Geomajas contributor guide

Geomajas Developers and Geosparc

Geomajas contributor guide

by Geomajas Developers and Geosparc

1.10.0

Table of Contents

List of Figures

List of Tables

List of Examples

Chapter 1. Developers information

1. maven compilation, targets, profiles, variables

When doing an initial compilation of Geomajas, you may need to start compilation from the "build-tools" and then the "backend" directories. Only when these are compiled, compilation from the project root will succeed.

```
cd build-tools
mvn install
cd backend
mvn install
cd ..
mvn install
```

The source contains one main pom which allows building of the Geomajas framework and each of the sample applications in one go.

You can also choose to build them individually.

There are a couple of profiles defined which should help during development:

- `-DskipShrink`: do not use shrinking when building or using the dojo face. When not specified, a shrinked version of the javascript files is used. The files are compressed and combined for faster loading and better caching.
- -DskipDocs: do not build the documentation module. Can speed up the build a little.
- -DskipGwt: skip the GWT compilation phases. Useful when you just want a quick compile or don't want/need the compile to JavaScript.
- `-Dfull-build`: from the root project, this enables inclusion of the build tools and documentation in the build. This is actually enabled by default (to disable use `-Dhudson`"),
- `-Dhudson`: profile for running the selenium integration tests on the Jenkins (previously Hudson) continuous integration server. As long as running the tests on the ci server proves problematic, this will disable these tests.

1.1. GWT build

For faster compilation during testing (when not using development mode), it can be useful to compile only for the browser used for testing. This will reduce the number of compilation steps by a factor 6. Removing supported languages can further remove compilation steps. Include the following excerpt in your `Xxx.gwt.xml` file to set your target browser.

```
<!-- set target browser to compile for, use this to limit to the browser used f
<!-- where value = "ie6/opera/gecko1_8/safari/gecko" , "gecko1_8" is FireFox 3
<set-property name="user.agent" value="gecko1_8" />
```

1.2. dojo build

For development using the dojo face, apart from using the "-DskipShrink" setting mentioned higher, you may also want to configure the ResourceController to try to directly read the JavaScript files from disk before looking at the classpath (it also changes the cache headers). This allows a simple refresh in the browser to load the changed versions. You can configure this using a init-param for the dispatcher servlet, like in this example.

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-c
    <init-param>
        <param-name>files-location</param-name>
        <param-value>/home/me/apps/java/geomajas/geomajas/geomajas-dojo-client/
        <description>
            When this is specified, files are searched here first.
            Files which are found at this locations are not cached.
        </description>
    </init-param>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*:META-INF/geomajasWebContext.xml</param-value>
        <description>Spring Web-MVC specific (additional) context files.</descr
    </init-param>
    <load-on-startup>3</load-on-startup>
</servlet>
```

1.3. Running the example applications

Once you have done a "`mvn install`" on either the entire tree or the "Geomajas" directory, you can use maven to run the example applications.

For the dojo face, you can run the examples using (when in the geomajas-dojo-example directory)

```
mvn jetty:run
```

For the gwt face, you have three options. Once in the geomajas-gwt-simple directory, you can run the application in development mode using

```
mvn gwt:run
```

Note

Due to classpath problems and the gwt-maven-plugin which does not properly handle excluded dependencies (the "provided" scope), this can fail on some systems.

You can also use jetty to run the normally built application

```
mvn jetty:run
```

Alternatively you can run the actual war using

```
mvn jetty:run-war
```

Note

It can be advisable to run "mvn clean" between "gwt:run" and "jetty:run-war" or the classpath problem from the previous footnote may appear again.

2. Documentation

The general documentation is split in three books.

- developers guide: guide for developer who want to use Geomajas in their application.
- contributors guide: guide for people who want to contribute to the project or want to know more about the functioning of the project (this one).
- end user guide: documentation for end users of applications built using Geomajas.

Apart from that, each face and each plug-in has their own documentation.

All documentation is written in docbook format to allow both PDF and HTML output formats. The sources can be found in the "documentation" directory of the project.

For editing the docbook files, we recommend using XMLMind [http://www.xmlmind.com/xmleditor/]. The personal version is free and can (at the time of writing) be used for editing open source documentation.

The docbook files are currently formatted using XMLMind. When using another tool for editing, please keep the current formatting to assure diffs remain usable.

The documentation includes a lot of examples which are excerpts from the source of the example applications. This prevents copy-paste mistakes. The build process for the documentation automatically updates these excepts. The directories which have to be scanned for excepts are specified in the pom. When this includes code which is not in the current versioned entity (the root directory for the face or plug-in), then the source needs to be obtained from a dependency and unpacked. Excerpts can be annotated using annotations like

```
<!-- @extract-start AllowAllSecurity, Allow full access to everybody -->
<bean name="security.securityInfo" class="org.geomajas.security.SecurityInfo">
    <property name="loopAllServices" value="false"/>
</bean>
<!-- @extract-end -->
```

for XML or

```
// @extract-start filename, title
for (String line : lines) {
    // do something
}
// @extract-end
```

for java files. The start annotation includes the filename which should be used (all files are placed in the "listing" directory) and optionally a title for the example.

3. API contract

The Geomajas project has a very strong API contract. To assure the project adheres to this contract, we have the following requirements;

- No API classes or interfaces may be removed.
- No API classes or interfaces may be renamed.
- No API classes or interfaces may have their package name modified.
- No API methods may be removed.
- No API methods may have their signature changed.
- No methods may be added to classes annotated using "`@UserImplemented`".
- Each class on which a "@Api" annotation is added should have a "@since" javadoc comment.

- Each method on which a "@Api" annotation is added should have a "@since" javadoc comment.
- Each public method which is added in a class which is annotated with "@Api(allMethods = true)", should have a "@since" javadoc comment.

The checkstyle configuration which is used for the project (which is defined in the geomajas-parent parent) tries to check the API contract. This required a api.txt file in src/main/resources which contains the API for the previous release version. The API for the compiled version is put in target/api.txt.

Note that apart from the class and method signatures, the behaviour should also remain constant (especially when documented or tested). Just keep a method and throwing `NotImplementedException` cannot be considered "maintaining a stable API".

4. Versioning

Version have a major.minor.patch structure.

- major: indicates that this release has major advances over previous releases. New major versions do not need to be backwards compatible.
- minor: indicates that there are important new features that do not break compatibility with previous versions with the same major number. Even minor versions are used for "stable" versions which will be supported by Geosparc. Odd minor versions are used for work-in-progress and stabilisation efforts.
- patch: bugfixes and smaller improvements.

5. subversion, commits

New committers need to sign an agreement which hands over copyright to Geosparc. Policies are needed for assigning commit rights (see below).

All SVN commits should include the JIRA issue number at the start of the commit message, and a short description of the work done. The JIRA issue number allows linking the commits with the issues (as can be seen in JIRA), the short message allows persons to know what is happening without referring to JIRA. The only times JIRA issue number are not needed is for making "obvious" changes like fixing typos.

Commits should be grouped by issue as much as possible/sensible (better two commits than one commit for fixing two issues, better one commit of five files than five commits of one file (for one issue)).

Development of the "latest-and-greatest" version happens in "trunk".

Continued development on earlier versions (when not "latest-and-greatest") occur in branches with the future version number as name.

When a release is cut, a tag with the release version as name is created. The release should be built from the tagged files.

After each commit, the system should still compile and all test cases should still succeed. There is a continuous integration engine (Jenkins) which verifies this and send messages to the commit mailing list on failures.

6. Coding

Note that details about coding style and naming are on the Chapter 2, *Coding quality and style* page.

6.1. Logging

- When inserting debug statements, parameterized messages should be used to prevent the need/usefulness of `isDebugEnabled()`.
- all logging is done through slf4f, logger is created using

