Reading C++ Documentation

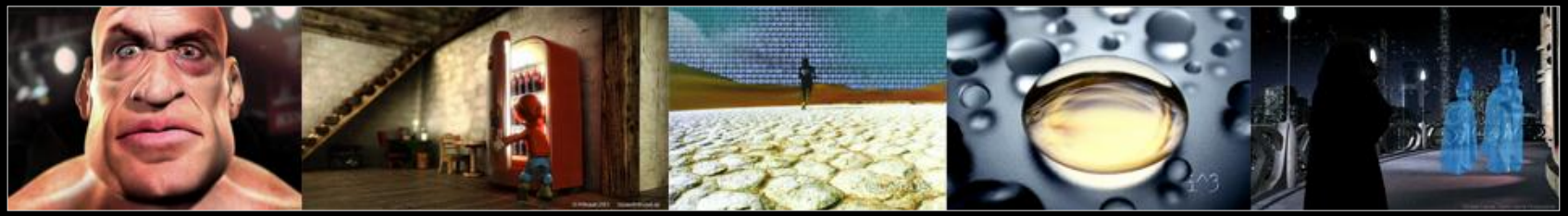
Kristine Middlemiss, *Senior Developer Consultant*
Autodesk Developer Network (ADN)

Autodesk

Session Agenda

- Docs are written for C++
- Differences between C++ and Python API

Docs are written for C++



Autodesk®

Python API Documentation

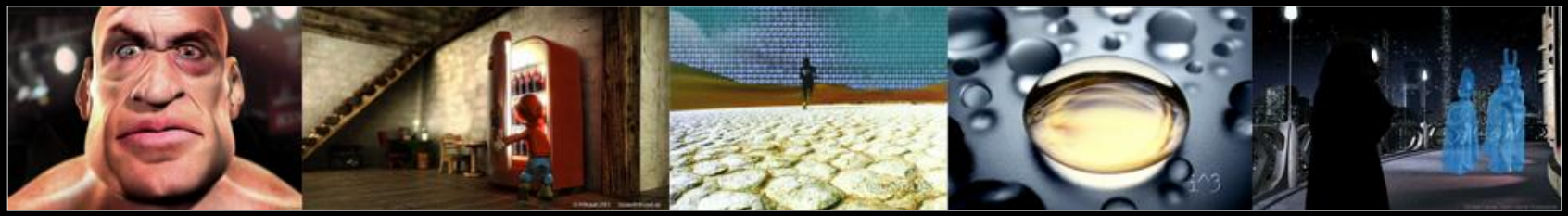
- No Python-specific API class documentation
- Methods not supported in Python:
  - Marked by: NO SCRIPT SUPPORT
- In most cases, alternate forms are provided

C++ Learning Curve in Docs

Public Member Functions

	MSelectionList ()
	MSelectionList (const MSelectionList &src)
virtual	~MSelectionList ()
MStatus	clear ()
bool	isEmpty (MStatus *ReturnStatus=NULL) const
unsigned int	length (MStatus *ReturnStatus=NULL) const
MStatus	getDependNode (unsigned int index, MObject &depNode) const
MStatus	getDagPath (unsigned int index, MDagPath &dagPath, MObject &component=MObject::kNullObj) const
MStatus	getPlug (unsigned int index, MPlug &plug) const
MStatus	add (const MObject &object, const bool mergeWithExisting=false)
MStatus	add (const MDagPath &object, const MObject &component=MObject::kNullObj, const bool mergeWithExisting=false)
MStatus	add (const MString &matchString, const bool searchChildNamespacesToo=false)
MStatus	add (const MPlug &plug, const bool mergeWithExisting=false)
MStatus	remove (unsigned int index)
MStatus	replace (unsigned int index, const MObject &item)
MStatus	replace (unsigned int index, const MDagPath &item, const MObject &component=MObject::kNullObj)
MStatus	replace (unsigned int index, const MPlug &plug)
bool	hasItem (const MObject &item, MStatus *ReturnStatus=NULL) const
bool	hasItem (const MDagPath &item, const MObject &component=MObject::kNullObj, MStatus *ReturnStatus=NULL) const
bool	hasItem (const MPlug &plug, MStatus *ReturnStatus=NULL) const
bool	hasItemPartly (const MDagPath &item, const MObject &component, MStatus *ReturnStatus=NULL) const

