

אופציונמאנאזער

OptionsManager

v. 1.0.2

User Guide

Table of Contents

1	Table of Contents	i
2	Quick Intro	1
3	Maven Usage	3
4	User Guide	4
5	Project Reports	13
5.1	Project License	14
5.2	Project Team	20
5.3	Issue Tracking	21
5.4	Source Repository	22

1 Quick Intro

Version 1.0.2: Bug fix: now also handles inherited options.

1.1 Short Description

Manages application/tool options. Options are mapped to Java Bean properties using annotations. Help texts can be specified for options also using annotations and a complete help text can be generated by the options manager. Options can be loaded from command line arg String[] array, an XML file, or a java properties file.

Please note that even if this tool can load XML data into a JavaBean it is not an XML to Java tool! Even if it can be used as such it is very limited for that use. It is only intended for mapping user options/configuration from some source into a structure of JavaBeans for which one such source happens to be XML. For an XML - JavaBean mapper I recommend either JAXB (<https://jaxb.dev.java.net/>) or Simple (<http://simple.sf.net/>) or XOB (<http://xob.sf.net/>) which are all intended for such use.

1.1.1 Simple example

The following is a simple options model:

```
@OptionsModel(name="")
public class BreadOptions {

    @Option(description="Specifies the oven temperature",
            required="true")
    private float ovenTemp;

    @Option(name="bakeTime",
            description="The amount of time to bake.")
    private int bakeTimeMinutes = 120;

    @Option(description="Displays help.", flag=true)
    private boolean help = false;

    ...

    // Setters & Getters

    ...
}
```

If you don't like annotations with a long list of attributes the following example is identical in functionality:

```

@OptionsModel(name="")
public class BreadOptions {

    @Option
    @Description("Specifies the oven temperature")
    @Required
    private float ovenTemp;

    @Option
    @Name("bakeTime")
    @Description("The amount of time to bake.")
    @Optional
    private int bakeTimeMinutes = 120;

    @Option
    @Description("Displays help.")
    @Flag
    private boolean help = false;

    ...

    // Setters & Getters

    ...
}

```

This model would be loaded from a command line tool with:

```

public void main(String[] args) {
    CommandLineOptionsManager<BreadOptions> clom =
        new CommandLineOptionsManager(BreadOptions.class);

    // Make our options "--" prefixed.
    BreadOptions breadOptions = clom.loadOptions("--", args);

    if (breadOptions.isHelp()) {
        clom.printHelpText("--", "", System.out);
    }
    else {
        ...
    }
}

```

This allows the user to pass arguments like "--ovenTemp 85 --bakeTime 120".

Please note the `@OptionsModel(name="")` on the `BreadOptions` class. If we didn't specify that annotation the name would be "breadOptions" and the arguments would be "--breadOptions-ovenTemp 85 --breadOptions-bakeTime 120 ...". In the case of command line argument options you don't usually want to use the name of the options model as part of the argument specification.

See the User Guide for more detailed information.

2 Maven Usage

2.1 Maven Usage

If you are using maven, add the following to your pom:

```
<project>
  ...
  <repositories>
    ...
    <repository>
      <id>maven-biltmore-se</id>
      <name>Biltmore maven repository</name>
      <url>http://maven.biltmore.se/mavrepl</url>
    </repository>
  </repositories>
  ...
</project>
```

Then add the following dependency:

```
<dependency>
  <groupId>se.biltmore.tools.optionsmgr</groupId>
  <artifactId>OptionsManager</artifactId>
  <version>1.0</version>
</dependency>
```

Thats it!

3 User Guide

3.1 Introduction

The idea for OptionsManager came (as usual) from a need. When working with another project I needed to have one set of options models and be able to populate them from maven, ant, and a command line tool. Maven and ant being as nice as they are didn't fight me on this. But when it came to command line arguments I had to manually parse the args string array and populate the model. That really annoyed me. I started thinking, what if you could load models automatically from command line arguments also ? The idea of OptionsManager was born.

OptionsManager supports instantiating and populating annotated (decided to require annotations to offer more flexibility) models from different forms of input. There are 4 variants provided with the package (so far): Command line, properties, XML elements only (like maven), XML elements and attributes (like ant). It is quite simple to create new variants for other type of input if needed.

These are very simple to use for handling options/config of both simple and complex structure. Each option can be documented using the annotations and a help text can be automatically generated from that.

