SD473708

Introduction to Maya Customization

Lanh Hong
Autodesk

Learning Objectives

- Explain the benefits of using the API to customize and extend Maya's functionalities.
- Create a simple plug-in using the Python API.
- Discover how to automate common tasks.
- Locate resources and documentation to continue programming in Maya.

Description

Autodesk Maya is a popular 3D modeling, animation, rendering, and visual effects software in the Media & Entertainment industry. It provides a powerful API that allows users to customize and extend Maya's functionalities in order to create efficient workflows and become even more productive.

This class is an introduction to Maya customization for customers who would like to improve efficiency in using the product or to create automation. Some programming experience will be helpful. You will learn about the benefits of customizing Maya and gain confidence to create your first plug-in using the Maya Python API. Note that all concepts can easily be reproduced in the C++ environment as well.

Speaker(s)

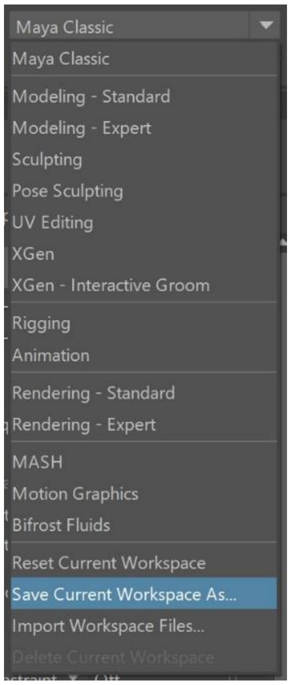
Lanh Hong is part of the Media & Entertainment DevTech team at Autodesk where she provides support for members in the Autodesk Developer Network and customers using the Autodesk Forge web services. She joined Autodesk in 2018 after completing her bachelor’s degree in Computer Science from the University of California, Davis. Since then, her primary focus is on helping customers customize and extend Autodesk Maya’s functionalities using the Maya API.

Customizing Maya

Customizing can be as simple as moving panels around Maya’s user interface (UI) or adding tools to a custom shelf. For more advanced customization, Maya provides the ability to use programming to develop plug-ins, scripts, and applications. Before I introduce customization using programming, I will explain briefly on some of the ways you can customize Maya without diving much into programming concepts.

User Interface (UI)

First, you can customize the look and feel of Maya’s UI by dragging panels around and docking them anywhere you like. By moving the panels around, you can change up how your workspace looks like to fit your style and preference. To create a custom workspace, you can select *Save Current Workspace As…* in the workspace dropdown list.

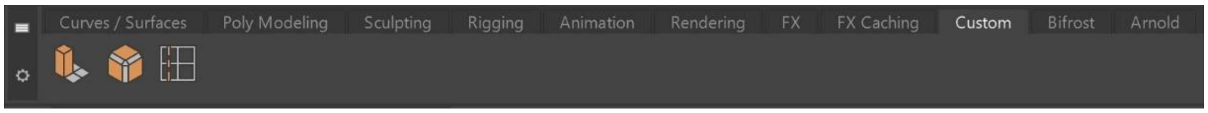


SELECT “SAVE CURRENT WORKSPACE AS…” TO SAVE YOUR CUSTOM WORKSPACE.

Furthermore, you can customize the colors of Maya’s UI and view panel. If you are not a fan of Maya’s default colors, you can change them in the Color Settings window located in *Windows > Settings/Preferences > Color Settings*.

Shelves

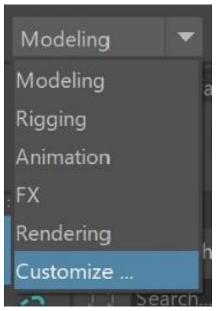
If you want a place to store your tools that you use most often and be able to access them quickly, you can utilize the custom shelf. A custom shelf can contain any tools, menu items, commands or scripts that you add to it. Any shelf item can be customized by giving it a name and uploading an icon. To customize your shelves, head over to the Shelf Editor window located in *Windows > Settings/Preferences > Shelf Editor*.



ADD TOOLS TO YOUR CUSTOM SHELF.

Menu Sets

Even menu sets in Maya can be customized. If you prefer to choose the menu items that appears in your menu bar, you can create your own menu set. Any customization of menu sets can be done by selecting *Customize* in the menu set dropdown list. This will open the Menu Set Editor where you can add, delete, or rearrange items in your menu sets.



SELECT "CUSTOMIZE" TO CREATE OR EDIT MENU SETS.

Other Customizations

There are other ways to customize Maya such as customizing hotkeys, marking menu, and the hotbox. However, I will not go into much detail about them. To find more information about these customizations mentioned and to explore what else is possible, head over to the Maya documentation page.

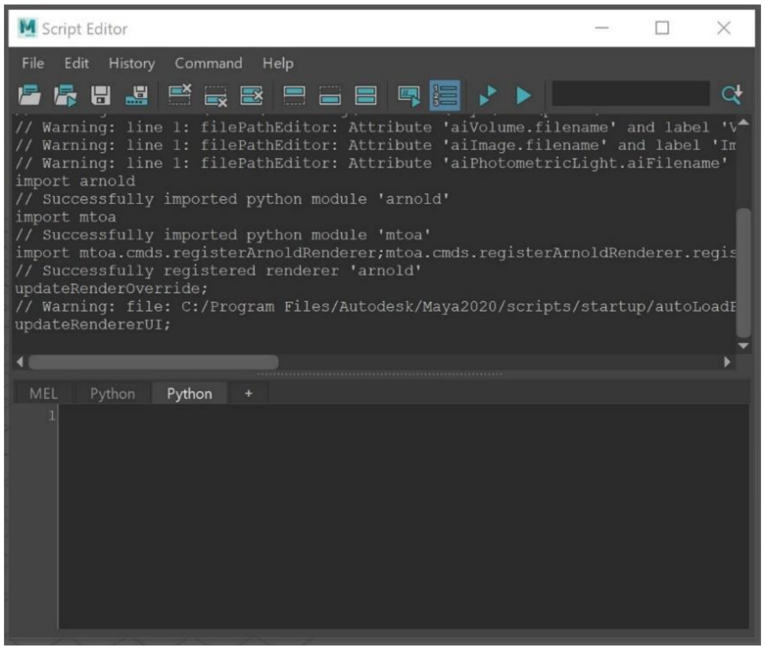
For this class, the primary focus is to introduce customization by using the Maya Application Programming Interface (API).

Customizing Maya Using Programming

Maya provides the ability to use programing to customize and extend its capabilities. You can customize existing features of Maya or create your own to extend its functionality. With Maya's tools, you can develop plug-ins, scripts, and applications that will work seamlessly with Maya.

