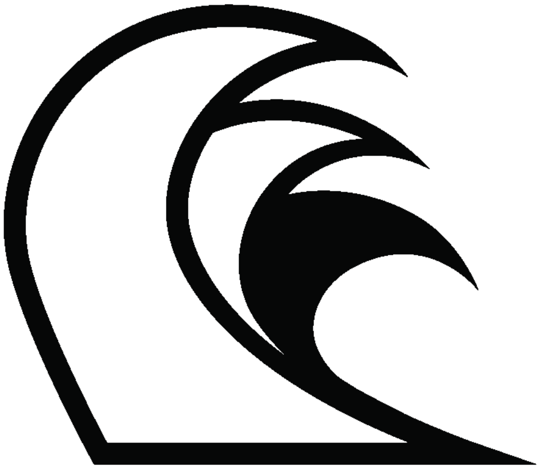
Programming in MotionBuilder || Focusing on Python
Autodesk MotionBuilder 2013

Autodesk Developer Network
May 2012

Module 2: The Beginning: Python SDK



AUTODESK®
MOTIONBUILDER

Contents

Programming in MotionBuilder || Focusing on Python

Autodesk Developer Network

Module 2: The Beginning: Python SDK

Agenda

- Working with the Python Version
- Navigating MotionBuilder Install folders
- How to Execute Scripts
- The Python Editor
- Programming Preferences
- MotionBuilder Python Modules
- Starting to Build Code for MotionBuilder
- Helpful coding standards

2.0 Working with the Python Version

Python Version in MotionBuilder

- MotionBuilder 2013 uses Python version 2.6.4 (make sure you check the Python version in every release of MotionBuilder as it can change). However you will find we will always have the same Python version as Autodesk Maya.
- There is no way to change the Python version in MotionBuilder you have to use the version that it comes with.

- Starting MotionBuilder 2010, the python installation shipped with MotionBuilder includes the entire standard Python install package – a benefit to users who no longer need to download the entire package from http://www.python.org to make use of specific functions that ship with the standard Python install. You can find the standard python modules here:

  C:\Program Files\Autodesk\MotionBuilder 2013\bin\x64\python\DLLs

Including External Python Modules in MotionBuilder

- Setting up external modules so that MotionBuilder will locate them, there are two options:

  1. MotionBuilder uses the environment variable PYTHONPATH and PATH, so you can create or download any modules from the internet and point the PYTHONPATH to their location and MotionBuilder will find them so that you may use them inside of MotionBuilder.

     If you have MotionBuilder open when you set the environment variables you need to restart MotionBuilder as it reads the environmental variables on startup only.

  2. The other option for MotionBuilder is copying the regular *.py* modules into the MotionBuilder 'Python' folder located here:

     C:\Program Files\Autodesk\MotionBuilder 2013\bin\config\Python

     However if you don't have administrative access to this folder on windows 7, you could put your module into
     %USERPROFILE%\Documents\MB\2013-x64\config\Python
     or

     %USERPROFILE%\Documents\MB\2013-x64\config\PythonStartup

For Binary modules, such as *.pyd* they must be placed in a specific folder located here:

> *C:\Program Files\Autodesk\MotionBuilder 2013\bin\x64\python\DLLs*

This assumes you have administrative rights on this folder, if not, you will need to go with the first option.

2.1 Navigating MotionBuilder Install Folders

There are two main folders that you need to be familiar with when programming in MotionBuilder, the folders that we will explore in depth are:

1. bin
2. OpenRealitySDK

From the default installation location all folders sit inside of this folder:

C:\Program Files\Autodesk\MotionBuilder 2013

For any of the below folder or files we will look at, if you edit any of them while MotionBuilder is open you need to restart MotionBuilder for the changes to take effect. These folders and files are only read on MotionBuilder application startup.

The 'bin' Folder

This folder is for customizing MotionBuilder, setting preferences, and working with Python, think of it as the main hub of the MotionBuilder installation, everything else is frills. Inside the bin folder there are two sub-folders that are important to explore, these folders are:

1. *config*
2. *x64 (or win32)*

1. *The 'config' Folder*
    - *The 'Python' Folder*
        - This is the Python install base information and modules that MotionBuilder Python uses.
        - When importing MotionBuilder Python modules this is where the application looks for them.
        - *The 'pyfbsdk_additions.py' File*

        - There are two modules that make up MotionBuilder Python SDK, this is one and the other one ‘pyfbsdk.pyd’ (explained below).
        - This module is imported when you want to create and work with Python UI.
    - modules are used inside the Python Editor tool window
        - *The ‘completion.py’ File*, Used for Auto Completion
        - *The ‘mbdebugger.py’ File*, Used to Debug Python Scripts
        - *The ‘mbsyntax.py’ File*, Used for Syntax Highlighting
        - *The ‘PythonKeyboard.txt’ File*
          This controls the keyboard shortcuts for the Python Editor, the Python editor needs to have the focus for you to be able to execute the shortcuts (don’t confuse this with the application shortcuts)
    - *The ‘KeyboardDocumentation.txt’ File*
        - This documentation tells you what all the Keyboard Shortcuts are, explains them in sentences.
    - *The ‘ToolManager.py’ File*
        - This file is utility script that is used to open and re-open tools that you place in this folder when a user in the application closes them. This is a MotionBuilder default script, do not delete because it is called by the Python Editor ‘Pop tool manger’ button (deleting it will cause you to lose this functionality).
- *The ‘PythonStartup’ Folder*
    - Place your Python tools here so that they are available for users when the MotionBuilder application startups up. To have the tools be physically open on startup you have to save them in the default ‘Layout’ or you will have to use the ‘Pop tool manager’ button inside the Python Editor to make them appear. They do not instantly appear if you have done neither of the above things.
- *The ‘Scripts’ Folder*
    - This is the Python samples folder, we have sub divided the samples into types scripts they are.
    - *The ‘ActionScript.txt’ File*
        - This file is used to associate scripts to keyboard shortcuts defined in the keyboard shortcuts mapping files.