3.2 Terms used by this document

3.2.1 Options Model

This refers to all annotated JavaBean models whose top level model Class object is passed to the constructor of each OptionsManager subclass and that will be instantiated and loaded with the option values.

3.2.2 Options path

This refers to a path from the top level options model down to a JavaBean property value. Internally each part of the path is separated by a '.'. This can be translated to something else externally by the OptionsManager subclass (or the user). For example the CommandLineOptionsManager sets it to "-". This allows arguments to be paths like "--thirdparty-license-type" where "Thirdparty", and "License" are options models, and "type" is a JavaBean property in the "License" model.

All exception messages that are result of bad indata when loading includes the options path to the item that failed, separated by the external path separator for the specific OptionsManager.

All key values in the input, no matter what type it is must match in some way the options path to a value. For OptionsManager subclasses that handles input in random order the options path is always a full path. For example CommandLineOptionsManager. For those that takes input in a sequential structured order the options path can be relative. For example the 2 XML OptionsManager variants where each child element can be seen as a relative path to its parent element.

3.2.3 Top level model

This is the options model that is specified as T in the generics declaration and also passed to the OptionsManager subclass constructor. This model can in turn have submodels. To differentiate between submodels and this model the term "top level (options) model" is used.

3.3 Options Models

The options models must be java beans with some supported extensions. Either the private field of the bean property or the public setter method can be annotated. The setter usually offers more flexibility, but in some cases a field is more useful. In general it is a matter of personal taste. It is always the setter that is used to populate a bean property with a value. You can change the name of a property either using `@Option(name="name")` or using `@Name("name")`. The setter must however match the annotated name so this is only useful if you annotate fields.

3.3.1 Supported types

The following bean property types are supported.

- All Java primitives + their object equivalents.
- `java.lang.String`
- `java.util.Date`

When it comes to Date `OptionsManager` needs to know the format of the date string to be able to parse it into a Date object. By default the following formats are supported:

- `YYYY-MM-DD HH:MM`
- `YYYY-MM-DD`
- `HH:MM`

If you need to support other Date formats, use the `addSimpleDateFormat("...")` or the `addDateFormat(DateFormat format)` method on the `OptionsManager` instance before calling `loadOptions(...)`. `OptionsManager` will take each format in order and try until it succeeds or fails all.

- `java.net.URL`
- `java.io.File`
- Any enum

When loading a value for an enum property the value is case insensitive. Enum properties automatically validates input. An `OptionsException` will be thrown if the loaded value does not match one of the enum constants.

3.3.2 Arrays and Maps

Currently arrays and Map types are not supported.

Collections are supported, so use those instead of arrays.

For Map there is a possible workaround. Make a `JavaBean` model that internally sits on a `Map`, and has a key and value setter. When a key is received it should be temporarily saved, and when a value is received the key and value should be added to the map. This will only work with `OptionsManager` subclasses handling sequential input like XML. In this case annotate the key and value setter methods rather than the `Map` field.

3.3.3 Collections

Both value bean properties and submodel bean properties can be of a `'java.util.Collection'` type for multiple entries. In this case the type of the object in the `'Collection'` must be annotated since `OptionsManager` have no way of knowing what to put in there otherwise. (Generic declarations does unfortunately not reflect).

There are 3 ways to provide setters for `'Collection'` type properties:

- 1 A setter that takes the whole 'Collection'. This is what you would expect when following the JavaBean standard.

In the Collection case, if the field name or name annotation is 'picture' then a setter method name of 'setPictures' will still be found.

For this type of setter there are 2 variants for specifying input:

- A parent entry of name 'list' and entries matching the annotated collection content type.
 - A parent entry name that matches the JavaBean property name and setter name and that must end with 's' with entries matching the annotated collection content type. If the content type matches the parent name but without the 's' it will be compatible with maven's way of loading collections.
- 2 A setter that takes just the type of the objects in the collection (that is a setter for a non 'Collection' property) but that adds each set object to the internal 'Collection'. In this case the type annotation is not needed since OptionsManager gets it from the setter. This is still JavaBean legal as far as I know since the JavaBean specification only specifies setters and getters the internal storage is up to the bean, again as far as I know.

