Autodesk Maya API | White Paper

This white paper introduces Autodesk® Maya® software and its architecture to programmers who may be unfamiliar with its APIs (application programming interfaces).

Contents

CONTENTS **1**

INTRODUCTION **3**

AN OVERVIEW OF THE MAYA ARCHITECTURE **3**

MAYA API AND MAYA PYTHON API **4**

PLUG-INS **5**

Loading and Unloading 5

The simplest Maya Plug-in 6

A more complex example 8

PLUG-IN INTEGRATION WITH MAYA **11**

Types of Plug-ins 11

Plug-in access to Maya scene data 12

IMPLEMENTATION OF THE MAYA API **12**

Objects and Function Sets 12

Objects 13

Function Sets 13

Transient Objects 14

Math Classes 15

Proxy Objects 15

UI 15

Access to the Dependency Graph 15

A Note on the API and MEL 16

MAYA API DOCUMENTATION AND RESOURCES 16

CONCLUSIONS 17

APPENDIX A: MAYA API CLASSES 17

APPEDIX B: MAYA API CLASS HIERARCHY 25

APPENDIX C: SELECTED LIST OF SAMPLE PLUG-INS 29

Introduction

The Maya® Software Development Kit (SDK) contains a rich set of facilities that can help you add new functionality to the Maya software. The SDK includes a C++ API called the Maya API that provides functionality for querying and changing the Maya model along with the ability to add new Maya objects to the Maya model. In addition, the SDK contains Python® bindings to the Maya API. These bindings allow a developer to call the Maya API from Python. We refer to this as the Maya Python API. Using either API, you can add new elements to Maya such as: commands(accessible from MEL [Maya Embedded Language] or Python script), file translators, shaders, animation nodes, 3D graphical manipulators, geometry shapes, types of deformations, dynamic fields, particle emitters, inverse-kinematic solvers and any type of custom node.

These functionalities are, for the most part, indistinguishable from native Maya features. Functionality implemented in the C++ Maya API will perform almost as fast as a native Maya feature. While they are powerful, the Maya APIs are not very complicated to learn. Any programmer should be able to quickly write their first simple plug-in using either API.

Examples of plug-ins that have been built using the C++ Maya API are:

- Maya Classic Cloth
- Maya Live
- Maya Fur
- MTOR from Pixar (Maya to Renderman)
- Arete Software's Digital Nature Tools

Note: Throughout this document, "Windows®" is used to refer to the Windows operating systems that we support for the release. Likewise, “Macintosh®” refers to Mac OS® X.

An Overview of the Maya Architecture

Maya is designed to be an open, flexible and extensible product.

The APIs of Maya and its command scripting languages (MEL and Python) are key to the software being open and extensible. During the initial design of Maya, Maya and its C++ API were built together in order to maximize its extensibility. To fully appreciate this relationship, you must first understand the architecture of Maya.

At the lowest level, Maya is a very efficient database for storing graphical information. This database is called the Dependency Graph (DG). Information in the DG is stored in objects called nodes. Nodes have properties called attributes that store the configurable characteristics of each node. Similar types of attributes can be connected together, letting data flow from one node to another.

For example, a nurbsCurve node has an attribute that contains a NURBS curve. This attribute can be connected to the input curve attribute of a revolve node. The revolve node also has input attributes that describe the sweep angle and the axis that it would revolve around. The output attribute of the revolve node is a NURBS surface. You can connect this attribute to the input of a nurbsSurface node that knows how to draw a surface. You can then make the nurbsSurface node a child of a transform node that knows how to position objects in 3D space.

This data flow is, in fact, a mechanism for implementing construction history. If you change one of the inputs to any node that participates in the construction of the revolved surface, the

DG recomputes only the parts of the history that are affected in order to update the surface. The DG keeps a clear record of what affects what, so that it only recomputes the graph as necessary.

There are over 600 built-in nodes shipped as part of the Maya system. The Maya Help comes with documentation called the Node and Attribute Reference that describes each of these nodes and their attributes.

Working directly at the DG level gives you the maximum amount of flexibility and power, but it is limiting. Fortunately, you are not required to work directly at this level because Maya also comes with over 900 commands that can do this for you. These commands create DG nodes, set and connect their attributes, and create the transform nodes that position objects. For example, there is a revolve command that takes sweep angles and an axis as arguments, and builds the revolve network described above from a selected NURBS curve. This command handles all the low level details for you. The Maya Help comes with documentation called the MEL Command Reference that describes all the built-in Maya commands and their arguments.

About 200 of the Maya commands are UI creation commands that let you build windows, menus, buttons, icons, and so on. Maya itself contains a large number of scripts that it uses to build its UI. In addition to the revolve node and the revolve command, there is a menu item called "Revolve", and a revolve icon on the UI shelf. The menu item or icon executes the command that in turn creates the DG network. Scripting is used to implement approximately 98% of the UI of Maya.

At a high level (for example, for technical directors), Maya provides an extremely customizable environment. MEL and Python are very powerful scripting languages that can be used to expand an artist's palette of tools. Both are also very powerful UI creation languages. For example, you can create a specific UI for each character being animated in your scene and this can significantly increase your productivity.

Finally, if the script you want to write requires a Maya command that does not exist, or if your script would be much simpler to write if a node with a certain set of characteristics existed, you can add such commands and nodes to Maya using the API.

Maya API and Maya Python API

The APIs of Maya closely mirror the underlying architecture of Maya so that the scene graph and DG, which manage the way data is processed, are visible through the Maya API in a manner consistent with their underlying implementation.

The Maya API is a C++ interface with which developers can write plug-in shared objects that may be loaded into Maya at runtime. When they are no longer needed, plug-ins can be unloaded. As a result, you can efficiently develop a plug-in by compiling it, loading it, testing it, unloading it, changing its source code, recompiling and reloading. The same can be done for plug-ins that have been implemented using the Maya Python API.

Plug-ins (coded in C++) developed for Maya work in tandem with standard debugging environments on Linux® and Windows operating systems. On Linux, you can launch Maya with the `-d` flag and it will start under the control of the debugger. You can also set breakpoints in your plug-in and get stack traces as you would with any other Unix® application. On Windows, you can launch Maya directly from the Microsoft® Visual Studio® development system and debug the plug-in as you would any other Windows application.

The APIs of Maya provide access to a significant portion of the functionality of Maya. This includes, but is not limited to, the ability to query and modify existing data such as geometry, transforms, hierarchies, scene graph, and DG nodes. Also, these APIs let you create new objects such as shader types available to the renderer, and let you draw in OpenGL®. The OpenMaya API also includes a class that permits the customization of a shape's hardware

rendered appearance so that tools can be developed to let artists see how their work will appear in a real time environment. Plug-in features can be added and removed at any time using scripting language commands. Plug-ins can operate in either of the two modes of operation of Maya: interactive or batch mode. The Maya API is available in a standalone form for writing separate standalone applications. These applications can read and write the ASCII and binary scene files of Maya and perform operations on the data therein.

As described in the previous section, you normally add a new command or node (or both) to Maya using one of the APIs and then create a script that provides a UI for the new feature. For example, you can use one of the APIs to create a new type of node, then write a new command that creates an instance of the node and places it in the scene and connects it to the appropriate nodes. Finally, you can write a MEL or Python script that inserts the command in a menu so that you can access it.

The Maya API and the Maya Python API are designed to be platform neutral. In most cases, no source code changes are required to "port" a plug-in between Linux, Windows or Mac OS X. Normally, platform specific code is only required when creating or manipulating windows without using the platform independent UI creation abilities of scripting, or when using a platform specific 3rd party library.

Both the Maya API and the Maya Python API provide source code compatibility. Therefore, C++ plug-ins written for earlier Maya releases can recompile without any source code changes in the current version of Maya. An existing Python plug-in should import and run correctly without source code changes if source code compatibility is maintained. If source code incompatibility occurs, the documentation for the new release will contain detailed instructions on what changes need to be made to the plug-ins.