Scripts

Scripts can be written in MEL or Python. They are mostly comprised of commands that perform some task in Maya. In fact, almost everything you do in the Maya interface is a MEL command. Commands can be executed in the Command Line or Maya's interactive Script Editor window located in *Windows* > *General Editors* > *Script Editor*. The Script Editor is a place where you can run single line or multiple lines of MEL or Python code. You can also see a history of commands and results in the top pane.



```
// Warning: line 1: filePathEditor: Attribute 'aiVolume.filename' and label 'V
// Warning: line 1: filePathEditor: Attribute 'aiImage.filename' and label 'Im
// Warning: line 1: filePathEditor: Attribute 'aiPhotometricLight.aiFilename'
import arnold
// Successfully imported python module 'arnold'
import mtoa
// Successfully imported python module 'mtoa'
import mtoa.cmds.registerArnoldRenderer;mtoa.cmds.registerArnoldRenderer.regis
// Successfully registered renderer 'arnold'
updateRenderOverride;
// Warning: file: C:/Program Files/Autodesk/Maya2020/scripts/startup/autoLoadP
updateRendererUI;
```

SCRIPT EDITOR

Plug-ins

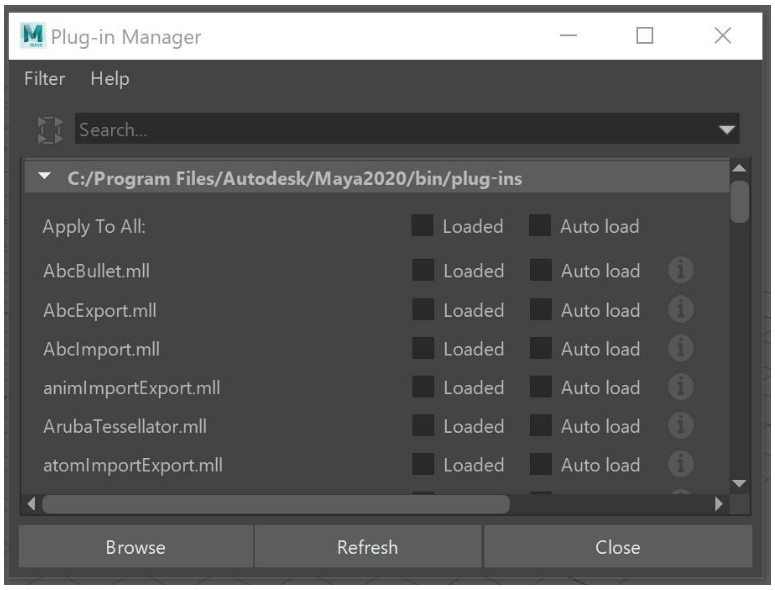
Plug-ins can be written in C++ or Python. They are add-ons to Maya that extends the software's capabilities. You can create new tools, features, commands, and nodes as a plug-in and use them inside Maya. Maya also comes with pre-built plug-ins that you can take advantage of immediately. However, all pre-built or custom plug-ins must be loaded first before using. You can do so in the Plug-in Manager window located in *Windows > Settings/Preferences > Plug-in Manager*.

Command Plug-ins

A command plug-in defines a new MEL command that can be used just like any other commands in Maya. The command plug-in can be invoked using both MEL and Python. It can also be added to any scripts or paired with dependency plug-ins.

Dependency Graph Plug-ins

A dependency graph is a network of nodes. Each node has connection points that allows it to connect to other nodes. A dependency graph plug-in defines a new dependency graph node that works with other nodes in Maya. It allows you to create your own nodes such as deformers, manipulators, shapes, and shaders just to name a few. It involves taking in a set of inputs and computing an output. These input and output plugs can be connected to other nodes in the dependency graph.



PLUG-IN MANAGER

Benefits of Customization

Customizing Maya using programming is an investment of time, but it is worth the effort to learn and understand. By building tools and features that plugs right into Maya, you can tailor Maya to fit your needs. You can automate repetitive and common tasks to increase your productivity when using the software. It will also help streamline your workflows and increase your efficiency.

Programming Languages

There are several programming languages that Maya mainly uses – Maya Embedded Language (MEL), Python, and C++.

MEL

MEL is a scripting language for Maya, and it is Maya's foundation. Maya's UI is created using MEL, and any tasks performed in Maya are just MEL commands. Use MEL to create scripts and automate common tasks in Maya. You can easily run MEL commands in the Command Line or Script Editor. Compared to the other languages, MEL is the easiest to learn and understand. However, MEL is limited to scripting only and not for plug-in development.

C++

The Maya API is available in C++ and Python. For plug-in development in Maya, most people will choose C++ because it is a popular language being used in the M&E industry. It is more complicated and difficult to learn compared to MEL and Python. Any plug-ins written in C++ must be compiled before using. Since it is a lower level language, it runs quickly and gives better performance to your plug-ins.

Python

Python is the best of both worlds. You can create scripts or build plug-ins using Python. To execute commands in Python you must use the `maya.cmds` module. It is a wrapper for MEL commands and can be use in place of MEL. As for plug-ins, the Python API wraps around the C++ API. Unlike C++, plug-ins do not need to be compiled before use, but the performance will be hindered.

Python will be the language of choice when we create a plug-in later because it will be easier to understand for an introduction class.

Maya Application Programming Interface (API)

You have heard me mention the Maya API a few times already, but what is it exactly? An API is a way for applications to access and interact with another application. For plug-ins to work seamlessly and behave like a native Maya feature, it needs a way to access and handle Maya's resources. This is where the Maya API comes in. When you are building your plug-in, you will use the Maya API to access and manipulate objects or create new objects that will work just like the internal objects.

The Maya API comes in C++ and Python. The Python API is just a wrapper wound the C++ API. It can do many of the things that the C++ API can do, but it is not mapped perfectly 1 to 1. Few changes were made to the Python bindings for integration purposes. You may find that the Python API will have missing functionality, but most of it is available. Performance wise, the C++ API performs better and runs faster in general. However, Python is easier to use and understand. You can use either one to develop your plug-ins, but for this class, we will be using Python to create the plug-in.

Python API

There are two versions of the Maya Python API – 1.0 and 2.0. The 2.0 API is the newest version. It provides a more Pythonic workflow compared to 1.0 API and with improved performance. Most people will use the 2.0 API, and the Maya Python API reference page only references the 2.0 version. Between 1.0 and 2.0, most of the class and method names are kept the same so it could get confusing on which one you are using. The Maya Developer documentation and devkit plug-ins have both versions as samples.

Before you can use the APIs, you need to import the modules. The Python API 1.0 modules can be found in `maya` while Python API 2.0 modules can be found in `maya.api`. For example:

```
import maya.OpenMaya as om1

import maya.api.OpenMaya as om2
```

For Python API 2.0, you must define an additional function called `maya_useNewAPI` so that Maya knows to pass objects from the 2.0 API.

```
def maya_useNewAPI():
        pass
```

API Basics
There are many classes that makes up the Maya API. They mainly consist of objects, function sets, proxies, wrappers and iterators.

Objects (MObject)
Maya's model objects are represented by the `MObject` class. Model objects are the internal modelling, animation, and rendering objects, as well as all the dependency graph nodes and their attributes. The `MObject` class provides an interface to access these model objects. Each `MObject` has a specified type, and they work together with function set classes to manipulate a specific model object.

Function sets (MFn)
Function set classes have the `MFn` prefix. Function set classes provide a set of functions for operating on a specific object type. For example, the `MFnNurbsCurve` class contains functions for manipulating NURBS curves. It can also create `MObject`, and the `MObject` that it creates acquires that function set's type. As an example, `MFnTypedAttribute` creates `MObjects` with `kTypedAttribute` type.

Proxies (MPx)
Proxy classes have the `MPx` prefix. Proxy classes provide a way to define your own model object that Maya is aware of. Any custom node or command must extend a proxy class. For example, creating a custom command involves extending the `MPxCommand` class and a custom node will need to extend `MPxNode`. Extending a proxy class gives you the function sets to manipulate your object.

