reporting>
    <outputDirectory>${basedir}/target/site</outputDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>2.0.1</version>
      </plugin>
      <!-- JUnit Test Results & EMMA Coverage Reporting -->
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>emma-maven-plugin</artifactId>
        <inherited>true</inherited>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>surefire-report-maven-plugin</artifactId>
        <inherited>true</inherited>
      </plugin>
      <!-- Generating JavaDoc Report -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
        <configuration>
          <minmemory>128m</minmemory>
          <maxmemory>512m</maxmemory>
          <encoding>UTF-8</encoding>
          <docencoding>UTF-8</docencoding>
          <charset>UTF-8</charset>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
```

Samples

pom.xml - example

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>egovframework.dev.com</groupId>
  <artifactId>egovframework-dev-com</artifactId>
  <version>1.0</version>
  <packaging>war</packaging>
  <name>egovframework-dev-com Maven Webapp</name>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.4</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.5</source>
        <target>1.5</target>
      </configuration>
    </plugin>
  </plugins>
</project>
```

pom.xml –example (in Hudson)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.jvnet.hudson</groupId>
    <artifactId>hudson</artifactId>
    <version>1.6</version>
    <relativePath>../pom.xml</relativePath>
  </parent>

  <groupId>org.jvnet.hudson.main</groupId>
  <artifactId>pom</artifactId>
  <version>1.274</version>
  <packaging>pom</packaging>

  <name>Hudson main module</name>
  <description>The module that constitutes the main hudson.war</description>

  <modules>
    <module>remoting</module>
    <module>core</module>
    <module>maven-agent</module>
    <module>maven-interceptor</module>
    <module>war</module>
    <module>test</module>
  </modules>

  <scm>
    <connection>scm:svn:https://svn.dev.java.net/svn/hudson/tags/hudson-1_274</connection>
    <developerConnection>scm:svn:https://svn.dev.java.net/svn/hudson/tags/hudson-1_274</developerConnection>
    <url>https://hudson.dev.java.net/source/browse/hudson/tags/hudson-1_274</url>
  </scm>

  <build>
    <defaultGoal>install</defaultGoal>
    <plugins>
      <plugin>
        <artifactId>maven-release-plugin</artifactId>
        <version>2.0-beta-8</version>
        <configuration>
          <!-- enable release profile during the release, create IPS package, and sign bits. -->
          <prepareVerifyArgs>-P release,ips,sign</prepareVerifyArgs>
          <!--
            also run assembly during the release.
            http://www.nabble.com/Release-Plugin%3A-Include-assemblies-for-deploying-tf2642295s177.html#a7377938
           -->
          <goals>install javadoc:javadoc animal-sniffer:check assembly:attached deploy</goals>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-assembly-plugin</artifactId>
        <inherited>false</inherited>
        <configuration>
          <finalName>hudson-${version}</finalName>
          <descriptors>
            <descriptor>assembly-src.xml</descriptor>
          </descriptors>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-remote-resources-plugin</artifactId>
        <executions>
          <execution>
            <goals>
              <goal>process</goal>
            </goals>
            <configuration>
              <resourceBundles>
                <resourceBundle>org.jvnet.hudson:license:1.0</resourceBundle>
              </resourceBundles>
            </configuration>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <!-- make sure our code doesn't have 1.6 dependencies except where we know it -->
```

```
            <groupId>org.jvnet</groupId>
            <artifactId>animal-sniffer</artifactId>
            <version>1.2</version>
            <configuration>
              <signature>
                <groupId>org.jvnet.animal-sniffer</groupId>
                <artifactId>java1.5</artifactId>
                <version>1.0</version>
              </signature>
            </configuration>
          </plugin>

          <!--<plugin>
            <groupId>org.jvnet.fix1600</groupId>
            <artifactId>fix1600</artifactId>
            <executions>
              <execution>
                <goals>
                  <goal>fix</goal>
                </goals>
              </execution>
            </executions>
          </plugin>-->
        </plugins>
      </build>

      <dependencies>
        <dependency>
          <!-- for JRE requirement check annotation -->
          <groupId>org.jvnet</groupId>
          <artifactId>animal-sniffer-annotation</artifactId>
          <version>1.0</version>
          <optional>true</optional><!-- no need to have this at runtime -->
        </dependency>
      </dependencies>

      <properties>
        <maven.version>2.0.4</maven.version>
      </properties>

      <profiles>
        <profile>
          <id>debug</id>
          <activation>
            <activeByDefault>true</activeByDefault>
          </activation>
          <properties>
            <hudson.sign.alias>hudson</hudson.sign.alias>
            <hudson.sign.keystore>../dummy.keystore</hudson.sign.keystore>
            <hudson.sign.storepass>hudson</hudson.sign.storepass>
          </properties>
        </profile>
      </profiles>

      <repositories>
        <repository>
          <id>m.g.o-public</id>
          <url>http://maven.glassfish.org/content/groups/public/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </repository>
      </repositories>

      <pluginRepositories>
        <pluginRepository>
          <id>m.g.o-public</id>
          <url>http://maven.glassfish.org/content/groups/public/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>

      <licenses>
        <license>
          <name>The MIT license</name>
          <url>http://www.opensource.org/licenses/mit-license.php</url>
          <distribution>repo</distribution>
        </license>
      </licenses>

      <distributionManagement>
        <site>
          <id>hudson-www</id>
          <url>java-net:/hudson/trunk/www/maven-site/</url>
        </site>
      </distributionManagement>
    </project>
```

References

Apache Maven project [http://maven.apache.org/]