2. *The ‘Platform’ Folder (For example ‘x64)*
    - *The ‘plugins’ Folder*
        - Put your compiled plug-ins here (unless you change your Preferences > SDK to point to somewhere else).
        - On startup MotionBuilder registers your Plug-in and puts them in the correct location in the application. For example if your plug-in is a tool it will show up in the File Menu “Open Reality > Tools”, if your plug-in is a custom constraints it will be located in the Asset Browser > Constraints, custom filters will be in Window > Filters, etc.
    - *The ‘Python’ Folder*
        - *The ‘DLLs’ Folder*
            - This is where all the Python Standard Library is installed, such as socket, hashlib, tkinter etc.
        - *The ‘x64\python\lib\\pyfbsdk.pyd’ File*
            - This is where the Python SDK complete functionality module is located, when you call ‘import pyfbsdk’ in your scripts this is what file it is accessing.
        - This is also where you can store platform specific Python modules that you would like to import via MotionBuilder.

The ‘OpenRealitySDK’ Folder

1. *The ‘include’ Folder*
    - This is where all the Open Reality SDK header files are located, you need these for compiling OR SDK plug-ins
2. *The ‘lib’ Folder*
    - This is where all the Open Reality import libraries are located, you need these for linking to your OR SDK plug-ins
3. *The ‘Samples’ Folder*
    - This is where all the Open Reality Samples are located; make sure you use it a lot as it is a gold mine of information.
    - Inside this folder they are subdivided in to separate folders of like plug-in types.
4. *The ‘Scenes’ Folder*

  - These are the scenes and accessory files that are used by certain plug-in in the above ‘Samples’ folder for the Open Reality SDK to demonstrate certain functionality.

5. The “Scripts” Folder
  - These are the scripts you would use to test on some Open Reality plug-in samples. For example, orpyfbsdk_template.py in this folder is the testing script included in the pyfbsdk_template sample project.

Re-iterating the point of how important the sample folders are

You need to use the samples that come with MotionBuilder for both OpenReality and Python to see how to use MotionBuilder programming workflows, if you are not use any samples your learning curve maybe be very steep.

User Configuration Directory

Other than these above folders we talked about under MotionBuilder default installation location, there is also a very important directory you need to know and may use it quite often in your production activity. This is a new feature introduced with MotionBuilder 2012. It is called “User Configuration Directory”:

In the latest version of Windows, only administrative users have the permission to write to the Program File directory. To enable non-administrative users to customize MotionBuilder’s configuration files, MotionBuilder creates the following directory for each user that opens MotionBuilder.

C:\Users\[user]\Documents\MB\2013-x64\config\

Under this directory, you will find it has almost the same folder structure as MotionBuilder’s installation “config” directory. It has “CharacterTemplate”, “Keyboard” ,“Layout” and “PythonStartup” subfolders etc, which allows you to specify your own keyboard shortcuts, startup programs and scripts, character templates and layouts. For example, you can put python scripts into the “PythonStartup” folder under this directory to have your scripts run when MotionBuilder starts up. Also, you can centralized all your working/testing scripts by putting them into the “Scripts” folder under this directory.

Besides these folders, there are several text files prefixed with your computer name, let's take a look at two important ones:

- *The '{MyComputerName}.Application.txt' File*
    - There is nothing programming specific stored in this file, but I list it here to bring it to your attention. This is where a lot of preferences are stored from MotionBuilder. There are several occasions where you call a function's value and it retrieves it's default values from this file. For example, if you do not set the SamplingPeriod from the class FBDevice, it will take the value from this file.
- *The '{MyComputerName}.Python.txt' File*
    - This file is used by the Python Editor to store what scripts you had open the last time you closed MotionBuilder. When you re-open MotionBuilder as long as the scripts are in the same path it will re-open the scripts again. This is a system file so you do not need to worry about editing it.

With MotionBuilder 2012, we also introduced a new environment variable MB_CONFIG_DIR, which you can optionally set to override the default location of the MotionBuilder configuration files.

2.2 How to Execute Scripts

There are 7 main ways to execute a Python scripts in MotionBuilder, they follow under these categories:

1. Launching Script from Command Line Startup
2. Drag and Drop Script from the Asset Browser
3. From the Python Editor
4. Places Python Tools in 'PythonStartup' Folder
5. Running Script on Hot Key Initiation
6. Using the Function 'ExecuteScript' in the class 'FBApplication'
7. Triggering a script via a script device inside a Relation Constraint

Launching Script from Command Line Startup

Running MotionBuilder by command line can be a very effective way to assist you in optimizing your pipeline and helping you automate certain tasks. To helping you automate tasks Python is very handy to help you out, you can start MotionBuilder up with a Python script and the script will be executed once MotionBuilder is fully launched.

For launching a script via command line a generic syntax would look like this.

```
motionbuilder.exe [flags] [Python script or filename]
```

The flags are optional (for a full list of flags, refer to Appendix 1)

Drag and Drop Script from the Asset Browser

Drag and drop the script from the Asset Browser to the Viewer, which gives you this option box:

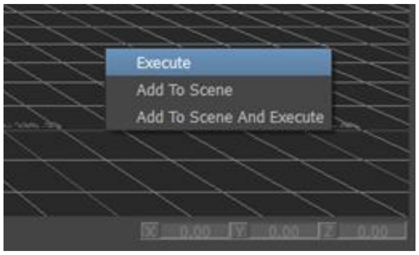


Figure 1 | This is the Option Box you get when you Drag and Drop a Script to the Viewer

Here you can choose from one of three script execution options:

1. Execute
    - This executes the script immediately.
2. Add to Scene
    - This adds it to the Scene you are working on, you can see it in the Navigator under a section called 'Scripts'

   - The reason you would want to add the script to the scene is for two reasons:
      1. If you are running a particular script a lot, it makes it to add it to the scene and so you can easily execute the script by dragging and dropping it in the Viewer.
      2. You can add scripts to relation constraints, and the script can be triggered on certain conditions that you set up. For example if you want a script to execute only when a particular property has a value of 1.
   - Keep in mind that even though the script appears to be embedded in the FBX file, it still has a path property back to the original location that it is from, so if you move the file you need to update the path property to the correct path or the script will no longer run.
3. Add to Scene and Execute
   - This option does both 1 and 2 at the same time.