  ```
  private final Logger log = LoggerFactory.getLogger( ContainingClassName.class
  ```

- logging levels

  ### Table 1.1. logging levels

log level	default on	use
ERROR	yes	major problems, should always be visible in logs and are likely to require action from a person (to fix the condition or assure it does not happen again). Indicates that something is seriously wrong.
WARN	yes	warning about potential problems. Should always be visible in logs and a person will probably need to assess whether this is harmless or should be treated as an error.
INFO	yes	important information. You can assume this level is on in production, so it should be carefully considered whether this level is appropriate. In general only used to indicate service status (started, stopped).
DEBUG	no	logging information which is detailed enough to know what is happening in the system, without flooding the logs.
TRACE	no	very detailed logging, probably only making sense to the developer of the code.

- When an exception is caught and (another exception) thrown you should not log the exception. You should however include the cause in the newly thrown exception.

6.2. Unit testing

Unit testing: At least each class implementing the public API should have a unit test, testing all methods. For testing JUnit is used.

- Advantages of unit testing:
  - Capturing a JIRA [http://jira.geomajas.org/] bug report in a reproducible manner.
  - Allowing you to specify exactly the behaviour you want, before you start coding.
- How unit testing should be done:

  - If you are testing src/main/java/org/geomajas/ToBeTestedClass.java, create a class src/test/java/org/geomajas/ToBeTestedClassTest.java. Actual test methods have a name starting with "test". The class itself should extend jnit.framework.TestCase.
  - The test will automatically be run when running "`mvn install`".
- Integration tests should also be provided. These can also be used for testing the user interface (thanks to selenium).

6.3. Exception handling

Never throw away exception, either log them or throw them again (possibly wrapped). Do not log and throw, this only clutters log files with duplicate exceptions.

Do not wrap exceptions unnecessarily (so no `GeomajasException` caused by a `GeomajasException`) unless you add additional information in the message.

When wrapping an exception, always include the cause.

6.4. Refactoring

Changes in the (public) API use a "deprecate, then remove" cycle. It should be marked "deprecated" in at least one minor version before it can be removed in the next major version.

6.5. File encoding

All source files, including .properties files should use UTF-8 encoding.

6.6. Other

For the directory structure and file locations, we follow standard maven conventions (see http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html).

Chapter 2. Coding quality and style

As a general note, the coding style and naming conventions should be adhered to. Some parts are even checked by the checkstyle maven plug-in. However, deviations are always allowed when this enhances code readability.

Formatters are available for the style as described here (see bottom of document). You can be liberal on applying this on new code, but be prudent when applying these to the existing code base. Code style changes make revision changes a lot more difficult and should thus be limited. If there is a need to reformat existing code, then this should be done in a separate commit.

1. Class, method and variable names

Rules

- Use meaningful names. Especially class and method names should explain their purpose.
- For class, method and (non-static) variable names, use camelCase to separate the words, not underscores. For abbreviations, capitalize he first letter, lower case for the others.
- Class names start with a capital, for example "CommandDispatcher".
- Method and (non-static) variable names start lower case, for example "getEmptyCommandResponse".
- All static variables should have capitalized names with words separated by underscores.
- Package names are all lower case and should be singular.
- Use get/set/isXxx.
- Abbreviations and acronyms should not be uppercase when used as name (for example, use "exportHtml()").
- All names should be written in English.
- The terms get/set must be used where an attribute is accessed directly.
- "is" prefix should be used for boolean variables and methods. In some cases, when this is more readable, "has", "can" or "should" can also be used as prefix.
- Complement names must be used for complement entities. These include get/set, add/remove, create/destroy, start/stop, insert/delete, increment/decrement, old/new, begin/end, first/last, up/down, min/max, next/previous, old/new, open/close, show/hide, suspend/resume, etc.
- Exception classes should be suffixed with Exception.

Recommendations

- Usually class names are nouns and method names are verbs.
- Generic variables should have the same name as their type.
- Variables with a large scope should have long names, variables with a small scope can have short names. Scratch variables used for temporary storage or indices are best kept short. A programmer reading such variables should be able to assume that its value is not used outside a few lines of code. Common scratch variables for integers are i, j, k, m, n and for characters c and d.
- The name of the object is implicit, and should be avoided in a method name. For example, use "line.getLength()" instead of "line.getLineLength()". The latter might seem natural in the class declaration, but proves superfluous in use, as shown in the example.

- The term compute can be used in methods where something is computed.
- The term find can be used in methods where something is looked up.
- The term initialize can be used where an object or a concept is established.
- Plural form should be used on names representing a collection of objects.
- Negated boolean variable names must be avoided.
- Default interface implementations can be prefixed by Default. However, if it is not expected that there will even be another implementation, it can be a lot more natural to suffix with "Impl" instead.
- Singleton classes should return their sole instance through method getInstance, should have a private constructor and be declared final.
- Functions (methods returning an object) should be named after what they return and procedures (void methods) after what they do.
- Data transfer objects sometimes exist in two flavors, one which contains the Geomajas geometry dto's and one which contains JTS geometry objects. In that case, the variant with the geometry dto's should use the natural name, and the variant with JTS geometry objects should have a class name which has the "JG" suffix (JG stands for Jts Geometry).

1.1. Comment

Each file should have the correct copyright notice at the start of the file.

```
/*
 * This is part of Geomajas, a GIS framework, http://www.geomajas.org/.
 *
 * Copyright 2008-2012 Geosparc nv, http://www.geosparc.com/, Belgium.
 *
 * The program is available in open source according to the GNU Affero
 * General Public License. All contributions in this program are covered
 * by the Geomajas Contributors License Agreement. For full licensing
 * details, see LICENSE.txt in the project root.
 */
```

Note that the end year (shown here is 2010) should always be the current year. All headers will be updated at the beginning of each year.

- The copyright message should be at the top of the file. However, for js files, it is allowed to have the "dojo.provide" line above the copyright as this helps for debugging.
- Each class and interface should have class comments indicating the purpose of the class.
- Public methods should be commented if the meaning is not entirely clear from method and parameter names (is it ever?). When the method overrides or implements a method, then repeating the javadoc is not needed.
- Comments in the code are recommended when they explain a block of code or when they explain why things are done in a certain way. Repeating the code in human readable wording is wasteful.
- Use "@todo" comments to indicate shortcuts or hacks which should be fixed. Better still is just to do it right and not have the shortcut.
- All comments should be written in English.
- Comments should be indented relative to their position in the code.
- Javadoc comments should be active, not descriptive (for exampe on method "getXxx()" the comment could be "Get xxx").

- All classes and interfaces need javadoc class comments.
- All classes and interfaces in the geomajas-api module need full javadoc comments on all methods.
- All classes, interfaces and methods which have a "`@Api`" annotation needs a "`@since`" javadoc comment to indicate the version in which the class or method was added. This is also the case for methods which are added in classes with "`@Api(allMethods = true)`" annotation.

1.2. Claim your code

Be proud of your code and take responsibility of your changes. When making any kind of significant changes (not for reformatting, fixing typing errors or renaming), add your full name at the bottom of the authors list in the class comments.

1.3. Code layout

See the example below

```
/*
 * This is part of Geomajas, a GIS framework, http://www.geomajas.org/.
 *
 * Copyright 2008-2012 Geosparc nv, http://www.geosparc.com/, Belgium.
 *
 * The program is available in open source according to the GNU Affero
 * General Public License. All contributions in this program are covered
 * by the Geomajas Contributors License Agreement. For full licensing
 * details, see LICENSE.txt in the project root.
 */

package org.geomajas.bladibla;

/**
 * Short description of the purpose of this class.
 *
 * @author Author's name
 * @author Another Author's name
 */
@Annotation(param1 = "value1", param2 = "value2")
public class Foo implements Serializable {

    int[] x = new int[] {1, 3, 5, 6, 7, 87, 1213, 2};

    /**
     * Do something
     *
     * @param x some data
     * @param y more data
     */
    public void foo(int x, int y) throws Exception {
        for (int i = 0; i < x; i++) {
            y += (y ^ 0x123) << 2;
        }
        do {
            try {
                if (0 < x && x < 10) {
                    while (x != y) {
                        x = f(x * 3 + 5);
                    }
                } else {
                    synchronized (this) {
                        switch (e.getCode()) {
                            //...
                        }
                    }
                }
            }
            catch (MyException e) {}
            finally {
                int[] arr = (int[]) g(y);
                x = y >= 0 ? arr[y] : -1;
            }
        }
        while (true);
    }
}
```

- The code is written with the right margin at 120 characters and lines should not be longer than that if possible.
- Tabs should be used for all indents. We assume a tab is four spaces for determining line length.
- When lines are split because they are too long, a double indentation should be used.
- Opening braces on the same line as the declaration/for/if..., so not aligned with the closing brace.
- No spaces inside brackets.
- Spaces around operators.
- No wildcards allowed on import statements.
- Always a space before braces.
- Always use braces (and thus multiple lines) for if, while, do-while.
- Array specifiers must be attached to the type not the variable.
- Class variables should never be declared public.
- Logical units within a block should be separated by one blank line.

We have both an eclipse [http://files.geomajas.org/geomajas_formatter_eclipse.xml] and IntelliJ IDEA [http://files.geomajas.org/geomajas_formatter_intellij.xml] formatter which can be used. However, be careful not to change the entire formatting of a class.

Chapter 3. Spring usage in Geomajas

1. Spring dependency injection

To assure the spring dependency injection is used, you should obtain beans through either injection (possibly autowiring) or the application context. When you directly instantiate classes which require spring dependency injection, you are likely to encounter NullPointerException or other problems.

