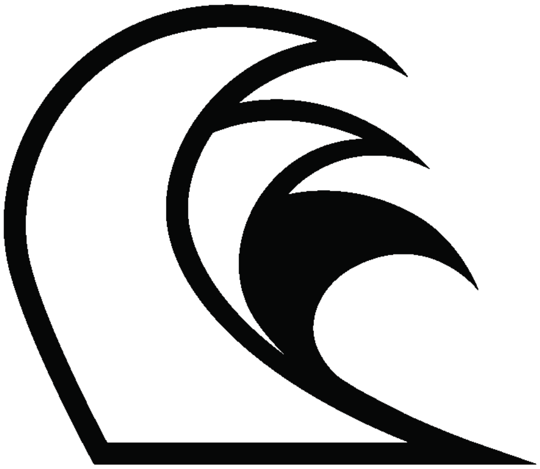
# Programming in MotionBuilder || Focusing on Python

## Autodesk MotionBuilder 2013

Autodesk Developer Network
May 2012
**Module 5: Python UI**



AUTODESK®
MOTIONBUILDER

# Contents

# Programming in MotionBuilder || Focusing on Python

Autodesk Developer Network

Module 5: Python UI

## Agenda

- Pre-Built UI Functionality
- Keeping the user informed via the Viewer Progress Bar
- Building UI Functionality from Scratch (aka a world of possibilities)
- Creating and Registering your Tool
- Creating UI components, buttons, lists, tabs, etc.
- Attaching and playing your UI Components in your Tool
- Additional Helper Classes
- Available Callback Classes for your UI
- Tool Manager and startup scripts
- Other possible UI tool kits and MotionBuilder

## 6.0 Pre-Built UI Functionality

Prior to MotionBuilder 2009, there was not any extensive UI functionality available in Python; however there were a couple pre-built UI components, which allowed for basic data gathering and sharing. These are still available in MotionBuilder 2013, and are recommended if you desire their functionality, why re-invent the wheel. These three pre-pre-build UI components are:

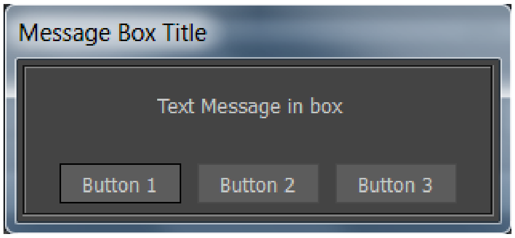
1. Message Box with up to Three Buttons
2. Message Box with up to Three Buttons and a Check Box Option
3. Message Box with up to Three Buttons and allows for user data

All these functions are available from the module pyfbsdk.

### *The 'FBMessageBox' Function*

This creates as a message UI with up to three buttons that you can use to communicate with the user.

**Example Image**



Message Box Title

Text Message in box

Figure 1 | FBMessageBox User Interface

## Function Definition

Int = FBMessageBox (str pBoxTitle, str pMessage, str pButton1Str, str pButton2Str=None, str pButton3Str=None, int pDefaultButton=0, bool pScrolledMessage=False)

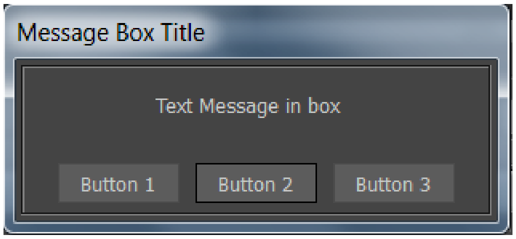
## Example

```
from pyfbsdk import *

indexOfButtonSelected = FBMessageBox( "Message Box Title", "Text Message in box", "Button 1", "Button 2", "Button 3", 0, True )
print indexOfButtonSelected
```

## How the index integer works

- Whichever button you select (1, 2, or 3), this value will be returned by the function call, so if you select “Button 2” the integer 2 will be set to indexOfButtonSelected in the above example.
- This same explanation applies to the 6th parameter where it asks for the default index of which button you would like to pre-select, in other words when you run the script the button that gets selected if a user just hits enter. This is visually represented by the button having a bold line around it:
    - Button 1 has an index of 1
    - Button 2 has an index of 2
    - Button 3 has an index of 3
    - index of 0 means there is no pre-selected button