TIP: If this is the first time using MotionBuilder you will need to add the path to your script directory in the Asset Browser.

- Right click on the folder view on the left side of the Asset Browser
- Select 'Add favorite paths

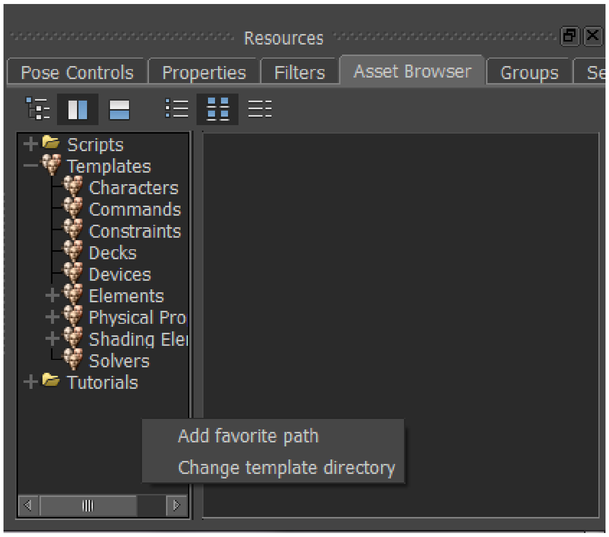


Figure 2 | How to Add Folders to the Asset Browser

- Using the Brower Explorer navigate to your scripts directory on your computer (for me it is ..\bin\config\Scripts)

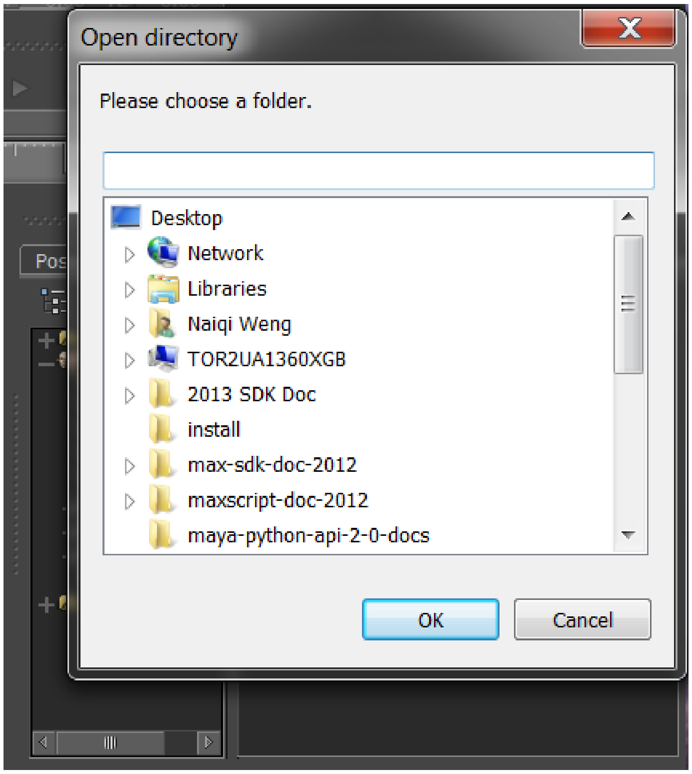


Figure 3 | Navigating to the Folder you would like to Add

From the Python Editor

MotionBuilder now comes with a built-in Python Editor that is very convenient and efficient to work with, no more having an external editor open and working back and forth between it and MotionBuilder to test your scripts. To open the new Python Editor you select Windows > Python Editor in the MotionBuilder menu. Inside the Editor there are 3 main sways to execute your Python scripts and code:

1. "Execute Active Work Area" Button
    - Scripts
        - To execute your script you must save the file every time before you try and execute it, if you do not a popup will occur asking you to save it.
        - If the mouse focus is in the Python Editor, the hotkey Ctrl+S works to save the script, it's nice to get into the habit of using this, and it makes your workflow more efficient.
        - All scripts must start with by importing of the MotionBuilder Python module for them to execute correctly without errors.
    - Code
        - To execute particular parts of your code, you can do so by highlighting the code and clicking the 'Execute Active Work Area' button or even more conveniently using the hotkey Ctrl+Enter.
        - When executing highlighted code you do not need to have the import MotionBuilder Python modules statement as part of the highlighted code to have it execute, it is automatically imported.
2. "Add to Scene as Script" Button
    - This adds it to the Scene you are working on, you can see it in the Navigator under a section called 'Scripts'
    - The reason you would want to add the script to the scene is for two reasons:
        - If you are running a particular script a lot, it makes it to add it to the scene and so you can easily execute the script by dragging and dropping it in the Viewer.
        - You can add scripts to relation constraints, and the script can be triggered on certain conditions that you set up. For example if you want a script to execute only when a particular property has a value of 1.
    - Keep in mind that even though the script appears to be embedded in the FBX file, it still has a path property back to the original location that it is from, so if you move the file you need to update the path property to the correct path or the script will no longer run.
3. Use Command Line Python, Top Half (console) of Python Editor
    - This console window behaves just like the Python Interpreter IDLE; it is a single line interpreter, so it allows for quick easy testing without having to save a file.

- You need to execute the import MotionBuilder Python modules statement as it is not automatically imported in this command line Python console.

Place Python Tools in 'PythonStartup' Folder

Put your Python tools (code that have a python UI) in the default location or where your 'PythonStartup' folder is set to in the Python Preferences (for more information see the below section 2.4 Programming Preferences). When MB launches the script it will automatically be available to use in the 'Pop tool manager' or in a custom layout.