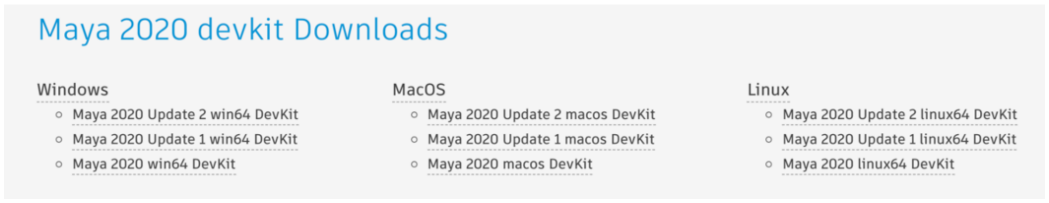
Wrappers (M)
Wrapper classes have the `M` prefix. They are simple objects such as vectors and arrays (e.g. `MVector` and `MIntArray`). These objects can be manipulated directly.

Iterators (MIt)
Iterator classes have the `MIt` prefix. Iterators iterate over items and works on `MObjects` of a particular type. For example, `MItCurveCV` class can iterate over a NURBS curve control vertices (CVs) if it is given an entire curve or a group of CVs.

Maya Developer Kit (Devkit)

The Maya Software Development Kit (SDK), commonly known as the Maya Developer Kit (devkit), are a set of tools and libraries that you need to customize and extend Maya's capabilities. Inside the devkit, you will find C++ and Python APIs, as well as plug-in samples. You will use the devkit to create custom plug-ins, scripts, and applications for Maya.

The devkit download links can be found in the Maya Developer Center [https://www.autodesk.com/developmaya]. Make sure to download the devkit that corresponds to your operating system, Maya version, and the update number. Downloading the correct devkit guarantees binary compatibility with Maya's C++ architecture. Python plug-ins are very portable, meaning they don't require any changes between devkits. However, C++ plug-in must be recompiled for every major release.



Maya 2020 devkit Downloads

Windows	MacOS	Linux
Maya 2020 Update 2 win64 DevKit	Maya 2020 Update 2 macos DevKit	Maya 2020 Update 2 linux64 DevKit
Maya 2020 Update 1 win64 DevKit	Maya 2020 Update 1 macos DevKit	Maya 2020 Update 1 linux64 DevKit
Maya 2020 win64 DevKit	Maya 2020 macos DevKit	Maya 2020 linux64 DevKit

MAYA DEVKIT DOWNLOAD LINKS IN THE MAYA DEVELOPER CENTER

Plug-in Samples

After downloading the devkit from the Maya Developer Center, you can find C++ and Python plug-in samples inside the devkit directory. These samples are a good starting point when you are developing your plug-in.

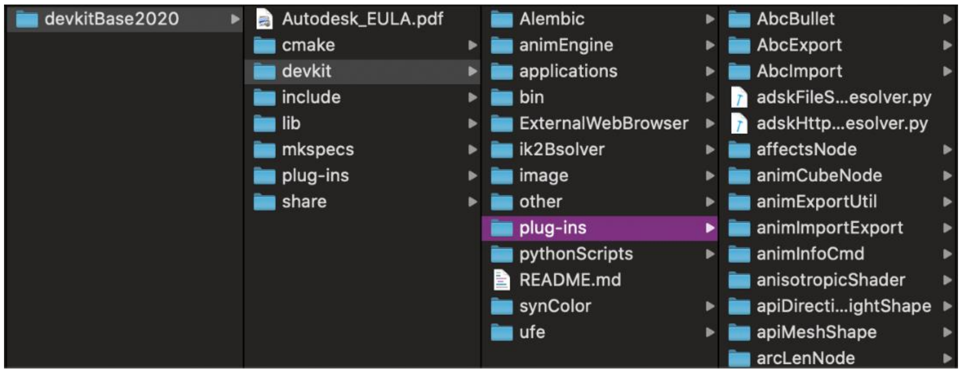
Python Samples

Python plug-in samples are in the `devkitBase/devkit/plug-ins/scripted/` directory. Samples that begin with "py" (e.g. `pyApiMeshShape.py`) uses Python API 2.0. All other samples use Python API 1.0.

C++ Samples

C++ plug-in samples are located in the `devkitBase/devkit/plug-ins/` directory.

Unlike Python, C++ plug-ins must be compiled before they can be used. Instructions on building C++ samples can be found in the Developer Help documentation [http://help.autodesk.com/view/MAYAUL/2020/ENU/?guid=__developer_Maya_SDK_MERGED_ExamplesAndSamples_Building_Sample_Plugins_html].



PLUG-IN EXAMPLES IN THE DEVKIT DIRECTORY

Your First Plug-in

Time to create your first plug-in using Python! For this class, I will guide you through a dependency node plug-in using the Python API 2.0.

Before you start, let us understand the dependency graph and how a dependency graph node fits in Maya. A dependency graph is a network of nodes. Each node has input and output connection points, known as *plugs*, that can be connected to other nodes. A node's behavior is to simply take in a set of inputs, use the inputs to do some computation, and give the results as outputs. A dependency graph plug-in defines a new node in the dependency graph.

Setup

Let us set up and get ready to create your plug-in. Make sure you have the following items before we start.

- **Text Editor** – Choose any text editor – Visual Studio Code, Sublime Text, Notepad++, or other ones you are familiar with. This is where you will be writing your plug-in code.

- **Devkit and Environment Variables** – Download the devkit from the Maya Developer Center and set up your environment variables. Follow the exact steps for your system in the Developer Help documentation [http://help.autodesk.com/view/MAYAUL/2020/ENU/?guid=__developer_Maya_SDK_MERGED_Setting_up_your_build_html].

- **Node ID block** – Register for your own node ID block. Dependency graph nodes require you to use your own IDs to prevent any conflicts with other nodes in Maya. The IDs will be unique to you and will ensure that your plug-ins will not have any problems loading. You can find the link to register in the Maya Developer Center or go to directly here [https://mayaid.autodesk.io]. The IDs are not needed for command plug-ins.

  After registering, your node IDs will be sent to you via email. The node IDs comes in a hexadecimal form with `0x` prefix. Each digit can have 16 different values instead of the normal 10 digit that goes from 0 to 9. A hexadecimal goes from 0 to 9, then 10 to 15 is represented as A to F. An example node ID will look like this – `0x087EFE`.

Create the plug-in

You will be creating a new node called `arcLenNode`. This node will take in a NURBS curve as an input, calculate the arc length of the curve, and give the resulting length as the output.

This plug-in is a C++ sample that you can find in the devkit, but we will be recreating it using Python. At the end, you can compare this Python version and the C++ version to see the differences between them.

1. Start with a template

Let us start off with a template. As a starting point, copy the Python API 2.0 code from the documentation in *Maya Developer Help > Maya Python API > Maya Python Plug-in Learning Path > Dependency Graph Plug-ins Basics > Dependency Graph Plug-ins* [http://help.autodesk.com/view/MAYAUL/2020/ENU//?guid=__developer_Maya_SDK_MERGED_Maya_Python_API_Maya_Python_Plug_in_Learning_Dependency_Graph_Plug_in_Basics_Dependency_Graph_Plug_ins_html].