Message Box Title

Text Message in box

Figure 2 | The FBMessageBox Default Button User Interface

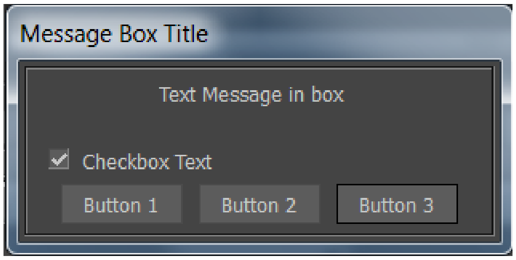
### FBMessageBox Gotchas

- You can’t resume work in MotionBuilder until you click one of the buttons on the Message Box.
- Giving any of the buttons the same name, causes MotionBuilder to freeze.

## *The ‘FBMessageBoxWithCheck’ Function*

This creates as a message UI with up to three buttons and a check box option that you can use to communicate with the user.

### Example Image



Message Box Title

Text Message in box

☑ Checkbox Text

Figure 3 | The FBMessageBoxWithCheck User Interface

### Function Definition

tuple< int, bool > = FBMessageBoxWithCheck (str pBoxTitle, str pMessage, str pButton1Str, str pButton2Str, str pButton3Str, str pCheckBoxStr, bool pCheckBoxValue, int pDefaultButton=0, bool pScrolledMessage=False)

### Example

```
from pyfbsdk import *

ButtonAndCheckBoxResult = FBMessageBoxWithCheck("Message Box
Title","Text Message in box", "Button 1", "Button 2", "Button 3", "Checkbox Text",
True, 3, True)
print "Index of the Button that was selected-> %s \nThe Checkbox's value-> %s"
% (ButtonAndCheckBoxResult[0],ButtonAndCheckBoxResult[1])
```

### How the index integer works

- The index works the same as FBMessageBox

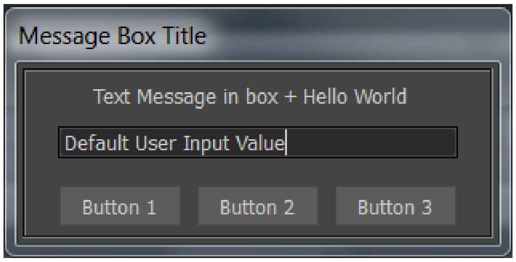
### FBMessageBoxWithCheck Gotchas

- The return value is a tuple containing the index of the button that was selected and the true or false value of the check box.
- You can’t resume work in MotionBuilder until you click one of the buttons on the Message Box.
- Giving any of the buttons the same name, causes MotionBuilder to freeze.

## *The ‘FBMessageBoxGetUserValue’ Function*

This creates as a message UI with up to three buttons that you can use to communicate status with the user and it allows for user input so that you can add custom functionality based on their input.

### Example Image



Message Box Title

Text Message in box + Hello World

Default User Input Value

Figure 4 | The FBMessageBoxGetUserValue User Interface

### Function Definition

tuple< int, str > = FBMessageBoxGetUserValue (str pBoxTitle, str pMessage, object pValue, FBPopupInputType pValueType, str pButton1Str, str pButton2Str=None, str pButton3Str=None, int pDefaultButton=0)

**Example**

```
from pyfbsdk import *

    value = "Hello World"
    ButtonAndUserInputResult = FBMessageBoxGetUserValue( "Message Box
    Title", "Text Message in box + %s"% value, "Default User Input Value",
    FBPopupInputType.kFBPopupString, "Button 1", "Button 2", "Button 3", 1 )
    print "Index of the Button that was selected-> %s \nThe user value that was
    entered-> %s" % (ButtonAndUserInputResult[0],ButtonAndUserInputResult[1])
```