One reason for doing this is to make the input structure smaller. If we take the 'Collection' setter described above as an example we would have something like this:

```
<data>
  <pictures>
    <picture .../>
    <picture .../>
    ...
  </pictures>
</data>
```

But using the method described here would give you:

```
<data>
  <picture .../>
  <picture .../>
  ...
</data>
```

- 3 The same as the previous alternative, but instead of setMyProp(...) you have an addMyProp(...) which makes the code clearer than the previous setter. This of course completely breaks the JavaBean standard.

3.3.4 The Annotations

With the exception of @Option all other annotations are optional and most are alternatives for attributes of @option. All @Option attributes are optional (though some are required in certain situations).

- Option(...) - This is required on each field/setter that are supposed to be loaded by OptionsManager.
- OptionsModel(name="...", description="...") - This is used on a options model class and is optional. This can be used on submodels, but is of most use on the top level model.
 - name="..." - Sets the name of the options model. The default will be the name of the property that holds the model in the parent or if this is the top level options model the name of the model class with the first character decapitalized.

You want to set the `name=""` when you don't want the top level options model name to be part of the option path. For example, when using the `CommandLineOptionsManager` you don't want to specify `--option_model_name-option_property_name`. It is much nicer for the user to have to specify `--option_property_name`. Setting the top level options model name to `""` accomplishes that.

The `@Name("name")` annotation can be used as an alternative to this.

- `description="..."` - Supplies a description of the model. This is only used for the top level options model and will be displayed in the generated help text before any options help texts. If the options model is not the top level model this will simply be ignored.

Please note that when you use the `@OptionsModel` on the top level model and only provide a description you will get a default name of `""` which will still override the default non annotated name! This might not be what you intended.

The `@Description("description")` annotation can be used as an alternative to this, and will not affect the name.

3.3.4.1 `@Option(name="...") / @Name("...")`

This specifies the name of the option. If `name` is not set the name of the field is used if it is a field that is annotated. If it is a setter that is annotated the setter name without "set" and decapitalized will be used. Please note that if you annotate a field a different name than the fieldname can be specified for `name="..."`. The setter must however follow the annotated name. Due to that if it is a setter that is annotated the name cannot differ from the setter name, and thus this is not very useful on a setter.

3.3.4.2 `@Option(description="...") / @Description("...")`

This provides a description of the option and are used when generating a help text.

It is possible to influence the formatting of the description on help output. The following are available:

- `\\` or `Description.NL`
Forces a line break.
- `\\ \\` or `Description.NLNL`
Forces 2 consecutive line breaks. This creates one empty line.
- `\\m\\` or `Description.ManualLineBreaks`
Here after no automatic line breaks will be done when line is larger than specified length.
- `\\a\\` or `Description.AutoLineBreaks`
Here after automatic line breaks will be done again when line is larger than the specified length.

No more than 2 consecutive line breaks are allowed. Any more than that will simply be ignored.

Example:

```
@Description("This is my first paragraph." + Description.NLNL +
    "This is my second paragraph." +
    Description.ManualLineBreaks +
    "My very long and unbroken line until I break it.")
```

3.3.4.3 `@Option(required=true) / @Required`

This says that the option is required, and `validateLoadedOptions()` after a `loadOptions(...)` will throw an `OptionsException` if a required option has not been set.

3.3.4.4 @Optional

This is the opposite of @Required. All options are however optional unless specified as required, so this does not supply any functionality, it only has retention SOURCE and are completely ignored. It is only documentative / cosmetic. It has no equivalent attribute in @Option.

3.3.4.5 @Option(type=MyType.class) / @Type(MyType.class)

If you have a JavaBean property whose type is a java.util.Collection then this must be specified to indicate what type is put into the collection.

3.3.4.6 @Option(validate="regular expression") / @Validate("regular expression")

This allows you to constrain values for options. When options are loaded the value will be matched against the regular expression and an OptionsException thrown if it does not match. This can be used for any type of option since all values are provided as strings when loaded and this match is done before the value is converted to the real type.

3.3.4.7 @Option(validValues={"value", ...}) / @ValidValues({"value", ...})

This is an alternative to validate, and simply provides a list of valid values.

3.4 The options managers

3.4.1 CommandLineOptionsManager

This is intended for handling command line options, but it really only takes a String array of "key, value, ..., key, value" where the key is an options path possibly prefixed with custom specified prefix (usually "-" or "--").

Multiple values for a single key is currently not supported, but something like "value,value,value" can be passed as a single value and your options model can parse it into separate values as a workaround.