The APIs of Maya use objects and function sets to access internal objects of Maya. Objects are very lightweight classes that provide RTTI (Run Time Type Identification) and can be used in a type-less manner. A function set is a user owned structure that allows operations on Maya owned objects. Since an object is simply a handle that knows its type, a function set provides you with a tool that acts upon objects of the right type. These function sets usually have names that match the Maya features, such as `MFnSkinCluster` and `MFnNurbsSurface`. For example, the `MFnSkinCluster` class can be instantiated with an object of type `kSkinCluster`, and subsequently can be used to perform operations on that object. The object itself is not able to perform any functions without the aid of its function set.

Proxies let you develop new types of objects. Proxy object classes let you create functionality that integrates into Maya as first-class citizens. Examples of proxies are: commands, file translators, and new types of shaders.

The APIs of Maya are very flexible. This flexibility lets you accomplish a particular task in a number of ways, and lets you determine where to make the tradeoff between speed of development and the performance of the plug-in. With either API, if the performance of the plug-in is not critical, then it is possible to quickly prototype a solution to a particular problem. If performance is critical, the C++ Maya API should be utilized.

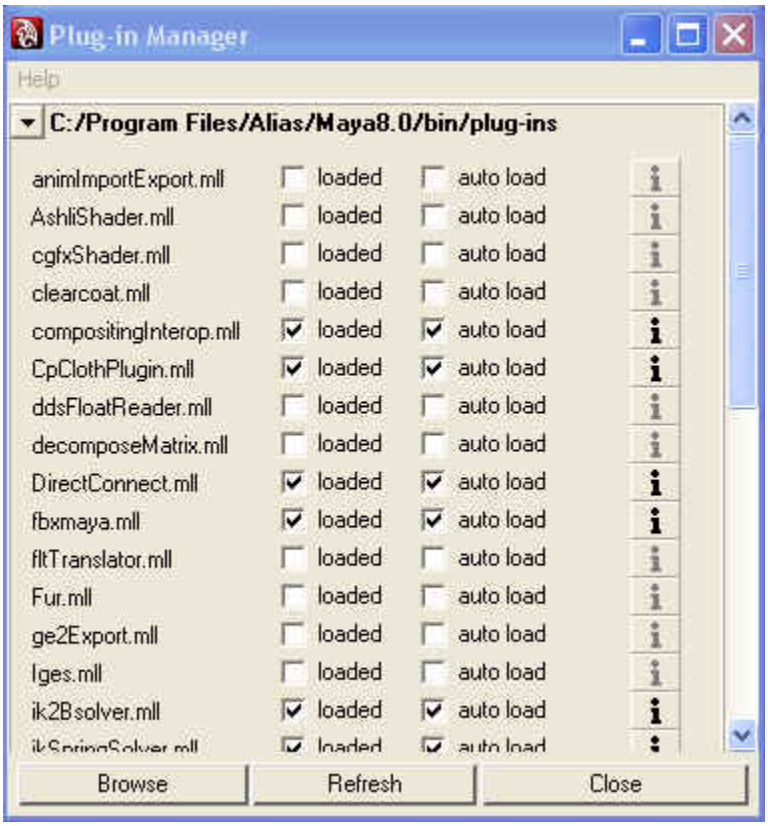
Plug-ins

This section describes some simple Maya plug-ins and how they can be used.

Loading and Unloading

Plug-ins can either be binary if implemented in the C++ Maya API or scripted if the Maya Python API is used. Either type of plug-in can be loaded and unloaded using the Plug-in

Manager Window or by using script (MEL or Python). The Plug-in Manager provides a quick and easy way to load and unload both C++ and Python implemented plug-ins.



Plug-in Manager Window

If a plug-in has been loaded, information about the item can be found by clicking the "i" button. For developers who wish to write scripts for loading and unloading plug-ins, the following MEL commands are available:

```
loadPlugin "name"
```

The "name" parameter identifies the plug-in file. For example, the command:

```
loadPlugin feature;
```

loads the plug-in named "feature". The environment variable MAYA_PLUG_IN_PATH is searched for a file named "feature.so" on Linux, "feature.mll" on Windows, and "feature.dynlib" on Mac OS X. The command could also be used to load `feature.py` on any of our supported platforms.

Plug-ins are unloaded with the MEL command:

```
unloadPlugin "name"
```

The parameter "name" is the name of the plug-in. Equivalent commands are available in Python.

The simplest Maya Plug-in

When learning a new computer language, the first program you are likely to see is a "Hello World" program. The following is the plug-in code required for outputting "Hello World" to the script editor window in Maya.

```
#include <maya/MSimple.h>
#include <maya/MGlobal.h>
```

```
DeclareSimpleCommand( helloWorld, "Autodesk", "8.5");
MStatus helloWorld::doIt( const MArgList& )
{
    MGlobal::displayInfo("Hello World\n");
    return MS::kSuccess;
}
```

The Maya Python API version is as follows:

```
import sys
import maya.OpenMaya as OpenMaya
import maya.OpenMayaMPx as OpenMayaMPx

kPluginCmdName = "spHelloWorld"

# command
class scriptedCommand(OpenMayaMPx.MPxCommand):
     def __init__(self):
            OpenMayaMPx.MPxCommand.__init__(self)
     def doIt(self,argList):
            print "Hello World!"

# Creator
def cmdCreator():
     return OpenMayaMPx.asMPxPtr( scriptedCommand() )

# Initialize the script plug-in
def initializePlugin(mobject):
     mplugin = OpenMayaMPx.MFnPlugin(mobject)
     try:
            mplugin.registerCommand( kPluginCmdName, cmdCreator )
     except:
            sys.stderr.write( "Failed to register command: %s\n" %
kPluginCmdName )
            raise

# Uninitialize the script plug-in
def uninitializePlugin(mobject):
     mplugin = OpenMayaMPx.MFnPlugin(mobject)
     try:
            mplugin.deregisterCommand( kPluginCmdName )
     except:
            sys.stderr.write( "Failed to unregister command: %s\n" %
kPluginCmdName )
            raise
```

This example defines a new command “spHelloWorld”. We use the “sp” prefix since this is a scripted plug-in.

Once the plug-in is compiled and/or accessible through MAYA_PLUG_IN_PATH, type the following MEL commands:

```
loadPlugin "helloWorld";
helloWorld;
unloadPlugin "helloWorld";
```

Either binary or scripted plug-ins can be loaded and unloaded with the commands above.

A more complex example

As outlined in the `helloWorld` example, it is quite easy to implement the standard `helloWorld` plug-in in Maya. The `helixCmd` plug-in listed below is a more complex example of a Maya plug-in.

```
#include <math.h>
#include <maya/MSimple.h>
#include <maya/MFnNurbsCurve.h>
#include <maya/MPointArray.h>
#include <maya/MDoubleArray.h>
#include <maya/MPoint.h>
#include <maya/MIOStream.h>

DeclareSimpleCommand( helix, "Autodesk", "8.5");

MStatus helix::doIt( const MArgList& args )
{
     MStatus stat;
     const unsigned deg = 3;              // Curve Degree
     const unsigned ncvs = 20;     // Number of CVs
     const unsigned spans = ncvs - deg;   // Number of spans
     const unsigned nknots= spans+2*deg-1;       // Number of knots
     double  radius = 4.0;  // Helix radius
     double  pitch = 0.5;   // Helix pitch
     unsigned       i;

     // Parse the arguments.
     for ( i = 0; i < args.length(); i++ )
            if ( MString( "-p" ) == args.asString( i, &stat )
                          && MS::kSuccess == stat)
            {
                   double tmp = args.asDouble( ++i, &stat );
                   if ( MS::kSuccess == stat )
                          pitch = tmp;
            }
            else if ( MString( "-r" ) == args.asString( i, &stat )
                          && MS::kSuccess == stat)
            {
                   double tmp = args.asDouble( ++i, &stat );
                   if ( MS::kSuccess == stat )
                          radius = tmp;
            }

     MPointArray controlVertices;
     MDoubleArray knotSequences;

     // Set up cvs and knots for the helix
     for (i = 0; i < ncvs; i++)
            controlVertices.append( MPoint( radius * cos( (double)i ),
                   pitch * (double)i, radius * sin( (double)i ) ) );

     for (i = 0; i < nknots; i++)
            knotSequences.append( (double)i );

     // Now create the curve
     MFnNurbsCurve curveFn;
     MObject curve =
```

```
                curveFn.create( controlVertices, knotSequences, deg,
                                MFnNurbsCurve::kOpen, false, false,
                                MObject::kNullObj, &stat );

        if ( MS::kSuccess != stat )
                cout << "Error creating curve.\n";

        return stat;
}
```

This example illustrates how to use the API to parse arguments passed to a command, how to use array classes, how to create a curve and how to check the return status. These operations are standard tasks for plug-in development.