**How the FBPopupInputType works**

- Whichever enumeration type you choose from this class will be the data type of the user input, your options are:
    - kFBPopupBool, True or False
    - kFBPopupChar, one character
    - kFBPopupString, a string
    - kFBPopupInt, integer
    - kFBPopupFloat, float
    - kFBPopupDouble, double
    - kFBPopupPassword, string value shown with '*'

**FBMessageBoxGetUserValue Gotchas**

- The return value is a tuple containing the index of the button that was selected and the value containing the user input of data type specified in the function call.
- You need to make sure you indicate in the functional call the datatype of the user input you desire, or you will not get the correct value passed back. For example if you expect a string and you put kFBPopupInt, you will not get the string you are looking for but rather an integer, which is incorrect.
- You can't resume work in MotionBuilder until you click one of the buttons on the Message Box.
- Giving any of the buttons the same name, causes MotionBuilder to freeze.

## 6.1 Keeping the User Informed via the Viewer Progress Bar

Whether you are creating filters, constraints or tools you can keep the user updated of the status by using the progress bar inside the Viewer, this is controlled by the class FBProgress. This allows for text to be inserted as well as the percentage completed ticker bar on timely operations, such as scene load or save.

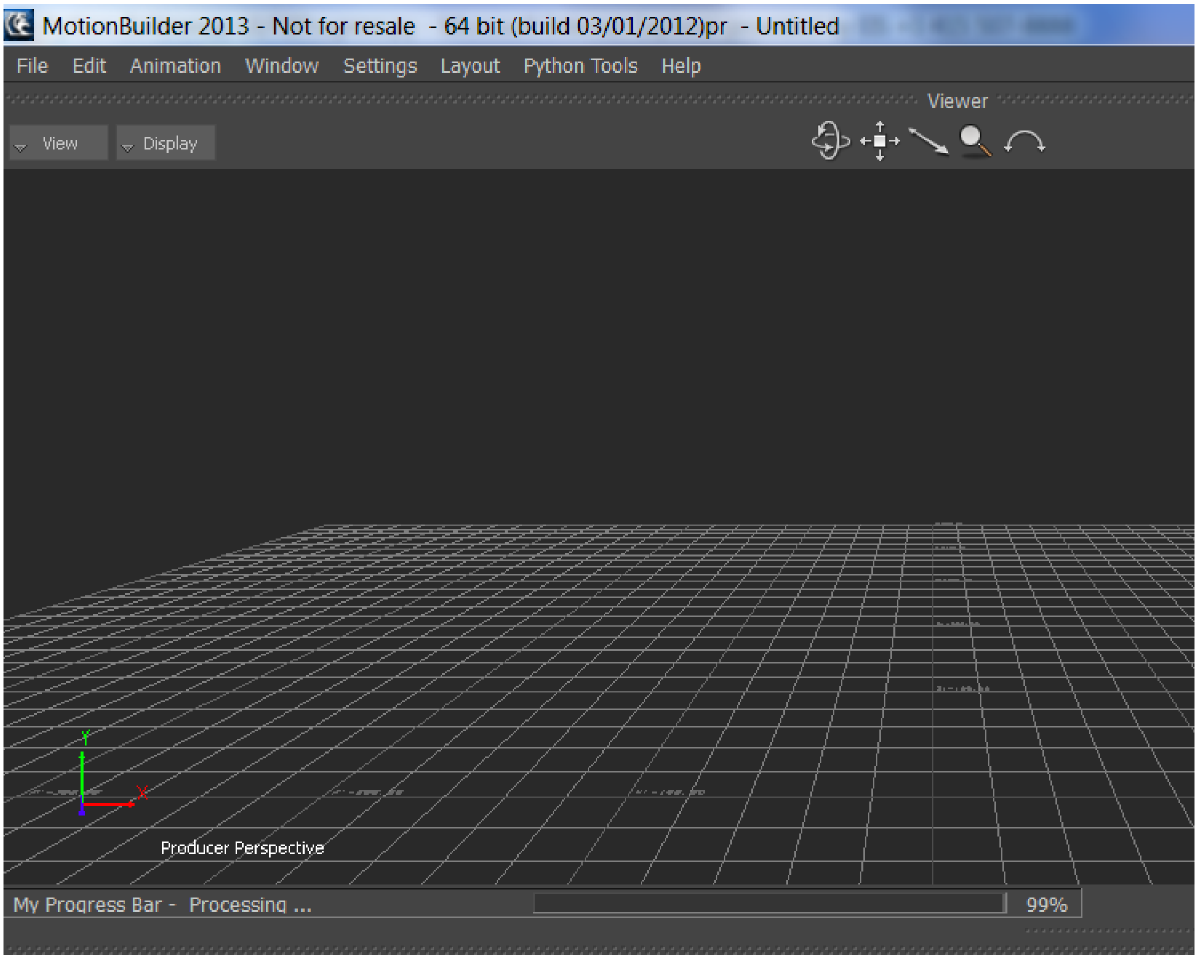

MotionBuilder 2013 - Not for resale - 64 bit (build 03/01/2012)pr - Untitled
File
Edit
Animation
Window
Settings
Layout
Python Tools
Help
Viewer
View
Display
Y
X
Producer Perspective
My Progress Bar - Processing ...
99%

Figure 5 | The Location in the Viewer of the Progress Bar Output

The available attributes are, caption or text to be displayed for the progress bar, the text to be displayed on the progress bar and the percentage completed for your operation that you are implementing this functionality for.

**Example**

```
from pyfbsdk import *
```

```
    lFbp = FBProgress()
    lFbp.Caption = "My Progress Bar"
    lFbp.Text = " Processing ..."
    lCount = 0.0

    lFbp.ProgressBegin()lComps = FBSystem().Scene.Components
