Apollo code guidelines

Revision number: 1

Date:

Name: P. van der Velde

# Introduction

* Use pre- and postconditions. Indicate for each method / property. For public methods / properties check the pre-conditions with an exception. For non-public methods check the pre-conditions with an assert.
* Make everything internal / private as much as possible. Figure out how to test private / internal parts.
* Reduce storage to a minimum. Try not to store global data on a class of which we'll have lots of copies (e.g. vertices, or points etc.)
* Reduce usage of strings.
* All public methods must check for invoke rights?
* If an element is unclear in use stop to check the design and improve documentation
* Use the debugger attributes where appropriate
* Define copy/clone constructors where it makes sense. Try not to use IClonable because it’s unclear what this should do
* Provide read & write accessor methods for global fields!

General

* Use 4 spaces for indentation. Do not use tabs
* Use m\_ for class fields. Use s\_ for static fields. Use g\_ for global variables. This way developers can easily see what a variable is and they show up in intellisense nicely.
* Do not use booleans for method parameters, use a properly named enumeration instead.
* Place a copyright at the top of each file

Language specific

General

* Stick to the coding guidelines for the individual languages, unless otherwise specified.

C#

* All structs should implement IEquatable<T> for performance reasons

F#

Resources

* Only put UI resources in a resource file. Create a default resource and bundle this with the assembly.
* Non-UI assemblies should not have any translatable resources. These resources should never be visible in the UI.

File I/O

XML

* Write all elements in CamelCase

Binary

* Add CRC’s to all sections to improve robustness
* Ensure that files written are divided into sections that are independent from each other

Documentation

* Never refer to the class/method/property etc. itself on its own documentation
* Always refer to the class/method/property by a <see ...> element the first time. After that use <code> tags
* Write proper documentation. If things are unclear read the documentation first, if the documentation is not clear then check the source code and improve the documentation.
* For each public method document
  + Purpose of method
  + Parameters & their ranges
  + Pre-conditions for the method
  + Post-conditions for the method
  + Return values
  + All exceptions
* Document original sources with the <source> tag
* Document design decisions with the <design> tag
* Document possible todo items with the <todo> tag. Note that this is not to be used for actual missing code. Either implement that code or create an issue in the issue tracker.

Testing

* Properly comment test cases.
* Test code is also source code. Treat it the same way as normal code!