Example usage:

```
public void main(String[] args) {

    CommandLineOptionsManager<MyOptModel> clom =
        new CommandLineOptionsManager(MyOptModel.class);

    MyOptModel myOptModel =
        clom.loadOptions("--", "-", args);

    if (myOptModel.isHelp()) {
        clom.printHelpText("--", "-", System.out);
    }
    else {
        ...
    }
}
```

3.4.2 PropertiesOptionsManager

This loads an option model from a properties file or the properties XML format. Each key in the properties file is an options path separated by '.'.

Example usage:

```

PropertiesOptionsManager<MyOptModel> pom =
    new PropertiesOptionsManager(MyOptModel.class);

MyOptModel myOptModel =
    pom.loadOptions(myPropertiesFile);

```

3.4.3 XMLElementOptionsManager

This loads an option model from an XML file containing only elements (like the maven pom). This should actually be more or less compatible with the *configuration* section of a maven plugin configuration. That is you should be able to copy the content of the *configuration* section, stick it in its own xml file, and then load it with this class into the same model as a maven plugin does. That said this class has nothing to do with maven, it just uses the same XML format.

The XML file does not need to be DTD och XML Schema validated at all. It can be the simplest form of XML. It gets validated against the model when loaded, and model options can specify regular expressions that loaded values must match.

Using this or XMLAttributeOptionsManager is just a matter of personal taste in how you like to have your XML.

Example usage:

```

XMLElementOptionsManager<MyOptmodel> xeom =
    new XMLElementOptionsManager(MyOptModel.class);

MyOptModel myOptModel =
    xeom.loadOptions(myXMLStream);

```

3.4.4 XMLAttributeOptionsManager

This loads an option model from an XML file containing elements representing models and attributes representing value JavaBean proerties, and sub elements representing sub models. This is more or less the same format that Ant tasks use.

The XML file does not need to be DTD och XML Schema validated at all. It can be the simplest form of XML. It gets validated against the model when loaded, and model options can specify regular expressions that loaded values must match.

Using this or XMLElementOptionsManager is just a matter of personal taste in how you like to have your XML.

Example usage:

```

XMLAttributeOptionsManager<MyOptmodel> xaom =
    new XMLAttributeOptionsManager(MyOptModel.class);

MyOptModel myOptModel =
    xaom.loadOptions(myXMLStream);

```

3.4.5 Producing help text

There are 2 variants of help text generators. All take a prefix, path separator, and a PrintStream or a PrintWriter to produce output on.

- printHelpText(...)

This variant produces help text for only leafs of the model tree.

- `printHelpTextFull(...)`

This variant produces help text for all branches and leafs of the model tree.

Annotating the top level model with `@OptionsModel` and providing a description using the `description` attribute or `@Description("...")` will cause this to be displayed before all other properties descriptions. This is useful for a general help description.

3.5 Exceptions (`OptionsException`, `OptionsModelException`, `IOException`)

All public APIs throws `OptionsException`. You can get this on bad input when loading, problems parsing model when instantiating, and problems with model when loading.

There is however a subclass to `OptionsException` called `OptionsModelException` and this is thrown when there are any problems with the model class specified, like not finding a setter, not being allowed to instantiate a model, etc. Both the constructor and `loadOptions(...)` will throw this on model problems.

Model problems means that your model(s) are not entirely correct! A plain `OptionsException` means the end user has supplied bad input. You should consider handling these 2 cases separately in your code!

`OptionsManager` subclasses that do IO operations when loading options will also throw `IOException`.

3.6 Writing your own `OptionsManager` subclass for other inputs

Please note that the javadoc on the project site only shows the public user APIs, not the internals needed when doing your own subclass!

Declare a class extending `OptionsManager`.

```
public class MyOptionsManager<T> extends OptionsManager<T> {  
  
    public MyOptionsManager(Class optionsModelClass)  
        throws OptionsException {  
        super(optionsModelClass);  
    }  
  
    ...  
}
```

Then you need to do an inner class holding all the arguments passed to your `loadOptions(...)` methods. This class must implement the empty marker interface "Arguments".