    for lComp in lComps:
        print lComp.Name
        lCount += 1
        lVal = lCount / len(lComps) * 100
        FBSleep(20)
        lFbp.Percent = int(lVal)

lFbp.ProgressDone()
```

## 6.2 Building UI Functionality from Scratch (aka a world of possibilities)

As of MotionBuilder 2010 we have the ability to customize and create your own Python UI for all your Python scripts that you are creating. We're going to start building the UI like building a house from the frame to adding all the components after.

All these classes and functions are available from the modules pyfbsdk and pyfbsdk_additions.

We are going to look at building UI in a 3 stage process:

1. Registering and Creating your Tool
2. Creating UI components, buttons, lists, tabs, etc.
3. Attaching and playing your UI Components in your Tool
    - Three Available Workflows

## 6.3 Creating and Registering your Tool

These classes and functions are available from the modules pyfbsdk_additions, for additional information; please refer the module located here for additional coding comments:

C:\Program Files\Autodesk\MotionBuilder 2013\bin\config\Python\pyfbsdk_additions.py

If you are familiar with creating UI's in the Open Reality, this is where this step differs in Python, as we have created helper functions and classes that you must use to make sure everything gets registered internally with MotionBuilder, so your new tool will work as you expect it to.

## *Creating and Registering your tool*

To create a tool, use one of the two functions (FBCreateTool()and CreateUniqueTool()FBCreateUniqueTool()) provided in the pyfbsdk_additions module. Using these functions ensures that the tools are added to the tool list and that the Tool Manager knows about them.

Typically when you are developing a tool, you must execute it repeatedly when testing it. The problem is that each time you execute a script creating a tool; it adds a new tool to MotionBuilder. The previously-created tool is not replaced.

For example, if your script creates a tool with the function FBCreateTool: tool = FBCreateTool("tool example"), then each time the script is run new tools are created, for example, "tool example 1, "tool example 2" and so on.

