Plantlore Programming Guidelines

1. SVN Repository

Repository location

Plantlore SVN repository is located at BerliOS [www.berlios.de]

Repository structure

Repository contains sources as well as other documents and materials used for development. This is the current structure of folders in the repository:

- *analysis* - contains all files specifying how Plantlore should work
  - *BioCASE* – Information about BioCASE provider software
  - *Database* – Database model and DB related stuff
  - *Examples* – various examples (mostly java sources)
- *documentation* - contains all documents saying how Plantlore work
- *src* - contains Plantlore Java sources
- *test* - contains JUnit tests for classes in src directory
- *lib* - contains all libraries used by Plantlore

Using Plantlore Repository

General information about using CVS can be found at [here be dragons]. To get fresh version of the sources and other Plantlore related documents, use "`svn checkout svn://svn.berlios.de/plantlore/trunk plantlore`". In order to receive SVN commit notifications, join plantlore-dev@lists.berlios.de mailing list. This list is used for general development discussion as well. This is the list of Plantlore specific rules for using SVN:

- When adding or committing files, always include appropriate description of what changes were made.
- Always commit any changes to the HEAD branch.
- In case of any questions, contact the "Repository manager" Jakub at **fraktalek@users.berlios.de**

2. Source Code Style Guidelines

Source code should conform to the *Java Coding Style Guide w*hich can be found at http://www.sun.com/software/sundev/whitepapers/java-style.pdf. Please read (or at least skim through) this document before writing or modifying Plantlore source code.
This file contains selection of the most important rules from the document as well as Plantlore specific rules.

1. Source file style

- *File name* — First character of each word capitalized
No whitespaces or special characters in the file name
File extension: .java
- *Character encoding* — UTF8 (Unicode)
- *Language* — English
- *File size* — Less than 2000 lines
- *Indentation size* — 4 characters
- *Special characters* — (Such as greek alphabet ;-) Not forbidden but not recommended.

2. Naming conventions

- *Packages* - net.sf.plantlore.<package>.<subpackage>...; Current package structure:

    - net.sf.plantlore.client – sources for the GUI client application
    - net.sf.plantlore.server – sources for server-side application
    - net.sf.plantlore.common – common classes used by many other parts of application.
    - net.sf.plantlore.l10n – common classes and resources for localization
- *Class/interface* - First character of each word capitalized, no underscores. When creating MVC pattern, append “View” / “Ctrl” to the name of the class representing view / controller. In case of model there's no need to append anything. Example:
    - AuthorManager.java - Model of the Author manager MVC.
    - AuthorManagerView.java - View of the Author manager MVC.
    - AuthorManagerCtrl.java - Controller of the Author manager MVC.
- *Methods* - Each but first character of every word capitalized. Should be imperative verbs or verb phrases. Should describe function as much as possible
- *Fields* - Each but first character of every word capitalized
- *Constants (static final)* - All upper-case

3. Comments

To be removed comments should start with: XXX, e.g. // XXX: This is to be removed
Parts where something is to be done before releasing should start with: TODO

4. Miscelaneous

Do not use import wildcards (java.util.*). Always import only required class.

5. JavaDoc

For a full description of JavaDoc guidelines see http://java.sun.com/j2se/javadoc/writingdoccomments/index.html. Plantlore source code should conform to the rules and recommendations in this document.

Tags that should always be used (if applicable)

@param	@version
@return	@see
@throws	@since
@author	

How to write tag comments:

@author	Full name of the author
@version	
@throws	All checked exceptions (method declaration contains keyword throws) Unchecked exceptions that caller might want to catch
@since	Plantlore version in which the feature was first implemented

6. Logger

Plantlore uses log4j library for logging. For more information about log4j see http://logging.apache.org/log4j/docs/ or a document about using log4j from the Plantlore repository. Log4j provides 5 basic logging levels. Here is how they should be used:

- ***DEBUG*** – This level should be used for the lowest level debug information and debug messages. It is completely up to you what you need to log on this level.
- ***INFO*** – These messages should describe the progress of an application on a higher level, for example opening new dialog window, saving some records, completing a job.
- ***WARN*** – Warnings describe situations where an error didn't occurr but the situation could potentially lead to an error. For example getting an empty result set (when there should be at least something) is not an error but might lead to one later in the execution of application.

- ***ERROR*** – Error messages describe invalid application states which are mostly recoverable, although for example some functionality might not be accessible. This level should also be used for programming errors (e.g. your code detected an error in some API it is using)
- ***FATAL*** – Fatal logger messages should be used when application encountered a serious problem, often unrecoverable or only hardly recoverable. It should be used for example when database connection was lost unexpectedly, saving or opening of a file did not succeed etc.

7. L10N Properties

The properties allow us to specify localized messages, tool tips, titles, errors, warnings and descriptions. In order to keep the names of the properties lucid (easily understandable), the following rules must be abode.

The name comprises two parts: ***the namespace*** and ***the name*** of the property.

The concept of **namespaces** was introduced to be able to track the use of the property easily.

- Namespaces should correspond to packages or complex entities that must be presented to the User.
  For example: *Import*, *Export*, *Login*, *Record*, *Metadata*, *Publication*, *User*.
- Complex namespaces use the *dot* notation to separate its parts.
  For example: *Login.Authorization*.
- Some namespaces are more general and are supposed to be used by several packages, so that the way some events are reported is the same in the whole System.
  For example: *Error*, *Warning*, *Question*, *Common*.

The **name** describes the problem, message, or description of the property. It should be brief and self-explaining.

- For example: *Error.InvalidDBLayer*, *Error.FileNotFound*, *Error.InsufficientAccessRights*, *Login.Authorization.Password*, *Login.Authorization.UserName*, *Question.AbortImport*.
- Names of entities should correspond with their names in the source code.
  For example: *Occurrence.YearCollected*, *Plant.CzechName*, *User.Password*.

Some examples of the naming conventions:

Record.Property = Property
Record.ForeignKey = Foreign Key
Habitat.Country = Country
Author.WholeName = The whole name
Common.OK = OK
Common.Help = Help
Import.Replace = Replace
Import.Title = Import
Error.InvalidDBLayer = The database layer is not valid!\n It cannot be null.
Question.AbortImport = Import will be aborted.