Running Script on Hot Key Initiation

If you want your script to be executed when a user performs a certain keystroke or sequence you can easily do this will a little set up in these two files:

1. Editing 'ActionScript.txt' File
   - This is the file you store the path and the name of the script you want to execute as a hotkey, please see the documentation in the file as it explains in much more detail how the process works.
   - The 'ActionScript.txt' file is located in the \bin\config\Scripts\ directory.
2. Editing 'Keyboard Configuration' File
   - This file assigns a keyboard shortcut mapping to your script that you specified in the 'ActionScript.txt' file.
   - The keyboard shortcut mapping files are located in the \bin\config\Keyboard directory.
   - The keyboard shortcut mapping file to use (MotionBuilder, MotionBuilder Classic, Maya, etc.) depends on your preferences settings (Settings >Keyboard Configuration).

This feature was working with MotionBuilder 2010 and previous versions. Unfortunately there is a bug introduced and this approach does not work properly for current version MotionBuilder 2013.

Using the Function FBApplication::ExecuteScript

In the class FBApplication (see section 2.6 Starting to Build Code for MotionBuilder)

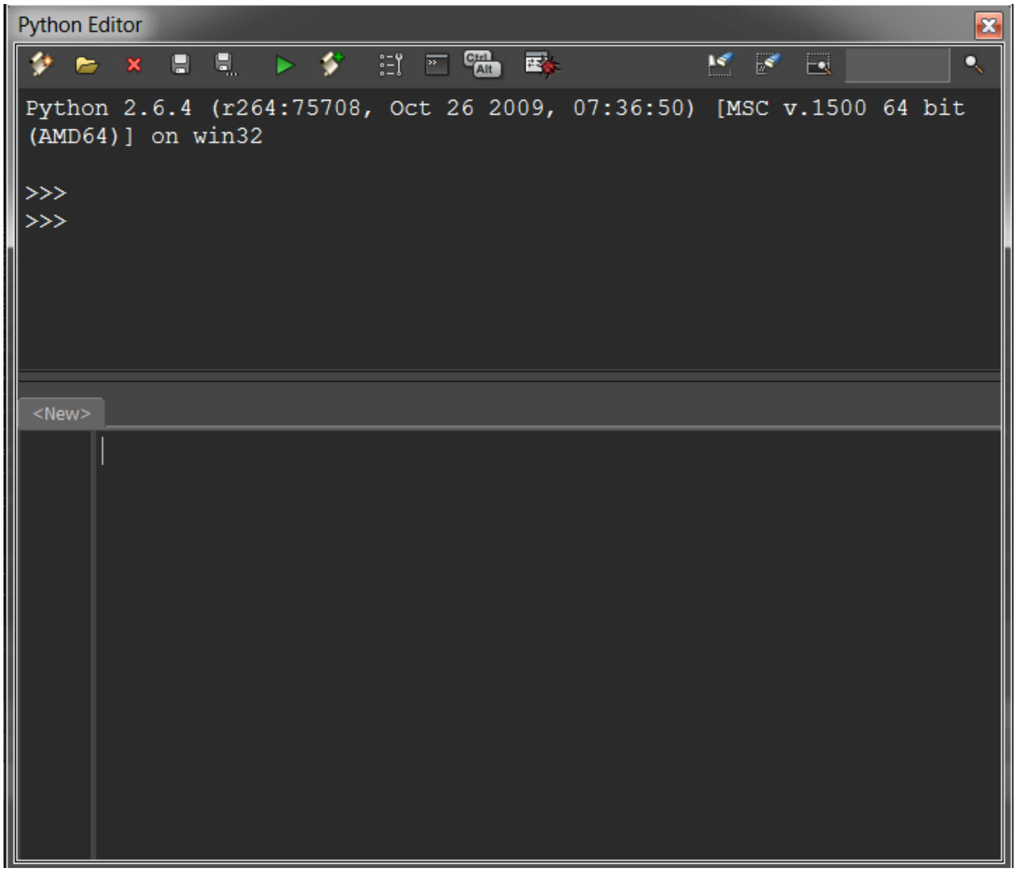
Triggering a script via a script device inside a Relation Constraint

For more details on this see Module 8: Constraints.

2.3 The Python Editor

We have made a lot of changes Here we will go on a little UI tour of the editor so you know how to use it to the fullest.

To access the new Python Editor, you can locate it in Windows > Python Editor, at the very bottom of the list.



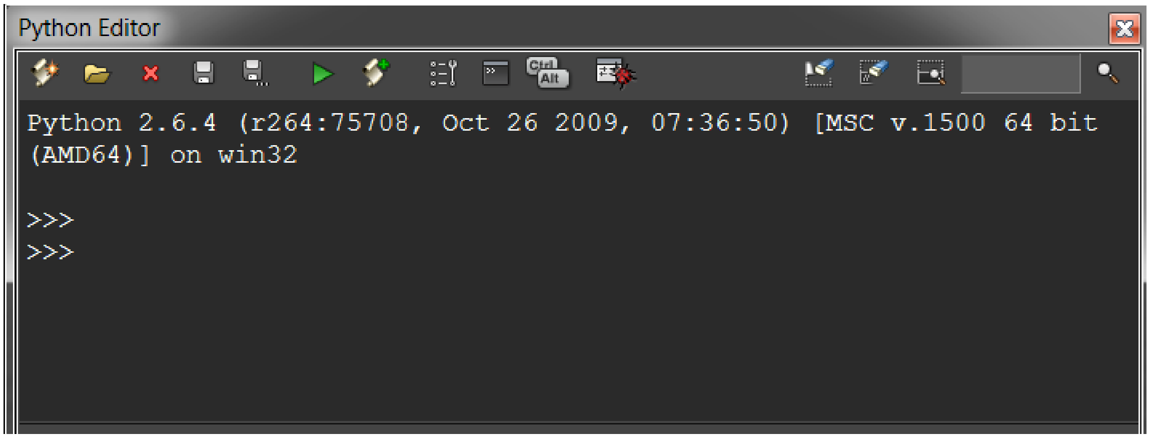
```
Python 2.6.4 (r264:75708, Oct 26 2009, 07:36:50) [MSC v.1500 64 bit
(AMD64)] on win32

>>>
>>>
```

