Getting Started:

The Beginners Guide to CSQC

(for Darkplaces and FTE QuakeWorld)

A guide by Marco “eukara” Hladik

Introduction

Thousands of people have touched upon the greatness that is *QuakeC*.
If you are unfamilar with *Clientside QuakeC*, or *CSQC* in short, then you only might have worked with what we nowadays refer to as *Serverside QuakeC*, aka *SSQC*.

Serverside QuakeC runs on the server, just like the name suggests. The server runs the essential logic required to make the gameplay happen fairly across all clients.
The server is the master of the game, whereas the client is just a terminal.
How the game is interpreted visually is not decided by the server.

This of course, created quite a few limitations for modders back in the day. Custom menus and heads-up-displays were not really possible. While you could spam print calls to specific clients to create ‘fake’ menus, these caused network lag and do not interface with the keyboard/mouse directly.

And that is where CSQC comes in.
The idea of a clientside QuakeC system had already been discussed way back when the Quake engine code was about to be released. These plans never went anywhere until David “Spike” Walton, came along.
He originally created the current specification of CSQC for his Quake engine fork **FTE QuakeWorld**.

Since then, the arguably most popular engine, **Darkplaces**, has partially adopted his spec.

While I try to write this guide to target both engines, I do encourage you to use FTE QuakeWorld for working with advanced CSQC. Darkplaces has strong limitations for when it comes to networking information (Scoreboards, etc.) and is missing certain callback functions and builtins that generally make working with CSQC a great experience.
In short, not all things CSQC can do will be possible to see in Darkplaces.

If you do not know basic QuakeC syntax, you absolutely have to learn that first.
Do you know what a field in QuakeC is? Then read along!

What You Need

The days of vqcc, fastqcc and frikqcc are long gone.
If you want to code seriously in QuakeC nowadays, you've got two options: *fteqcc and gmqcc*.
Note that gmqcc is not exactly designed for Quake, although it supports it.
You'd have to compile your code with the **-std=fteqcc** flag. If you are going to compile with that one though, you might as well just use fteqcc.

From now on this guide assumes you are using a compiler that is compatible with fteqcc's additions.

For compiling you need two files, next to the compiler:

> *csprogsdefs.qc*

as well as a new file called

> *progs.src*

progs.src works the same as it did in the old SSQC.
You specify which files you want to include in the order that they belong in.

Setting Up Your Environment

Since SSQC and CSQC are two seperate entities, I encourage you to put them into their own seperate directories as well.
In this documentation I am assuming you have a subfolder in your mod directory containing the source code. For example: ***id1/csprogs_src/***

Inside the new, blank progs.src you will now have to include the list of files that you are going to use.
Let's start off with this template:

```
#pragma progs_dat  "../csprogs.dat"
#define CSQC
#includelist
csprogsdefs.qc
// Your own files below
#endlist
```

Now, try compiling that, you will most likely receive the following errors:

> *error: function CSQC_Init has no body*
> *error: function CSQC_Shutdown has no body*
> *error: function CSQC_InputEvent has no body*
> *error: function CSQC_UpdateView has no body*
> *error: function CSQC_ConsoleCommand has no body*

That's because these are the bare minimum callback functions that you need to implement in order for CSQC to work.
This is like having a progs.dat without worldspawn, PlayerPreThink and alike.

Luckily, these aren't too many functions.
Create a new (or multiple if you'd like to categorise them) .qc file. In this example, I will call it main.qc.

Let's go through the 5 callbacks that need to be implemented, one by one.

```
void CSQC_Init( void ) {


}
```

This is the equivalent to worldspawn in SSQC. If you want to do precaches of any sort, put them here.
You can precache assets just like in SSQC via **precache_model** and **precache_sound**.
In order to draw graphics clean and quickly, there is a new precache builtin called **precache_pic**.
That will be used in an example later on.

```
void CSQC_Shutdown( void ) {


}
```

This is called when CSQC ends, for example when you disconnect a server. Rarely used.

```
float CSQC_InputEvent( float flEventType, float flKey, flo
at flCharacter ) {
        return FALSE;
}
```

This is the direct interface to your input hardware. Can detect between keyboard, mouse and touch based inputs with possible more to come.
Only really used when you want to create interfaces.

```
void CSQC_UpdateView( float flWidth, float flHeight ) {
    clearscene();
    // Manipulate view, add entities below
    renderscene();
    // Any 2D elements will be below
}
```

The bare minimum for updating the view, run every single frame.
If you do not call **renderscene()** the entire output would be black.
The comment between **clearscene()** and **renderscene()** indicate where you want your code to be that is going to manipulate the way the game is drawn in the 3D view.
The last comment indicates where you want to draw your overlays, like the heads up display.

```
float CSQC_ConsoleCommand( string strCommand ) {
    return FALSE;
}
```

Makes it possible to intercept and interpret custom console commands.