Please note that this is a private inner class used nowhere else, so I consider it completely OK to not use setters and getters on this, but reference the fields directly, which can be done even if they are private.

```
private static class MyArguments implements Arguments {

    /** The prefix used to indicate an option. */
    private String argPrefix = "--";

    /** The command line args. */
    private String[] args;

    /** The delimiter of a component argument name that maps
        to a model structure. */
    private String modelSeparator = "-";

}
```

Provide a public `loadOptions(...)` user method for loading options, create an instance of your `Arguments`, copy arg data to it, and then pass it to `loadOptions(Arguments)` in base class. If the loading of options from your input does not throw any `IOException` you can call the `loadOptionsNoIO(Arguments)` instead.

```
public T loadOptions(String argPrefix, String modSep,
    String[] args) throws OptionsException {

    MyArguments arguments = new MyArguments();
    arguments.argPrefix = argPrefix;
    arguments.modelSeparator = modelSeparator;
    arguments.args = args;

    return loadOptions(arguments);
}
```

Now you have to override one of the following 2 methods and provide your loading implementation in it.

```
protected void loadOptions(OptionInfos optionInfos,
    Arguments arguments) throws Exception;
protected void loadOptions(OptionModelInfo optionModelInfo,
    Arguments arguments) throws Exception;
```

The first variant receives an `OptionInfos` object which contains all `OptionInfo` and `OptionModelInfo` created from parsing the model class. These are also used for instantiating and populating the model.

The `OptionInfos` class contains several methods for accessing the `OptionInfo` objects:

```
OptionInfo getOptionInfoByName(String name);

OptionInfo getOptionInfoByPublicPath(Path publicPath);

List<OptionInfo> getRequiredOptionInfos();

List<OptionInfo> getAllValueOptionInfos();

List<OptionInfo> getAllOptionInfos();
```

This is very useful when you are loading input of random paths. In this case you should also call

```
void assureModelInstance() ;
```

on an `OptionInfo` before trying to set a value on it (see further down).

The other variant that receives an `OptionModelInfo` is more useful for loading input of sequential paths. The `OptionModelInfo` received is the one representing the top level options model. It has 2 useful methods in addition to those inherited from `OptionInfo`:

```
List<OptionInfo> getChildren();  
  
OptionInfo getChildByName(String name);
```

These either return all immediate children or a specific child by name, where the name can be seen as a relative options path.

`OptionsModelInfo` also contains the following method:

```
Object instantiateModelAndAddToParent();
```

This will create an instance of the model represented by the `OptionsModelInfo` and add that instance to its parent model (which is expected to have already been instantiated).

No matter if you have random or sequential input the setting of an option value is the same. The `OptionInfo` class (which `OptionModelInfo` extends) contains the following method:

```
void setValueAsString(String value) throws OptionsException;
```

This will validate the value if a `@Option(validate="...")` or `@Validate("...")` have been specified for the value, and then try to convert the string into the type of the JavaBean property, and finally call the property setter with the value.

The easiest way is to download the sourcecode and look at the existing `OptionManager` subclasses.

4 Project Reports

4.1 Project Information

This document provides an overview of the various documents and links that are part of this project's general information. All of this content is automatically generated by **Maven** on behalf of the project.

4.1.1 Overview

Document	Description
Project License	This is a link to the definitions of project licenses.
Project Team	This document provides information on the members of this project. These are the individuals who have contributed to the project in one form or another.
Issue Tracking	This is a link to the issue management system for this project. Issues (bugs, features, change requests) can be created and queried using this link.
Source Repository	This is a link to the online source repository that can be viewed via a web browser.

5 Project License

5.1 Overview

Typically the licenses listed for the project are that of the project itself, and not of dependencies.

5.2 Project License

5.2.1 Apache 2.0

[Original text]

Copy of the license follows.

- [Home](#)
- [Foundation](#)
- [Projects](#)
- [People](#)
- [Get Involved](#)
- [Support Apache](#)
- [Download](#)
- [ASF Blog](#)

Apache License, Version 2.0

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

- 1 You must give any other recipients of the Work or Derivative Works a copy of this License; and
- 2 You must cause any modified files to carry prominent notices stating that You changed the files; and
- 3 You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- 4 If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part

of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

5.2.1.1 APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner] Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0> Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License. [Apache Projects](#)