A similar example using the Maya Python API follows:

```
import maya.OpenMaya as OpenMaya
import maya.OpenMayaMPx as OpenMayaMPx
import sys, math

kPluginCmdName="spHelix"

kPitchFlag = "-p"
kPitchLongFlag = "-pitch"
kRadiusFlag = "-r"
kRadiusLongFlag = "-radius"

# command
class scriptedCommand(OpenMayaMPx.MPxCommand):
     def __init__(self):
            OpenMayaMPx.MPxCommand.__init__(self)

     def doIt(self, args):
            deg = 3
            ncvs = 20
            spans = ncvs - deg
            nknots = spans+2*deg-1
            radius = 4.0
            pitch = 0.5

            # Parse the arguments.
            argData = OpenMaya.MArgDatabase(self.syntax(), args)
            if argData.isFlagSet(kPitchFlag):
                   pitch = argData.flagArgumentDouble(kPitchFlag, 0)
            if argData.isFlagSet(kRadiusFlag):
                   radius = argData.flagArgumentDouble(kRadiusFlag, 0)

            controlVertices = OpenMaya.MPointArray()
            knotSequences = OpenMaya.MDoubleArray()

            # Set up cvs and knots for the helix
            #
            for i in range(0, ncvs):
                   controlVertices.append(
                          OpenMaya.MPoint( radius * math.cos(i),
                          pitch * i, radius * math.sin(i) ) )
```

```
                for i in range(0, nknots):
                        knotSequences.append( i )

                # Now create the curve
                #
                curveFn = OpenMaya.MFnNurbsCurve()

                nullObj = OpenMaya.MObject()

                try:
                        curveFn.create( controlVertices,
                                        knotSequences, deg,
                                        OpenMaya.MFnNurbsCurve.kOpen,
                                        0, 0,
                                        nullObj )
                except:
                        sys.stderr.write( "Error creating curve.\n" )
                        raise

# Creator
def cmdCreator():
     # Create the command
     return OpenMayaMPx.asMPxPtr( scriptedCommand() )

# Syntax creator
def syntaxCreator():
     syntax = OpenMaya.MSyntax()
     syntax.addFlag(kPitchFlag, kPitchLongFlag,
OpenMaya.MSyntax.kDouble)
     syntax.addFlag(kRadiusFlag, kRadiusLongFlag,
OpenMaya.MSyntax.kDouble)
     return syntax

# Initialize the script plug-in
def initializePlugin(mobject):
     mplugin = OpenMayaMPx.MFnPlugin(mobject, "Autodesk", "8.5", "Any")
     try:
             mplugin.registerCommand( kPluginCmdName, cmdCreator,
syntaxCreator )
     except:
             sys.stderr.write( "Failed to register command: %s\n" %
kPluginCmdName )
             raise

# Uninitialize the script plug-in
def uninitializePlugin(mobject):
     mplugin = OpenMayaMPx.MFnPlugin(mobject)
     try:
             mplugin.deregisterCommand( kPluginCmdName )
     except:
             sys.stderr.write( "Failed to unregister command: %s\n" %
kPluginCmdName )
             raise
```

Plug-in integration with Maya

Features implemented as plug-ins are, for the most part, virtually indistinguishable from built-in Maya features. You can query a command or node to see if it is implemented by a plug-in, but normally neither Maya nor features implemented by other plug-ins need to be aware of the distinction.

Types of Plug-ins

Through the APIs of Maya, plug-ins can implement the following: new types of objects, tools with interaction, data file translators, and new scripting language commands. For new commands, plug-in developers can choose to implement undo and redo, batch mode operation, and UI with the same look and feel as the Maya UI. Plug-in commands can be used freely in MEL scripts with Maya built-in commands and other plug-in commands.

In Maya 8.5, plug-ins can implement the following:

- commands(accessible from MEL or Python)
- file translators (e.g. exporters for game engines, or third-party renderers)
- drawing in OpenGL
- inverse-kinematic solvers
- shaders, lights, textures
- hardware shaders
- procedural animation
- simulators (e.g. Maya Classic Cloth was written entirely using the API)
- user-defined deformations
- 3D graphical manipulators
- geometry shapes
- dynamic fields
- particle emitters
- node and plug locking
- monitors that listen to Maya messages
- custom model views
- user defined transformations
- motion capture servers
- manipulators
- locators
- user defined fluids

- swatches for plug-in shaders
- viewport overriding
- any other type of custom node

It is important to note that almost all of the information one would need from a Maya scene is available through the API. Essentially, if the data exists in the DG, there is most likely a way to access it through a command or file translator, or plug into it with a custom node. This opens up a great number of opportunities for modifying Maya to meet specific workflow needs, or for getting exactly the information out of Maya that one needs for a specific application.

Plug-in access to Maya scene data

The APIs of Maya let you access the contents of the scene graph, contents of the selection list, contents of the DG and so on.

Access is provided by means of:

- Maya objects that represent a "node" in a Maya scene file. The methods needed to work with these objects live separately from them in classes called function sets. Creation and deletion of Maya objects is the responsibility of Maya.
- Transient objects that are used as base software tools (iterators, math library classes, selection lists, DAG paths, etc.). You can create and destroy these freely by using the Maya API. Unlike Maya objects that require a function set to operate, these objects contain their own methods.

Implementation of the Maya API

The APIs of Maya are implemented with several types of objects: Maya objects, function sets, transient objects, and proxy objects.

Objects and Function Sets

Most API development can be done through objects and function sets. This approach to handling objects is type-free. This makes for rapid creation of prototypes without sacrificing strong typing when this information is necessary in production code.

Objects are references (pointers) to the internal objects in Maya. They provide a means of referencing the major types in Maya such as DAG nodes, dependency nodes, geometry and so on. A function set is a group of methods that operate on an object.

Below is a brief C++ example that demonstrates the difference between objects and function sets:

```
MObject obj = dagFn.child();
if ( obj.hasFn( MFn::kCamera ) )
{
     MFnCamera cam( obj );
     cam.setHorizontalFieldOfView(cam.horizontalFieldOfView()/2.0);
}
```

In this example, the `MObject` is a reference to a camera node in the scene and `MFnCamera` is a function set that operates on a camera node.

It is also possible to create new instances of objects using a function set. For example, in C++:

```
MFnCamera cam;
MObject camera = cam.create( /* args... */ );
```

This creates a new camera and adds it to the Maya scene.

Objects and function sets let you create, query, and modify objects. When additional types of objects are introduced in future releases, they will also be made available through this mechanism.

Objects

An "object" is a reference to a Maya object. You can access them through the APIs' `MObject` class. The only operations that can be performed directly on a `MObject` are the determination of type, and of the function sets that can operate on this type of MObject. Once the MObject's type has been determined, you can bind the function set for that type, or any base type, to that MObject.

For example, given an MObject called 'obj' one can do the following:

```
if ( obj.apiType() == MFn::kCamera )...
```

which determines whether or not the object is exactly a camera, or

```
if ( obj.hasFn(MFn::kCamera) )...
```

which determines if the object is compatible with the camera function set.

Function Sets

A function set is a class that operates on an object. It contains the set of methods necessary to query, modify, and create a particular type of object.

Any instance of a `MObject` can have one and only one Maya Object type. All Function Sets have one and only one Function Set type. But an `MObject` can, however, be compatible with many types of Function Sets. This compatibility follows the class hierarchy of the Function Sets. That is why an `MObject` with the Maya Object type `MFn::kNurbsSurface` is compatible with `MFnNurbsSurface`, `MFnDagNode`, `MFnDependencyNode`, and `MFnBase`.

One could do the following:

```
if ( obj.hasFn(MFn::kCamera) )
{
// Create a camera function set.
MFnCamera camera( obj );
// Determine the fStop of the camera object.
double fStop;
camera.fStop(fStop);
...
}
```

Note: Function sets accept any object regardless of whether they can operate on it. In the above code, the camera function set could have been created using a surface object instead. In this case, the function set would be put into an invalid state and all methods would return error values but would not cause a fatal error.