```
@Component
public class MyClass {

    @Autowired
    private ApplicationContainer applicationContainer;

    public void myMethod() {
        Command command = applicationContext.getBean("controller.general.LogComm
        .....
```

We recommend using the annotations when possible.

You cannot assume that (auto) wired services are initialized while the application context is being built. If you need to do some initialization of the bean state, this should be removed from the setters which are called while building the context, and moved to a post construct method.

```
@PostConstruct
private postConstruct() {
    // dome some stuff here
}
```

1.1. Bean naming convention

Bean names match the (fully qualified name of the) interface they implement if there is only one implementation to be used. When this is not the case, the bean name is the (fully qualified) classname. When the bean name starts with "org.geomajas.", this is cut off. Interfaces which are expected to have several alternate implementations should be annotated with the "@ExpectAlternatives" interface.

There is a "GeomajasBeanNameGenerator" class which tries to automatically determine the bean names, assuring that you don't need to mention this explicitly in the "@Component" annotation. If the first interface which is implemented by the class does not have the "@ExpectAlternatives" annotation, then the fully qualified name of the first interface is used as bean name. For all other beans, and for beans which are in a "command" package and don't have a class name starting with "Default" the fully qualified class name is used. In all cases the bean name has the "org.geomajas." prefix removed is present (using the "GeomajasBeanNameGenerator.simplify()" method.

Note that these rules are built to easily replace instantiation based on class names by instantiating based on bean names. For the same class name, you can often replace the code

```
Class.forName(className).newInstance();
```

by

```
applicationContext.getBean(GeomajasBeanNameGenerator.simplify(className));
```

1.2. Initialising the application context

For servlets, you should configure the Spring context loader listener in the web.xml file. You should also add the request context listener to assure session scoped beans work and can access the request. This is shown in the code below.

Example 3.1. web.xml configuration to initialise application context

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//E
<web-app>
    <display-name>Geomajas application</display-name>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            classpath:org/geomajas/spring/geomajasContext.xml
            WEB-INF/applicationContext.xml
        </param-value>
    </context-param>

    <listener>
         <listener-class>org.springframework.web.context.ContextLoaderListener<
    </listener>
    <listener>
         <listener-class>org.springframework.web.context.request.RequestContext
    </listener>
    .....
```

root context for geomajas
additional context for your application
assures the application context is available
assures the request can be accessed

The "contextConfigLocation" context parameter is a space separated list of Spring context files. The root Geomajas context - which is contained in the Geomajas backend - should always be put on top of this list. The root context will load all the predefined back-end services and automatically import plug-in context files of all the available plug-ins on the class path. Below that you may specify additional application context definition files which are needed for your application. You can include several files by separating them using whitespace. Each location can include the protocol/location used to find the file. Ant-style wild cards can be used. If no protocol is specified, the web application context root is searched. The following are examples of allowed patterns:

```
WEB-INF/applicationContext.xml
WEB-INF/layer*.xml
file:C:/some/path/*-context.xml
classpath:com/mycompany/**/applicationContext.xml
classpath*:conf/appContext.xml
```

The classpath* pattern is specific in that it will combine all the resources that match this exact pattern in the class path, not just the first one.

The web application context can conveniently be retrieved from the servlet configuration in other servlets:

```
public void init(ServletConfig config) throws ServletException {
    return WebApplicationContextUtils.getWebApplicationContext(servletContext);
    .....
```

Chapter 4. Face or plug-in

Geomajas is an extensible frameworks which can be extended by including additional plug-ins on the class path when the application is started.

Some of the possible extensions include

- adding security services.
- providing specific rendering pipeline which modify the default rendering.
- additional services which may be used (also by by other plug-ins), for example printing support.
- a different face (in principle a face is just another plug-in, the term "face" is used when the plug-in produces data or makes data available to the outside world).
- access to a kind of data store (these are referred to as "layer" plug-ins, they consume data).

1. Plug-in structure

Some conventions are in use to make plug-ins easily accessible and auto-register, and to make plug-ins good citizens of the Geomajas project.

1.1. Plug-in application context

Each plug-in can have a configuration file in `META-INF/geomajasContext`.xml which is automatically included in the application context (after the main `geomajasContext` which comes from the `impl` module, but before all files which are explicitly added (through `web.xml`)).

This context file should at least declare the plug-in, the plug-ins and dependent version it depends on, and the copyright and/or license information for all other dependencies. It also has to indicate the API version which is used. This is also version which is used for the back-end (which includes the API) which is used in the pom. Assuming this compiles and that you only used

The dependencies are used to check compatibility of the plug-in with the back-end and required plug-ins. If you only access them using the API, this should assure that everything stays compatible.

Example 4.1. Plug-in declaration in geomajasContext.xml

```
<bean class="org.geomajas.global.PluginInfo">
    <property name="version">
        <bean class="org.geomajas.global.PluginVersionInfo">
            <property name="name" value="Plug-in name" />
            <property name="version" value="${project.version}" />
        </bean>
    </property>
    <property name="backendVersion" value="1.7.1" />
    <property name="dependencies">
        <list>
            <bean class="org.geomajas.global.PluginVersionInfo">
                <property name="name" value="Static security" />
                <property name="version" value="1.7.1" />
            </bean>
        </list>
    </property>
    <property name="copyrightInfo">
        <list>
            <bean class="org.geomajas.global.CopyrightInfo">
                <property name="key" value="Geomajas"/>
                <property name="copyright" value="(c) 2008-2011 Geosparc nv"/>
                <property name="licenseName" value="GNU Affero General Public L
                <property name="licenseUrl" value="http://www.gnu.org/licenses/
            </bean>
            <bean class="org.geomajas.global.CopyrightInfo">
                <property name="key" value="Apache commons"/>
                <property name="copyright" value=""/>
                <property name="licenseName" value="Apache License, Version 2.0
                <property name="licenseUrl" value="http://www.apache.org/licens
            </bean>
        </list>
    </property>
</bean>
```

You can add any other configuration which is necessary in this file, for example configure pipelines, register services.

Note that when adding dependencies, you should run dependency:tree (or similar) to check for sub-dependencies and assure the copyrightInfo list remains complete with copyright and license details for the dependent libraries.

1.2. Plug-in web context

Each plug-in can have a configuration file in `META-INF/geomajasWebContext.xml` which is automatically included in the web context for the dispatcher servlet. This is used to allow plug-ins to define additional web endpoints without the need to define servlet entries in web.xml.

The DispatcherServlet allows use of Spring MVC for defining your controllers and views. Any definitions which are specific to the web tier, should be put in the web context file. The services which are defined in the application context can also be used.

A typical context will define the package to scan (note that if the package which contains the controllers was already scanned in geomajasContext.xml, you will still need to redeclare the scanning to allow controllers to be picked up). The example context as used for the ResourceController looks like this:

Example 4.2. geomajasWebContext.xml for ResourceController

```
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/sche
http://www.springframework.org/schema/context http://www.springframework.org/sc
http://www.springframework.org/schema/util http://www.springframework.org/schem

    <context:component-scan base-package="org.geomajas.servlet"/>

</beans>
```

1.3. Plug-in pom

The pom needs to be complete to allow proper release of the plug-in. This is basically done by including the geomajas-parent as parent pom (alternatively, you can use geomajas-al-parent for Apache licensed modules.

The following sections need to be included:

- name
- description
- repositories
- pluginRepositories

This is for the main pom for the plug-in. All other modules should include this main (-all) pom as parent (except documentation which requires geomajas-doc-parent).

Version management for all dependencies and plug-ins should be done in the plug-in parent (example modules may be exceptions to this though it is strongly discouraged).

1.3.1. When not using the Geomajas parent

When not using the Geomajas parent, you should consider the following:

The following sections need to be filled in:

- description
- scm
- organization
- mailinglists
- licenses
- issueManagement
- ciManagement
- developers

- repositories
- pluginRepositories

The build should also include the following settings

- properties should contain "<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>".
- the following compiler build plug-in declaration should be used:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
        <encoding>utf8</encoding>
        <source>1.5</source>
        <target>1.5</target>
    </configuration>
</plugin>
```

- The checkstyle plug-in should be activated, using the latest Geomajas style.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>2.5-DF</version>
    <configuration>
        <configLocation>config/geomajas-checkstyle.xml</configLocation>
    </configuration>
    <executions>
        <execution>
            <phase>verify</phase>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
    <dependencies>
        <dependency>
            <groupId>org.geomajas</groupId>
            <artifactId>geomajas-checkstyle</artifactId>
            <version>1.0.6</version>
        </dependency>
    </dependencies>
</plugin>
```

- A source jar should be produced.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-source-plugin</artifactId>
    <version>2.1.2</version>
    <executions>
        <execution>
            <goals>
                <goal>jar</goal>
            </goals>
            <configuration>
                <includePom>true</includePom>
            </configuration>
        </execution>
    </executions>
</plugin>
```

- The jar should include indices.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <configuration>
        <archive>
            <manifest>
                <addDefaultImplementationEntries>true</addDefaultImplementatio
            </manifest>
            <manifestEntries>
                <geomajas-version>${project.version}</geomajas-version>
                <license>AGPLv3</license>
                <more-info>http://www.geomajas.org/ and http://www.geosparc.co
            </manifestEntries>
            <compress>true</compress>
            <index>true</index>
        </archive>
    </configuration>
</plugin>
```

Many of these requirements can be met by inheriting from the `geomajas-parent` project.

