Changing the default application configuration file

When an Orx game or application starts, the matching config data file is expected to reside in the same folder as the executable, and to have the same name as the executable, minus the extension. For example:

```
bin/AlienAttack.exe
bin/AlienAttack.ini
```

When first executing the `AlienAttack.exe` file, the default bootstrapping seeks for the `AlienAttack.ini` file and begins processing it before Orx processes the init() function.

This default behaviour can be changed, allowing you to load your root configuration file from a completely different location, and optionally with a completely different name.

This is done by adding a Bootstrapping function like so:

```
orxSTATUS orxFASTCALL Bootstrap()
{

    // Add "../../data/config" to the list of locations that config files can be loaded from
    orxResource_AddStorage(orxCONFIG_KZ_RESOURCE_GROUP, "../../data/config", orxFALSE);

    // Loads a specific config file
    orxConfig_Load("MyRootConfig.ini");

    return orxSTATUS_FAILURE;
}
```

orxConfig_Load is used to call the new root config file of `MyRootConfig.ini`. Because we added the new config resource path previously, Orx knows where to try and find the config file.

The role of returning orxSTATUS_FAILURE or orxSTATUS_SUCCESS

In the previous example, the return status is set to `orxSTATUS_FAILURE`. Try to load `MyRootConfig.ini` manually and give up after that. This effectively stops Orx from continuing to look for a default config (based on the chosen build config) after the Bootstrap function has completed.

If `orxSTATUS_SUCCESS` is returned, Orx will still try and load the default config located next to the executable.

What does this mean?

Depending on whether you select Debug, Profile or Release when compiling, Orx will try to automatically find your named ini file.

For example, if your executable was: `AlienAttack.exe`, and you compiled as Debug, and you didn't try to load a specific file, then Orx will try to load `AlienAttackd.ini` for you. Here's a code example:

```
orxSTATUS orxFASTCALL Bootstrap()
{

    // Add "../../data/config" to the list of locations that config files
can be loaded from
    orxResource_AddStorage(orxCONFIG_KZ_RESOURCE_GROUP, "../../data/config",
orxFALSE);

    return orxSTATUS_SUCCESS;
}
```

In the above example, tell Orx where to find the ini file, but return `orxSTATUS_SUCCESS` so that Orx still use the selected build config to look for the named file.

Executing the bootstrap function

Next, in the main() function, set the Bootstrapping function before the *orxExecute* line:

```
int main(int argc, char **argv)
{
  orxConfig_SetBootstrap(Bootstrap);

  orx_Execute(argc, argv, Init, Run, Exit);

  return EXIT_SUCCESS;
}
```

Your application will now call its root .ini file from some other location relative to the .exe.