Differences between C++ and Python API



Autodesk®

Maya Python API Plug-ins

- Several modules were constructed that expose most of the C++ API functionality:
  - OpenMaya.pyc
  - OpenMayaAnim.pyc
  - OpenMayaRender.pyc
  - OpenMayaUI.pyc
  - OpenMayaMPx.pyc
  - OpenMayaCloth.pyc

OpenMaya Key Points

- Parellels C++ libraries, except for maya.OpenMayaMPx
- All MPx proxy classes are collected into maya.OpenMayaMPx for technical reasons.
- Python API is a thin wrapper around C++ API
- Full support of C++ Maya API functionality
- SWIG based bindings created from header files
- Modules: $MAYA_LOCATION/Python/lib/site-packages/maya

Python API Scripts: Example

- Most code translates readily from C++ to Python

```
# DG modifier to change perspective camera translateX
import maya.OpenMaya as om
sel = om.MSelectionList()
om.MGlobal.getSelectionListByName( "persp“, sel )
dagPath = om.MDagPath()
sel.getDagPath( 0, dagPath )
dagPath.fullPathName()
# Result: |persp #

mod = om.MDGModifier()
mod.commandToExecute( "setAttr persp.tx 5" )
mod.doIt()
maya.cmds.getAttr( "persp.tx" )
# Result: 5 #
mod.undoIt()
# Result: 28 #
```

Python API vs. C++ API

- No MStatus class - use exceptions instead

```
try:
    fnPlugin.registerCommand( “spLs", cmdCreator )
except:
    sys.stderr.write( "Command registration failed")
    raise
```

- Catch error if registerCommand() fails

Python API vs. C++ API

- Detect error but allow code to keep running

```
try:
    fnPlugin.registerCommand( “spLs", cmdCreator )
except:
    sys.stderr.write( "Command registration failed")
    pass
```

- Keyword pass used instead of raise

Python API vs. C++ API

- MString and MStringArray classes have been replaced by Python native strings and string lists

```
import maya.OpenMaya as om
sel = om.MSelectionList();
om.MGlobal.getSelectionListByName( "persp", sel )
path = om.MDagPath()
sel.getDagPath(0, path )
myString = path.fullPathName()
print "Path is %s" % myString
# Path is |persp #
```

```
myStrings = []
sel.getSelectionStrings( myStrings )
print myStrings
# [u'persp'] #
```

Python API vs. C++ API

- Python does not have a concept of pointers
- Use MScriptUtils for working with pointers and references
  - Reference treated as a pointer
- MScriptUtils
  - creates objects that can be passed as pointer or reference parameters
  - convenience methods for transferring values between these objects and native Python datatypes

Python API vs. C++ API

(C++) int MImage::getSize(unsigned int& width, unsigned int& height)

(Python)

```
img = OpenMaya.MImage()
img.create( 512, 256 )

util = OpenMaya.MScriptUtil()
util2 = OpenMaya.MScriptUtil()
wPtr = util.asUintPtr() # creates a ptr object
hPtr = util2.asUintPtr() # creates another ptr object
OpenMaya.MScriptUtil.setUint( wPtr, 0 )
OpenMaya.MScriptUtil.setUint( hPtr, 0 )
img.getSize( wPtr, hPtr )
width = OpenMaya.MScriptUtil.getUint( wPtr ) # 512
height = OpenMaya.MScriptUtil.getUint( hPtr ) # 256
```

Python API vs. C++ API

- MPxCommand has to use MSyntax

- In C++, it is possible to use the argument parameter (MArgList) of MPxCommand::doIt() to parse arguments.

- In Python, you must use the MSyntax and MArgParser classes to support arguments within a scripted MPxCommand.

Python API Scripts: Caveats

- Careful about undo
  - API functionality is not automatically undoable in the same way that MEL commands are
  - With Python API code operating outside of an MPxCommand-derived class, there is no formal interface to allow you to implement your own undo behaviour
- Careful with scripted plug-ins
  - Importing the .py file is not the same as loading it from the Plug-in Manager
  - Will not register new commands/nodes

Autodesk