You will continue to build on top of it and make changes along the way. This way, it will be easier than starting from scratch. Here is what you are starting with:

```
import sys
import maya.api.OpenMaya as OpenMaya
# ... additional imports here ...

def maya_useNewAPI():
    """
    The presence of this function tells Maya that the plugin produces, and
    expects to be passed, objects created using the Maya Python API 2.0.
    """
    pass

# Plug-in information:
kPluginNodeName = 'myNodeName'              # The name of the node.
kPluginNodeClassify = 'utility/general'     # Where this node will be found in the Maya UI.
kPluginNodeId = OpenMaya.MTypeId( 0x87EFE ) # A unique ID associated to this node type.

# Default attribute values
sampleDefaultValue = 1

##########################################################
# Plug-in
##########################################################
class myNode(OpenMaya.MPxNode):
    # Static variables which will later be replaced by the node's attributes.
    sampleInAttribute = OpenMaya.MObject()
    sampleOutAttribute = OpenMaya.MObject()

    def __init__(self):
        ''' Constructor. '''
        OpenMaya.MPxNode.__init__(self)

    def compute(self, pPlug, pDataBlock):
        '''
        Node computation method.
        - pPlug: A connection point related to one of our node attributes (could be an input or an output)
        - pDataBlock: Contains the data on which we will base our computations.
        '''
        if( pPlug == myNode.sampleOutAttribute ):
            # Obtain the data handles for each attribute
            sampleInDataHandle = pDataBlock.inputValue( myNode.sampleInAttribute )
            sampleOutDataHandle = pDataBlock.outputValue( myNode.sampleOutAttribute )
            # Extract the actual value associated to our sample input attribute
            # (we have defined it as a float)
            sampleInValue = sampleInDataHandle.asFloat()

            # ... perform the desired computation ...

            # Set the output value.
            # As an example, we just set the output value to be equal to the input value.
            sampleOutDataHandle.setFloat( sampleInValue )
            # Mark the output data handle as being clean; it need not be computed given its input.
            sampleOutDataHandle.setClean()
        else:
            return OpenMaya.kUnknownParameter

##########################################################
# Plug-in initialization.
##########################################################
def nodeCreator():
    ''' Creates an instance of our node class and delivers it to Maya as a pointer. '''
    return  myNode()

def nodeInitializer():
    ''' Defines the input and output attributes as static variables in our plug-in class. '''
    # The following MFnNumericAttribute function set will allow us to create our attributes.
    numericAttributeFn = OpenMaya.MFnNumericAttribute()

    #==================================
    # INPUT NODE ATTRIBUTE(S)
    #==================================
    global sampleDefaultValue
    myNode.sampleInAttribute = numericAttributeFn.create( 'myInputAttribute', 'i',
                                                          OpenMaya.MFnNumericData.kFloat,
                                                          sampleDefaultValue )
    numericAttributeFn.writable = True
    numericAttributeFn.storable = True
    numericAttributeFn.hidden = False
    myNode.addAttribute( myNode.sampleInAttribute ) # Calls the MPxNode.addAttribute function.

    #==================================
    # OUTPUT NODE ATTRIBUTE(S)
    #==================================
    myNode.sampleOutAttribute = numericAttributeFn.create( 'myOutputAttribute', 'o',
                                                           OpenMaya.MFnNumericData.kFloat )
    numericAttributeFn.storable = False
    numericAttributeFn.writable = False
    numericAttributeFn.readable = True
    numericAttributeFn.hidden = False
    myNode.addAttribute( myNode.sampleOutAttribute )

    #==================================
    # NODE ATTRIBUTE DEPENDENCIES
    #==================================
    # If sampleInAttribute changes, the sampleOutAttribute data must be recomputed.
    myNode.attributeAffects( myNode.sampleInAttribute, myNode.sampleOutAttribute )

def initializePlugin( mobject ):
    ''' Initialize the plug-in '''
    mplugin = OpenMaya.MFnPlugin( mobject )
    try:
        mplugin.registerNode( kPluginNodeName, kPluginNodeId, nodeCreator,
                              nodeInitializer, OpenMaya.MPxNode.kDependNode, kPluginNodeClassify )
    except:
        sys.stderr.write( 'Failed to register node: ' + kPluginNodeName )
        raise

def uninitializePlugin( mobject ):
    ''' Uninitializes the plug-in '''
    mplugin = OpenMaya.MFnPlugin( mobject )
    try:
        mplugin.deregisterNode( kPluginNodeId )
    except:
        sys.stderr.write( 'Failed to deregister node: ' + kPluginNodeName )
        raise
```

2. Replace node ID with your own

Change the node ID (`0x87EFE`) to use one of your own. The node ID is a hexadecimal number with a `0x` prefix. You should have received a range of numbers in your email after registering for your node ID block.

```
kPluginNodeId = OpenMaya.MTypeId( 0x87EFE ) # A unique ID associated to this node type.
```

3. Change the node name

Replace the node name from `myNodeName` to `arcLenNode`. The plug-in node name is the name that Maya will use to create the node.

Then change the class definition from `myNode` to `arcLenNode`. The name can be anything you like, as long as it is intuitive to you. Here is a simplified sample of what your code will look like afterwards.

```
...
kPluginNodeName = 'arcLenNode'              # The name of the node.
...
class arcLenNode(OpenMaya.MPxNode):
     ...
    def compute(self, pPlug, pDataBlock):
        if( pPlug == arcLenNode.sampleOutAttribute ):
            sampleInDataHandle = pDataBlock.inputValue( arcLenNode.sampleInAttribute )
            sampleOutDataHandle = pDataBlock.outputValue( arcLenNode.sampleOutAttribute )
            ...
...
def nodeCreator():
    return  arcLenNode()

def nodeInitializer():
    ...
    arcLenNode.sampleInAttribute = numericAttributeFn.create( 'myInputAttribute', 'i',
                                                      OpenMaya.MFnNumericData.kFloat,
                                                      sampleDefaultValue )
    ...
    arcLenNode.addAttribute( arcLenNode.sampleInAttribute ) # Calls the MPxNode.addAttribute function.
    ...
    arcLenNode.sampleOutAttribute = numericAttributeFn.create( 'myOutputAttribute', 'o',
                                                       OpenMaya.MFnNumericData.kFloat )
    ...
    arcLenNode.addAttribute( arcLenNode.sampleOutAttribute )
    ...
    arcLenNode.attributeAffects( arcLenNode.sampleInAttribute, arcLenNode.sampleOutAttribute )
```

4. Modify the input and output attributes

First, let us change the input and output attribute variable names. You can use any name you that want. Here, I will change `sampleInAttribute` to `inputCurve` and `sampleOutAttribute` to `output`.

You will also change the data handle variables `sampleInDataHandle` to `inputCurveDataHandle` and `sampleOutDataHandle` to `outputDataHandle`.

Then, you will change the long name that's passed into the create function – `myInputAttribute` to `inputCurve` and `myOutputAttribute` to `output`.

```
class arcLenNode(OpenMaya.MPxNode):

    inputCurve  = OpenMaya.MObject()
    output = OpenMaya.MObject()
    ...
    def compute(self, pPlug, pDataBlock):
        if( pPlug == arcLenNode.output ):

            inputCurveDataHandle = pDataBlock.inputValue( arcLenNode.inputCurve  )
            outputDataHandle = pDataBlock.outputValue( arcLenNode.output )

            sampleInValue = inputCurveDataHandle.asFloat()

            outputDataHandle.setFloat( sampleInValue )

            outputDataHandle.setClean()
            ...
...
def nodeInitializer():
    ...
    arcLenNode.inputCurve  = numericAttributeFn.create( 'inputCurve', 'i',
                                                         OpenMaya.MFnNumericData.kFloat,
                                                         sampleDefaultValue )
    ...
    arcLenNode.addAttribute( arcLenNode.inputCurve  ) # Calls the MPxNode.addAttribute function.
    ...
    arcLenNode.output = numericAttributeFn.create( 'output', 'o',
                                                   OpenMaya.MFnNumericData.kFloat )
    ...
    arcLenNode.addAttribute( arcLenNode.output )
    ...
    arcLenNode.attributeAffects( arcLenNode.inputCurve , arcLenNode.output )
```

Lastly, change the input to receive a typed attribute. Currently, the node's input attribute is a floating-point number. You will change this so that the node will take in a NURBS curve instead. In the `nodeInitializer` function, add a typed attribute function set (`MFnTypedAttribute`) to create the input attribute. Since the input will be a NURBS curve and not a float number, use the typed attribute function set to create it instead of `MFnNumericAttribute`.