- [HTTP Server](#)
- [Abdera](#)
- [ActiveMQ](#)
- [Ant](#)
- [APR](#)
- [Archiva](#)
- [Buildr](#)
- [Camel](#)
- [Cayenne](#)
- [Cocoon](#)
- [Commons](#)
- [Continuum](#)
- [CouchDB](#)
- [CXF](#)
- [DB](#)
- [Directory](#)
- [Excalibur](#)
- [Felix](#)
- [Forrest](#)
- [Geronimo](#)
- [Gump](#)
- [Hadoop](#)
- [Harmony](#)
- [HttpComponents](#)
- [iBATIS](#)
- [Incubator](#)
- [Jackrabbit](#)
- [Jakarta](#)
- [James](#)
- [Labs](#)
- [Lenya](#)
- [Logging](#)
- [Lucene](#)
- [Maven](#)
- [Mina](#)
- [MyFaces](#)
- [ODE](#)

- [OFBiz](#)
- [OpenEJB](#)
- [OpenJPA](#)
- [OpenWebBeans](#)
- [PDFBox](#)
- [Perl](#)
- [Pivot](#)
- [POI](#)
- [Portals](#)
- [Qpid](#)
- [Roller](#)
- [Santuario](#)
- [ServiceMix](#)
- [Sling](#)
- [SpamAssassin](#)
- [STDCXX](#)
- [Struts](#)
- [Synapse](#)
- [Tapestry](#)
- [TCL](#)
- [Tiles](#)
- [Tomcat](#)
- [Turbine](#)
- [Tuscany](#)
- [Velocity](#)
- [Wicket](#)
- [Web Services](#)
- [Xalan](#)
- [Xerces](#)
- [XML](#)
- [XMLBeans](#)
- [XML Graphics](#)
- [Attic](#)

5. *Foundation*

- [FAQ](#)
- [Licenses](#)
- [News](#)
- [Press Inquiries](#)
- [Public Records](#)
- [Sponsorship](#)
- [Donations](#)
- [Buy Stuff](#)
- [Thanks](#)
- [Contact](#)

5.Foundation Projects

- [Conferences](#)
- [Infrastructure](#)
- [JCP](#)
- [Legal Affairs](#)
- [Public Relations](#)
- [Security](#)
- [Travel Assistance](#)

5.How it works

- [Introduction](#)
- [Meritocracy](#)
- [Structure](#)
- [Roles](#)
- [Collaboration](#)
- [Infrastructure](#)
- [Incubator](#)
- [Public Relations Committee](#)
- [Other entities](#)
- [Glossary](#)
- [Voting](#)

5. Get Involved

- [Mailing Lists](#)
- [Version Control](#)
- [Developer Info](#)

5.Download

- [from a mirror](#)

5.Community

- [Memorials](#)

5.Related Sites

- [ApacheCon](#)
- [Feathercast](#)
- [PlanetApache](#)

Copyright © 2009 The Apache Software Foundation, Licensed under the [Apache License, Version 2.0](#).

Apache and the Apache feather logo are trademarks of The Apache Software Foundation.

6 Project Team

6.1 The Team

A successful project requires many people to play many roles. Some members write code or documentation, while others are valuable as testers, submitting patches and suggestions.

The team is comprised of Members and Contributors. Members have direct access to the source of a project and actively evolve the code-base. Contributors improve the project through submission of patches and suggestions to the Members. The number of Contributors to the project is unbounded. Get involved today. All contributions to the project are greatly appreciated.

6.1.1 Members

The following is a list of developers with commit privileges that have directly contributed to the project in one way or another.

Name	Email
Tommy Svensson	opensource@biltmore.se

6.1.2 Contributors

There are no contributors listed for this project. Please check back again later.

7 Issue Tracking

7.1 Overview

This project uses sourceforge to manage its issues.

7.2 Issue Tracking

Issues, bugs, and feature requests should be submitted to the following issue tracking system for this project.

```
http://sourceforge.net/projects/optionsmanager/support
```

8 Source Repository

8.1 Overview

This project uses a Source Content Management System to manage its source code.

8.2 Web Access

The following is a link to the online source repository.

```
http://optionsmanager.hg.sourceforge.net/hgweb/optionsmanager/
```

8.3 Anonymous access

Refer to the documentation of the SCM used for more information about anonymously check out. The connection url is:

```
http://optionsmanager.hg.sourceforge.net:8000/hgroot/optionsmanager/optionsmanager
```

8.4 Developer access

Refer to the documentation of the SCM used for more information about developer check out. The connection url is:

```
ssh://user@optionsmanager.hg.sourceforge.net/hgroot/optionsmanager/optionsmanager
```

8.5 Access from behind a firewall

Refer to the documentation of the SCM used for more information about access behind a firewall.