To address this issue, the function CreateUniqueTool is included in pyfbsdk_additions. This function checks whether a tool with a similar name exists, and if so, it destroys that tool. It then creates a new tool with the same name. This means that each time you re-execute your script during testing; there is only one tool instantiated.

```
# Tool creation will serve as the hub for all other controls
global t
t = FBCreateUniqueTool("My New Tool")
```

Once you have finished testing a tool, replace the call to FBCreateUniqueTool with a call to FBCreateTool, so as not to use the tool destruction functionality. And if you use the startup scripts directory to create the tool (or if you execute the script creation tool only once) you won't have multiple tool instantiation.

```
# Tool creation will serve as the hub for all other controls
global t
t = FBCreateTool("My New Tool")
```

## *Setting the Size of your Tool Window*

To set the width and height of your tool you can use the attributes available on your newly created tool object.

```
t.StartSizeX = 300
t.StartSizeY = 143
```

Some other read write attributes you can use are:

```
Int StartSizeX
Int StartSizeY
Int MinSizeX
Int MinSizeY
Int MaxSizeX
Int MaxSizeY
char ToolName
```

## *Setting your Window to Show*

**Prior to this step we would create all the UI within our Tool, including implementing any functionality that it requires (however we are explaining this a little out of order so see steps 2, Creating the Framework of your UI and 3, Adding components to your UI, buttons, lists, tabs, etc. below for more details.

For your tool to show up, you need to call the ShowTool funciton on your tool object.

```
ShowTool(t)
```

If for convenience reason, a user prefers to drag and drop a script into viewer to “Show a tool”, there is a good way to do it that will ensure the Tool will be created only once. SafeToolCreationExample.py in C:\Program Files\Autodesk\MotionBuilder 2013\bin\config\Scripts\UI demonstrates how to create a tool that will be only instantiated once even if dragged multiple times in the Viewer.

## *FBGetTools Helper Function*

There is a function that returns a list of Python tools instantiate in MotionBuilder that you can use if you would to use this information in some manner:

```
from pyfbsdk_additions import *

print FBGetTools()
```

## 6.4 Creating UI components, buttons, lists, tabs, etc.

Here is the entire UI component that you can create inside of your tool; the base elements that can be created from FBComponent are the following:

| ELEMENT | CLASS | DESCRIPTION |
|---|---|---|
| **Arrow Button** | FBArrowButton | Button with an Arrow |
| **Button** | FBButton | Button |
| **List** | FBList | List of items |
| **Container** | FBContainer | Container (similar to list) |
| **Edit** | FBEdit | Text edit box |
| **Edit Number** | FBEditNumber | Number edit box |
| **Edit Color** | FBEditColor | Color edit tool |
| **Edit Vector** | FBEditVector | Vector edit tool |
| **Edit Property** | FBEditProperty | Property Editing |
| **Updated Edit Property** | FBEditPropertyModern | Updated Property Editing |
| **Image** | FBImage | UI Image |
| **Label** | FBLabel | Text label |
| **List** | FBList | A List of Items |
| **Memo** | FBMemo | An editable file |
| **Scroll Box** | FBScrollBox | An area that scrolls |
| **Slider** | FBSlider | Slider |
| **Spreadsheet** | FBSpread | Spreadsheet |
| **Tab Panel** | FBTabPanel | Panel with tabs |
| **Thermometer** | FBThermometer | Graphical tool |
| **Time Code** | FBTimeCode | Time Display |
| **Tree** | FBTree | Hierarchal viewer |

### *Creating each element type*

Creating the elements you want in your UI is very straight forward it is just like creating and FBModelCube and setting the public attributes for it. There are lots and lots of examples that come with MotionBuilder for creating these data types, so I will not go over them all.

To create a label in your UI, you create an instance of the FBLabel class, then set all the public attributes to your liking.

```
l = FBLabel()
l.Caption = "This is my label"
```

```
l.Style = FBTextStyle.kFBTextStyleBold
l.WordWrap = True
```

To create a button in your UI, you can create an instance of FBButton, and then set all the public attributes to your liking. You will need to set up an OnClick callback that calls a function as a parameter that you have defined in your script.

```
b = FBButton()
b.Caption = "Browse..."
b.Justify = FBTextJustify.kFBTextJustifyCenter
#attach to your layout
b.OnClick.Add(BrowseCallback)
```

## *Spreadsheets*

The spreadsheets in MotionBuilder are data structures that function as traditional spreadsheets, but with certain areas of the spreadsheets pre-defined and configured to contain extra data.