You will also change the attribute type from `MFnNumericData.kFloat` to `MFnData.kNurbsCurve`. You can remove the default value since it is optional, and a typed attribute will have a value of null object by default.

```
def nodeInitializer():

    numericAttributeFn = OpenMaya.MFnNumericAttribute()
    typedAttributeFn = OpenMaya.MFnTypedAttribute()

    arcLenNode.inputCurve = typedAttributeFn.create( 'inputCurve', 'i',
                                                     OpenMaya.MFnData.kNurbsCurve )
    ...
```

5. Calculate the arc length

Since the input attribute can now receive NURBS curve, you need to change the way it is being extracted. Instead of the method `asFloat()`, use `asNurbsCurveTransformed()` instead. This method will extract the actual value as a curve, so you can change the variable name from `sampleInValue` to `curve` for it be more readable.

To calculate the arc length of the NURBS curve, use the function set `MFnNurbsCurve`. It contains a method called `length` that returns the arc length of the curve. The resulting length will be set as the output.

```
class arcLenNode(OpenMaya.MPxNode):
    ...
    def compute(self, pPlug, pDataBlock):
        if( pPlug == arcLenNode.output ):
            ...

            curve = inputCurveDataHandle.asNurbsCurveTransformed()

            # ... perform the desired computation ...
            curveFn = OpenMaya.MFnNurbsCurve( curve )
            arcLenResult = curveFn.length()

            # Set the output value.
            outputDataHandle.setFloat( arcLenResult )
            ...
```

Final plug-in code

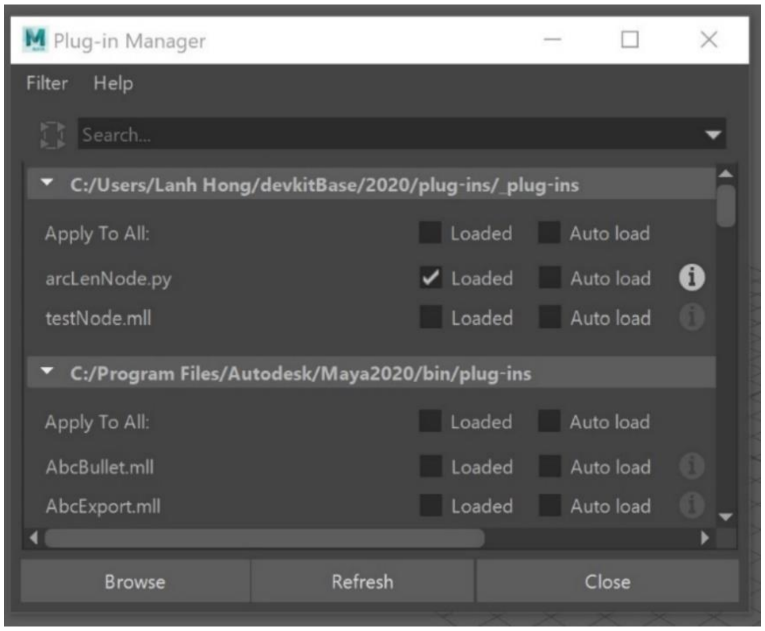
Congratulations! You are done with your plug-in! You just recreated the `arcLenNode` C++ sample in Python. Your final plug-in may look like this:

```
import sys
import maya.api.OpenMaya as OpenMaya
# ... additional imports here ...

def maya_useNewAPI():
    """
    The presence of this function tells Maya that the plugin produces, and
    expects to be passed, objects created using the Maya Python API 2.0.
    """
    pass

# Plug-in information:
kPluginNodeName = 'arcLenNode'              # The name of the node.
kPluginNodeClassify = 'utility/general'     # Where this node will be found in the Maya UI.
kPluginNodeId = OpenMaya.MTypeId( 0x00136300 ) # A unique ID associated to this node type.

##########################################################
# Plug-in
##########################################################
class arcLenNode(OpenMaya.MPxNode):
    # Static variables which will later be replaced by the node's attributes.
    inputCurve = OpenMaya.MObject()
    output = OpenMaya.MObject()

    def __init__(self):
        ''' Constructor. '''
        OpenMaya.MPxNode.__init__(self)

    def compute(self, pPlug, pDataBlock):
        '''
        Node computation method.
        - pPlug: A connection point related to one of our node attributes (could be an input or an output)
        - pDataBlock: Contains the data on which we will base our computations.
        '''
        if( pPlug == arcLenNode.output ):
            # Obtain the data handles for each attribute
            inputCurveDataHandle = pDataBlock.inputValue( arcLenNode.inputCurve )
            outputDataHandle = pDataBlock.outputValue( arcLenNode.output )

            # Extract the actual value associated to our input attribute
            # (we have defined it as a nurbs curve)
            curve = inputCurveDataHandle.asNurbsCurveTransformed()

            # Get the arc length
            curveFn = OpenMaya.MFnNurbsCurve( curve )
            arcLenResult = curveFn.length()

            # Set the output value.
            outputDataHandle.setFloat( arcLenResult )

            # Mark the output data handle as being clean; it need not be computed given its input.
            outputDataHandle.setClean()
        else:
            return OpenMaya.kUnknownParameter

##########################################################
# Plug-in initialization.
##########################################################
def nodeCreator():
    ''' Creates an instance of our node class and delivers it to Maya as a pointer. '''
    return  arcLenNode()

def nodeInitializer():
    ''' Defines the input and output attributes as static variables in our plug-in class. '''
    # The following function sets will allow us to create our attributes.
    numericAttributeFn = OpenMaya.MFnNumericAttribute()
    typedAttributeFn = OpenMaya.MFnTypedAttribute()

    #==================================
    # INPUT NODE ATTRIBUTE(S)
    #==================================
    arcLenNode.inputCurve = typedAttributeFn.create( 'inputCurve', 'i',
                                                     OpenMaya.MFnData.kNurbsCurve )
    typedAttributeFn.writable = True
    typedAttributeFn.storable = True
    typedAttributeFn.hidden = False
    arcLenNode.addAttribute( arcLenNode.inputCurve ) # Calls the MPxNode.addAttribute function.

    #==================================
    # OUTPUT NODE ATTRIBUTE(S)
    #==================================
    arcLenNode.output = numericAttributeFn.create( 'output', 'o',
                                                   OpenMaya.MFnNumericData.kFloat )
    numericAttributeFn.storable = False
    numericAttributeFn.writable = False
    numericAttributeFn.readable = True
    numericAttributeFn.hidden = False
    arcLenNode.addAttribute( arcLenNode.output )

    #==================================
    # NODE ATTRIBUTE DEPENDENCIES
    #==================================
    # If inputCurve changes, the output data must be recomputed.
    arcLenNode.attributeAffects( arcLenNode.inputCurve, arcLenNode.output )

def initializePlugin( mobject ):
    ''' Initialize the plug-in '''
    mplugin = OpenMaya.MFnPlugin( mobject )
    try:
        mplugin.registerNode( kPluginNodeName, kPluginNodeId, nodeCreator,
                              nodeInitializer, OpenMaya.MPxNode.kDependNode, kPluginNodeClassify )
    except:
        sys.stderr.write( 'Failed to register node: ' + kPluginNodeName )
        raise

def uninitializePlugin( mobject ):
    ''' Uninitializes the plug-in '''
    mplugin = OpenMaya.MFnPlugin( mobject )
    try:
        mplugin.deregisterNode( kPluginNodeId )
    except:
        sys.stderr.write( 'Failed to deregister node: ' + kPluginNodeName )
        raise
```

Load and test the plug-in

To load it in Maya, make sure your Python file is saved in the correct directory. Earlier, you had to set the path variable `MAYA_PLUG_IN_PATH` when setting up your environment variables. This is the path that Maya will look for your custom plug-ins. The file name must have the “py” extension. I named mine `arcLenNode.py`.