1.4. Plug-in modules

All plug-ins consist of at least three modules, possibly more.

There is a -all module which is the main module for the plug-in. This one is used for compiling, releasing etc. It should be possible to checkout this module on an empty machine and compile the plug-in (the other modules need not be individually buildable on an empty machine).

One module contains the documentation for the plug-in in docbook format. A template module is generated when you use the `geomajas-plugin-archetype`.

The actual work should be done in one or more modules. You need more than one module when there is face specific code in the plug-in.

2. Plug-in creation

To add a plug-in to the Geomajas project, you should write a proposal which is sent to the Geomajas developers mailing list (majas-dev). It will be discussed and once some kind of consensus seems to be reached, you can initiate a vote to allow creation of the plug-in. The vote should contain the following details

- plug-in name
- plug-in lead
- general description
- technical description

If the persons developing the plug-in don't have commit rights yet, they can get a directory in the sandbox (a part of our version control system) where they can prove their skills until they get full commit rights.

When the vote is accepted and commit rights are in place, the plug-in can be moved to trunk and a JIRA module and continuous integration can be set up. The module should also be added to the aggregate.sh file (which assures all documentation can be found in one place), and it should be added in the geomajas-dep pom (until the first release, it should be commented in that file).

To start the actual coding, we have provided a plug-in archetype which can be used using the following command line (to use the latest release):

Example 4.3. Create project using GWT Maven archetype

```
mvn archetype:generate -DarchetypeCatalog=http://files.geomajas.org/archetype-c
```

Alternatively, you can use the very latest (snapshot) archetype using the following command.

Example 4.4. Create project using GWT Maven archetype

```
mvn archetype:generate -DarchetypeCatalog=http://files.geomajas.org/archetype-l
```

You first have to select the archetype you want to build (geomajas-plugin-archetype). Then it will ask you the "groupId", "artifactId", version and base package. Once you confirmed the settings, the project will be created in a sub-directory with a name equalling the "artifactId" you choose.

3. Plug-in state

A Geomajas plug-in has a "state" which indicates the maturity.

- *incubating*: work-in-progress plug-in which has not reached graduation criteria yet.
- *graduated*: the plug-in is considered stable, development is active and there is sufficient documentation to be usable and testing to prove it works.
- *retired*: t he plug-in is no longer maintained. It can be deprecated or development just stopped for some reason. Both graduated and incubation plug-ins can become retired, so this does not give an indication of quality.

All plug-ins start at in the incubating state.

3.1. Plug-in graduation

The process for a plug-in to move state from incubation to graduated, is called graduation. In order for a plug-in to graduate, several criteria need to be met.

The following is a list of plug-in graduation criteria:

- A plug-in requires a maintainer. This is the contact-person for the plug-in. He should watch the mailing lists and be available for user questions.
- All code should oblige to the programming rules as laid out in the Geomajas contributor guide (code style, javadoc, check-style, author tags,...).
- A check must be made to assure all dependencies of the plug-in have their licenses respected. Examples of issues to consider are compatibility of the license (with the AGPL license for the module) and possible copyright/license display requirements. All the relevant information needs to be supplied in the META-INF/geomajasContext.xml file for the plug-in.
- If the plug-in is a face, the copyright information for all plug-ins needs to be included in the user interface (for example in an "about" box).

- There must be enough documentation for users to easily start using the plug-in without having to ask the basic questions and the documentation needs to be in the expected location and format (to allow inclusion in project documentation).
- There must be enough tests available to prove code stability.

Graduation is an all-or-nothing process. A plug-in either meets all criteria, or it does not. The plug-in maintainer can propose to graduate a plug-in on the majas-dev mailing list. When there is community agreement on the proposal, he or she can initiate a PSC vote. A request for graduation can only be vetoed by including the steps which need to be taken to graduate. Once these steps are taken, the plug-in maintainer can again propose to graduate.

3.2. Plug-in retirement

Plug-in retirement is also handled by a PSC vote. This will typically happen when a plug-in is deprecated (focus moves to a different plug-in which supersedes the retired one), or when a plug-in maintainer wants to quit without having someone to follow up. However, anyone can propose to retire a module. This will normally be denied if the plug-in maintainer is still actively maintaining the module.

Both incubation and graduated plug-in can become retired. Reactivation of a retired plug-in, is of course possible when a new maintainer can be found. In this case the plug-in becomes an incubation plug-in again (and the maintainer must have signed a CLA).

Chapter 5. JIRA conventions

1. Basic issue tracker rules

1.1. One problem one issue

When you report a problem, please submit one issue per problem. There are various reasons for this, amongst them:

- The more crowded an issue is, the more likely is it that some problems may get lost over time.
- Different problems are likely to be handled by different people. The more problems you put into the issue, the more difficult is this issue to handle for all involved parties.

In particular, if you're going to write sentences like "Besides this, I noticed that..." or "There are several problems with....", then please seriously ask yourself whether you should submit multiple issues instead of a single one.

If you don't follow this rule, be prepared for people asking you to split up your issue.

1.2. Provide a meaningful summary

Providing a meaningful summary helps the committers to easily recognize an issue in a list of dozens of others. Since duplicate issues are draining a lot of work from committers, you should always check if the issue you wish to report hasn't already been reported. Of course this works best if the summaries of the existing issues are as descriptive as possible.

1.3. Provide a clear description

You, as the submitter of a problem, know exactly what you were doing when you were hit by the problem. However, most other people probably don't. For instance, they may have a completely different workflow for doing the same things you are doing.

In order to prevent committers to have to ask back how exactly an issue can be reproduced, it is the task of the issue's submitter to be as clear on this as possible - preferably by given a step-by-step description.

2. Filling out the JIRA form

In order to create a new issue, you need to log in to the JIRA issue tracker [http://jira.geomajas.org/].

When creating a new issue, the first thing you will be asked, is to select the project and issue type:

- Project: the project you wish to report an issue for (please try to use the correct module, only use the Geomajas when it doesn't fit a specific module - if it fits multiple modules, please add to all).
- Issue Type: the type of issue you want to report. Is it a bug, task or a question? Please be correct in this.

Then a new form appears with new fields to fill in. The summary and description have been discussed earlier. As for the other fields:

- Priority: how urgent is the issue? This value can always be changed by the Geomajas committers if they feel that the priority does not match the issue's impact.
- Due date: not used.

- Components: What component do you think the issue relates to? Not necessary to fill this in.
- Affects version: In what version of Geomajas did you encounter the issue?
- Assignee: Assign the issue to someone you believe is best suited to fix the issue.
- CLA: Whether you agree to the CLA. This needs to be ticked when you include a patch with the issue and haven't sent a signed CLA to the project.
- The rest is not used.

Chapter 6. Setting up your development environment

1. Prerequisites

1.1. Maven

Geomajas is uses the Apache Maven project management tool for its build and documentation process. Maven can be downloaded from the Apache project site: http://maven.apache.org [http://maven.apache.org] Installing Maven is quite simple: just unzip the distribution file in the directory of your choice and make some environment changes so you can access the executable. More information for your specific OS can be found at the bottom of http://maven.apache.org/download.html [http://maven.apache.org/download.html]

1.2. Subversion

Geomajas uses subversion as its version control system. Accessing subversion requires you to at least install a compatible client. There are numerous client solutions available, some as standalone clients and some as IDE plug-ins:

- Tortoise SVN: an excellent SVN client for Windows (http://tortoisesvn.tigris.org/)
- Subversive: Eclipse plug-in, can be found on the following Eclipse update site (http://download.eclipse.org/releases/helios [http://download.eclipse.org/releases/galileo] > Collaboration Tools)
- Subclipse: Eclipse plug-in, can be found on the following Eclipse update site (http://subclipse.tigris.org/update_1.6.x [http://subclipse.tigris.org/update_1.6.x])
- IDEA SVN plug-in (part of the default IDEA installation)

The Geomajas repository can be found at https://svn.geomajas.org/majas. The standard SVN repository layout is followed: trunk, tags and branches. For the latest and greatest code (including GWT face) you should check out the trunk:https://svn.geomajas.org/majas/trunk.