Figure 4 | The Python Editor

The Python Console

The Python console is the top half of the Python Editor; it is the area that allows you to execute one line Python code just like the command line Interpreter called IDLE that comes with the Standard version of Python 2.6. When using the console the MotionBuilder Python modules are automatically imported so you do not need to do this. This console is here to allow for quick and easy code testing without the hassle of importing and saving script files. The feature list of the python console is:

- single or multi-line entry
- command history
- code auto-completion
- error feedback on executed scripts or code



```
Python 2.6.4 (r264:75708, Oct 26 2009, 07:36:50) [MSC v.1500 64 bit
(AMD64)] on win32

>>>
>>>
```

Figure 5 | The Python console

To View a list of Python Editor Shortcuts, enter each of these lines followed by the enter key:

```
import pythonidelib
print pythonidelib.GetShortcutInfo()
```

You will find the output results are difficult to read. To make it easier, execute the following line as a whole (selecting the lines and press enter key):

```
import pythonidelib
d = pythonidelib.GetShortcutInfo()
for k,v in d.iteritems():
    print k, ":", v
```

you will see the following outputs in the Python console, they are the Python console shortcuts.

```
Custom command 4 : {SHFT:F4*DN}
Custom command 8 : {SHFT:F8*DN}
debug current script : {NONE:F7*DN}
find : {CTRL:F*DN}
Custom command 10 : {SHFT:F10*DN}
close script : {CTRL:-*DN}
uncomment selection : {ALT:U*DN}
open script : {CTRL:O*DN}
pop tool manager : {NONE:F10*DN}
execute script : {NONE:F5*DN}
```

```
save as script : {NONE:F2*DN}
new script : {CTRL:T*DN}
next tab : {CTRL:.*DN}
indent block of text : {NONE:TAB*DN}
next command : {ALT:N*DN}
unindent block of text : {SHFT:TAB*DN}
save script : {CTRL:S*DN}
Custom command 7 : {SHFT:F7*DN}
comment out selection : {ALT:C*DN}
go to line : {CTRL:G*DN}
Search and replace : {CTRL:H*DN}
add script to scene : {CTRL:=*DN}
Custom command 9 : {SHFT:F9*DN}
previous command : {ALT:P*DN}
show all shortcuts : {NONE:F1*DN}
Custom command 2 : {SHFT:F2*DN}
Custom command 3 : {SHFT:F3*DN}
Custom command 1 : {SHFT:F1*DN}
Custom command 6 : {SHFT:F6*DN}
find next : {NONE:F3*DN}
previous tab : {CTRL:,*DN}
Custom command 5 : {SHFT:F5*DN}
```

The Python Active Work Area

This is the main area that you will spend almost all of your Python programming time in. As this in essence is the Python Editor. It has several nice usability functions that make it competitive to any commercial scripting editor, these are:

- line numbers
- code auto-completion
- code colorization
- code comment and uncomment
- code indent and de-dent
- drag and drop scripts
- multiple script workspaces

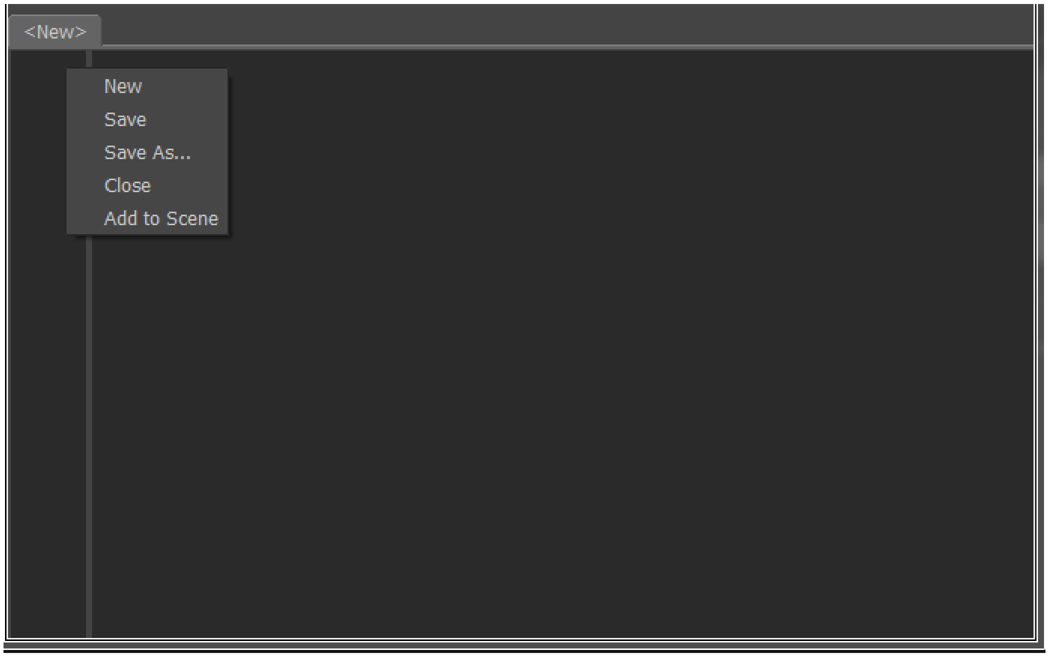


Figure 6 | The Active Work Area

If you right click your mouse on the tab of your script, (in this case it is where the words <New> are) you will see an additional menu that allows you all the options that are available in the toolbar at the top of the Python Editor. This allows for a different method to access these functionalities, purely here to accommodate different users workflow styles. The one exception is the ‘Save As’ option, this is not available in the toolbar after you save your script the first time, so this is very convenient to have if you would like to rename your script once it is saved.

The main difference from toolbar at the top of the Python Editor vs. the right click of the tab, is that you do not need to make the tab active first, you can just right click on any tab by hovering your mouse over it (so in theory it never became the active tab). However, for the toolbar at the top of the Python Editor, you can only perform these actions if the tab is active first.

