Scripting

Important notes

Avoid Empty Methods
Never implement an empty callback method. Calling the method takes quite a bit of cpu power compared to just skipping it. This is especially important for Update, PostUpdate, OnDraw and the like that are called every update / frame.
So if you want to remove function do NOT:

```
void Update(float afTimeStep)
{
  return;
  UpdateStuff(afTimeStep);
}
```

Instead to DO like this:

```
/*void Update(float afTimeStep)
{
  UpdateStuff(afTimeStep);
}*/
```

Callbacks

When doing setting a callback for a function, the callback function will either be searched for in the Entity or the current map script (depends on the function, but most will be the map script.

However, if you want to call a global function, that is one that is outside of a class. Then you can use the prefix `$`, to do that. For example: `SomeFunction(…, "$CallbackFuncDecl")`. Note that this class must be in the same file (or in one of the included files) as the class objects that would otherwise would have been searched exist in. So if you do `$SomeFunc` for a callback that checks the map file normally, then this will call the global func `void SomeFunc()` in the map script.

To be clear, here is the difference between a global function and a class method:

```
//The following is a global functuion
void DoStuff(){
  ...
}

class cAClass {
   //this is a class method.
   void DoStuff()
   {
      ....
```

```
    }
}
```

Base classes

Map

General

This is the basic structure for a level script. It should control all level specific code.

Callback Methods

These are the methods that are called from the Map:

void OnStart()
Called the first time the map is loaded.
void OnEnter()
Called every time the map is loaded
void OnLeave()
Called when player leaves the map
void CreateData()
All special data needed for the map (and not saved) are created here.
void DestroyData()
All special data needed for the map (and not saved) are destroyed here.
void Update(float afTimeStep)
Called every update tick
void PostUpdate(float afTimeStep)
Called after all normal Update during a tick has been called
void OnGui(float afTimeStep)
When ImGui_* functions should be called.
void OnAction(int alAction, bool abPressed)
When an action is made.
void OnAnalogInput(int alAnalogId, cVector3f &in avAmount)
When an analog action is made.
void OnPlayerDead(int aType, const tString&in asSource)
When the player has died from damage. asSource is the name of the killer entity
float DrawDebugOutput(cGuiSet @apSet,iFontData @apFont,float afStartY)
This only used for pure debug purposes. Use cLux_DrawDebugText(…) for easily outputting messages. “Show Map Info” in the F1 menu must be turned on for it to show up.
void OnRenderSolid(cRendererCallbackFunctions@ apFunctions)
Useful for drawing debug output or if you want some really specific effect in the level.

User Module

General

A user module is a class that has all kinds of functionality. It is really just base class that can be filled to take care of some kind of behavior. Normally the modules are not used directly, but instead have helper functions that simply the calling of various methods.

Modules are added in “Modules.cfg”.

The games come with a pre-made ones that can be find in User Modules page.

Callback Methods

void Update(float afTimeStep)
Called every update tick.
void PostUpdate(float afTimeStep)
Called every update tick after normal update.
void Reset()
Called when game is reset.
void OnGui(float afTimeStep)
When ImGui_* functions should be called.
void OnDraw(float afFrameTime)
Called before 2D graphics are drawn.
void OnRenderSolid(cRendererCallbackFunctions@ apFunctions)
When the debug “draw entity info” is on, this will be called and can rendering debug geometry.
float DrawDebugOutput(cGuiSet @apSet,iFontData @apFont,float afStartY)
This only used for pure debug purposes. Use cLux_DrawDebugText(...) for easily outputting messages. “Show Map Info” in the F1 menu must be turned on for it to show up. ALso, if you want the text to be projected on the character, then use iLuxEntity::DrawProjDebugText(...)
void OnPostRender(float afFrameTime)
Called when a frame is rendered..
void LoadUserConfig()
When the user config is loaded.
void SaveUserConfig()
When all user content should be saved.
void OnMapEnter(cLuxMap @apMap)
When a map is finished loaded.
void OnMapLeave(cLuxMap @apMap)
When a map is exited.
void CreateWorldEntities(cLuxMap @apMap)
When all the world entities should be created.
void DestroyWorldEntities(cLuxMap @apMap)
Before map is destroyed and all world entities created in class should be destriyed.
void OnEnterContainer(const tString&in asOldContainer)
When the container that the module reside in is entered.
void OnLeaveContainer(const tString&in asNewContainer)
When the container that the module reside in is left.

void OnPlayerDead(int aType, const tString&in asSource)
When the player has died from damage. aSource is the name of the entity that killed the player

void OnExitPressed() When the exit button is pressed..
void OnAction(int alAction, bool abPressed)
When an action is made.
void OnAnalogInput(int alAnalogId, cVector3f &in avAmount)
When an analog action is made.
void AppGotInputFocus()
If the game got input focus.
void AppLostInputFocus()
If the game lost input focus.

Entity

General

`iLuxEntity` is the basic class for pretty much everything interactive in the game like Props, Areas, Agents, etc (see all below). It has a lot of basic functions that can be used and keeps track of all the data of the entity. `iLuxEntity` is never used directly, but you always use any of the classes that inherits from it.

Components

Sometimes the basic data that the entity (or any the classes that inherit from it), is not enough. This can then be extended by the use of components. Components are separate classes without any explicit connections and they communicate with `iLuxEntity` and one another through a message system. For a full list of the available components and more detail information see the Entity Components page.

Message System

The message system is used to send messages inside the entity and use mostly utilized by the Modules. A message is sent a long with a simple data structure (`cLuxEntityMessageData`) where the contents depend on the type of message sent. See the enum `eLuxEntityMessage` for all available messages.

Callbacks

For an extensive list of these, check the Entity Types page.