### Spread Parts

This is a generic class (FBSpreadPart) that can be used to represent any element of the spreadsheet. All of the objects in a spreadsheet inherit from this class, giving any element in the spreadsheet common characteristics.

### Header Row

The header row is the top row of the spreadsheet. It can be clicked or selected, and it can contain titles. The header row does not have all of the spread cell properties that data cells have.

### Column–1

Column-1 is the first column of row titles in the spreadsheet. It has special aspects that are slightly different in functionality from the other columns in the spreadsheet.

### Data cells

The data cells, depending on the settings for that particular cell, can contain and display various elements. The CellType property determines which variable is to be displayed.

In the following example, setting the style causes the spread cell to display the contents of its double variable.

```
mSpread.GetCell(0,1).Style = FBCellType.kFBCellStyleDouble;
```

However, the memory space is not shared between the different values. This allows a particular cell to contain many different values in the form of different types of values (integer, double, button states, and so on).

## 6.5 Attaching and playing your UI Components in your Tool

There are three workflows for creating your UI framework, this first workflow is the same workflow available in the Open Reality SDK, so if you’re familiar with creating UI in that OR SDK, you will likely choose this one. The second and third workflow are new workflow and are only available in Python, these are new classes located in the pyfbsdk_additions module file that are wrappers to workflow 1 in essence.

### *Workflow 1 – The Original Framework*

The user interface in MotionBuilder is managed using layout objects (class FBLayout). Think of layouts like the framework of your window, this is where ever thing gets attached too.
Layouts are responsible for managing areas of the screen called regions. UI components such as buttons, viewers, and edit boxes are bound to regions. In other words, when a UI component is bound to a region, the region defines how big it is and how it behaves when the layout is resized.

A region is an area in a layout that has been prepared and configured to hold a user interface element. In order to use a visual component, you need to add a region to the layout, and then set the control of this region to the visual component you would like. The following lines of code give a good example of this series of calls.

t is the tool object we created in step 1:

```
x = FBAddRegionParam(5,FBAttachType.kFBAttachLeft,"")
y = FBAddRegionParam(5,FBAttachType.kFBAttachTop,"")
w = FBAddRegionParam(300,FBAttachType.kFBAttachNone,"")
h = FBAddRegionParam(20,FBAttachType.kFBAttachNone,"")

t.AddRegion("lab","lab", x, y, w, h)
t.SetControl("lab",l)
```

A region is first defined using the FBLayout::AddRegion() function. Once a region is defined and the corresponding UI component is created, the component is bound to its region using the FBLayout::SetControl() function.

### Working with FBAddRegionParam

For each element you need to call this 4 times, for x and y spacing of the element compared to the layout, and for the width and height of the element we are trying to attach.

#### For Setting X and Positioning of the Element

For the x and y calls, the first parameter is the pixel spacing away from the object you are attaching it to, the second is where the starting point is of the object you are attaching it to and the third is the item you are attaching it to, if you are attaching it to the main layout, you do not need to specify anything so you leave empty quotes.

#### For Setting Width and Height of the Element

For the width and height, the first parameter is the width or height of the object, the second is what you you're going to attach it to, so we will always set the enumeration to FBAttachType.kFBAttachNone and the third parameter is also what were attaching it to, so for these two we will always keep empty strings there.

### Attaching this region to the main layout

You now need to attach your region that contains your element to the main layout by using the AddRegion function:

```
t.AddRegion("but","but", x, y, w, h)
```

### Setting the Control of your Element to your Layout

Then the last thing you need to do is to set the control of your element to your main layout by using the SetControl function:

```
t.SetControl("but",b2)
```