Exploring the Python Editor Toolbar and Functionality

New Tab – This opens a new empty script for you in a new tab in the lower half of the Python Editor

Open Script – This open a browser window for you to locate a script from your computer that you would like to open up in a new tab in the lower half of the Python Editor.

Close Script – This closes the script/tab that currently has the focus and is visible in the lower half of the Python Editor.

Save to File – This saves the script/tab that currently has focus and is visible in the lower half of the Python Editor. If you have not saved the script before, this button will behave like a Save As button and allow you to select the name and location on your computer of where you would like to save it.

Save As… -- This Opens the Save File browser where you specify the file name and location of the new script you want to save.

Execute Active Work Area – This allows you to execute code or a script, depending on whether any code is highlighted, if nothing is highlighted in your script/tab that has focus, it will execute the whole script for you.

Add to Scene as Script – This adds the current script that has focus to the current scene, if you have not saved your script it does not prompt you do so, so make sure you save before adding to the scene to ensure all your changes are captured.

Pop tool manager – This allows you to select from the tool scripts that you have available in the 'PythonStartup' folder.

Pop telnet console – This opens a local tenet console that is automatically connected to MotionBuilder, the MotionBuilder Python modules are already imported.

Contextual Help – This works similarly like the help command in Python, it displays in the console what you have selected, the keyboard shortcuts when the console is empty or highlighted text in the interactive console or work area.

Debug Current Script – This provided you a tool to debug the current, which allows you to create a log file to assist you in your debugging practices.

Clear Console – This clears the top half (console) of the Python Editor from any previously outputted history or text.

Clear Active Work Area – This clears the bottom half (active work area) of the Python Editor from any code. **Warning:** This deletes all of your code.

Exploring the Python Editor Search Functionality

Using the search in the Python Editor is a three step process which is explained below:

Step 1 or or *Search in Editor* – This is a toggle button which indicated where you want to search in the top half (console), the bottom half (active work area), or both parts in the Python Editor

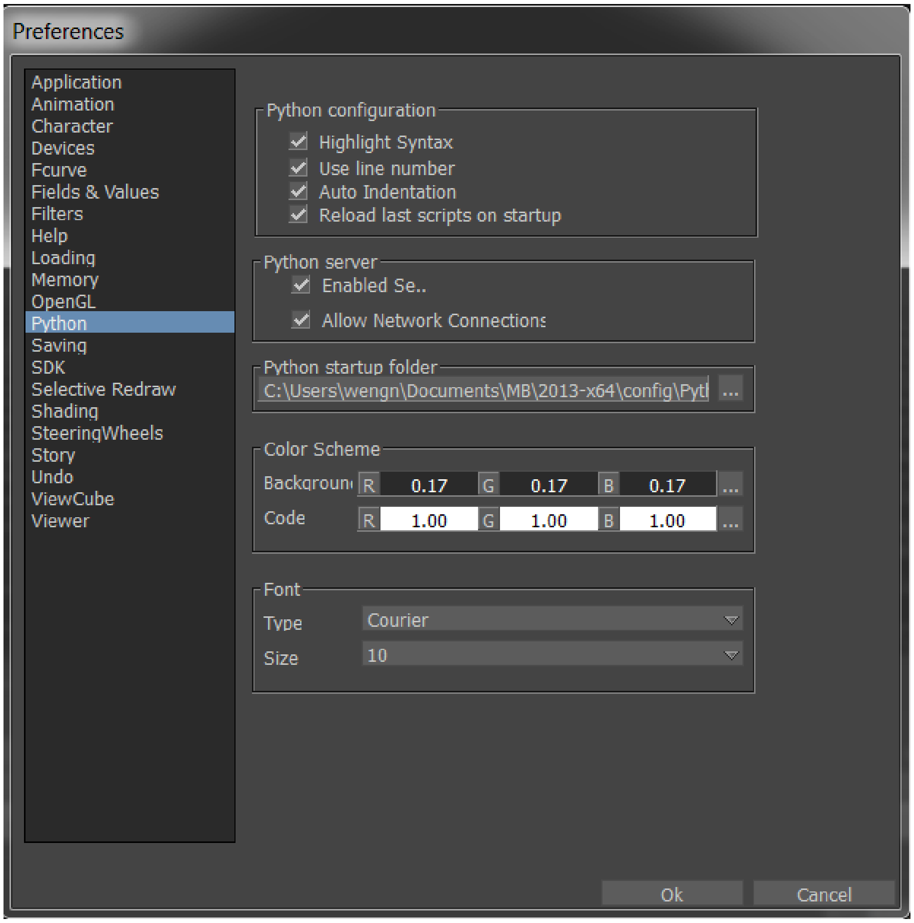
Step 2 *Search Text Field* – Type in the word you would like to look for in the Python Editor

Step 3 *Find Text* – Click this button when you have performed step 1 and 2, this will jump to the first instance of the matching word, and if you continue to click this button it will go through and find every instance of the word in the area that is indicated in step 1.

2.4 Programming Preferences

In this version of MotionBuilder we have created more control over programming than ever before. In Settings > Preferences… you can now create some customization around Python and the Open Reality SDK.

Python Preferences



Python configuration
☑ Highlight Syntax
☑ Use line number
☑ Auto Indentation
☑ Reload last scripts on startup

Python server
☑ Enabled Se..
☑ Allow Network Connections

Python startup folder
C:\Users\wengn\Documents\MB\2013-x64\config\Pyt ...

Color Scheme

	R		G		B	
Backgroun	R	0.17	G	0.17	B	0.17
Code	R	1.00	G	1.00	B	1.00

Font
Type: Courier
Size: 10

Figure 7 | Python Preferences

Python Configuration

- These are setting that you can choose from to customize the appearance and behaviours or the Python Editor

Python Server

- These setting are for working with the telnet console. If you are going to be running and sending MotionBuilder commands from another computer you need to enable these.