Inside Maya, you can load the plug-in in two different ways – using `loadPlugin` MEL command or the Plug-in Manager. The Plug-in Manager is located in *Windows > Settings/Preferences > Plug-in Manager*.



LOAD YOUR CUSTOM PLUG-INS FROM THE PLUG-IN MANAGER.

After you have loaded the plug-in, run the following MEL script in the Script Editor to test your plug-in. The script will create a new `arcLenNode` node and a NURBS circle. It will connect the NURBS curve to the `inputCurve` attribute, and then prints out the results to the Script Editor.

```
createNode arcLenNode;
CreateNURBSCircle;
connectAttr nurbsCircleShape1.local arcLenNode1.inputCurve;
getAttr arcLenNode1.output;
```

Explanation

Now that you are able to load and use the plug-in, let's briefly examine the code to deepen your understanding of what you've built.

Import modules

You started off my importing the modules needed to build the plug-in. Here, you imported the `OpenMaya` module from the Maya API. Since you are using Python API 2.0, you must define the function `maya_useNewAPI`.

```
import sys
import maya.api.OpenMaya as OpenMaya
# ... additional imports here ...

def maya_useNewAPI():
    """
    The presence of this function tells Maya that the plugin produces, and
    expects to be passed, objects created using the Maya Python API 2.0.
    """
    pass
```

Plug-in information

You defined some of the plug-in information, which includes the plug-in node name, its classification, and node ID. Since this node will be used as a general or utility node, you classified it as `utility/general`. The node ID is a unique number that help distinguish your node from another registered node.

```
# Plug-in information:
kPluginNodeName = 'arcLenNode'              # The name of the node.
kPluginNodeClassify = 'utility/general'     # Where this node will be found in the Maya UI.
kPluginNodeId = OpenMaya.MTypeId(  0x87EFE  ) # A unique ID associated to this node type.
```

Initialize and uninitialized plug-in

The `initializePlugin` and `uninitializePlugin` functions are the entry and exit points for your plug-in. When your plug-in gets loaded in Maya, it calls the `initializePlugin` and `uninitializePlugin` functions. These functions will register and deregister the node in Maya using the `MFnPlugin` function set and its `registerNode` and `deregisterNode` functions.

Registering the node requires some information about the node such as the name, ID, creator and initializer methods, node type, and classification.

```
def initializePlugin( mobject ):
    ''' Initialize the plug-in '''
    mplugin = OpenMaya.MFnPlugin( mobject )
    try:
        mplugin.registerNode( kPluginNodeName, kPluginNodeId, nodeCreator,
                              nodeInitializer, OpenMaya.MPxNode.kDependNode, kPluginNodeClassify )
    except:
        sys.stderr.write( 'Failed to register node: ' + kPluginNodeName )
        raise

def uninitializePlugin( mobject ):
    ''' Uninitializes the plug-in '''
    mplugin = OpenMaya.MFnPlugin( mobject )
    try:
        mplugin.deregisterNode( kPluginNodeId )
    except:
        sys.stderr.write( 'Failed to deregister node: ' + kPluginNodeName )
        raise
```

Creator function

Once **`initializePlugin`** gets called, Maya calls the **`nodeCreator`** function next. It gets called to create an instance of **`arcLenNode`** and returns it to Maya.

```
def nodeCreator():
    ''' Creates an instance of our node class and delivers it to Maya as a pointer. '''
    return  arcLenNode()
```

Initializer function

The **`nodeInitializer`** function gets called after the **`nodeCreator`** function, and this is where the node's input and output attributes get created and set up.

You have a single input attribute called **`inputCurve`** that takes in a NURBS curve. Since a NURBS curve is a typed attribute, you used the **`MFnTypedAttribute`** function set to create one. The **`output`** will be a floating-point number, therefore using **`MFnNumericAttribute`** to create the attribute.

When creating the attributes using the **`create`** function, you passed in some information such as the attribute's long and short name, its type, and an optional default value. You may set the attributes to be writable, storable, hidden, etc.

After creating the attributes, you add the attributes to the node and make it accessible to the dependency graph using the **`addAttribute`** function from **`MPxNode`** class.

Then you set up the dependencies for your attributes using **`attributeAffects`**. This will ensure that any changes to the inputs will recompute the value for the output attribute.

```
def nodeInitializer():
    ''' Defines the input and output attributes as static variables in our plug-in class. '''
    # The following function sets will allow us to create our attributes.
    numericAttributeFn = OpenMaya.MFnNumericAttribute()
    typedAttributeFn = OpenMaya.MFnTypedAttribute()

    #==================================
    # INPUT NODE ATTRIBUTE(S)
    #==================================
    arcLenNode.inputCurve = typedAttributeFn.create( 'inputCurve', 'i',
                                                      OpenMaya.MFnData.kNurbsCurve )
    typedAttributeFn.writable = True
    typedAttributeFn.storable = True
    typedAttributeFn.hidden = False
    arcLenNode.addAttribute( arcLenNode.inputCurve ) # Calls the MPxNode.addAttribute function.

    #==================================
    # OUTPUT NODE ATTRIBUTE(S)
    #==================================
    arcLenNode.output = numericAttributeFn.create( 'output', 'o',
                                                    OpenMaya.MFnNumericData.kFloat )
    numericAttributeFn.storable = False
    numericAttributeFn.writable = False
    numericAttributeFn.readable = True
    numericAttributeFn.hidden = False
    arcLenNode.addAttribute( arcLenNode.output )

    #==================================
    # NODE ATTRIBUTE DEPENDENCIES
    #==================================
    # If inputCurve changes, the output data must be recomputed.
    arcLenNode.attributeAffects( arcLenNode.inputCurve, arcLenNode.output )
```

Node definition

Here is where you define the `arcLen` node and its behavior. The `arcLenNode` class extends the proxy class `MPxNode` for Maya to recognize the plug-in as a node. The node's attributes are `inputCurve` and `output`, which were added to the node using the `addAttribute` function in the `nodeInitializer` function earlier.

The `__init__` function is a constructor, and it gets executed when you create an instance of `arcLenNode`. Since the class extends the `MPxNode` class, it will call the constructor for that as well.

The `compute` function is where the bulk of the work is done. The dependency graph uses an evaluation process that marks plugs as *dirty* when the output values are stale and needs to be recomputed. When a plug is *dirty*, Maya calls the `compute` function on the node and passes in the *dirty* plug and its corresponding data block. The variables `pPlug` and `pDataBlock` are instances of `MPlug` and `MDataBlock`.