### Example

```
def PopulateLayout(mainLyt):
   #1.  Creating the instruction title
   l = FBLabel()
   l.Caption = "Select the shot track definition file to use:"
```

```
    l.Style = FBTextStyle.kFBTextStyleBold
    l.WordWrap = True

    x = FBAddRegionParam(5,FBAttachType.kFBAttachLeft,"") # Create a label that is left justify
    y = FBAddRegionParam(5,FBAttachType.kFBAttachTop,"") # ... and at the top of the layout.
    w = FBAddRegionParam(300,FBAttachType.kFBAttachNone,"") # its width is fixed at 300 pixels
    h = FBAddRegionParam(20,FBAttachType.kFBAttachNone,"") # and its height is fixed at 15 pixels

    mainLyt.AddRegion("lab","lab", x, y, w, h)
    mainLyt.SetControl("lab",l)

    #2.  Creating a temp button for a place holder
    e = FBEdit()
    e.ReadOnly = True
    e.Text = ""

    x2 = FBAddRegionParam(5,FBAttachType.kFBAttachLeft,"")
    y2 = FBAddRegionParam(5,FBAttachType.kFBAttachBottom,"lab")
    w2 = FBAddRegionParam(220,FBAttachType.kFBAttachNone,"")
    h2 = FBAddRegionParam(20,FBAttachType.kFBAttachNone,"")

    mainLyt.AddRegion("path","path", x2, y2, w2, h2)
    mainLyt.SetControl("path",e)

    #3.  Creating a the browse button
    b2 = FBButton()
    b2.Caption = "Browse..."
    b2.Justify = FBTextJustify.kFBTextJustifyCenter

    x3 = FBAddRegionParam(225,FBAttachType.kFBAttachLeft,"path")
    y3 = FBAddRegionParam(5,FBAttachType.kFBAttachBottom,"lab")
    w3 = FBAddRegionParam(57,FBAttachType.kFBAttachNone,"")
    h3 = FBAddRegionParam(20,FBAttachType.kFBAttachNone,"")

    mainLyt.AddRegion("but","but", x3, y3, w3, h3)
    mainLyt.SetControl("but",b2)

    #4. Creating the 'create backplate' checkbox
    cb = FBButton()
    cb.Style = FBButtonStyle.kFBCheckbox
    cb.Caption = "Add backplates to each shot"

    x4 = FBAddRegionParam(60, FBAttachType.kFBAttachLeft, "")
    y4 = FBAddRegionParam(5, FBAttachType.kFBAttachBottom, "path")
    w4 = FBAddRegionParam(300,FBAttachType.kFBAttachNone,"")
```

```
h4 = FBAddRegionParam(25,FBAttachType.kFBAttachNone,"")

mainLyt.AddRegion("backp", "backp",x4,y4,w4,h4)
mainLyt.SetControl("backp",cb)

#5. Creating the button the automatically does everything
b3 = FBButton()
b3.Caption = "Setup Shots"
b3.Justify = FBTextJustify.kFBTextJustifyCenter

x5 = FBAddRegionParam(40,FBAttachType.kFBAttachLeft,"")
y5 = FBAddRegionParam(5,FBAttachType.kFBAttachBottom,"backp")
w5 = FBAddRegionParam(100,FBAttachType.kFBAttachNone,"")
h5 = FBAddRegionParam(25,FBAttachType.kFBAttachNone,"")

mainLyt.AddRegion("shots","shots", x5, y5, w5, h5)
mainLyt.SetControl("shots",b3)

x6 = FBAddRegionParam(20,FBAttachType.kFBAttachRight,"shots")
y6 = FBAddRegionParam(5,FBAttachType.kFBAttachBottom,"backp")
w6 = FBAddRegionParam(100,FBAttachType.kFBAttachNone,"")
h6 = FBAddRegionParam(25,FBAttachType.kFBAttachNone,"")
```

## *Workflow 2 – Box Layouts*

All these classes are introduced in MotionBuilder 2009, and are located in the module pyfbsdk_additions, for more details on specific functions please refer to the documentation inside of this module located here:

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

There are three types of of new layouts:

1. Horizontal Box Layout
2. Vertical Box Layout
3. Grid Layout

### The ‘FBHBoxLayout’ and the ‘FBVBoxLayout’ Class

There is a new terminology introduced here called Box Layouts, this class is made to ease the creation of Tool in Python. It manages all the 'FBLayout' region stuff (no need to use FBAddRegionParam if you do not want.

