Code Style Guide

Java code for the Geode project was originally developed using conventions based on The Elements of Java Style, by A. Vermeulen, S. Ambler, *et. al*.

As an open-source project we are following the Google Java Style guide, emphasizing a few of the more important points in this page.

General Principles

When modifying existing software, your changes should follow the style of the original code. Do not introduce a new coding style in a modification, and do not attempt to rewrite the old software just to make it match the new style. The use of different styles within a single source file produces code that is more difficult to read and comprehend. Rewriting old code simply to change its style may result in the introduction of costly yet avoidable defects.

Apply these rules to any code you write, not just code destined for production or only if it is visible to the user. This includes code documentation (javadocs).

Formatting Conventions

In order to create a source base that has a unified appearance and is easy to read and comprehend we include conventions for formatting Java code.

Most of the formatting conventions for this project center around the use of white space and the placement of brackets. Formatter settings for Eclipse and IntelliJ are covered first, followed by more detailed descriptions.

File encoding

Use UTF-8 file encoding and Unix line separators. Some source code files contain UTF-8 characters and will be damaged if you modify them with an editor that does not support UTF-8 encoding.

Line Length

Limit your source code line length to 100 characters.

Indentation

Improve code readability by grouping individual statements into block statements and uniformly indent the content of each block to set off its contents from the surrounding code.

Use spaces, not hard tabs, for indentation.

Use 2-space indentation in general.

Curly Braces

Always use braces, even around one-line `if`, `else` and other control statements.

Locate the opening brace '{' of each block statement in the last character position of the line that introduced the block. Place the closing brace '}' of a block on a line of its own, aligned with the first character of the line that introduced the block. The following examples illustrate how this rule applies to each of the various Java definition and control constructs.

```
// CORRECT: A public class definition
public class MyClass {
  ...
}

// WRONG:
public class MyClass
{
  ...
}

// WRONG:
public class MyClass {
  ...
  }

// CORRECT: A method definition
void method(int j) {
  ...
}

// CORRECT: A do/while
do {
  ...
} while (...)
```

Eclipse and IntelliJ formatter settings

Eclipse

On MacOS "Preferences..." can be found under the "Eclipse" menu.

Set the file encoding to UTF-8

```
Window > Preferences > General >
    Workspace > Text file encoding > Set explicitly to "UTF-8"
Window > Preferences > General >
    Workspace > New Text file line delimiter > Set explicitly to "Unix"
```

Set the editors to insert spaces instead of tabs

```
Window > Preferences > General >
    Editors > Text Editors > Check "Insert Spaces for tabs"
```

Import newline/brace/indentation formatter settings. This will create a new formatter profile which you should activate when working with project source files.

```
Window > Preferences > Java >
    Code Style > Formatter > Import >
        Import etc/eclipseFormatterProfile.xml
```

Then import the "imports" settings file

```
Window > Preferences > Java > Code Style > Organize Imports >
    Import etc/eclipseOrganizeImports.importorder
```

Avoid using auto-formatting - "Ctrl + Shift + F" for an entire file while changing a part of the file. Especially for an existing file as it makes it difficult to know the exact difference by comparing revisions.

IntelliJ

Open the Settings dialog with File -> Settings...

```
Select the Editor -> Code Style tab. Push the "Manage" button and then the "Import..." button
  Select IntelliJ format
    Import etc/intellijIdeaCodeStyle.xml
```

While IntelliJ supports importing of Eclipse formatter profiles you should not use this option because it does not properly configure IntelliJ for Geode. For instance, the arrangement of "import" statements will be incorrect if you use the Eclipse formatter profile.

Avoid using reformatting commands for an entire file while changing a part of the file. Especially for an existing file as it makes it difficult to know the exact difference by comparing revisions.