3Delight Extension Attributes in Custom Nodes

3Delight for Maya defines groups of extension attributes that can be added to any user-defined node type. The grouping follows the types of *Maya* built-in nodes for which *3Delight for Maya* adds some extension attributes. Once the attributes are added to a user-defined node type, that node's *Attribute Editor* template will automatically display the same group of gadgets as what is defined for Maya built-in nodes.

To list the available attribute groups, use this command:

```
delightExtensionAttr -listGroups;
```

To add groups of extension attributes to a node type:

```
// Adds the common light extension attributes
delightExtensionAttr -nodeType "myCustomAreaLight" "light";

// Adds the arealight-specific extension attributes
delightExtensionAttr -nodeType "myCustomAreaLight" "areaLight";
```

Adding the extension attributes at the right moment

One issue with custom shapes using *3Delight Extension Attributes* is that the custom shape is defined in a plug-in, and it requires extension attributes defined in another plug-in, *3Delight for Maya*. This can cause issues when opening scenes - if the custom shape plugin is loaded and initialized before *3Delight for Maya* is loaded, it will not be able to define the extension attributes on the custom shapes, and *Maya* will throw errors for each scene file statement setting a custom shape's extension attribute value.

One way to avoid this problem is to define a callback on the MSceneMessage::kAfterPluginLoad event. The callback calls a simple MEL procedure that adds the extension attributes on the node, if *3Delight for Maya* is already loaded. Here is how to do it:

1. In your plug-in, define the callback function:

```
static void afterPluginLoadCallback(const MStringArray& i_strings, void* i_data)
{
        // This will be called once per plug-in loaded by the scene file, so
        // avoid executing the command again after it successfully added the
        // extension attributes.
        //
    static int attributesAdded = 0;
    if(attributesAdded == 0)
    {
        MString cmd("addExtensionAttributes;");
        MGlobal::executeCommand(cmd, attributesAdded);
    }
}
```

2. Create a "addExtensionAttributes.mel" file, and define the MEL procedure that adds the extension attributes:

```
// Adds 3Delight extension attributes on "myCustomAreaLight".
// Returns 1 if the 3Delight Extension Attributes were added successfully
//
global proc int addExtensionAttributes()
{
        if(exists("delightExtensionAttr"))
        {
                delightExtensionAttr -nodeType "apiMesh" -addGroup "displacementShader";
            return 1;
        }

        return 0;
}
```

3. Register the callback somewhere in your plug-in's initialize() function:

```
MStatus status;
MSceneMessage sceneMessage;
sceneMessage.addStringArrayCallback(MSceneMessage::kAfterPluginLoad, afterPluginLoadCallback, NULL,
&status);
```

Maya appears to execute the MSceneMessage::kAfterPluginLoad callback after the initialize() function is ran, so the above example will make Maya call afterPluginLoadCallback() once right after your plug-in is initialized, and once after each plug-in required by the scene is loaded. You are thus sure to get afterPluginLoadCallback() called at some point after *3Delight for Maya* has been loaded.

It is not recommended to use the procedure passed to MFnPlugin::registerUI() to add extension attributes as it is ran only once, and it is not guaranteed that at the time it is ran, *3Delight for Maya* has been loaded.