Retrieving and Changing Config Values

Configuration files contain sections and properties that describe many items in Orx. But they can also be used to track values during runtime.

A `Level` property is a good example of a value that might be tracked and changed in your game. You could use a variable in your main code file to track variables like this, however there are advantages to moving these kinds of values to your config:

1. All your data will remain together
2. You can change and test your values from one place without needing to compile
3. You can have various override config files for different situations, eg a debugging config, a release config etc.

Let's work through how we can retrieve values from our configuration files and change them in memory.

Setting up a Property and Key Value

Begin by creating a common section for all your runtime related values. We'll call it `Runtime`. Though you can call it anything you like:

```
[Runtime]
Level = 1
```

So our `Runtime` section will contain a property called `Level` and will be set by default to 1.

Our code can retrieve this value very easily with `orxConfig_GetU32`:

```
    int level = 0;

    orxConfig_PushSection("Runtime");
    level = orxConfig_GetU32("Level");
    orxConfig_PopSection();
```

To make this reusable in your code, place it into a function:

```
int GetLevelFromConfig() {
    int level = 0;

    orxConfig_PushSection("Runtime");
    level = orxConfig_GetU32("Level");
    orxConfig_PopSection();

    return level;
}
```

In the same way, we can change the value with `orxConfig_SetU32`:

```
void SetLevelInConfig(int newLevel) {
    orxConfig_PushSection("Runtime");
    orxConfig_SetU32("Level", newLevel);
    orxConfig_PopSection();
}
```

Please note that this only changes the value in memory. Your `Level` property in your config file stored on disk is not affected. When your application is restarted, the original values are loaded back from the configuration files in your filesystem.

If you are interested in persisting values back to disk in a separate config file, see Saving Game information to the Config.

Lists of Strings

You can retrieve a value from a list of strings, floats, ints or vectors stored in your config. We'll have a go at retrieving string values.

Change your `Runtime` section to contain a list of level titles:

```
[Runtime]
Level = 1
Titles = Desert # Ocean # Mountains # Forest # City
```

If you wanted to get the third value from the list, which is Mountains, you can use:

```
    orxConfig_PushSection("Runtime");
    const orxSTRING value = orxConfig_GetListString("Titles", 2);
    orxConfig_PopSection();
```

You can also choose a random value from the list by supplying a -1 index value:

```
    orxConfig_PushSection("Runtime");
    const orxSTRING value = orxConfig_GetListString("Titles", -1);
    orxConfig_PopSection();
```

An alternative way to get a random value from a list is to instead use `orxConfig_GetString`:

```
    orxConfig_PushSection("Runtime");
    const orxSTRING value = orxConfig_GetString("Titles");
    orxConfig_PopSection();
```

Changing Orx Properties during runtime

Not only can you retrieve and update your own values, but also values of sections that define Orx objects (or any part of the Orx config).

Examples of use:

1. Changing the object that is spawned by a spawner
2. Changing the text of a text object for the next time it is displayed

In the case of changing a spawner's object, let's say you had the following section:

```
[AlienSpawner]
Object = MidSizeAlien
```

Change the object it uses:

```
    orxConfig_PushSection("AlienSpawner");
    orxConfig_SetString("Object", "BossAlien");
    orxConfig_PopSection();
```

The spawner will instantly switch to spawning out `BossAlien` objects.

We'll leave it there. This really just scratches the surface of what is possible. You can see more config related functions available in the API.