Creating a basic extension

We can create a very simple extension that translates the native Phong Maya shader to a standard Arnold shader since it is not currently supported by MtoA and has no any translator associated with it.
First, we need to create a new class that inherits from the MtoA translator we need. As this is a simple example, we begin inheriting from the most simple CNodeTranslator:

translator1.h

```
#include "translators/shader/ShaderTranslator.h"


class CTestTranslator
    : public CShaderTranslator
{
public:
    AtNode* CreateArnoldNodes();
    virtual void Export(AtNode*);
    static void* creator();
};
```

And we override the methods in a very simple way:

creator

> Just returns an instance of the defined class.

CreateArnoldNodes

> Creates the Arnold node that will be exported. We create an Arnold standard here as we will translate the Maya Phong into an Arnold standard.

Export

We export all the attributes are required for the render session. We will export here the "diffuse" Phong attribute to the standard_surface "base" attribute and the "color" attribute to the standard_surface "base_color".

translator1.cpp

```
#include "translator1.h"

#include <ai_msg.h>
#include <ai_nodes.h>


void CTestTranslator::Export(AtNode* shader)
{
    AiMsgInfo("[test extension] Exporting %s",
GetMayaNodeName().asChar());
    AiNodeSetFlt(shader, "base", FindMayaPlug("diffuse").
asFloat() );
    AiNodeSetRGB(shader, "base_color",
                 FindMayaPlug("colorR").asFloat(),
                 FindMayaPlug("colorG").asFloat(),
                 FindMayaPlug("colorB").asFloat());
}

AtNode* CTestTranslator::CreateArnoldNodes()
{
    return AddArnoldNode("standard_surface");
}

void* CTestTranslator::creator()
{
    return new CTestTranslator();
}
```

Note that this is only an example and does not correctly export the Phong shader to a similar standard_surface shader.

Lastly, we need the code that will register the translator for the Maya Phong shader:

extension1.cpp

```
#include "translator1.h"
#include "extension/Extension.h"


extern "C"
{
    DLLEXPORT void initializeExtension(CExtension
&plugin)
    {
        plugin.RegisterTranslator("phong",
                                  "",
                                  CTestTranslator::
creator);
    }
    DLLEXPORT void deinitializeExtension(CExtension
&plugin)
    {
    }
}
```

To compile this, set these environment variables: `ARNOLD_PATH`, `MAYA_PATH` and `MTOA_PATH`:

```
set ARNOLD_PATH="C:\solidangle\releases\Arnold-X.X.X.X-
platform"
set MTOA_PATH="C:\solidangle\mtoadeploy\20XX"
set MAYA_PATH="C:\Program Files\Autodesk\Maya20XX"
```

To compile the extension, do the following depending on your OS:

Windows

Open a Visual Studio command prompt and execute the following commands:

```
cl /c translator1.cpp /EHsc /MD /DNT_PLUGIN
/DREQUIRE_IOSTREAM /I%MAYA_PATH%\include /I%MTOA_PATH%
\include /I%ARNOLD_PATH%\include

cl /c extension1.cpp /EHsc /MD /DNT_PLUGIN
/DREQUIRE_IOSTREAM /I%MAYA_PATH%\include /I%MTOA_PATH%
\include /I%ARNOLD_PATH%\include

link /dll extension1.obj translator1.obj /LIBPATH:%
ARNOLD_PATH%\lib /LIBPATH:%MAYA_PATH%\lib /LIBPATH:%
MTOA_PATH%\lib ai.lib OpenGl32.lib glu32.lib Foundation.
lib OpenMaya.lib OpenMayaRender.lib OpenMayaUI.lib
OpenMayaAnim.lib OpenMayaFX.lib mtoa_api.lib
```

Linux

```
g++ -o translator1.os -c -fvisibility=hidden -Wno-
reorder -std=c++11 -fvisibility=hidden -Wno-reorder -
std=c++11 -Wall -Wsign-compare -g -fno-omit-frame-
pointer -fvisibility=hidden -std=c++0x -fPIC -D_LINUX -
D_BOOL -DREQUIRE_IOSTREAM -DLINUX -I${ARNOLD_PATH}
/include -I${MTOA_PATH}/include -I${MAYA_PATH}/include
translator1.cpp

g++ -o extension1.os -c -fvisibility=hidden -Wno-reorder
-std=c++11 -fvisibility=hidden -Wno-reorder -std=c++11 -
Wall -Wsign-compare -g -fno-omit-frame-pointer -
fvisibility=hidden -std=c++0x -fPIC -D_LINUX -D_BOOL -
DREQUIRE_IOSTREAM -DLINUX -I${ARNOLD_PATH}/include -
I${MTOA_PATH}/include -I${MAYA_PATH}/include extension1.
cpp

g++ -o extension1.so -fvisibility=hidden -z origin -g -
shared build/translator1.os build/extension1.os -
L${MTOA_PATH}/bin -L${ARNOLD_PATH}/bin -L${MAYA_PATH}
/lib -lai -lpthread -lFoundation -lOpenMaya -
lOpenMayaRender -lOpenMayaUI -lOpenMayaAnim -lOpenMayaFX
-lmtoa_api
```

After this, we need to make the generated `extension1.dll/.so/dylib` available to MtoA you can do this one of two ways:

- `copy the library to MTOA_PATH/extensions`
- `set the MTOA_EXTENSIONS_PATH to the loacation of the extension library, this would be the preferred option.`

Now, if we open Maya and assign any object a Phong shader, it will be rendered in an IPR session similar to a lambert. If we change the `color or diffuse` Phong attribute, it will update in the IPR.

If we export the scene, we will see that the Phong Maya shader has been exported like this:

```
...
standard_surface
{
        name phong1
        base 1.0
        base_color 0.8 0.8 0.8
}
...
```

From here, you can make the Export method as complex as you need it.