This makes it very easy to prototype code. For example:

```
// Walk through a list of objects, and get the fStop of each
// camera. Note that if something other than a camera is on
// the list, this code will still work.
double fStop;
MFnCamera camera;
MObjectArray objectArray = <...get a list ...>;
for ( int i = 0; i< objectArray.length(); i++)
{
     if (camera.setObject(objectArray[i]) == MS::kSuccess)
     {
            camera.fStop(fStop);
            ...
     }
}
```

The above example demonstrates that function sets let you work in an entirely type-less manner. Doing so reduces the efficiency of a plug-in by forcing additional unnecessary operations to be performed (each method of the function will try and then fail to operate on the invalid object).

It is also possible to program in an entirely typed manner. For example:

```
// Walk through a list of objects, and get the fStop of each
// camera. Note that if something other than a camera is on
// the list, this code will skip it.
double fStop;
MFnCamera camera;
MObjectArray objectArray = <...get a list ...>;
for ( int i = 0; i< objectArray.length(); i++)
{
     if (objectArray[i].hasFn(MFn::kCamera))
     {
            camera.setObject(objectArray[i]);
            camera.fStop(fStop);
            ...
     }
}
```

This code is more efficient than the previous one because the camera methods are only applied to the cameras in the list. Efficiency can also be achieved in the form of filtering when using the iterator classes. Iterators are prefixed with `MIt`. These classes are designed to loop through objects of a given type and further filtering is available to select among those objects. For example, you can use an iterator to process only 2D textures.

Transient Objects

The Maya API contains several types of basic objects that do not represent nodes in a Maya scene. These objects include the math objects for vectors and matrices, strings, DAG paths (Maya instancing support), selection lists, command argument lists. These objects are quite small and are transient in nature (they are not stored in a data file).

These objects do not require a function set class to operate upon them. Instead, they are directly accessible and provide all the methods necessary to operate on the object without the need for an additional function set class.

Math Classes

The Maya API supplies a set of math classes. These classes include, for example, `MMatrix`, `MVector`, `MPoint`, `MQuaternion`, and `MEulerRotation`. The math classes of the Maya API implement many operator methods. For example, adding two vectors can be coded as:

```
MVector v( 1, 0, 0 );
MVector w( 0, 1, 1 );
MVector x = v + w;
```

If a developer provides a conversion routine from their math class to a Maya math class, the Maya API can automatically work with their class. This allows developers to pass their math objects to Maya API methods.

Proxy Objects

Proxy objects are objects that the programmer defines and that Maya knows how to operate on. Proxy objects include commands, file translators, manipulators, contexts, dependency graph nodes, surface shapes, fluid emitters and modeling editors. For example, proxy DG nodes let you create new operations (such as a new revolve), but also allow for new types of objects such as shaders, fields, particle emitters, and geometry shapes. The APIs provide classes from which you can derive your own classes. These developer-defined classes are registered with Maya, allowing users to create instances of them. However, since these objects typically have a UI component to them (for example, they have UI to create them and to manipulate their values), you will have to provide a UI for these objects.

An example of a proxy object is a new kind of surface creation node. This user-defined node derives from the proxy DG node class, and uses code written by the developer to define the input and output attributes of the node. The developer would also provide the contents of a virtual method that is then called to provide the value of the output attribute based on its inputs. In this case, the node could access several input attributes and use this information to determine what the output NURBS surface will look like. A proxy node that creates a simple NURBS surface can easily be written with less than 100 lines of source code. Once the node itself has been written, you can write MEL or Python scripts to create the UI that will let users easily create an instance of the node and modify the attributes of existing instances.

Note: Proxy nodes are stored by Maya and must be differentiated from one another. For example, you may implement two different plug-in shader nodes and want to use/store both in a scene. To handle this differentiation, a node ID is used. Customers on a support contract can send requests to our Support team to receive a block of node IDs for plug-in development.

UI

Using MEL or Python, you can create UI components for plug-in commands using the same process that is used to create the UI for built-in features. MEL or Python also provide access to drawing methods and widgets that let you create very complex UI objects.

Finally, since scripting is platform-neutral, the UI only needs to be specified in MEL or Python once and it will work correctly on the Linux, Windows, and Mac OS X platforms.

Access to the Dependency Graph

The DG is the heart of Maya. While performing typical modeling, animation, and rendering tasks, most users do not need to be aware of the underlying implementation of Maya.

However, when working in Maya APIs, you may want to exploit the power of the DG to create optimized and robust tools that integrate well with the overall architecture.

The DG has a dataflow structure that supports animation, construction history, and constraints.

An API programmer can create new nodes that can be inserted into the DG. The Maya APIs provide the necessary methods to both add new nodes and examine nodes that already exist. By providing two levels of access to the DG, the Maya API does not limit the power programmer's abilities and does not overwhelm the casual programmer.

The scene graph provides you with a high level view where you need to know little about the DG but can still affect it. This is similar to the UI view where a user may keyframe an animation without knowing that they are creating or modifying DG nodes. Similarly, developers are able to keyframe animation without needing to know that they are creating and manipulating DG nodes.

A lower level view of Maya is provided by API function sets. Function sets let you view the DG so that you can see a node, the attributes of the node, the node connections, and the data and messages coming in and going out of the node. In this view, the treatment of all types of DG nodes is identical.

Access at either level of the animation system of Maya requires that a scene be evaluated efficiently in a given context, namely the context of a user-interface editing tool or the context of time. Time is represented in a DG node that can trigger events, cause expressions to be evaluated, or enable keyframing on virtually everything in a scene.

A Note on the API and Script

Almost everything that can be accomplished in script(MEL or Python commands) can be done in the C++ API and in many cases is executed much more quickly. This is because a plug-in is a compiled shared binary library while all scripts have to pass through the interpreter that adds a further level of processing. However, the API and script are not mutually exclusive. Often, the amount of time it takes to perform the C++ coding for duplicating a single script method greatly outweighs the performance benefits. In some situations, Maya script commands have broader access to scene data than might be available through an API class implementation. Since Maya script commands can be called from within plug-in code, you can use MEL or Python Maya commands when necessary and use the C++ or Python APIs for everything else.

Maya API Documentation and Resources

The Maya 8.5 API comes with the following documentation:

- Maya API Developers Manual (includes a section of the Maya Python API).
- Maya Motion Capture API Guide.
- API Class Reference documentation (287 classes - each class is documented).
- Source code examples (160 working examples comprising over 100,000 lines of source code)
- Makefiles are provided on Linux and Mac OS X
- Microsoft Developer Studio 2005 IDE solution files on Windows.

- A convenient "Plug-in Wizard" for quickly creating MS Visual Studio 2005 Maya plug-in solutions
- Xcode Project files on Mac OS X

Documentation is in HTML format and can be viewed with any web browser.

Additional resources for learning the Maya API are the following:

1. Maya API Developer Conference Notes (Available from the Autodesk Online Store)
2. Complete Maya Programming, An Extensive Guide to MEL and the C++ API; By David A. D. Gould
3. Complete Maya Programming, An In-depth Guide to 3D Fundamentals, Geometry, and Modeling. (Volume II); By David A. D. Gould

Conclusions

The APIs of Maya are easy to learn and provide an extremely powerful interface that lets you quickly extend Maya in almost any way you want. Many plug-ins have been successfully implemented using the Maya API. With the broad set of examples and documentation that is available, it is easy to get started with either API. Although this document focuses primarily on plug-ins, an alternate type of executable is supported. It is possible for developers to write standalone applications that run on the command line using either the Maya API or Maya Python API. Standalone applications can be used for file translation and batch operations such as checking node naming conventions and so on. Examples of standalone applications are included in our developer kit.

Once you decide that you want to start developing with the Maya API there are several routes that you can take. If you are developing in-house applications, then you need to purchase the Maya software that includes the Maya API and developer kit. If you want to write a commercial Maya plug-in, then consider joining our Autodesk® Sparks® third-party plug-in program. Information can be found at: www.Autodesk.com/sparks

Appendix A: Maya API Classes

This section contains an alphabetical list of the classes that are part of the Maya 8.5 API.