Python Startup Folder

- This is the location that MotionBuilder looks for your scripts on startup, the default location is here:

  C:\Users\[user]\Documents\MB\2013-x64\config\PythonStarup

Colour Scheme

- This allows you to customize the look of the Python Editor; the default is white background with black text.
  - A lot of people that are use to Linux and VI, they might change their colour scheme to a black background with white text.
- You must re-start MotionBuilder for colours to take affect

Font

- This allows you to control the font type and size of the Python Editor
- These settings are stored in the @.Application.txt file in the section title [Python]
- You must re-start MotionBuilder for new fonts to take affect

Environment Variable for Python Startup Directory

We introduced a new environment variable to set up MotionBuilder startup python directory, “MOTIONBUILDER_PYTHON_STARTUP_PATH”.
The list of directories contained in this environment variable determines the location from which MotionBuilder will load Python startup scripts

SDK Preferences

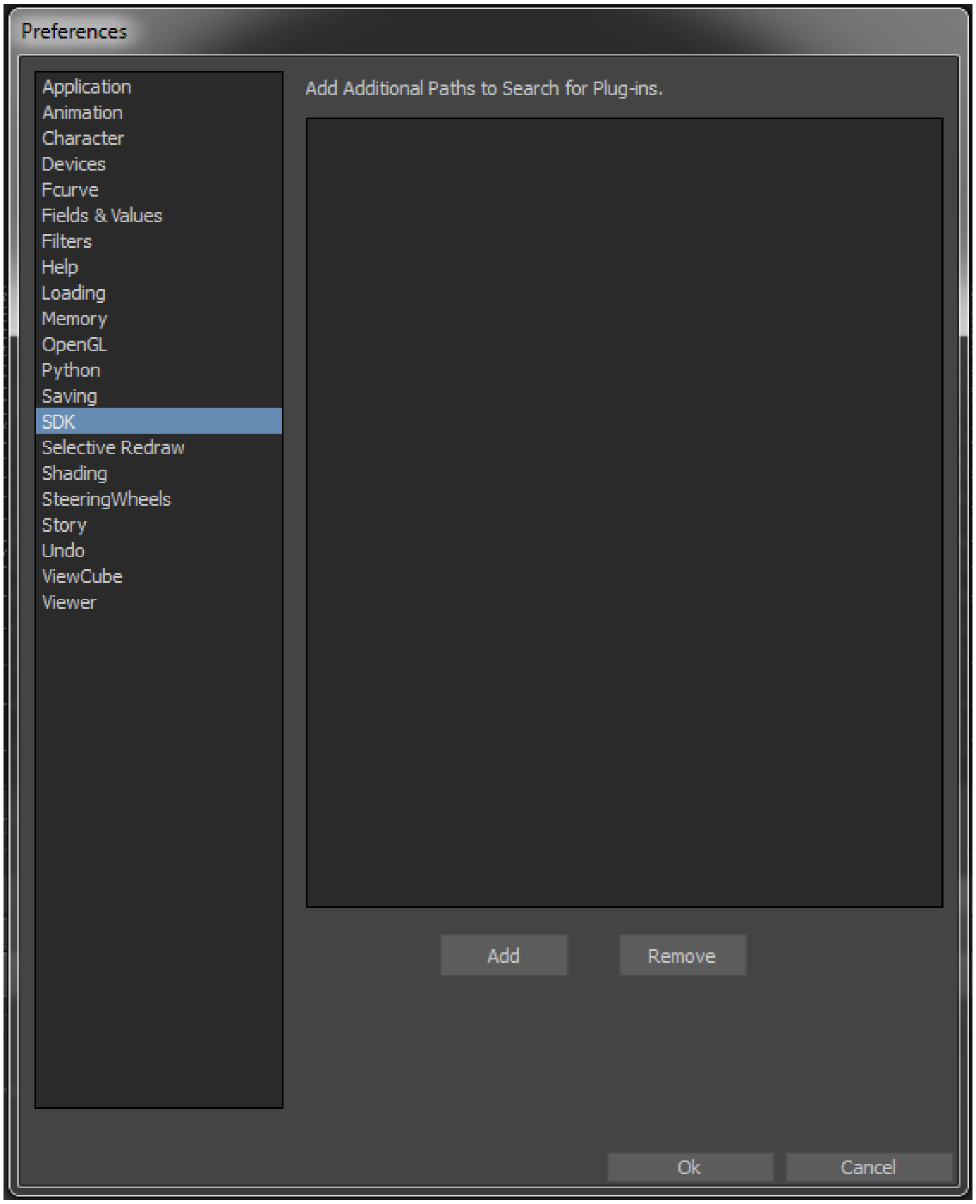


Figure 8 | SDK Preferences

Add Additional Paths to Search for Plug-ins

- This is the locations that MotionBuilder looks for your compiled plug-ins on startup, the default location is here:
  \bin\{platform}\plugins
- There is no limit to how many paths you would like to add, however if you added 100 hundred paths remember MotionBuilder needs to look in each of those locations before it starts up.

Environment Variable for Plug-ins Path

We also add this new environment variable MOTIONBUILDER_PLUGIN_PATH, which includes a list of directories representing location from which MotionBuilder will load compiled plugins

2.5 MotionBuilder Python SDK Modules

All of MotionBuilder functionality resides in two Python Modules. The functionality is very clearly divided between the two modules which make it easy for each programmer to remember which module they need to import when working with certain classes. These modules are:

1. **pyfbsdk**
    - This module is the major module (and was the only module prior to 2010), it contains all of the classes that are in the Python Reference Documentation, so a perfect rule of thumb is if the class is documented in the Python Reference Documentation, it belongs to this module. The MotionBuilder Python API is contained in the pyfbsdk module. This module was generated using the Boost.Python library, which binds C++ functions to Python functions. The functions contained in this module therefore have analogous signatures to the MotionBuilder C++ ORSDK.
2. **pyfbsdk_additions**
    - This module is used only for building UI for tools; it mainly consists of helper classes based on UI classes from the pyfbsdk module. There is no class documentation on this module so you will need to refer to the module file itself for documentation and functions (see section 2.1 Navigating MotionBuilder Install Folders for where it is located).

