Programming C++ in Style

Ruud van Gaal

October 7, 2011

Contents

Chapter 1

Intro

1.1 Introduction

Racer is the result of over 10 years of development, with more than 15+ years going into the lower-level libraries. These libraries have been developed to provide operating system independence, and also to provide useful utility classes.

This text explains the rules between the lines; the structure behind the source code, which has been keeping it all together.

It is assumed the reader has at least some basic knowledge on the C++ language.

Chapter 2

Racer decomposition

2.1 Libraries

The executable pulls together a bunch of libraries, most of which are developed in-house to provide abstract access to reusable classes.

2.1.1 Common libraries

Some libraries are not really specific to Racer, but are useful for all kinds of applications.

qlib The lowest-level library, abstracting the operating system. This provides classes such as windows, file writing abstractions, dialogs, event passing (for Windows, Mac and Linux) etc.

d3 A 3D library, able to open model files (the native format being DOF), import formats such as ASE, and do basic rendering.

world A higher level 3D world scene graph and rendering library. Uses D3 for its basic file management, but has its own methods for scene graph rendering.

nlib Networking library, mainly focused at UDP communication, although TCP is also supported. Also includes things such as a HTTP client and even server, if needed.

plib Physics library. Contains classes for filtering, Newton physics abstractions, but also standardized higher-level objects such as fuel tanks, and tire models.

util_mfc Various utility classes, such as a slideshow helper and a (magnetic) cardreader.

enet UDP network middleware.

jpeg Standard JPEG library for image loading and saving.

gll OpenGL extension handling.

2.1.2 Racer-specific libraries

Furthermore the following Racer-specific libraries exist:

racerlib Most of the building blocks of car simulation; classes like RCar, RWheel but also higher ones such as RRace.

racerlibu Utility modules and classes, such as database client/server links, specific user interface components (buttons/labels/listboxes...).

ui_public The menu modules; split at some point to make it easier to make up other types of game interfaces.

2.2 Symbol naming

An important factor in source code maintenance is to be consistent. In that light, the exact writing does not really matter, but some common styles are often used in most C++ code.

2.2.1 Variables

Variables are spelled in mixedcase, starting with lowercase, for example, 'chassisDamageFactor'.

2.3 Future subjects

- no negated names
- layout

2.4 References

- *http://geosoft.no/development/cppstyle.html* - a summary of in-depth C++ style. This article deviates a bit here and there from the tips given in that page.