Notes:

1. The main library that contains the bulk of the API code is the OpenMaya library. Unless otherwise indicated, the classes listed are accessible by linking with that library. The other libraries are named and enclosed in parentheses. (Note: Python import library information is not included below.)
2. Class names in bold are considered very important

M3dView	A 3-D view (OpenMayaUI)
MAngle	Manipulate Angular Data
MAnimControl	Control over animation playback and values (OpenMayaAnim)
MAnimCurveChange	Animation Curve Change Cache (OpenMayaAnim)
MAnimCurveClipboard	Control over the animation clipboard (OpenMayaAnim)

MAnimCurveClipboardItem	Wrapper for a clipboard item. (OpenMayaAnim)
MAnimCurveClipboardItemArray	Array of MAnimCurveClipboardItem data type (OpenMayaAnim)
ManimMessage	Animation messages (OpenMayaAnim)
MAnimUtil	Static class providing common animation helper methods (OpenMayaAnim)
MargDatabase	Command argument list parser
MArgList	Create and retrieve argument lists
MArgParser	Command argument list parser
MArrayDataBuilder	Array builder for arrays in data blocks
MArrayDataHandle	Data block handle for array data
MAttributeIndex	The index information for an attribute specification
MAttributeSpec	An attribute specification
MAttributeSpecArray	An attribute specification array
MboundingBox	Implementation of a 3D bounding box
MCallbackIdArray	Container class for callback IDs
MCloth*	A number of classes that allow the replacing of the Maya Cloth solver (Not an OpenMaya class)
MColor	A color math class
MColorArray	Array of MColor data type
MCommandMessage	Listen to messages related to MEL
MCommandResult	Result returned from executing a command
MCommonRenderSettingsData	Container class for common render settings
MCommonSystemUtils	Methods for getting and setting environment variables, make directories etc.
MComputation	Interrupt monitor for long computations
MConditionMessage	Condition change messages
MCursor	Manipulate Cursors (OpenMayaUI)
MDagMessage	Dependency graph messages
MDagModifier	DAG graph modifier
MDagPath	DAG Path
MDagPathArray	Indexable array of DAG paths
MDataBlock	Dependency node data block
MDataHandle	Data handle for information contained in a data block
MDeviceChannel	Input device channel (OpenMayaUI)
MDeviceState	Input device state (OpenMayaUI)
MDGContext	Dependency graph context class
MDGMessage	Dependency graph messages
MDGModifier	Dependency graph modifier
MDistance	Manipulate linear data
MDoubleArray	Array of doubles data type
MDrawData	Draw data used in the draw methods of MPxSurfaceShapeUI (OpenMayaUI)
MDrawInfo	Drawing state used in the draw methods of MPxSurfaceShapeUI (OpenMayaUI)

MDrawProcedureBase	Used to create custom hardware shader drawing effects
MDrawRequest	A draw request used in the draw methods of MPxSurfaceShapeUI (OpenMayaUI)
MDrawRequestQueue	Drawing queue used in MPxSurfaceShapeUI::getDrawRequests (OpenMayaUI)
MDrawTraversal	Utility class for interactive drawing
MDynamicsUtil	Utility methods for working with 2D textures with fluids and particles
MDynSweptLine	Class for evaluating curve segments as lines over time (OpenMayaFX)
MDynSweptTriangle	Class for evaluating surfaces as triangles over time (OpenMayaFX)
MEulerRotation	Euler rotation math
MEvent	System event information (OpenMayaUI)
MEventMessage	Event messages
MFeedbackLine	Feedback line (OpenMayaUI)
MFileIO	I/O operations on scene files
MFileObject	Manipulate Unix filenames and search paths
MFloatArray	Array of floats data type
MFloatMatrix	A matrix math class for 4x4 matrices of floats
MFloatPoint	Implementation of a point
MFloatPointArray	Array of MFloatPoint data type
MFloatVector	A vector math class for vectors of floats
MFloatVectorArray	Array of MFloatVectors data type
MFn	Function set type identifiers
MFnAirField	Function set for air fields (OpenMayaFX)
MFnAmbientLight	Manage ambient light dependency nodes
MFnAnimCurve	Animation curve function set (OpenMayaAnim)
MFnAnisotropy	Manage anisotropy shaders
MFnAreaLight	Manage area light dependency nodes
MFnArrayAttrsData	Function set for multiple arrays of attributes for dependency node data
MFnAttribute	Dependency node attribute function set
MFnBase	Function set base class
MFnBlendShapeDeformer	Blend shape deformer function set (OpenMayaAnim)
MFnBlinnShader	Manage Blinn shaders
MFnCamera	Function set for cameras
MFnCharacter	Function set for characters
MFnCircleSweepManip	Function set for circle sweep manipulator (OpenMayaUI)
MFnClip	Clip function set (OpenMayaAnim)
MFnComponent	Base class for component function sets
MFnComponentListData	Component list function set for dependency node data
MFnCompoundAttribute	Compound attribute function set
MFnCurveSegmentManip	Function set for curve segment manipulator (OpenMayaUI)
MFnDagNode	DAG node function set
MFnData	Parent class for dependency graph data function sets

MFnDependencyNode	Dependency node function set
MFnDirectionalLight	Manage directional light dependency nodes
MFnDirectionManip	Function set for direction manipulator (OpenMayaUI)
MFnDiscManip	Function set for disc manipulator (OpenMayaUI)
MFnDistanceManip	Function set for distance manipulator (OpenMayaUI)
MFnDoubleArrayData	Double array function set for dependency node data
MFnDoubleIndexedComponent	Double indexed component function set
MFnDragField	Function set for drag fields (OpenMayaFX)
MFnDynSweptGeometryData	Swept geometry function set for dependency node data (OpenMayaFX)
MFnEnumAttribute	Enumerated attribute function set
MFnExpression	Expression function set
MFnField	Function set for dynamic fields (OpenMayaFX)
MFnFluid	Function set for fluids(OpenMayaFX)
MFnFreePointTriadManip	Function set for free point triad manipulator (OpenMayaUI)
MFnGenericAttribute	Generic attribute function set
MFnGeometryData	Geometry data for dependency node data
MFnGeometryFilter	Geometry filter function set (OpenMayaAnim)
MFnGravityField	Function set for gravity fields (OpenMayaFX)
MFnHikEffector	Full body IK end effector function set
MFnIkEffector	Inverse kinematics end effector function set (OpenMayaAnim)
MFnIkHandle	Function set for inverse kinematics (IK) handles (OpenMayaAnim)
MFnIkJoint	Function set for joints (OpenMayaAnim)
MFnIkSolver	Function set for inverse kinematics (IK) solvers (OpenMayaAnim)
MFnInstancer	Function set for obtaining read-only information on a particle instancer node
MFnIntArrayData	Integer array function set for dependency node data
MFnKeyframeDelta	Base function set for keyframe deltas
MFnKeyframeDeltaAddRemove	Retrieve info on keyframe adds or removes
MFnKeyframeDeltaBlockAddRemove	Retrieve info on block keyframe add or removes
MFnKeyframeDeltaBreakdown	Retrieve info on changes to keyframe breakdown
MFnKeyframeDeltaMove	Retrieve info on changes to keyframe value or time
MFnKeyframeDeltaScale	Retrieve info on changes to keyframe scaling
MFnKeyframeDeltaTangent	Retrieve info on changes to keyframe tangent
MFnKeyframeDeltaWeighter	Retrieve info on changes to keyframe weighted state
MFnKeyframeDeltInfType	Retrieve info on changes to pre/post infinity type
MFnLambertShader	Manage Lambert shaders
MFnLattice	Lattice function set (OpenMayaAnim)
MFnLatticeData	Lattice data dependency graph type
MFnLatticeDeformer	FFD lattice deformer function set (OpenMayaAnim)

