Register a Translator for a Custom C4D Node

Let's assume you have your own C4D node (for example an object) which you want to translate to an Arnold node. If built-in Arnold nodes are not satisfactory then you have to implement your own Arnold node, but that's a different topic. So in this example let's translate the *MyObject* object to a *polyme sh* Arnold node.

1. Create a custom translator inherited from **AbstractTranslator**. You have to define a type string of the translator which should be the class name. You have to implement a constructor and all abstract methods.

```
#define MY_TRANSLATOR "MyTranslator"

class MyTranslator : public AbstractTranslator
{
   // constructor: pass the translator type string to the parent class
   MyTranslator(BaseList2D* node, RenderContext* context) : AbstractTranslator(MY_TRANSLATOR, node,
context)
   {
   }

   // Name of the Arnold node entry we want to export to.
   // Arnold node is created by the framework, override CreateArnoldNode() if needed.
   // The Arnold node name will be the name of the C4D node by default but you can use custom name
   // by overriding GetAiNodeName().
   virtual char* GetAiNodeEntryName()
   {
      return "polymesh";
   }

   // Called before the export starts to allocate array type Arnold parameters for motion blur.
   virtual void InitSteps(int nsteps)
   {
      // we don't have such parameters in the standard node
   }

   // Translation logic. Step is the motion blur step we are exporting at.
   virtual void Export(int step)
   {
      // m_aiNode is our Arnold node
      // GetC4DNode(true) returns the C4D node we are exporting
      BaseObject* object = (BaseObject*)GetC4DNode(true);
      BaseContainer* data = object->GetDataInstance();

      // export motion blur independent values in the first motion blur step
      if (step == 0)
      {
         Int32 myvalue = data->GetInt32(MY_PARAM);
         AiNodeSetBool(m_aiNode, "self_shadows", myvalue > 10);
      }
   }
};
```

2. Register the translator. Add a function to the main module of your plugin and handle the MSG_C4DTOA_EXTENSION_START message.

```
#include "c4dtoa_api.h"

// Implement your function to create custom translators.
AbstractTranslator* createTranslator(BaseList2D* node, RenderContext* context)
{
   if (node->GetType() == MyObject)
   {
      // C4DtoA will take ownership of the translator instance
      return new MyTranslator(node, context);
   }

   return 0;
}

Bool PluginMessage(Int32 id, void *data)
{
   switch (id)
   {
      case MSG_C4DTOA_MODULE_START:
      {
         const char* myModuleName = "mymodule";

         // C4DtoA module manager and API version number is passed as the argument
         ModuleManagerData* moduleManagerData = (ModuleManagerData*)data;
         ModuleManager* moduleManager = moduleManagerData->moduleManager;

         // check compatibility
         if (moduleManagerData->apiVersion != C4DTOA_API_VERSION)
         {
            GePrintF("[c4dtoa] %s module is incompatible (API version is %d, current is %d)",
               myModuleName, moduleManagerData->apiVersion, C4DTOA_API_VERSION);
            return TRUE;
         }

         // create a new module instance
         Module* module = new Module(myModuleName);

         // register your custom translator creator function
         module->RegisterTranslatorCreator(createTranslator);

         // register the module by the manager
         moduleManager->RegisterModule(module);

         break;
      }
   }

   return FALSE;
}
```

That's it! Your translator will work with the normal and IPR renderer.