In the `compute` function, it checks whether the plug passed in is an output plug before calculating a new result. The data block that is passed in contains data handles which are `MDataHandle` instances. They will store and handle the values for their respective attributes. Input data handles are for reading only and can extract the actual values of the input. Output data handles are for writing only and can set the output values. The input data handle uses the function `asNurbsCurveTransformed` to extract the NURBS curve while the output data handle uses `setFloat` to set the output.

The `MFnNurbsCurve` class is used to access and operate on the input curve. It has a function called `length` that returns the arc length. The output data handle will use that result and set it as the output value. At the end of all the computation, the output data handle removes the output plug's *dirty* flag by marking it as *clean*.

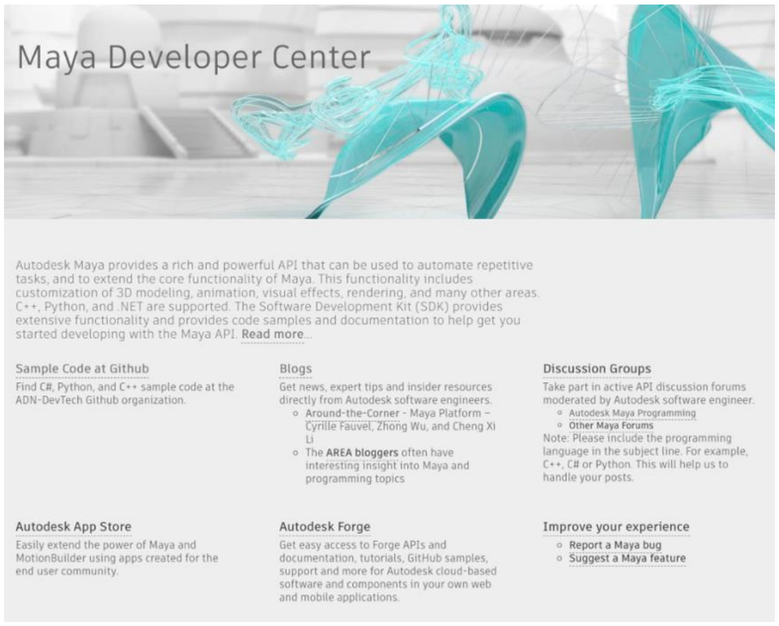
```
class arcLenNode(OpenMaya.MPxNode):
    # Static variables which will later be replaced by the node's attributes.
    inputCurve = OpenMaya.MObject()
    output = OpenMaya.MObject()

    def __init__(self):
        ''' Constructor. '''
        OpenMaya.MPxNode.__init__(self)

    def compute(self, pPlug, pDataBlock):
        '''
        Node computation method.
        - pPlug: A connection point related to one of our node attributes (could be an input or an output)
        - pDataBlock: Contains the data on which we will base our computations.
        '''
        if( pPlug == arcLenNode.output ):
            # Obtain the data handles for each attribute
            inputCurveDataHandle = pDataBlock.inputValue( arcLenNode.inputCurve )
            outputDataHandle = pDataBlock.outputValue( arcLenNode.output )

            # Extract the actual value associated to our input attribute
            # (we have defined it as a nurbs curve)
            curve = inputCurveDataHandle.asNurbsCurveTransformed()

            # Get the arc length
            curveFn = OpenMaya.MFnNurbsCurve( curve )
            arcLenResult = curveFn.length()

            # Set the output value.
            outputDataHandle.setFloat( arcLenResult )

            # Mark the output data handle as being clean; it need not be computed given its input.
            outputDataHandle.setClean()
        else:
            return OpenMaya.kUnknownParameter
```

Resources

Learning the Maya API can be tough, but there are many resources available to help you get started developing and continue this journey. I have only listed a few here, but it should be enough to help you get started and get through any issues you may be facing.

Maya Developer Center

[https://www.autodesk.com/developmaya]
The Maya Developer Center is an essential place to visit when you are developing your Maya plug-in. Here, you will find the download links for the devkit for recent Maya releases and updates. You will also find some learning materials to help you get started with your development journey. Links to code samples, blogs, whitepapers, and other resources are posted here, making it easy to find what you are looking for. Important tools for developing your plug-ins such as registering a Maya Node ID Block can be found here as well.



Maya Developer Center

Autodesk Maya provides a rich and powerful API that can be used to automate repetitive tasks, and to extend the core functionality of Maya. This functionality includes customization of 3D modeling, animation, visual effects, rendering, and many other areas. C++, Python, and .NET are supported. The Software Development Kit (SDK) provides extensive functionality and provides code samples and documentation to help get you started developing with the Maya API. **Read more**...

Sample Code at Github
Find C#, Python, and C++ sample code at the ADN-DevTech Github organization.

Blogs
Get news, expert tips and insider resources directly from Autodesk software engineers.
- Around-the-Corner - Maya Platform – Cyrille Fauvel, Zhong Wu, and Cheng Xi Li
- The **AREA bloggers** often have interesting insight into Maya and programming topics

Discussion Groups
Take part in active API discussion forums moderated by Autodesk software engineer.
- Autodesk Maya Programming
- Other Maya Forums

Note: Please include the programming language in the subject line. For example, C++, C# or Python. This will help us to handle your posts.

Autodesk App Store
Easily extend the power of Maya and MotionBuilder using apps created for the end user community.

Autodesk Forge
Get easy access to Forge APIs and documentation, tutorials, GitHub samples, support and more for Autodesk cloud-based software and components in your own web and mobile applications.

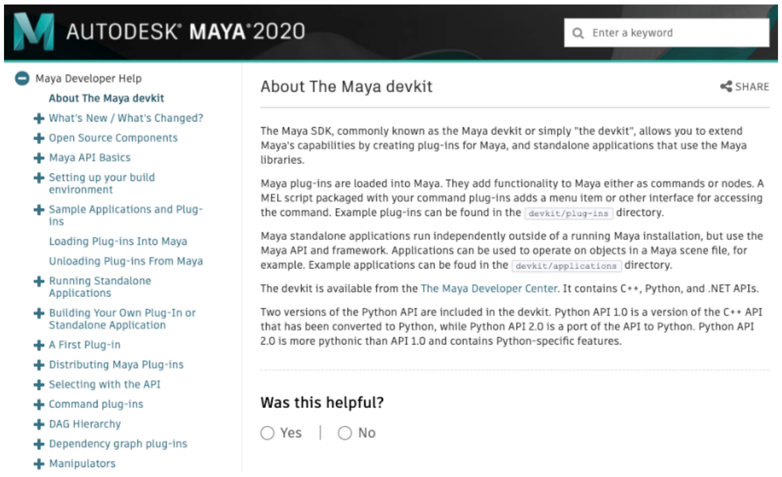
Improve your experience
- Report a Maya bug
- Suggest a Maya feature

MAYA DEVELOPER CENTER

Maya Developer Help Documentation

[https://www.autodesk.com/me-sdk-docs]
The best place to learn Maya development is in the Maya Developer Help documentation page. There are a ton of information that goes in depth about specific topics, along with many code samples. Links to the C++ and Python API references are located here, and you will be looking through these pages quite often. The API reference pages is where you'll find the classes and functions you need to build your plug-in.

The link above will take you to the documentation for the latest M&E product release. Just scroll down to Previous Releases to access the Maya 2020 developer documentation page.



