Horizon Game Engine  
Internal Code Documentation

Table of Contents

[Compiling 1](#_Toc430172646)

[Folder organization 1](#_Toc430172647)

[Code categories 2](#_Toc430172648)

[Code files 2](#_Toc430172649)

[Display 2](#_Toc430172650)

[Experimental 2](#_Toc430172651)

[Game 2](#_Toc430172652)

[Gui 2](#_Toc430172653)

[Input 3](#_Toc430172654)

[Main 3](#_Toc430172655)

[Math 3](#_Toc430172656)

[Resource 3](#_Toc430172657)

[Util 3](#_Toc430172658)

[Run-time available resources 3](#_Toc430172659)

# Compiling

The project builds by invoking “make” with no arguments from the “bin” directory. It relies on a special makefile that automatically compiles all the .cpp files found in the “source” directory using GCC and links them together. It uses an external program “depfmt.exe” to clean the dependency info provided by GCC so it’s digestible by Make.

Use “make clean” to remove temporary build files before committing to GIT or SVN.

Do not use compile.bat, it’s out of date.

# Folder organization

**Bin** Contains the utilities necessary to compile the program

**Resource** Contains non-code game data: textures, models, sounds, etc.

**Source** Contains .cpp files with code implementations

**Build** Contains temporary files used by Make. This can be safely deleted.

**Include** Contains .h files with only declarations, no implementation (excl. macros and single-file libs)

**Random** Contains unused or irrelevant files

# Code categories

The folders inside **Source** and **Include** directories group the code into several broad categories by function:

**Display** Contains the low-level code to display things on screen

**Experimental** Contains exploratory code and code expected to be unstable

**Game** Contains code that is used in a game-specific rather than generic setting

**GUI** Contains window toolkits and other interface doodats

**Input** Contains code to receive and handle input from keyboard, mouse, etc

**Main** Contains the program entry point and top-level program control code

**Math** Contains abstract mathematical code

**Resource** Contains code to read/write multimedia and game resource files

**Util** Contains general programming clutches

# Code files

By category:

## Display

**paint.cpp**

**renderable.cpp**

## Experimental

**terrain.cpp**

**testfuncs.cpp**

## Game

**camera.cpp**

**physics.cpp**

## Gui

**Gui2/**

**Gui4/**

**console.cpp**

**Gui.cpp**

**Gui2.cpp**

**Gui3.cpp**

**newconsole.cpp**

**toolbox.cpp**

**valscreen.cpp**

**window\_modeller.cpp**

## Input

**codetostring.cpp**

**input.cpp**

**keybinds.cpp**

## Main

**control.cpp**

**main.cpp**

## Math

**convexhull.cpp**

**quaternions.cpp**

**vectors.cpp**

## Resource

**fonts.cpp**

**models.cpp**

**textureloader.cpp**

## Util

**debug.cpp**

**functor.cpp**

**globals.cpp**

**hook.cpp**

**messenger.cpp**

# Run-time available resources

This is a description of systems and functions that are currently working, have been set up, turned on, and available to be used from mostly anywhere:

**GUI2**: an immediate-mode GUI/window toolkit. Available controls:

GUI2base: base class for all controls

GUI2button: can be clicked

GUI2checkbox: a control that can be checked or un-checked

GUI2colorbox: a 2d rectangular color-wheel

GUI2dropdownlist: allows selection from a drop-down list of items.

GUI2frame: rectangle used to contain other elements

GUI2label: a non-interactive printed text

GUI2Image: a non-interactive displayed image

GUI2listbox: used internally by dropdownlist

GUI2radiobutton: similar to checkbox, but round.

GUI2radiogroup: a logical group of radiobuttons that allows only one to be selected at once

GUI2scrollBar: a frame with a scroll-bar to display more than fits in the screen at once

GUI2scrollslidey: internally used by scrollBar

GUI2slider: a numeric value that can be changed by dragging a slider

GUI2spinner: a numeric value that can be changed using up/down buttons

GUI2textEntry: a clickable field where text can be entered

GUI2valuedisplay: every frame, reads a value from a pointer and displays it

# Usage and How-to’s

Here’s how to do stuff with the engine:

## GUI How-to

Creating a GUI from existing elements:

|  |
| --- |
| GUIx \*node = new GUIx();  node→… = …;  root→addChild(node); |

Iterating over and deleting elements:

|  |
| --- |
| for(auto \*ch:node→getChildren()){  /// returns all children except non-clients  if(ch->name == “hi”){ ch→close(); }  /// close() deletes the node on the next frame  } |

GUI layout (manual):

|  |
| --- |
| Node→setSize(vec2(100,100));  Node→invalidate()  /// invalidate() causes the node to recompute it’s layout |

GUI layout (automatic):

|  |
| --- |
| GUIgrid \*grid = new GUIgrid();  grid→bSizeToParent = true; // node will be as big as parent allows  grid->resize(2,2);  auto \*node = new GUIbutton(“hello”);  node→sizeToContents() // node will be just big enough to fit contents  grid→addChild(node);  grid→grid(node); // equivalent to grid(node, row=-1, col=0);  grid->invalidate();  root→addChild(grid); |