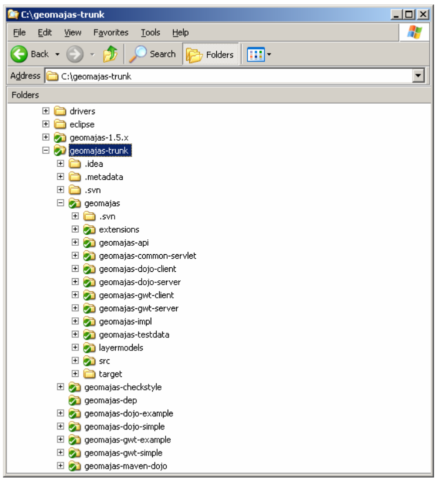
1.3. GWT

The GWT (Google Web Toolkit) software development kit (SDK) should be downloaded from the Google site: http://code.google.com/webtoolkit/download.html [http://code.google.com/webtoolkit/download.html]. After downloading you should unzip it in a directory of choice.

1.4. Build procedure

Start by recursively checking out the trunk directory to a new local folder with a name of your choice (e.g. geomajas-trunk). You will see that the source code layout follows the recommended hierarchical layout structure for multi module maven projects:

Figure 6.1. Hierarchical project layout



Build the code by running the install command on the pom in the top directory:

`geomajas-trunk>` **mvn install**

The install procedure will build all code, run all unit tests and install the artifacts in the repository. Integration tests based on Selenium will also be run.

2. Eclipse

The Eclipse setup is described in the "Getting Started" documentation (see documentation root). For framework development, take the setup described in "Running/debugging with the Google Plug-in for Eclipse (embedded Jetty option)". The embedded Jetty setup is needed to avoid problems with references to multiple versions of Geomajas artifacts like the backend. When using the direct GWT setup, these artifacts are not always correctly mediated.

2.1. Running/debugging with M2Eclipse (quickstart)

The following steps are required (see "Getting Started" for more detail)

- Install the GWT plugin
- Install the m2eclipse plugin
- Import the project as a maven project
- Check the "Use GWT" option in project properties
- Set "src/main/webapp" as webapp directory in the War tab
- Run/debug the embedded Jetty server (normal Java Application with JettyRunner as main class)
- Run/debug the GWT plugin with option (on external server)

2.2. Running/debugging with the maven Eclipse plugin

Eclipse project configurations can be alternatively be generated using the maven Eclipse plug-in. Running eclipse:eclipse fails on trunk because of an issue with maven filtering: http://jira.codehaus.org/browse/MECLIPSE-576. This will be fixed in the upcoming 2.9 release of the plugin, in the meantime you can back up to 2.6:

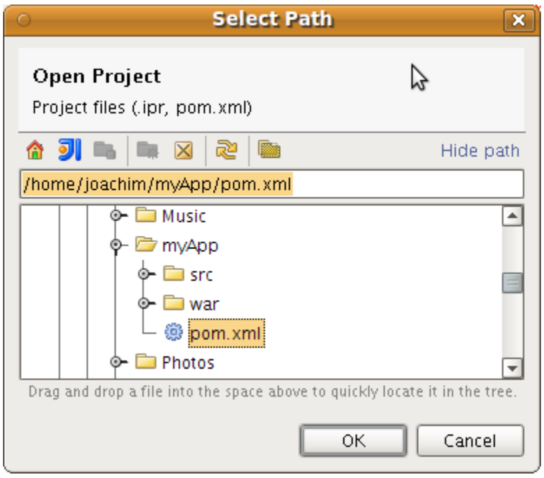
```
mvn -P noshrink eclipse:clean org.apache.maven.plugins:maven-eclipse-plugin:2.6
```

After the command has completed, Eclipse project definitions will have been generated for all subprojects (except the pom projects). These projects can now be imported into Eclipse.

3. IDEA

The setup in IntelliJ IDEA is quite straightforward and does not require running a separate maven command. Make sure you use the maven import wizard to open your project, it can be activated from the File menu "Open project" and selectthe root pom.xml file.

Figure 6.2. Open Geomajas project (replace root directory with your own)



Select Path

Open Project

Project files (.ipr, pom.xml)

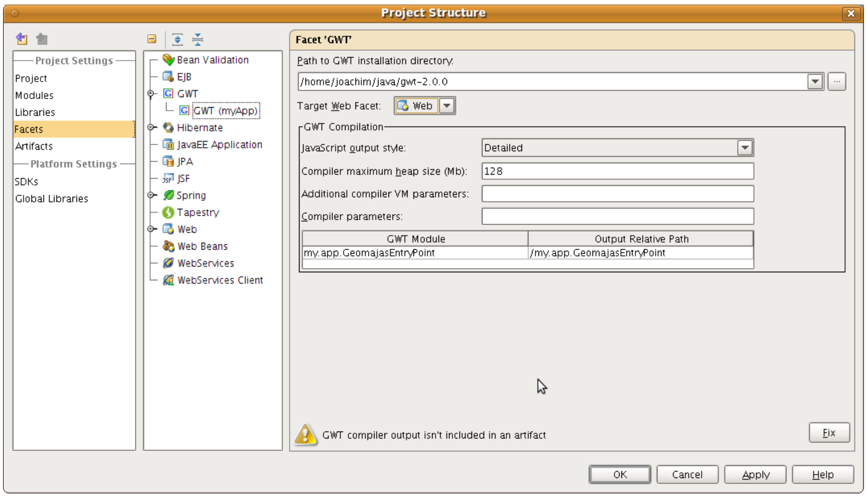
/home/joachim/myApp/pom.xml

Drag and drop a file into the space above to quickly locate it in the tree.

Developing with the GWT face will require you to install the latest version of IntelliJ IDEA (9.0) as this is the only version that supports GWT 2.0. The IDE will recognize the GWT projects and assign the correct facet but as always you will have to make your own run configuration (which is fortunately trivial).

Depending on the actual IDEA version, some additional settings have to be done in the project structure dialog. Apart from specifying the GWT installation directory, there is a specific project setting which has to be done manually, which is setting the target Web facet to "Web". The project structure for the simple GWT project should look as follows:

Figure 6.3. Project structure for simple GWT project



Project Structure

Facet 'GWT'

Path to GWT installation directory: /home/joachim/java/gwt-2.0.0

Target Web Facet: Web

GWT Compilation

JavaScript output style: Detailed

Compiler maximum heap size (Mb): 128

Additional compiler VM parameters:

Compiler parameters:

GWT Module	Output Relative Path
my.app.GeomajasEntryPoint	/my.app.GeomajasEntryPoint

GWT compiler output isn't included in an artifact

After this, you should be able to run and debug the project. Note that this setting is needed for each of the GWT modules you want to be able to run.

4. Maven

If you are working with another IDE or not using an IDE, it is always possible to run the example projects directly from maven. For the Dojo face (geomajas-dojo-simple and geomajas-dojo-example-war) the maven command is as follows:

```
geomajas-dojo-simple> mvn jetty:run
```

This command will start up the Jetty servlet engine, after which you can connect to the process for debugging.

In a GWT project, you should run the following goal:

```
geomajas-gwt-simple> mvn gwt:debug
```

This will start up GWT development mode, debugging should also be possible here.

Chapter 7. How to release Geomajas

The Geomajas project consists of many pieces which each have their own release cycle. The most important parts are the back-end, faces and plug-ins. The example programs, documentation and and build tools also have individual release cycles.

This chapter tries to explain how to release any of these modules. The procedure is similar for all modules, but there are some specific checks to be done which only apply for certain parts.

Make sure you have read through the entire chapter before actually attempting a release, so you know what to do and when to do it.

1. Pre-release checks

1.1. Sonatype Checks

1.1.1. Sonatype general

In order to create a successful release, it is necessary to perform a few checks before actually starting the release procedure.