MFnLayeredShader	Manage layered shaders
MFnLight	Manage dependency graph nodes representing lights
MFnLightDataAttribute	Light data attribute function set
MFnManip3D	3D manipulator function set (OpenMayaUI)
MFnMatrixAttribute	Matrix attribute function set
MFnMatrixData	Matrix function set for dependency node data
MFnMesh	Polygonal surface function set
MFnMeshData	Mesh function set for dependency node data
MFnMessageAttribute	Message attribute function set
MFnMotionPath	Motion path animation function set (OpenMayaAnim)
MFnNewtonField	Function set for Newton Fields (OpenMayaFX)
MFnNObjectData	Function set for Nucleus geometry data
MFnNonAmbientLight	Manage non-ambient light dependency nodes
MFnNonExtendedLight	Manage non-extended light dependency nodes
MFnNumericAttribute	Numeric attribute function set
MFnNumericData	Numeric data function set
MFnNurbsCurve	NURBS curve function set
MFnNurbsCurveData	NURBS curve function set for dependency node data
MFnNurbsSurface	NURBS surface function set
MFnNurbsSurfaceData	NURBS surface function set for dependency node data
MFnParticleSystem	Function set for particle information (OpenMayaFX)
MFnPartition	Function set for partitions of objects
MFnPfxGeometry	Function set for accessing paint effects render line information
MFnPhongEShader	Manage PhongE shaders
MFnPhongShader	Manage Phong shaders
MFnPlugin	Register and deregister plug-in services with Maya
MFnPluginData	User defined data function set for dependency node data
MFnPointArrayData	Point array function set for dependency node data
MFnPointLight	Manage point light dependency nodes
MFnPointOnCurveManip	PointOnCurveManip function set (OpenMayaUI)
MFnPointOnSurfaceManip	PointOnSurfaceManip function set (OpenMayaUI)
MFnRadialField	Function set for radial fields (OpenMayaFX)
MFnReflectShader	Manage reflective surface shaders
MFnRendenLayer	Manage render layer nodes
MFnRotateManip	Function set for rotation manipulator (OpenMayaUI)
MFnScaleManip	Function set for scale manipulator (OpenMayaUI)
MFnSet	Function set for sets of objects
MFnSingleIndexedComponent	Single indexed component function set
MFnSkinCluster	Skin cluster function set (OpenMayaAnim)
MFnSphereData	Sphere function set for dependency node data
MFnSpotLight	Manage spot light dependency nodes
MFnStateManip	StateManip function set (OpenMayaUI)
MFnStringArrayData	String array function set for dependency node data

MFnStringData	String function set for dependency node data
MFnSubd	Subdivision surface function set
MFnSubdData	NURBS surface function set for dependency node data
MFnSubdNames	Manipulate subdivision surface vertex, edge and face ids
MFnToggleManip	ToggleManip function set (OpenMayaUI)
MFnTransform	Create and access transform nodes
MFnTripleIndexedComponent	Triple indexed component function set
MFnTurbulenceField	Function set for turbulence fields (OpenMayaFX)
MFnTypedAttribute	Typed attribute function set
MFnUInt64ArrayData	MUint64 array function set for dependency node data
MFnUniformField	Function set for uniform fields (OpenMayaFX)
MFnUnitAttribute	Unit attribute function set
MFnVectorArrayData	Integer array function set for dependency node data
MFnVolumeAxisField	Function set for volumeAxis fields (OpenMayaFX)
MFnVolumeLight	Function set for volume lights
MFnVortexField	Function set for vortex Fields (OpenMayaFX)
MFnWeightGeometryFilter	Weight geometry filter function set (OpenMayaAnim)
MFnWireDeformer	Wire deformer function set (OpenMayaAnim)
MGeometry	The set of data elements which represent a Maya surface
MGeometryData	Storage class for data that will be used in drawing (OpenMayaRender)
MGeometryManager	Access to the Maya cached renderable geometry
MGeometryPrimitive	Describes the topology used for accessing MGeometryData
MGeometryRequirements	Provides methods for reading file images stored on disk
MGLFunctionTable	Wrapper class for OpenGL API
MGlobal	Static class providing common API global functions
MHardwareRenderer	Provides access to the hardware renderer (OpenMayaRender)
MHWShaderSwatchGenerator	Provides a mechanism for supplying a swatch for a hardware renderer plug-in (OpenMayaUI)
MHwTextureManager	Provides an interface for loading and using hardware textures
MIffFile	Provides access to Maya IFF parsing classes
MIffTag	Encapsulates IFF's 4 character block structure
MIkHandleGroup	IK handle groups (OpenMayaAnim)
MIkSystem	Inverse kinematics (IK) system class (OpenMayaAnim)
MImage	Image manipulation
MImageFileInfo	Provides methods for reading file images stored on disk
MIntArray	Array of integers data type
MItCurveCV	Iterator for NURBS curve CVs
MItDag	DAG Iterator
MItDependencyGraph	Dependency graph iterator
MItDependencyNodes	Dependency node iterator
MIteratorType	Object used for configuring some of the iterators
MItInstancer	Iterator class for instancer data
MItKeyframe	Keyframe iterator (OpenMayaAnim)

MItMeshEdge	Polygon edge iterator
MItMeshFaceVertex	Polygon face vertex iterator
MItMeshPolygon	Polygon iterator
MItMeshVertex	Polygon vertex iterator
MItSelectionList	Iterate over the items in the selection list
MItSubEdge	Iterate over subdivision surface edges
MItSubFace	Iterate over subdivision surface faces
MItSubVertex	Iterate over subdivision surface vertices
MItSurfaceCV	NURBS surface CV iterator
MLibrary	Standalone API application support class
MLightLinks	Provides read only light linking information
MLockMessage	Register callbacks for conditional node and plug locking
MManipData	Manipulator data (OpenMayaUI)
MMaterial	Hardware shading material class used in MPxSurfaceShapeUI (OpenMayaUI)
MMatrix	A matrix math class for 4x4 matrices of doubles
MMatrixArray	Implements an array of MMatrix data type
MMessage	Message base class
MModelMessage	Scene messages
MnCloth	Wrapper for N Cloth object that is used by Nucleus solver
MNodeMessage	Dependency node messages
MObject	Generic class for accessing internal Maya objects
MObjectArray	Array of MObjects data type
MObjectHandle	Wrapper for MObjects which contain validity information
MObjectSetMessage	Message class used to listen to changes to set membership
MPlug	Create and access dependency node plugs
MPlugArray	Array of MPlugs data type
MPoint	Implementation of a point
MPointArray	Array of MPoint data type
MPolyMessage	Message class used to listen to component ID changes
MProgressWindow	Class that provides access to progress window functionality
MPsdUtilities	Conversion and post render operation functionality
MPx3dModelView	Class for creating custom model views (OpenMayaUI)
MPxBakeEngine	For users to provide their own baking engine to bake advanced shading properties into a texture
MPxCommand	Base class for user commands
MPxComponentShape	High level interface for creating surface shapes with components
MPxContext	Base class for user defined contexts (OpenMayaUI)
MPxContextCommand	Base class for context creation commands (OpenMayaUI)
MPxControlCommand	Used for creating proxy UI control
MPxData	Base class for user-defined dependency graph data types
MPxDeformerNode	Base class for user defined deformers (OpenMayaAnim)
MPxDragAndDropBehavior	Base class for user defined drag and drop behavior
MPxEmitterNode	Base class for user defined particle emitters (OpenMayaFX)

MPxFieldNode	Base class for user defined fields (OpenMayaFX)
MPxFileTranslator	Base class for creating Maya file translators
MPxFluidEmitterNode	Based class for user defined fluid emitters
MPxGeometryData	Base class for user-defined dependency graph geometry data types
MPxGeometryIterator	Base class for user defined geometry iterators
MPxGlBuffer	Base class for user defined GL buffers (OpenMayaUI)
MPxHwShaderNode	Base class for user defined hardware shaders (OpenMayaUI)
MPxIkSolver	OBSOLETE CLASS: Base class for user defined IK solvers (OpenMayaAnim)
MPxIkSolverNode	Base class for user defined IK solvers (OpenMayaAnim)
MPxImageFile	Allows support of new fixed and floating point image file formats in Maya
MPxImagePlane	Allows support for custom types of image planes in Maya
MPxLocatorNode	Base class for user defined locators (OpenMayaUI)
MPxManipContainer	Base class for user defined manipulator containers (OpenMayaUI)
MPxMaterialInformation	Changes value for the Phong shader
MPxMayaAsciiFilter	Output filtered Maya ASCII files
MPxMayaAsciiFilterOutput	Writes buffer to the output stream of MPxMayaAsciiFilter
MPxMidiInputDevice	Midi input device (OpenMayaUI)
MPxModelEditorCommand	Base class for editor creation commands (OpenMayaUI)
MPxNode	Base class for user defined dependency nodes
MPxObjectSet	Base class for user defined object sets
MPxParticleAttributeMapperNode	Parent class of all user defined per particle attribute mapping nodes
MPxPolyTrg	Base class for user defined face triangulation of polygons
MPxPolyTweakUVCommand	Base class used for moving polygon UVs
MPxSelectionContext	Base class for interative selection tools (OpenMayaUI)
MPxSpringNode	Base class for user defined spring law (OpenMayaFX)
MPxSurfaceShape	Parent class of all user defined shapes
MPxSurfaceShapeUI	Drawing and selection for user defined shapes (OpenMayaUI)
MPxToolCommand	Base class for interactive tool commands (OpenMayaUI)
MPxTransform	Base class for user defined transformations
MPxTransformationMatrix	Base class for all user defined transformation matrices
MPxUIControl	Base class for control creation
MPxUITableControl	Base class for creating spreadsheet controls
MQuaternion	Quaternion math
MRampAttribute	Wrapper class for ramp data attributes
MRenderCallback	Rendering callbacks (OpenMayaRender)
MRenderData	Access rendering data (OpenMayaRender)
MRenderingInfo	Holds information about rendering into hardware render targets
MRenderLine	Retrieve information from a paint effects render line
MRenderLineArray.h	Retrieve an array of paint effects render line information