- Maya Developer Help
  - **About The Maya devkit**
  - What's New / What's Changed?
  - Open Source Components
  - Maya API Basics
  - Setting up your build environment
  - Sample Applications and Plug-ins
  - Loading Plug-ins Into Maya
  - Unloading Plug-ins From Maya
  - Running Standalone Applications
  - Building Your Own Plug-In or Standalone Application
  - A First Plug-in
  - Distributing Maya Plug-ins
  - Selecting with the API
  - Command plug-ins
  - DAG Hierarchy
  - Dependency graph plug-ins
  - Manipulators

About The Maya devkit

The Maya SDK, commonly known as the Maya devkit or simply "the devkit", allows you to extend Maya's capabilities by creating plug-ins for Maya, and standalone applications that use the Maya libraries.

Maya plug-ins are loaded into Maya. They add functionality to Maya either as commands or nodes. A MEL script packaged with your command plug-ins adds a menu item or other interface for accessing the command. Example plug-ins can be found in the `devkit/plug-ins` directory.

Maya standalone applications run independently outside of a running Maya installation, but use the Maya API and framework. Applications can be used to operate on objects in a Maya scene file, for example. Example applications can be foud in the `devkit/applications` directory.

The devkit is available from the The Maya Developer Center. It contains C++, Python, and .NET APIs.

Two versions of the Python API are included in the devkit. Python API 1.0 is a version of the C++ API that has been converted to Python, while Python API 2.0 is a port of the API to Python. Python API 2.0 is more pythonic than API 1.0 and contains Python-specific features.

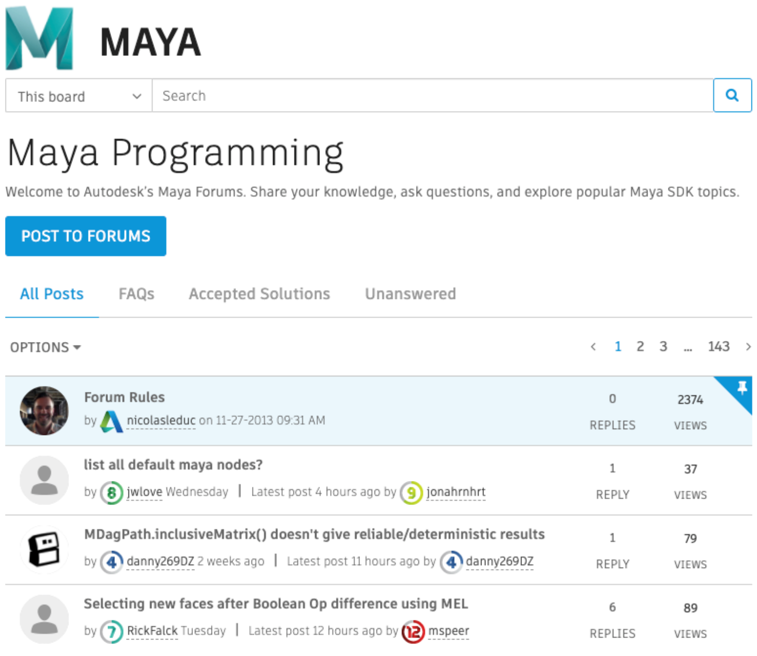
Was this helpful?

Yes | No

MAYA DEVELOPER HELP DOCUMENTATION

Maya Programming Forum

[https://forums.autodesk.com/t5/maya-programming/bd-p/area-b50]
The Maya programming forum is a community area for developers to get support from other community members. This is the place where you can share your knowledge and contribute to the Maya developer community. You may browse the forum to see if anyone else have the same questions as you, and you might find that there's already a solution. You can also post your questions or issues to the forum and get support from the community.



Maya Programming

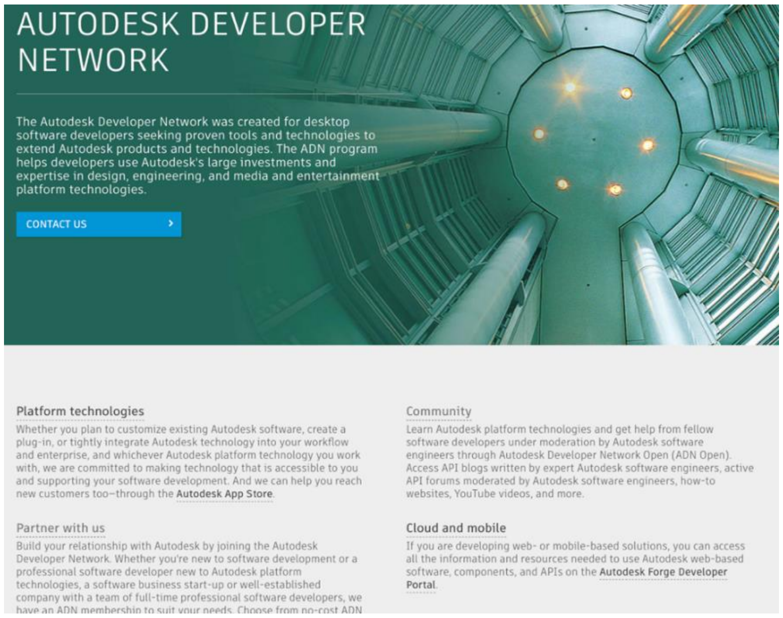
Welcome to Autodesk's Maya Forums. Share your knowledge, ask questions, and explore popular Maya SDK topics.

Forum Rules by nicolasleduc on 11-27-2013 09:31 AM	0 REPLIES	2374 VIEWS
list all default maya nodes? by 8 jwlove Wednesday | Latest post 4 hours ago by 9 jonahrnhrt	1 REPLY	37 VIEWS
MDagPath.inclusiveMatrix() doesn't give reliable/deterministic results by 4 danny269DZ 2 weeks ago | Latest post 11 hours ago by 4 danny269DZ	1 REPLY	79 VIEWS
Selecting new faces after Boolean Op difference using MEL by 7 RickFalck Tuesday | Latest post 12 hours ago by 12 mspeer	6 REPLIES	89 VIEWS

Maya Programming Forum

Autodesk Developer Network (ADN)

[https://www.autodesk.com/developer-network]
The Autodesk Developer Network (ADN) is a developer program aiming to support developers using Autodesk's technologies to extend and customize Autodesk desktop products. ADN Open is a no cost membership which consists of resources available to the public. If you need a more direct help for your development, you may upgrade to ADN Standard or ADN Professional to get one-on-one support.



AUTODESK DEVELOPER NETWORK

The Autodesk Developer Network was created for desktop software developers seeking proven tools and technologies to extend Autodesk products and technologies. The ADN program helps developers use Autodesk's large investments and expertise in design, engineering, and media and entertainment platform technologies.

Platform technologies

Whether you plan to customize existing Autodesk software, create a plug-in, or tightly integrate Autodesk technology into your workflow and enterprise, and whichever Autodesk platform technology you work with, we are committed to making technology that is accessible to you and supporting your software development. And we can help you reach new customers too—through the **Autodesk App Store**.

Partner with us

Build your relationship with Autodesk by joining the Autodesk Developer Network. Whether you're new to software development or a professional software developer new to Autodesk platform technologies, a software business start-up or well-established company with a team of full-time professional software developers, we have an ADN membership to suit your needs. Choose from no-cost ADN

Community

Learn Autodesk platform technologies and get help from fellow software developers under moderation by Autodesk software engineers through Autodesk Developer Network Open (ADN Open). Access API blogs written by expert Autodesk software engineers, active API forums moderated by Autodesk software engineers, how-to websites, YouTube videos, and more.

Cloud and mobile

If you are developing web- or mobile-based solutions, you can access all the information and resources needed to use Autodesk web-based software, components, and APIs on the **Autodesk Forge Developer Portal**.

AUTODESK DEVELOPER NETWORK