Before actually starting, know that creating a release means that the artifact in question will be placed in “staging” in the Sonatype nexus repository (which is synced to Maven Central). This in turn requires you to have an account, and a PGP signature. So make sure you have read both:

- Sonatype: Sonatype OSS Maven Repository Usage Guide [https://docs.sonatype.org/display/Repository/Sonatype+OSS+Maven+Repository+Usage+Guide]
- How to create a public PGP signature and distribute it on hkp://pgp.mit.edu. [http://www.sonatype.com/people/2010/01/how-to-generate-pgp-signatures-with-maven/]

1.1.2. Creating PGP key signatures

Authentication when uploading a release candidate on Sonatype is done through the PGP key signatures (as described in the link above). In order to create such a key for yourself do the following:

1. Create a key: `juven@juven-ubuntu:~$ gpg –gen-key`
2. List keys:

   ```
   juven@juven-ubuntu:~$ gpg –list-keys

   /home/juven/.gnupg/pubring.gpg

   ------------------------------

   pub 1024D/C6EED57A 2010-01-13

   uid Juven Xu (Juven Xu works at Sonatype) <juven@sonatype.com>

   sub 2048g/D704745C 2010-01-13
   ```

3. Upload key: `gpg --keyserver hkp://pgp.mit.edu --send-keys C6EED57A`

1.2. Maven Checks

Next you have to make sure that you have a personal SVN profile configured in your maven settings.xml. Concretely, make sure you have a settings.xml file in your maven directory (~/.m2/settings.xml) with the following content:

```
<settings>
    <servers>
        <server>
            <id>svn.geomajas.org</id>
            <username>my SVN username</username>
            <password>my SVN password</password>
        </server>
    </servers>
    <server>
        <id>sonatype-nexus-snapshots</id>
        <username>pieterdg</username>
        <password>my sonatype pasword</password>
    </server>
    <server>
        <id>sonatype-nexus-staging</id>
        <username>pieterdg</username>
        <password>my sonatype pasword</password>
    </server>
</settings>
```

2. General steps

Releasing is done in the following steps, each will be detailed below. Do also check the part specific notes below.

1. Prepare release (on trunk)
2. Put the release in staging
3. Finish the staging procedure
4. Start a vote for the release
5. Count votes
6. Finish the release
7. Announce

2.1. Prepare release

The pre-release checks above only need to be done the first time you wish to release a Geomajas artifact. Here we describe the checks that need to be done every time a single artifact is to be released.

These checks are the following:

- Make sure that there are now references to snapshot repositories in the pom.xml files. These must be commented out.
- Make sure there are no snapshot dependencies in the pom.xml. This can often be the case for the showcase application or the maven archetypes.
- Search for occurrences to the version number (with and without -SNAPSHOT" in the part. Fix when needed (not in the pom, this will be changed by the release procedure itself).
- Warn the Geomajas community that you are about to start a release, so they do not commit any changes. Changes in other plug-ins than the one you are releasing should not affect the release.

- Do a "mvn dependency:tree" and check that all dependencies (excluding the back-end/plug-in dependencies) are included with the license and/or copyright message.

Commit any changes you have had to make.

2.2. Put the release in staging

The basic procedure is quite easy: we let the maven release plug-in do all the work for us. Essentially, it is a two-step procedure: first prepare the release, then perform the release.

Make sure you start with a fresh checkout from trunk before actually attempting to release anything. Checkout command:

```
svn checkout https://svn.geomajas.org/majas/trunk <destination folder>
```

It might be that after the checkout you have to make sure that you have write access to all files and folders within the newly created <destination folder>.

2.2.1. Maven release-prepare

Go to the top folder of the plug-in you want to release and execute the following maven command:

```
mvn -DdevelopmentVersion=1.8.0-SNAPSHOT release:prepare
```

The version is not strictly required, but makes it easier. It points to the NEXT snapshot (after the release).

Should something go wrong, calling the release:prepare again, will take off where the previous call left off. In order to reset everything (mainly the committed changes), you can execute the following command:

```
mvn release:rollback
```

The release:prepare will execute the following steps (taken from the maven release plug-in website):

1. Check that there are no SNAPSHOT dependencies
2. Change the version in the POMs from x-SNAPSHOT to a new version (you will be prompted for the versions to use)
3. Transform the SCM information in the POM to include the final destination of the tag
4. Run the project tests against the modified POMs to confirm everything is in working order
5. Commit the modified POMs
6. Tag the code in the SCM with a version name (this will be prompted for)
7. Bump the version in the POMs to a new value y-SNAPSHOT (these values will also be prompted for)
8. Commit the modified POMs

2.2.2. Maven release-perform

The next step is to actually let maven perform the release. This can be done with the following command:

```
mvn release:perform
```

Note that you will be asked to give your PGP passphrase. Now go to the sonatype nexus (http://oss.sonatype.org/) and log in with your personal account. There, in the staging repositories, you will find the release in staging.

2.3. Finish the staging procedure

Now that the release has been placing in the staging area on sonatype, we can work towards finishing the staging procedure. In order to do so, the following steps need to be undertaken:

- Update trunk. Check if the versions have rolled to the next snapshot version.

- *(Only for final releases, not for milstones!)* For each of the modules in the part, the src/main/resources/api.txt file needs to be updated. The header at the top should remain, with the version updated. The rest of the file needs to be replaced by the contents of target/api.txt. This api.txt file must reflect the API status of the release so that it can be used for API regression checking.

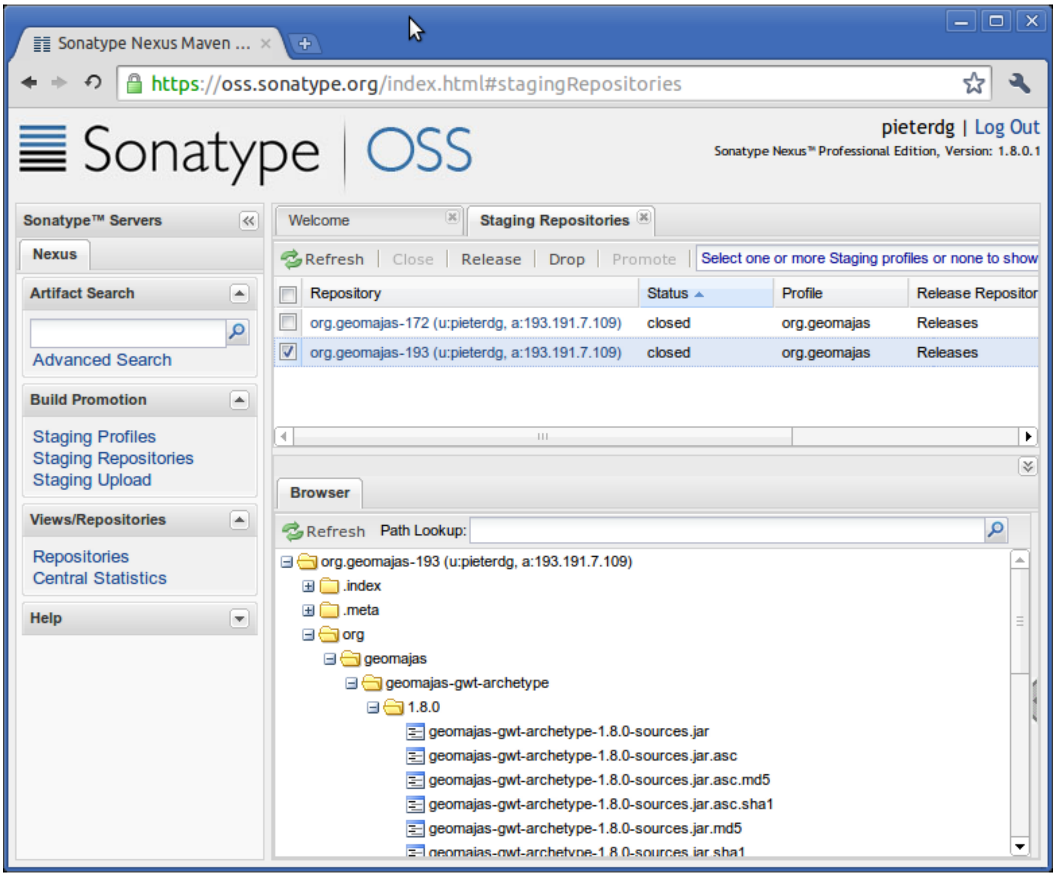
- The aggregate.sh need to be updated for the new snapshot versions respectively.

- In JIRA, create the next version for the project if that did not yet exist. Close any issues which are fixed in this release but only marked as resolved. Mark the project as released, moving issues which are not fixed to the next version.

- Login to http://oss.sonatype.org/. Check if the correct artifacts are in the staging repository, and check their contents. From sonatype, you can download the artifacts, and see if everything is fine (correct version, javadoc present, ...). If all is well, close it.

  The artifacts are now ready for testing. When staging several parts, it is recommended to close each separately. This allows more fine-grained promotion and/or dropping of artifacts.