MRenderShadowData	Access rendering shadow map data (OpenMayaRender)
MRenderTarget	Contains information about a render target
MRenderUtil	Static class providing common API rendering functions (OpenMayaRender)
MRenderView	Static class providing access to Maya Render View functionality
MSceneMessage	Scene messages callbacks such as File New, File Open
MScriptUtil	Wrapper class for working with basic types in script
MSelectInfo	Selection state information used in MPxSurfaceShapeUI::select (OpenMayaUI)
MSelectionList	A list of MObjects
MSelectionMask	Manage what is selectable in Maya
MStatus	Manipulate Maya API status codes
MStreamUtils	Wrapper class that provides iostream functionality in script
MString	Manipulate strings
MStringArray	Array of MStrings data type
MStringResource	Wrapper class for string look up in localized plug-ins
MStringResourceId	Wrapper class the provides a unique for a string resource
MSwatchRenderBase	Base class for plug-ins wanting to provide swatch images (OpenMayaRender)
MSwatchRenderRegister	Provides registering and unregistering of swatch rendering functions (OpenMayaRender)
MSyntax	Syntax for commands
MTesselationParams	Tessellation parameters
MTime	Set and retrieve animation time values
MTimeArray	Array of MTime data type
MTimer	Used for measuring time. Similar to MEL timerX command
MTimerMessage	Register messages based on a fixed time interval
MToolsInfo	Tool information (OpenMayaUI)
MTransformationMatrix	Transformation matrix
MTrimBoundary	Container class that holds MObjects which describe a trim boundary
MTypeId	Manage Maya object type identifiers
MUiMessage	UI messages (OpenMayaUI)
MUint64Array	Array of MUint64 data type
MUintArray	Array of unsigned int data type
MUserEvenMessage	Message class used for creating, posting and listening to user defined messages
MVector	A vector math class for vectors of doubles
MVectorArray	Array of MVectors data type
MViewportRenderer	Represents a hardware viewport renderer

Appedix B: Maya API Class Hierarchy

The following list illustrates class hierarchy of the Maya 8.5 API.

```
MAnimMessage < MMessage
MArgDatabase < MArgParser
MArgParser
MClothConstraintCmd < MClothConstraint
MClothConstraint
MCommandMessage < Mmessage
MCommonSystemUtils
MConditionMessage < MMessage
MDagMessage < MMessage
MDagModifier < MDGModifier
MDGMessage < MMessage
MDGModifier
MDrawInfo
MDrawTraversal
MDynamicsUtil
MEventMessage < MMessage
MFnAirField < MFnField < MFnDagNode < MFnDependencyNode < MFnBase
MFnAmbientLight < MFnLight < MFnDagNode < MFnDependencyNode < MFnBase
MFnAnimCurve < MFnDependencyNode < MFnBase
MFnAnisotropyShader < MFnDependencyNode < MFnBase
MFnAreaLight < MFnNonExtendedLight < MFnNonAmbientLight < MFnLight <
MFnDagNode < MFnDependencyNode < MFnBase
MFnArrayAttrsData < MFnData < MFnBase
MFnAttribute < MFnBase
MFnBase
MFnBlendShapeDeformer < MFnDependencyNode < MFnBase
MFnBlinnShader < MFnReflectShader < MFnLambertShader < MFnDependencyNode
< MFnBase
MFnCamera < MFnDagNode < MFnDependencyNode < MFnBase
MFnCharacter < MFnSet < MFnDependencyNode < MFnBase
MFnCircleSweepManip < MFnManip3D < MFnTransform < MFnDagNode <
MFnDependencyNode < MFnBase
MFnClip < MFnDependencyNode < MFnBase
MFnComponent < MFnBase
MFnComponentListData < MFnData < MFnBase
MFnCompoundAttribute < MFnAttribute < MFnBase
MFnCurveSegmentManip < MFnManip3D < MFnTransform < MFnDagNode <
MFnDependencyNode < MFnBase
MFnDagNode < MFnDependencyNode < MFnBase
MFnData < MFnBase
MFnDependencyNode < MFnBase
MFnDirectionalLight < MFnNonExtendedLight < MFnNonAmbientLight <
MFnLight < MFnDagNode < MFnDependencyNode < MFnBase
MFnDirectionManip < MFnManip3D < MFnTransform < MFnDagNode <
MFnDependencyNode < MFnBase
MFnDiscManip < MFnManip3D < MFnTransform < MFnDagNode <
MFnDependencyNode < MFnBase
MFnDistanceManip < MFnManip3D < MFnTransform < MFnDagNode <
MFnDependencyNode < MFnBase
MFnDoubleArrayData < MFnData < MFnBase
MFnDoubleIndexedComponent < MFnComponent < MFnBase
MFnDragField < MFnField < MFnDagNode < MFnDependencyNode < MFnBase
MFnDynSweptGeometryData < MFnData < MFnBase
MFnEnumAttribute < MFnAttribute < MFnBase
MFnExpression < MFnDependencyNode < MFnBase
MFnField < MFnDagNode < MFnDependencyNode < MFnBase
MFnFluid < MFnDagNode < MFnDependencyNode < MFnBase
MFnFreePointTriadManip < MFnManip3D < MFnTransform < MFnDagNode <
MFnDependencyNode < MFnBase
```

```
MFnGenericAttribute < MFnAttribute < MFnBase
MFnGeometryData < MFnData < MFnBase
MFnGeometryFilter < MFnDependencyNode < MFnBase
MFnGravityField < MFnField < MFnDagNode < MFnDependencyNode < MFnBase
MFnHikEffector < MFnTransform < MFnDagNode < MFnDependencyNode < MFnBase
MFnIkEffector < MFnTransform < MFnDagNode < MFnDependencyNode < MFnBase
MFnIkHandle < MFnTransform < MFnDagNode < MFnDependencyNode < MFnBase
MFnIkJoint < MFnTransform < MFnDagNode < MFnDependencyNode < MFnBase
MFnIkSolver < MFnDependencyNode < MFnBase
MFnIntArrayData < MFnData < MFnBase
MFnKeyframeDeltaAddRemove < MFnKeyframeDelta < MFnBase
MFnKeyframeDeltaBlockAddRemove < MFnKeyframeDelta < MFnBase
MFnKeyframeDeltaBreakdown < MFnKeyframeDelta < MFnBase
MFnKeyframeDelta < MFnBase
MFnKeyframeDeltaInfType < MFnKeyframeDelta < MFnBase
MFnKeyframeDeltaMove < MFnKeyframeDelta < MFnBase
MFnKeyframeDeltaScale < MFnKeyframeDelta < MFnBase
MFnKeyframeDeltaTangent < MFnKeyframeDelta < MFnBase
MFnKeyframeDeltaWeighted < MFnKeyframeDelta < MFnBase
MFnLambertShader < MFnDependencyNode < MFnBase
MFnLatticeData < MFnGeometryData < MFnData < MFnBase
MFnLatticeDeformer < MFnDependencyNode < MFnBase
MFnLattice < MFnDagNode < MFnDependencyNode < MFnBase
MFnLayeredShader < MFnDependencyNode < MFnBase
MFnLightDataAttribute < MFnAttribute < MFnBase
MFnLight < MFnDagNode < MFnDependencyNode < MFnBase
MFnManip3D < MFnTransform < MFnDagNode < MFnDependencyNode < MFnBase
MFnMatrixAttribute < MFnAttribute < MFnBase
MFnMatrixData < MFnData < MFnBase
MFnMeshData < MFnGeometryData < MFnData < MFnBase
MFnMesh < MFnDagNode < MFnDependencyNode < MFnBase
MFnMessageAttribute < MFnAttribute < MFnBase
MFnMotionPath < MFnDependencyNode < MFnBase
MFnNObjectData < MFnData < MFnBase
MFnNewtonField < MFnField < MFnDagNode < MFnDependencyNode < MFnBase
MFnNonAmbientLight < MFnLight < MFnDagNode < MFnDependencyNode < MFnBase
MFnNonExtendedLight < MFnNonAmbientLight < MFnLight < MFnDagNode <
MFnDependencyNode < MFnBase
MFnNumericAttribute < MFnAttribute < MFnBase
MFnNumericData < MFnData < MFnBase
MFnNurbsCurveData < MFnGeometryData < MFnData < MFnBase
MFnNurbsCurve < MFnDagNode < MFnDependencyNode < MFnBase
MFnNurbsSurfaceData < MFnGeometryData < MFnData < MFnBase
MFnNurbsSurface < MFnDagNode < MFnDependencyNode < MfnBase
MFnInstancer < MFnDagNode < MFnDependencyNode < MFnBase
MFnParticleSystem < MFnDagNode < MFnDependencyNode < MFnBase
MFnPartition < MFnDependencyNode < MFnBase
MFnPfxGeometry < MFnDagNode < MFnDependencyNode < MFnBase
MFnPhongEShader < MFnDependencyNode < MFnBase
MFnPhongShader < MFnReflectShader < MFnLambertShader < MFnDependencyNode
< MFnBase
MFnPluginData < MFnData < MFnBase
MFnPlugin < MFnBase
MFnPointArrayData < MFnData < MFnBase
MFnPointLight < MFnNonExtendedLight < MFnNonAmbientLight < MFnLight <
MFnDagNode < MFnDependencyNode < MFnBase
MFnPointOnCurveManip < MFnManip3D < MFnTransform < MFnDagNode <
MFnDependencyNode < MFnBase
```