You can add all the functions above into the **main.qc** file I have talked about earlier, then mention it in the progs.src under the **"// Your own files below"** comment.
Now run fteqcc inside the same directory as your progs.src, then launch Darkplaces/FTE QuakeWorld.

You may now observe that the entire world is being rendered, except for entities such as buttons, triggered walls, monsters and so on.

Basic Rendering

In the last chapter we've made it possible to draw a simple scene simply consisting of the BSP geometry. The world is always drawn by default.
In order do change this, you will have to add the following line between **clearscene()** and **renderscene()**:

```
setproperty( VF_DRAWWORLD, FALSE );
```

This is where you will learn about setproperty.
setproperty tells the engine to change the way the engine renders the 3D view.

Next to **VF_DRAWWORLD** you also can toggle it to enable drawing the engine statusbar with **VF_DRAWENGINESBAR**, or the crosshair with **VF_DRAWCROSSHAIR**.

```
// Manipulate view, add entities below
setproperty( VF_DRAWWORLD, TRUE );
setproperty( VF_DRAWENGINESBAR, TRUE );
setproperty( VF_DRAWCROSSHAIR, TRUE );
```

Will do just that. Compiling the csprogs.dat now will result in your seeing the world, the engine
heads-up-display as well as the crosshair (if its enabled in your engine).

Now, this does not add entities to the screen. In order to fix that, I will now teach you about another
builtin, called **addentities()**.
This builtin takes **one argument** that targets **all entities with that .drawmask flag**.
By default, every SSQC entitiy that is visible in standard Quake has a .drawmask value of **MASK_ENGINE**.

```
addentities( MASK_ENGINE );
```

This will cleanly tell the engine to draw all the objects that it would be drawing if **csprogs.dat** didn't exist.

Compiling the **csprogs.dat** now will now show all entities in the game world. Making Quake fully playable once more.

Now that we are seeing everything the server is seeing, we have to make our *viewmodel (the first-person weapon model)* appear. Since the model is not drawn on the server, but directly inside the engine, it does not have a drawmask of **MASK_ENGINE**.
Modifying the addentities line to include **MASK_ENGINEVIEWMODELS**, will fix this:

```
addentities( MASK_ENGINEVIEWMODELS | MASK_ENGINE );
```

This will tell the engine to draw both types of entities.
If you want to be really fancy and disable the viewmodel for the intermission screen, there is a global in CSQC, called **intermission** that you can check against:

```
addentities( ( intermission ? 0 : MASK_ENGINEVIEWMODELS )
| MASK_ENGINE );
```

Compiling and running the game now will result the engine in rendering Quake fully.

Here are the final contents of progs.src:

```
#pragma progs_dat  "../csprogs.dat"
#pragma target fte
#define CSQC

#includelist
csprogsdefs.qc
main.qc
#endlist
```

Here are the final contents of main.qc:

```
void CSQC_Init( void ) {

}

void CSQC_Shutdown( void ) {

}

float CSQC_InputEvent( float flEventType, float flKey, flo
at flCharacter ) {
```

```
    return FALSE;
}

void CSQC_UpdateView( float flWidth, float flHeight ) {
    clearscene();

    // Manipulate view, add entities here
    setproperty( VF_DRAWWORLD, TRUE );
    setproperty( VF_DRAWENGINESBAR, TRUE );
    setproperty( VF_DRAWCROSSHAIR, TRUE );
    addentities( ( intermission ? 0 : MASK_ENGINEVIEWMODEL
S ) | MASK_ENGINE );

    renderscene();

    // Any 2D elements will be here
}

float CSQC_ConsoleCommand( string strCommand ) {
    return FALSE;
}
```

This covers the basics of how to get CSQC up and running.