- Search for all occurrences to the previous snapshot version in trunk, replace by the released version (yes this will cause a dependency on the staged artifacts).



https://oss.sonatype.org/index.html#stagingRepositories

	Repository	Status	Profile	Release Repositor
☐	org.geomajas-172 (u:pieterdg, a:193.191.7.109)	closed	org.geomajas	Releases
☑	org.geomajas-193 (u:pieterdg, a:193.191.7.109)	closed	org.geomajas	Releases

- org.geomajas-193 (u:pieterdg, a:193.191.7.109)
  - .index
  - .meta
  - org
    - geomajas
      - geomajas-gwt-archetype
        - 1.8.0
          - geomajas-gwt-archetype-1.8.0-sources.jar
          - geomajas-gwt-archetype-1.8.0-sources.jar.asc
          - geomajas-gwt-archetype-1.8.0-sources.jar.asc.md5
          - geomajas-gwt-archetype-1.8.0-sources.jar.asc.sha1
          - geomajas-gwt-archetype-1.8.0-sources.jar.md5
          - geomajas-gwt-archetype-1.8.0-sources.jar.sha1

2.4. Start a vote for the release

Start a vote on majas-dev for release of the part. The voting period should be at least two working days and span a weekend. The voting message should read something like "VOTE: release blabla plug-in v1.0.0" (please note capitalization).

The body of the vote needs to include:

- Short introduction to the released part.
- Major improvements of the release.
- Link to release notes.
- Link to migration notes if any.
- An indication of how to test, typically a text like the following will do:

  > The artifacts are available in the staging repository: https://oss.sonatype.org/content/groups/staging
  >
  > To make sure you can use the artifacts you have to add this repository in your local maven settings: ~/.m2/settings.xml:

  ```
  <settigs>
    <profiles>
      <profile>
        <id>mine</id>
        <!-- add staging repository -->
        <repositories>
          <repository>
            <id>oss-staging</id>
            <url>https://oss.sonatype.org/content/groups/staging</url>
          </repository>
        </repositories>
      </profile>
    </profiles>
    <activeProfiles>
      <activeProfile>mine</activeProfile>
    </activeProfiles>
  </settings>
  ```

- Your vote.

2.5. Count votes

When the voting period has ended, count the votes. The vote is successful when there are at lease two +1 binding votes and no -1 binding votes.

Reply to your original vote message on majas-dev, change the title to start with "VOTE SUCCESSFUL" or "VOTE FAILED" (please note capitalization). In the body of the vote, include the full name of all voters and their vote.

2.6. Finish the release

When the vote failed, drop the repository. All the references to the release version in trunk need to be rolled to the new snapshot version. The version number is skipped. The release date in JIRA is actually the staging date.

The geomajas-dep pom and aggregate.sh need to be updated for the newly released versions respectively.

2.7. Announce

Build announcement message, it is recommended this contains the following:

- Short introduction to the released part.
- Major improvements in the release.
- Link to release notes.
- Link to migration notes if any.

Send announcement to majas-dev (plain text).

Publish on general forum.

Announce on freecode.

Create news item on geomajas.org site.

Send mail to jan.pote@geosparc.com to allow sending a press release.

3. Additional notes for releasing the back-end

When releasing the back-end core, you will have problems building the javadocs. Once the "release:perform" failed, execute the following steps.

```
cd target/checkout
mvn install
cd ../..
mvn release:perform
```

At this moment, the back-end should be available in sonatype. Download one of the jars and check the contents to see if all is well. If it is, close the staging repository to place the back-end officially in staging.

When releasing the back-end, you must also be aware that this releases only the modules within the backend folder. The following modules might need to be released separately:

- documentation (user guide, contributor guide, getting started guide)
- geomajas-dep
- GWT client
- GWT archetype
- GWT showcase

3.1. Announce

Create download image for this version (278x61 pixels).

Add release on download page (remember to name the page "release_1.5.0" with correct version number).

Update the download block on the Geomajas site.

Update the javadoc links on the documentation page.

Update Geomajas wikipedia page.

3.2. Documentation

The documentation uses the example applications for extracting code which is included in the manual. This is a circular dependency when it includes the part to be released. It may be useful to do a local build using the next release version locally, to allow the release the work for the documentation part. You can do the actual release of the example application at the end.

4. Additional notes for releasing the GWT face

Before releasing the GWT face, make sure that the “getVersion” method in the Geomajas.java class has been set to the correct version.

5. Additional notes for releasing the GTW archetype

The GWT archetype is usually released after the release of the GWT client. Before actually starting, make sure that the dependency versions are correct in:

```
geomajas-gwt-archetype/src/main/resources/archetype-resources/
pom.xml
```

After the archetype has been released, one must be able to test it while it is in staging. In order to do so the pointer to the correct repository for the archetype must used. This pointer is provided by the file: http://files.geomajas.org/archetype-staging.xml

This file must therefore be updated to make use of the newest version. It should look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<archetype-catalog
  xmlns="http://maven.apache.org/plugins/maven-archetype-plugin/archetype-catal
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/plugins/maven-archetype-plugin/ar
  http://maven.apache.org/xsd/archetype-catalog-1.0.0.xsd">
  <archetypes>
    <archetype>
      <groupId>org.geomajas</groupId>
      <artifactId>geomajas-gwt-archetype</artifactId>
      <version>1.8.0</version>
      <description>Geomajas GWT application archetype</description>
      <repository>https://oss.sonatype.org/content/groups/staging</repository>
    </archetype>
  </archetypes>
</archetype-catalog>
```

The correct command to create a new Geomajas project using the archetype in staging is:

```
mvn archetype:generate -DarchetypeCatalog=http://files.geomajas.org/archetype-s
```

Make sure that this command is also mentioned in the vote mail on the majas-dev mailing list.

5.1. Finish the staging procedure

On gaya, update/var/www/files.geomajas.org/htdocs/archetype-latest.xml. Make sure the version is correct.

5.2. Start a vote for the release

On gaya, update/var/www/files.geomajas.org/htdocs/archetype-staging.xml. Make sure the version is correct.

Include the command to try the archetype in the vote message.

It is recommended to start this vote only when all modules which are referenced are already released. In that case, you don't have to mention the staging repository in the vote message.

5.3. Finish the release

On gaya, update/var/www/files.geomajas.org/htdocs/archetype-catalog.xml. Make sure the version is correct.

6. Additional notes for releasing the GWT showcase

Just like the geomajas-dep and the archetype, the showcase must be built upon the new stable releases of the back-end and GWT client.

6.1. Finish the release

Upload war files to sourceforge download area.

Update the showcase on gaya. Make sure the Google API key is set.

7. Additional notes for releasing Geomajas-dep

Update the aggregate.sh as well as the dependencies in the pom before releasing. Don't forget the the version of geomajas-dep should reflect the <major>.<year>.<week>, where the week number should be even. Example: 1.11.4, means the 4th week in 2011. The major number should be equal to the back-end version referenced in the pom.xml.

8. Possible problems and solutions

When releasing the back-end core, there may be a problem building the javadocs. The solution can be to do a local build of the back-end using the next release version before doing the actual release.

The documentation uses the example applications for extracting code which is included in the manual. Sometimes (it shouldn't) this is a circular dependency when it includes the part to be released. It may be useful to do a local build using the next release version locally, to allow the release the work for the documentation part. You can do the actual release of the example application at the end.

9. Example announcement

title: Geomajas *1.5.0 technology preview/release candidate/stable* released

The Geomajas project is proud to release Geomajas 1.5.0, a technology preview showcasing the progress we are making towards our next stable build.

The major advances in this version include *(indicate major contributors when appropriate)*

- modularization of the system
- introduction of a GWT face

For the full list of changes, see http://jira.geomajas.org/jira/secure/ReleaseNote.jspa?version=10131&styleName=Html&projectId=10000&Create=Create

Documentation for this release can be found at http://files.geomajas.org/maven/1.5.0/geomajas/userguide.html .

Download links can be found at http://geomajas.org/release_1.5.0 .

For the next release we plan to include the following features

- absorb CO2 from the air to reduce global warming
- remove need for system to be powered

Please note that this is an unstable release, all the new features since the previous stable release may still change and we some new bugs may have been introduced.

If you want to help us, join the discussions on the developer list, list bugs in jira and make feature requests in our fora. See http://www.geomajas.org/gis-development .

Geomajas is the extensible open source web mapping framework, enabling integrated GIS solutions for businesses and government.

Appendix A. Geomajas Contributor License Agreement

In order for N.V. Geosparc (hereinafter “Geosparc”), a company under Belgian Law having its registered office at Gaston Crommenlaan 10, box 101, 9050, Gent, Belgium which is registered at the commercial register in Ghent, n° 0808.353.458, to have a clear understanding on the intellectual property rights associated with the Geomajas software library (hereinafter “Geomajas Project”) and to clearly determine the responsibilities and obligations associated with the Contributions (as defined hereinafter), Geosparc must receive a signed Geomajas Contributor License Agreement of the Contributor (as defined hereinafter) indicating that the Contributor agrees with the terms and conditions as defined hereunder. This Geomajas Contributor License Agreement (hereinafter “the Agreement”) intends to protect the Contributor as well as Geosparc.

Contributor hereby accepts and agrees to the following terms and conditions with regard to past, current and future Contributions submitted by Contributor to Geosparc, and has accepted the policy “Geomajas Contributions Policy”

1. Definitions

When used in this Agreement the following words and or expressions shall have the meaning as stated hereunder unless the context expressly requires otherwise:

1. “Contributor” means 1/ any individual and/or legal entity that voluntarily submits (a) Contribution(s) to the Geomajas Project or 2/ any individual legally representing his/her Company.

2. “Contribution” means any original work, including any modification and/or addition to the existing work that is submitted for introduction in, or documentation of, any of the products owned or managed by Geosparc, where such work originates from a Contributor. A Contribution may be submitted in any form of electronic, verbal and/or written communication or documentation, including without limitation, communication on electronic mailing lists, source code control systems and issue tracking systems that are managed by or on behalf of Geosparc for the purpose of discussion and improving the results of the Geomajas Project.

2. Granted Rights - Representations

1. For the benefit of Geosparc, the Contributor hereby:

    a. irrevocably assigns, transfers and conveys to Geosparc all right, title and interest in and to the Contribution(s). Such assignment includes copyrights (to the extent permitted by applicable mandatory law) and all other intellectual property rights other than patents and patent applications (“Patent”), together with all causes of actions accrued in favour for infringement thereof, recognized by any jurisdiction (“Proprietary Rights”). Without limitation of the foregoing, Geosparc shall be entitled to determine in its sole discretion whether or not to use the Contribution(s) and to use, sell, distribute, license, re-produce, re-use, modify, update, edit or otherwise make available the Contribution(s) as it sees fit, in any manner currently known or in the future discovered and for any and all purposes;

    b. grants (to the extent that under applicable mandatory law, Proprietary Rights cannot be assigned, transferred or conveyed) to Geosparc and to the recipients of the software incorporating the Contribution(s) an irrevocable, worldwide, non-exclusive, fully paid-up and royalty-free copyright license to reproduce, modify, prepare derivative works of, (publicly) display, perform, sub license and distribute the Contribution(s);

    c. grants to Geosparc and to recipients of software distributed by Geosparc a worldwide, non-exclusive, fully paid-up, royalty-free, irrevocable (except as stated in this Agreement) Patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Contribution(s), where such license applies only to the Patent claims licensable by Contributor that are necessarily infringed by the Contributor's Contribution(s) alone or by combination of such Contribution(s) with other work of Geosparc. Contributor furthermore agrees to immediately notify Geosparc of any patents that Contributor knows or comes to know are likely infringed by the Contribution(s) and/or are not licensable by the Contributor. If any entity institutes patent litigation against the Contributor or any other entity (including a cross-claim or counterclaim in a lawsuit) alleging that Contributor's Contribution(s) or the Geomajas Project work to which the Contributor has contributed constitutes direct or contributory patent infringement, then any Patent licenses granted under this Agreement for that Contribution or Geomajas Project work shall immediately terminate as of the date such litigation is filed.

2. Upon the assignment of the Proprietary Rights and the grant of the license as set forth in this article 2, Geosparc hereby grants a non-exclusive, worldwide, fully-paid up, royalty-free license to make, use reproduce, distribute, modify and prepare derivative works based on the Contribution(s) of Contributor.

3. Contributor hereby represents and warrants that:

   a. In the case that the Contributor is an individual who works in his/her own name the Contributor guarantees that he/she is legally entitled to assign the Proprietary Rights and to grant the above license.

   b. In the case that the Contributor is an employee the Contributor guarantees that he/she can legally represent the Company and is entitled to assign the Proprietary Rights and to grant the above license.

   c. In the case the Contributor is a Company and the Contributor's employee(s) or consultant(s) have rights to intellectual property the Contributor warrants that its employee(s) has waived such rights;

   d. each Contribution is the original creation of the Contributor. Contributor represents that each submission of a contribution includes complete details of any third-party license or other restrictions of which you are aware and which are associated with any part of the Contribution(s);

   e. no claim or dispute has been threatened or filed in connection with the ownership, use or distribution of the Contribution(s); and

   f. the execution of this Agreement does not constitute a breach under any other agreement to which Contributor and/or its employer is a party, does not require the consent, approval or waiver from or notice to any third party and does not violate any law or regulation.

   Contributor shall immediately inform Geosparc of any facts and/or circumstances of which Contributor becomes aware that would make the representations and warranties inaccurate or untrue in any respect.

   Contributor further agrees that Contributor shall at no time hereafter dispute, contest or aid or assist third party in disputing and/or contesting, either directly or indirectly, the right, title and interest in any and all Contributions of Geosparc as detailed in this Agreement.

4. In case that under applicable mandatory law the Contributor retains the moral rights or other inalienable rights to the Contributions, the Contributor agrees not to exercise such rights without the prior written permission of Geosparc.

5. In order to ensure that Geosparc will be able to acquire, use and protect its Proprietary Rights as detailed in this article 2, Contributor will (i) sign any documents to assist Geosparc in the documentation, perfection and enforcement of its rights, and (ii) provide Geosparc with support and reasonable access to information for applying, securing, protecting, perfecting and enforcing its rights.

3. Warranties

EXCEPT FOR THE EXPRESS WARRANTIES DETAILED IN ARTICLE 2, THE CONTRIBUTION(S) ARE PROVIDED “AS IS” AND NEITHER CONTRIBUTOR NOR THE Geosparc MAKES ANY WARRANTIES OF ANY KIND TO THE OTHER PARTY, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION OF ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

4. Miscellaneous

1. This Agreement shall enter into force upon execution of this document by Contributor. This Agreement may be terminated by a party if the other party commits a breach of this Agreement provided that if the breach is capable of remedy termination shall not occur if the breach shall not have been remedied within 90 days of such other party having been given notice in writing specifying the breach and requiring it to be remedied. The termination of this Agreement shall however remain in full force and effect with respect to any Contribution submitted prior to the termination date of the Agreement.

2. This Agreement contains the entire agreement between the parties and supersedes all prior or contemporaneous agreements or understanding, whether written or oral, relating to its subject matter. If any provision of this Agreement shall be deemed invalid or unenforceable, the validity and enforceability of the remaining provisions of this Agreement shall not be affected and such provision shall be deemed modified only to the extent necessary to make such provision consistent with applicable law.

3. 4.3.The Agreement is governed by the laws of Belgium, without reference to its conflict of law principles.

4. Geosparc shall have the right to assign its rights and obligations hereunder to any successor or assignee of its business or assets to which this Agreement relates, whether by merger, establishment of a legal entity, acquisition, operation of law or otherwise without the prior written consent of the Contributor.

Please execute (2) original copies of the above document and send this to the following recipient:

Address of recipient

Geosparc

Gaston Crommenlaan 10/101

BE-9050 Ghent

BELGIUM

The Contributor:

Name:

Title (if applicable in case of legal entity):

Full name of legal entity and address registered office (if applicable):

Date:

Signature:

The Company:

Name:

Title (if applicable in case of legal entity):

Full name of legal entity and address registered office (if applicable):

Date:

Signature:

N.V. Geosparc – Register N° BE 0808.353.458

Appendix B. Maven repository

The project use the nexus repository manager [http://nexus.sonatype.org/] to store all Geomajas jars and all dependencies.

The following configuration can be used in your maven profile :

```
<repositories>
    <repository>
        <id>Geomajas</id>
        <name>Geomajas repository</name>
        <url>http://maven.geomajas.org/</url>
    </repository>

    <!-- uncomment if you want to use Geomajas snapshots, comment for faster bu
    <repository>
        <id>Geomajas snapshots</id>
        <name>Geomajas repository</name>
        <url>http://maven.geomajas.org/</url>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
</repositories>
```

If you do not need access to the snapshot releases, then it is recommended to remove that repository from your pom (it will make your compilation a little faster).

The Geomajas build has quite a few dependencies which are gathered from several repositories.

Our nexus instance functions as a proxy for the following repositories ;

- maven central: http://repo1.maven.org/maven2/
- java.net repo: http://download.java.net/maven/2/
- jts4gwt: http://jts4gwt.sourceforge.net/maven/repository/
- OSGeo: http://download.osgeo.org/webdav/geotools/
- OpenGeo: http://repo.opengeo.org/
- geotools: http://download.osgeo.org/webdav/geotools/
- smartgwt: http://www.smartclient.com/maven2
- spring milestones: http://repository.springsource.com/maven/bundles/milestone
- spring releases: http://repository.springsource.com/maven/bundles/release
- selenium: http://nexus.openqa.org/content/repositories/releases
- selenium snapshots: http://nexus.openqa.org/content/repositories/snapshots
- hibernate-spatial: http://www.hibernatespatial.org/repository
- alfresco: http://maven.alfresco.com/nexus/content/repositories/public/
- vaadin-addons: http://maven.vaadin.com/vaadin-addons