Search

Config

This section is dedicated to the Config System, which is the “data-driven” side of Orx. Config greatly reduces the amount of code and logic required in your project.

Intro

The config system is part of Orx's core modules. It's the main module that makes Orx a data-driven game engine.
As it permits to do on-the-fly tweaks and modifications, it is one of the most important modules to learn.
When you get comfortable with it, your development time will be drastically shortened.

Its format is basically the one used in INI files. However, some concepts such as inheritance and override have been added.
A couple of specific operators have also been added [1] and will be covered in the syntax section.

Config Syntax

To learn about the basic syntax, how to use inheritance, how to include other config files, value types and lists, see the syntax page.

Main Settings

Main settings refer to config used to set up overall game properties. The config modules that belong to main are:

Config (settings about config), Console, Clock, Display, Input, iOS, Locale, Mouse, Param, Physics, Render, Resource, Screenshot, SoundSystem

Structure Settings

Structures settings refer to “structures” for use in your game. Structures are objects, cameras, bodies, etc. The config modules that belong to structure are:

Animation, Body, Joint, Camera, Clock, FX, Graphic, Object, Shader, Sound, Spawner, Text, Viewport, Timeline Track

Basic information

When Orx is initialized, it will use the executable name to deduce the main config file name.
Basically it will remove the extension from your executable name, if any, and add the `.ini` extension: if your program is called `MyNewGame.exe`, the main config file that will be loaded when Orx is initialized will be the `MyNewGame.ini`.

In that file you can include as many other config files [2)] as you want.

```
@SomeOtherConfigFile.ini@
```

Advanced information

The basic information above is all you need to get going with config in Orx. However, much more can be done with configuration files in your code.

You can load config files directly in code.

```
orxConfig_Load("MyConfigFile.cfg");
```

Once a config file is loaded, accessing its information will be done in memory, the file won't be read from disk again unless specifically asked with calls to `orxConfig_Load()` or `orxConfig_ReloadHistory()`.

You can even save the content of the config to a file, either completely or partially. Here's the function you need to call in order to do so:

```
orxConfig_Save("MyConfigFile.cfg", bEncrypt, MySaveFilter);
```

`MySaveFilter` is a callback that will be called for every section and every key to let you decide if you want to save this info to the file or not.
If you pass `orxNULL` instead of a valid callback, every single key/value pair will be saved.
Please note that the original comments of originally loaded files won't be saved as they're ignored during the loading and never stored in memory.

In your `MySaveFilter` callback, when provided with a name for a section and `orxNULL` as a key, it is for you to decide if you want to save some parts of this section (by returning `orxTRUE`) or if you'd rather skip the whole section completely (by returning `orxFALSE`).
In the first case, you will then be asked for each key/value pair of this section if you want to save it or not.

This filter system is very handy when you want to save partial data, like for handling save games as you'll see in the savegame section.

You can find the config module API in doxygen format via Orx's doxygen doc page or directly here.
We believe the API is self explanatory and shouldn't be much troublesome to learn.

Config's module use, data-wise, is thoroughly described in below sections.
The last section shows how to adapt it for handling save games.

There are many ways in which the config module can be helpful: for example, it already is the base for the localization module.

It's now up to you to find other clever ways of using it depending on your needs. 

Most of Orx's behaviors can be controlled via the config system.
They can be separated in two sections: default settings for controlling aspects such as Display, Physics, Input, ... and structure creation for controlling the creation of Objects, Sounds, Graphics, Shaders, etc...

Encrypting Config

See Encryption & orxCrypt program in order to learn how to encrypt / decrypt config, and the software tools available.

1)
for handling lists, randoms or included files, for example
2)
with any kind of extension, `.ini` isn't mandatory