Overview

The CryScriptSystem abstracts a LUA virtual machine for usage to the other systems and to the game code.

The functionality provided includes:

- Calling script functions.
- Exposing C++ based variables and functions to the scripts.
- Creating script tables stored in the virtual machine memory.

CryScriptSystem is based on Lua 5. More information on the Lua language can be found at http://www.lua.org.

Accessing Script Tables and Calling Script Functions from C++

A global script table can be retrieved by calling IScriptSystem::GetGlobalValue(). The IScriptTable is used to represent all script tables/variables.

Exposing C++ Functions and Values to the Scripts

In order to expose C++ functions and variable to the scripts, it's necessary to implement a new class. The easiest way to do this is by deriving the CScriptableBase class which provides most of the functionality.

Exposing constants to the scripts

To expose any constant values to the scripts, the IScriptSystem::SetGlobalValue() should be used. For example, to expose a constant named MTL_LAYER_FROZEN to our scripts, use the following code:

```
gEnv->pScriptSystem->SetGlobalValue("MTL_LAYER_FROZEN", MTL_LAYER_FROZEN);
```

Exposing functions to the scripts

To expose C++ functions to the scripts, it's recommended that you write a new class which derives from the CScriptableBase.

```
class CScriptBind_Game :
  public CScriptableBase
{
public:

  CScriptBind_Game( ISystem* pSystem );
  virtual ~CScriptBind_Game() {}

  int GameLog(IFunctionHandler* pH, char* pText);

};
```

Inside the constructor of this class, the following code should be present:

```
  Init(pSystem->GetIScriptSystem(), pSystem);
  SetGlobalName("Game");

  #undef SCRIPT_REG_CLASSNAME
  #define SCRIPT_REG_CLASSNAME &CScriptBind_Game::

  SCRIPT_REG_TEMPLFUNC(GameLog, "text");
```

The following example shows how this new ScriptBind function will be accessible to a Lua scripts.

```
Game.GameLog("a message");
```