3. Import modules

To use the Python MotionBuilder API in your scripts, include the following lines:

```
from pyfbsdk import *

from pyfbsdk_additions import *
```

NOTE: there is a third module you can use called ‘pythonidelib’, but this is purely a utility module for viewing shortcuts and small functionality for the Python Editor (see section 2.3 The Python Console for more details)

TIP: Ever wonder why things in MotionBuilder have the acronym ‘FB’? This is because prior to 2002 MotionBuilder use to be called FilmBox, i.e. FBModel.

2.6 Starting to Build Code for MotionBuilder

Finally hooray!! We're here, only 35 pages later and were finally ready to dive in and start programming in MotionBuilder (that's the course title, right?), we will start slow and from the beginning since this is a beginner training, however the way it will work is we will be learn some programming then some theory then some more programming hopefully continually building and understanding how MotionBuilder works, so that you can get to the point where if you don't know the answer you can figure it out based on all your knowledge you know.

The 'FBApplication' Class

- This class manages input and output routines, new, open, import, export, merge, render and batch, think of the class roughly as what the File menu in the UI can do this class can do.
- The FBApplication class is primarily used for file input and output operations such as importing, exporting, merging, and batch file processing.
- It also can be used to do application level operations such as minimizing, maximizing and closing MotionBuilder application.

Examples

Creating a new scene file

```
from pyfbsdk import *

#Using the class FBApplication

lApp = FBApplication()

lApp.FileNew()
```

The 'FBConfigFile' Class

- This class allows you to generate, modify and query configuration files. Config files will be automatically created when needed. They will be located in the C:\Users\[user]\Documents\MB\2013-x64\config folder or an explicitely specified folder depending on the constructor used.

Examples:

```
from pyfbsdk import *

lApp = FBApplication()
lConf = FBConfigFile("@Application.txt")

lConf.Set("Python","FontSize","16","Default is 10")
lApp.FileExit()
#FontSize = 14; by default it is 10
```

2.7 Helpful Coding Standards

1. Document your code, so that you and others can clearly read it. You can't imagine how many times people come back to code after a few months and it take them twice as long to figure out what is going on in the code because they did not document it and now they can't remember.
2. Document at the top your scripts what the script is about and your name in comments.
3. Name your variables appropriately; our sample code follows a variable naming convention where "l" is used as a prefix for local variables, "m" for class member variables, and "p" for parameter variables.
4. Use spaces, it makes your code easier to read, everything doesn't need to be bunched up, it doesn't go through the interpreter any faster, and you can use it freely, why? Because it's free!

Appendix One: MotionBuilder Command Line Startup Flags

These are additional flags you can use when starting up MotionBuilder from the command line (motionbuilder.exe on Windows). These flags are optional; the flags do not need to be used at all times unless you desire the effect that the flag causes. The generic syntax would look like this:

motionbuilder.exe [flags] [Python script or filename]

Note: A command line cannot consist of a user specifying a python script and a filename for startup because for efficiency your Python script can open the file you want on startup using FBApplication if you need both.

-console	This opens an output window used by FBTrace in the OR SDK, where the appropriate stdout/err stream goes. If you choose to use this console output window for Python output you would need to also specify the –verbosePython flag.
-F [filename]	This flag is not necessary to specify a filename, however if you choose to not use it you will always have to have the filename you want to open at startup last in the parameter list.
-g [width] [heigh]	Sets the window size of MotionBuilder to the values you have specified. Default value will be as large as the screen size.
-S	Starts MotionBuilder in Full Screen Mode, this is the same as choosing Display > Full Screen inside the Viewer. To exit out of Full Screen Mode, hit Alt+Enter.
-setAsFrontMostApp	For Macintosh only (MotionBuilder 7.5). When launching MotionBuilder from a terminal, MotionBuilder looses focus to the terminal. This can cause problems when you are trying to automate with scripts. This flag keeps the focus on MotionBuilder and set it as the front most application.
-suspendMessages	Disables all the warnings and dialogs that pop-up. This flag is useful for automation purposes when you do not want the script to be interrupted by dialog boxes. By default all warnings and dialogs boxes are shown.
-T[UI Name]	Finds a tool with the matching name among the tools that

	MotionBuilder has registered, and if it is found it activates it. This flag parameter is case sensitive. Also note there is not space between the flag name and the UI Name parameter unlike the other flags.
-verbosePython	Outputs all python messages to the appropriate stdout/err stream. This puts the Python print messages to the window that you activate using the console flag as well as to the Python Editor, this is the same location that FBTrace outputs to when using the OR SDK. By default we do not output python output to stdout/err only to the Python Editor.

TIP: If you know every single time you run MotionBuilder you always want a flag to be executed, instead of always running from command line which can be inefficient you can edit your Windows shortcut for the MotionBuilder application to include the flag parameter you want so that when you double click the application icon the flag is executed. A good use of this is for the –console flag.

Examples:

```
//Opening the file mia_blue.fbx on MotionBuilder startup
motionbuilder.exe mia_blue.fbx

//Starts MotionBuilder in Full Screen opening the scene mia_blue.fbx
motionbuilder.exe -S mia_blue.fbx

//Launches the script testScript.py on startup and suppresses all messages boxes that the script might generate.
motionbuilder.exe -suspendMessages testScript.py

//Launches the script Script.py and sends the output to the console output window.
motionbuilder.exe -console -verbosePython Script.py

//Specifies you want the full screen mode to be 500 by 500 with scene mia_blue.fbx open
motionbuilder.exe -g 500 500 -S mia_blue.fbx

//This launches the tool Audio from the OR SDK samples (you need to compile it first) before start up
```

```
motionbuilder.exe -TAudio

//This launches the tool Python Editor and console opening the scene mia_blue.fbx
motionbuilder.exe -console "-TPython Editor" mia_blue.fbx
```