The Command Module

The Command Module is a seemly complex beast and at first it isn't quite clear what benefit it provides when developing games using Orx.

So what does the Command Module actually give us?

1. The Command Module provides a **Console** that allows you to enter commands to manipulate Objects, the Config, and other parts of the Orx system without having to change your code, or your config, and without having to recompile/re-run. It allows you to do a “dress rehearsal” of Object positions, alpha, or any other parameter in realtime to see the effect of a change instantly. The Console can be considered somewhat similar to Immediate Mode in Visual Studio, or even Direct Mode in the old Amiga AMOS programming environment.

2. The Command Module presents a **Command Syntax**. What you enter into the Console can be taken and used to build Tracks. Tracks are groups of manipulations to objects or config at set time intervals. Without tracks, chaining lots of effects in a sequence would otherwise have be managed by variables and events, which can become complex and messy. Tracks will be dealt with in the Tracks article.

This article will cover how to use the command syntax in the Command Module's Console, and how to perform manipulations.

Starting with a basic project

If you don't have a basic project to work with, head over to here to set one up.

Define a sprite object in your config:

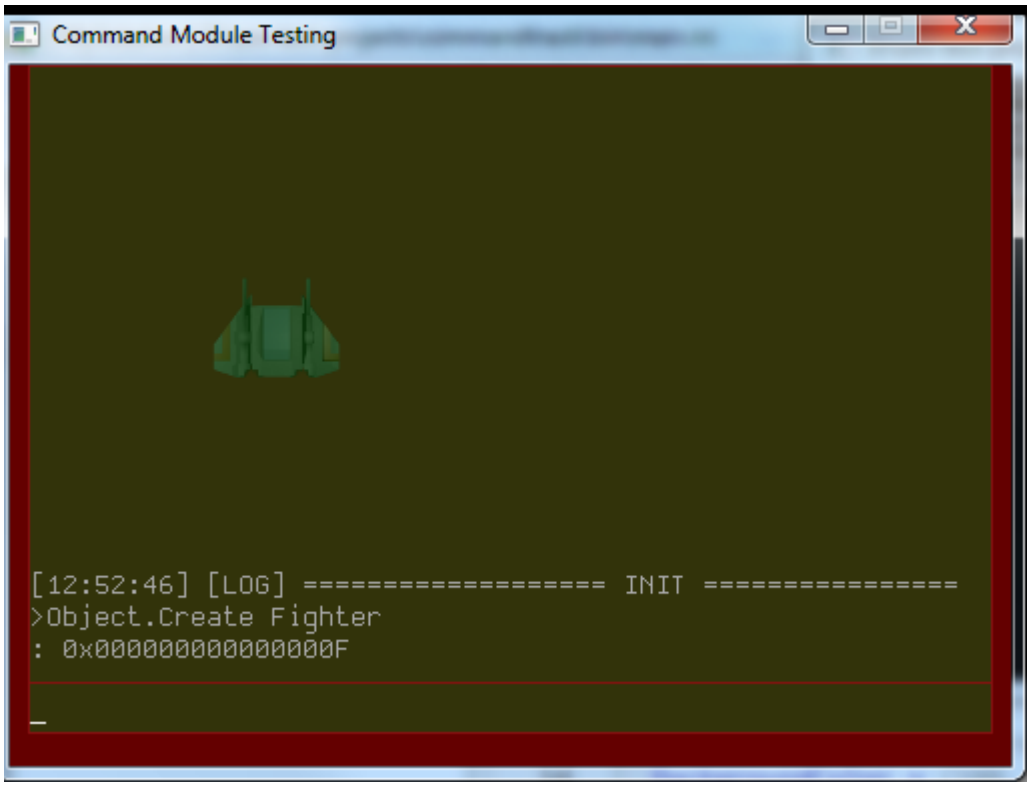
```
[Fighter]
Graphic = FighterGraphic
Position = (100, 100, 0)

[FighterGraphic]
Texture = fighter.png
```

And here is an example sprite image to use:

Press ` to enable the Console overlay.

The first command we will try is creating an object. To do that, enter:

```
> Object.Create Fighter
```



This will create an instance of the Fighter object and place it at vector (100, 100, 0) as defined in the config.

Notice the '>' symbol before the Object.Create command? This means to push the result to the stack. Why would you do this? We have no variables in the Command Module, so the stack is used. It's very handy. Think of the left hand side as the result of a method, ie: result = Object.Create. The “result” is pushed to the stack with the '>' symbol.

And '<' is to pop off the stack. < will take the Object reference from the stack and allow you to do something with it. Let's try an exercise to demonstrate:

Exercise One - Create an object and move it

```
> Object.Create Fighter
Object.SetPosition < (200, 200, 0)
```

So in the above, a Fighter was created and placed at (100, 100, 0) because of the config for it. The object reference was then placed on the stack from the '>'.

Then, the SetPosition was called. The '<' was used to pop the object reference back off the stack and feed it to the SetPosition command. The new coordinates were also supplied. The object moves to (200, 200, 0).

At this point, the sequence is over. Because we popped it off the stack to set the position, we can't use '<' again. If you wish to affect the same object over and over, you need to push it back on the

stack. Alternatively, you can use the tab key to substitute in the last ID.

Let's try both techniques in exercise two:

Exercise Two - Create an object and move it twice

```
> Object.Create Fighter
> Object.SetPosition < (200, 200, 0)
Object.SetPosition < (300, 300, 0)
```

Right. Notice the > on the second line. We pop the object off the stack, set it to (200, 200, 0) and push it back onto the stack again with the '>' at the beginning. It is now available with the next command.

What if you didn't push the object back on the stack for the second SetPosition command? No problem, just use the tab key to call the last ID:

```
> Object.Create Fighter
Object.SetPosition *press-tab-key* (200, 200, 0)
Object.SetPosition *press-tab-key* (300, 300, 0)
```

This is a handy way to continuously adjust the positioning of objects in your game, and getting the correct coordinates to then transfer to your code or config.

Exercise Three - Create an object, fade it out, and destroy it after 5 seconds

You will need a fadeout FX in your config for this exercise. Here a typical one:

```
[FadeOutFXSlot]
SlotList = FadeOutFX

[FadeOutFX]
Type        = alpha
StartTime   = 0.0
EndTime     = 0.5
Curve       = linear
Absolute    = true
Period = 0.5
EndValue  = 0;
StartValue   = 1;
```

Now onto the commands:

```
> Object.Create Fighter
> Object.AddFx < FadeOutFXSlot
Object.SetLifeTime < 5
```

Exercise Four - Create three fighters and destroy them in reverse order

```
> Object.Create Fighter
> Object.Create Fighter
> Object.SetPosition < (200, 200, 0)
> Object.Create Fighter
> Object.SetPosition < (300, 300, 0)
Object.Delete <
Object.Delete <
Object.Delete <
```

Now that you know a little about the command syntax, let's try using these command sequences in a track and call it from code.