Object Traversing

Imagine you have this INI file (not everything is shown in it):

```
[Scene]
ChildList = Terrain # Tank # Enemies

[Terrain]
Graphic    = @
Texture = grass.png

[Tank]
Graphic = @
Texture = tank.png
GSide = player

[Enemies]
ChildList = Turret1 # Turret2

[Turret1]
Graphic = TurretTexture
GSide = enemy

[Turret2]
Graphic = TurretTexture
GSide = enemy
```

Then in your code you load scene object as a whole:

```
orxSTATUS orxFASTCALL Init()
{
    orxViewport_CreateFromConfig("Viewport");
    _scene = orxObject_CreateFromConfig("Scene");
...
}
```

In the code above you rely on orx to traverse the object tree and create it appropriately. However, in our INI file we have added a new key GSide with values “player” and “enemy”. GSide stands for game side. Our game code needs to initialize its internal data state according to the value of the GSide. Some objects may not have GSide key at all. Traversing INI file with config calls is possible, but it is not simple.

You can traverse [Scene] object child tree with the following recursive function:

```
void InitObject(orxOBJECT *obj) {
    orxLOG("%s: name", orxObject_GetName(obj));
}

typedef void (*ObjectHandler)(orxOBJECT *obj);
```

```
void traverseScene(orxOBJECT *child, ObjectHandler objHandler) {
    orxOBJECT *sibling;
    while ((child = orxObject_GetOwnedChild(child))) {
        objHandler(child);
        sibling = child;
        while ((sibling = orxObject_GetOwnedSibling(sibling))) {
            objHandler(sibling);
            traverseScene(sibling, objHandler);
        }
    }
}
```

This code traverses through all of the [Scene] children, including their children and so on.

In the code above the specific functionality to handling of the object is delegated to InitObject function. Put your custom logic into it and you are done.

"iarwain" has made a few points to keep in mind

- This will not traverse any objects that are not connected to scene hierarchy
- Objects that have been created as part of the Scene hierarchy can decide to exclude themselves from that hierarchy at any point (usually done for permanent objects, UI objects, etc…)
- You might want to look at Scroll, which is a thin C++ layer on top of orx.

Orx does bookkeeping on all the orxSTRUCTURE derivative that are created, if you want to iterate through all the orxOBJECTS, you can do so like this:

```
for (orxOBJECT *pstObject =
orxOBJECT(orxStructure_GetFirst(orxSTRUCTURE_ID_OBJECT));
     pstObject != orxNULL;
     pstObject = orxOBJECT(orxStructure_GetNext(pstObject)))
```

Alternative to Traversing when Loading from Config

A better approach would be to attach your data when your object is created, by listening to the orxOBJECT_EVENT_CREATE event.

Another alternative is to use scroll C++ wrapper as it takes care of orxOBJECT_EVENT_CREATE with its binding mechanism and other tasks.

See also

Get OrxObject by Traversing Structures