

Learning Your Way Around Torque

Objectives Upon finishing this exercise you should be able to:

1. Have a feel for the overall structure of Torque game assets.

Resources

1. Navigate to the `examples` folder in your Torque Game Engine installation.

Open the Windows Explorer. Navigate to the directory where you installed the Torque Game Engine (C:\Torque\TGE_1_5_2 by default). Navigate down to `example`. Take note of the folders `starter.fps`, `starter.racing`, and `tutorial.base`. These three folders are the roots of sample games provided by GarageGames as part of the TGE. They are, unsurprisingly, a first-person shooter, a racer, and a simple tutorial.

2. The executable and DLL files for running the Torque demo are in the `example` directory.

`torqueDemo.exe` and the five `.dll` files are the executable file for running Torque Game Engine games. A game is defined using *Torque Script*; Torque Script files use a `.cs` extension.

When the `torqueDemo.exe` program begins, it looks in the current directory for the `main.cs` file, using that script to start the game. The `-game` command-line parameter takes an absolute path or a path relative to the current directory to use instead of the current directory when looking for `main.cs`.

Thus, the following command-line would start the `starter.fps` version of `main.cs`:

```
1 C:\Torque\TGE_1_5_2\example> torqueDemo.exe -game starter.fps
```

The `creator`, `demo`, and `show` folders are base folders for the world editor and other demonstration tools. The `common` folder defines many common script functions and objects.

The other files in this directory are DOS Batch and `bash` shell scripts which start the demo engine with the various game folders set as current.

3. Navigate down to `starter.fps`.

A game folder typically has `main.cs` and three subfolders:

```
1 starter.fps
2 |- main.cs
3 |-- client
4   |-- <script files for client side>
5 |-- data
6   |-- <game assets for both client/server>
7 |-- server
8   |-- <script files for the server side>
```

Client/Server? What is a client? What is a server? Why do I need them if I am going to build a simple, one-player game? Game engines which support multiple players often use a client/server architecture. The server is responsible for all actual game events such as spawning critters, dealing damage, assigning scores, and the like; the client is responsible for drawing pretty pictures on the user's screen.

This separation of responsibilities permits the creation of dedicated servers (machines where a server is running but no clients are running), shared servers (a server where a client is running on the same machine and connected to the server), and dedicated clients (machines where clients are running, connected to a server running on another machine). The shared server is also the configuration for playing a single player game.

What data? The `data` folder has subfolders for terrain, water, special effects, shapes (models), interiors (buildings), sounds, and missions. Looking at the file extensions of most of the files in these folders makes clear what they are (textures, sounds, 3D models, etc.). The `missions` folder can be a little bit confusing with two files for each named mission: `barebones.mis` and `barebones.ter`. The `.mis` file is a script file describing the mission It is generated by the mission editor. The `.ter` file is a terrain description saved by the terrain editor in a binary format.

It is important that you (and your team!) come up with a consistent naming and storage convention. Torque will let you specify arbitrary file paths to load scripts and resources; you should use standard storage locations so that when you need to replace or edit some shape, texture, or sound, you have some idea where to go.

4. Navigate to the `engine` folder to see the C++ code.

The `engine` folder, sibling to the `example` folder, is the base folder for the Torque Game Engine's C++ code. This folder has subfolders for all of the game engine subsystems including `audio`, `collision`, `gui`, and `platform`. The comments are not as complete as someone new to the codebase might wish but there are header comments in most files.

Reading the C++ is interesting; there are several very important macros (`DECLARE_CONOBJECT`, `ConsoleFunction`) which it is worthwhile to understand. Look at `console\console.h` for definitions of the `ConsoleFunction*` collection of macros and `console\consoleObject.h` for the `CONOBJECT` macros. The console header files are better commented than many; read the comments on what the macros are for and how to use them.

Note: Many lines of comments are more than 100 characters across; keep this in mind when selecting an editor.