**OMPL Style Guide**

This document provides a brief overview of OMPL coding standards. It is meant to assist developers who are contributing code to OMPL.

**Spacing**

* Each source block should be indented by 4 space characters. No tab characters should appear in source code.
* Each enclosing brace should reside on its own line, but braces can be omitted if the enclosed block consists of exactly one line.

The following is an example of desirable code spacing.

for (unsigned int i = 0 ; i < 50 ; ++i)

{

  if (i % 2 == 0)

      std::cout << i << " ";

}

**Filenames & Macros**

* Source filenames should begin each word with a capital letter, and there should be no underscores between words.
* Header files should use the .h extension, and implementation files should use the .cpp extension.
* Each implementation file should reside in a src/ directory, immediately below the directory containing the corresponding header file.
* Include guards in header files should be of the form OMPL\_PATH\_FILENAME\_, where the words in the filename are separated by underscores.

For example, the header and implementation files in which [**ompl::RNG**](http://ompl.kavrakilab.org/classompl_1_1RNG.html) is defined are [**ompl/util/RandomNumbers.h**](http://ompl.kavrakilab.org/RandomNumbers_8h_source.html) and [**ompl/util/src/RandomNumbers.cpp**](http://ompl.kavrakilab.org/RandomNumbers_8cpp_source.html), respectively. The header file begins with the include guard

#ifndef OMPL\_UTIL\_RANDOM\_NUMBERS\_

**Functions & Classes**

* Function names should begin with a lower-case letter and should begin each subsequent word with a capital letter, and there should be no underscores between words (e.g., [**ompl::base::StateSpace::setName()**](http://ompl.kavrakilab.org/classompl_1_1base_1_1StateSpace.html#ad12cc022ef531dcb563f6d3d13b847ad)).
* Functions accepting no parameters should have void in place of a parameter list.
* Class names should begin each word with a capital letter, again with no underscores (e.g., [**ompl::base::StateSpace**](http://ompl.kavrakilab.org/classompl_1_1base_1_1StateSpace.html)).
* Scoping directives (*public*, *protected*, *private*) should be at the same level of indentation as the class declaration.
* Class member variable names should begin with a lower-case letter and should begin each subsequent word with a capital letter. They should end with a single underscore (e.g., [**ompl::base::StateSpace::longestValidSegment\_**](http://ompl.kavrakilab.org/classompl_1_1base_1_1StateSpace.html#a8967afde4dc99333aef63eb85554bf81)).
* Names of constants and static variables should be in all capital letters, with an underscore between each word (e.g.,[**ompl::base::StateSpace::DEFAULT\_PROJECTION\_NAME**](http://ompl.kavrakilab.org/classompl_1_1base_1_1StateSpace.html#a45e65cf86a2c5cb1e4394dd8aafb4d3b)).

For example, consider the following source code, which follows the above guidelines.

class SampleObject

{

public:

  SampleObject(void) : objectTag\_(NUM\_INSTANCES++)

  {

  }

  int getObjectTag(void) const

  {

      return objectTag\_;

  }

  static int NUM\_INSTANCES;

private:

  const int objectTag\_;

};

int SampleObject::NUM\_INSTANCES = 0;

**Other Coding Guidelines**

* Compiler specific features should never be used (avoid use of #ifdef).
* Code must compile without warnings.
* If a class member function can be marked *const*, then it should be marked *const*.
* Member functions marked as *const* must be thread safe. If a function is intended to be thread safe, it should be marked as *const* (one can use the *mutable* keyword if needed).
* All classes, methods and member variables must be documented (Doxygen style). High-level documentation can be written either in [**MarkDown format**](http://www.stack.nl/~dimitri/doxygen/markdown.html) or in Doxygen format.
* Forward declaration of types should be done with the OMPL\_CLASS\_FORWARD() macro when shared pointers to the type are also needed: OMPL\_CLASS\_FORWARD() will define a shared pointer to the type using the *Ptr* suffix (e.g., [**ompl::base::StateSpacePtr**](http://ompl.kavrakilab.org/classompl_1_1base_1_1StateSpacePtr.html)).

[Physical and Biological Computing Group](http://www.kavrakilab.org/) **•** [Department of Computer Science](http://www.cs.rice.edu/) **•** [Rice University](http://www.rice.edu/)

Generated on Tue Jan 22 2013 11:06:06 by [doxygen](http://www.doxygen.org/index.html) 1.8.3

In matlab code for classes that have constant properties there should be a "costant\_property\_constructor" function that gets the properties from user or predefined values and distribute them to the appropriate variables. This is because we do not have static variables in matlab