MFnPointOnSurfaceManip < MFnManip3D < MFnTransform < MFnDagNode < MFnDependencyNode < MFnBase
MFnRadialField < MFnField < MFnDagNode < MFnDependencyNode < MFnBase
MFnReflectShader < MFnLambertShader < MFnDependencyNode < MFnBase
MFnRenderLayer < MFnBase
MFnRotateManip < MFnManip3D < MFnTransform < MFnDagNode < MFnDependencyNode < MFnBase
MFnScaleManip < MFnManip3D < MFnTransform < MFnDagNode < MFnDependencyNode < MFnBase
MFnSet < MFnDependencyNode < MFnBase
MFnSingleIndexedComponent < MFnComponent < MFnBase
MFnSkinCluster < MFnGeometryFilter < MFnDependencyNode < MFnBase
MFnSphereData < MFnData < MFnBase
MFnSpotLight < MFnNonExtendedLight < MFnNonAmbientLight < MFnLight < MFnDagNode < MFnDependencyNode < MFnBase
MFnStateManip < MFnManip3D < MFnTransform < MFnDagNode < MFnDependencyNode < MFnBase
MFnStringArrayData < MFnData < MFnBase
MFnStringData < MFnData < MFnBase
MFnSubdData < MFnGeometryData < MFnData < MFnBase
MFnSubd < MFnDagNode < MFnDependencyNode < MFnBase
MFnToggleManip < MFnManip3D < MFnTransform < MFnDagNode < MFnDependencyNode < MFnBase
MFnTransform < MFnDagNode < MFnDependencyNode < MFnBase
MFnTripleIndexedComponent < MFnComponent < MFnBase
MFnTurbulenceField < MFnField < MFnDagNode < MFnDependencyNode < MFnBase
MFnTypedAttribute < MFnAttribute < MFnBase
MFnUInt64ArrayData < MFnData < MFnBase
MFnUniformField < MFnField < MFnDagNode < MFnDependencyNode < MFnBase
MFnUnitAttribute < MFnAttribute < MFnBase
MFnVectorArrayData < MFnData < MFnBase
MFnVolumeAxisField < MFnField < MFnDagNode < MFnDependencyNode < MFnBase
MFnVolumeLight < MFnPointLight < MFnNonExtendedLight < MFnNonAmbientLight < MFnLight < MFnDagNode < MFnDependencyNode < MFnBase
MFnVortexField < MFnField < MFnDagNode < MFnDependencyNode < MFnBase
MFnWeightGeometryFilter < MFnGeometryFilter < MFnDependencyNode < MFnBase
MFnWireDeformer < MFnDependencyNode < MfnBase
MGeometry
MGeometryManager
MGeometryRequirements
MGeometryPrimitive
MGLFunctionTable
MHWShaderSwatchGenerator < MswatchRenderBase
MHwTextureManager
MIffFile
MiffTag
MImageFileInfo
MLockMessage < Mmessage
MMatrixArray
MMessage
MModelMessage < MMessage
MnCloth
MNodeMessage < MMessage
MObjectSetMessage < MMessage
MPolyMessage < Mmessage
MPxBakeEngine
MPxCommand
MPxComponentShape < MPxSurfaceShape < MPxNode

```
MpxContext
MPxControlCommand
MPxData
MPxDeformerNode < MPxNode
MPxEmitterNode < MPxNode
MPxFieldNode < MPxNode
MPxFluidEmitterNode < MPxEmitterNode < MPxNode
MPxGeometryData < MPxData
MPxHwShaderNode < MPxNode
MPxIkSolverNode < MPxNode
MPxImageFile < MPxNode
MPxLocatorNode < MPxNode
MPxManipContainer < MPxNode
MPxMaterialInformation
MPxMayaAsciiFilter < MPxFileTranslator
MPxMayaAsciiFilterOutput
MPxNode
MPxObjectSet < MPxNode
MPxParticleAttributeMapperNode < MPxNode
MPxPolyTrg < MPxNode
MPxPolyTweakUVCommand < MPxCommand
MPxSelectionContext < MPxContext
MPxSpringNode < MPxNode
MPxSurfaceShape < MPxNode
MPxToolCommand < MPxCommand
MPxTransform < MPxNode
MPxUIControl
MPxUITableControl < MPxUIControl
MRenderTarget
MRenderingInfo
MSceneMessage < MMessage
MSelectInfo < MDrawInfo
MSimple < MPxCommand
MScriptUtil
MStreamUtils
MStringResource
MStringResourceId
MString
MSwatchRenderBase
MTimerMessage < MMessage
MViewportRenderer
MUiMessage < MMessage
MUserEventMessage < Mmessage
```

Appendix C: Selected list of sample plug-ins

The following is a list of selected sample plug-ins in our developer kit. A general description of what the sample does is also provided. This list is provided as an aid for learning different aspects of the Maya API and Maya Python API.

Description	Name
Simple plug-in	`helloWorld.cpp` | `helloWorld.py`

Simple dependency node plug-in	`sineNode.cpp` | `sineNode.py`
Simple MEL command plug-in	`whatisCmd.cpp` | `whatisCmd.pyq`
Software shader plug-in	`lambertShader.cpp` | `slopeShader.py`
Hardware shader plug-in	`hwUnlitShader.cpp`
Translator plug-in	`maTranslator.cpp`
Picking	`whatisCmd.cpp` | `whatisCmd.py`
Particles	`ownerEmitter.cpp` and `particleSystemInfoCmd.cpp`
Meshes	`polyWriter.cpp` and `shellNode.cpp` | `splitUVCmd.py`
UVs	`flipUVCmd.cpp`
Messages	`dagMessageCmd.cpp` and `nodeMessageCmd.cpp` | `parentAddedMsgCmd.py`
Surface shapes	`apiMesh*.cpp` (there is more than one file)

Autodesk