This class manages a BoxLayout Horizontal. You still need to add this new box layout to the main tool layout so you will need to use workflow 1 once in this manner:

```
lyt = FBVBoxLayout()

x = FBAddRegionParam(0,FBAttachType.kFBAttachLeft,"")
y = FBAddRegionParam(0,FBAttachType.kFBAttachTop,"")
w = FBAddRegionParam(50,FBAttachType.kFBAttachNone,"")
h = FBAddRegionParam(0,FBAttachType.kFBAttachBottom,"")

mainLyt.AddRegion("main","main", x, y, w, h)
mainLyt.SetControl("main",lyt);
```

Once you have done this you can start working with the Box Layout, by using the function Add. There are 2 kinds of Add: Add with fixed size and AddRelative which ensure the control Added will occupy a 'percentage' of the available space after the fixed space has been assigned.

```
l = FBLabel()
l.Caption = "This is a label!!!!"
l.Style = FBTextStyle.kFBTextStyleBold
l.WordWrap = True
lyt.Add(l, 25)
```

As you can see here we have replaced, these six lines:

```
x2 = FBAddRegionParam(5,FBAttachType.kFBAttachLeft,"")
y2 = FBAddRegionParam(5,FBAttachType.kFBAttachBottom,"lab")
w2 = FBAddRegionParam(220,FBAttachType.kFBAttachNone,"")
h2 = FBAddRegionParam(20,FBAttachType.kFBAttachNone,"")

mainLyt.AddRegion("path","path", x2, y2, w2, h2)
mainLyt.SetControl("path",e)
```

With this one line:

```
lyt.Add(l, 25)
```

One thing to point out is you cannot center an element in a Box Layout.

### Previous Region Example converted to Box Layouts

```
def PopulateLayout(mainLyt):
    VBox = FBVBoxLayout()

    x = FBAddRegionParam(5,FBAttachType.kFBAttachLeft,"") # Create a label that is left justify
    y = FBAddRegionParam(5,FBAttachType.kFBAttachTop,"") # ... and at the top of the layout.
    w = FBAddRegionParam(5,FBAttachType.kFBAttachRight,"") # its width is fixed at 300 pixels
    h = FBAddRegionParam(5,FBAttachType.kFBAttachBottom,"") # and its height is fixed at 15 pixels

    mainLyt.AddRegion("main","main", x, y, w, h)
    mainLyt.SetControl("main",VBox)

    #1.  Creating the instruction title
    l = FBLabel()
    l.Caption = "Select the shot track definition file to use:"
    l.Style = FBTextStyle.kFBTextStyleBold
    l.WordWrap = True
    VBox.Add(l,20, space=5, width=300)


    HBox = FBHBoxLayout()
    VBox.Add(HBox, 20)

    #2.  Creating a temp button for a place holder
    e = FBEdit()
    e.ReadOnly = True
    e.Text = ""
    HBox.Add(e, 210, height=20)

    #3.  Creating a the browse button
    b2 = FBButton()
    b2.Caption = "Browse..."
    b2.Justify = FBTextJustify.kFBTextJustifyCenter

    HBox.Add(b2, 55, space=5, height=20)

    #4. Creating the 'create backplate' checkbox
    cb = FBButton()
    cb.Style = FBButtonStyle.kFBCheckbox
    cb.Caption = "Add backplates to each shot"

    VBox.Add(cb,20, space=5, width=300)

    #5. Creating the button the automatically does everything
    b3 = FBButton()
    b3.Caption = "Setup Shots"
    b3.Justify = FBTextJustify.kFBTextJustifyCenter
```

```
VBox.Add(b3,25, space=5, width=100)
```

## *Workflow 3 – Grid Layouts*

Gives you more control, however it can be also more complex because it works off of a grid with columns and rows, so you need to keep track of how wide and high your elements are in your columns and rows so everything fits nicely.

This is similar to workflow 2, as the calls are simplified like it.

**Previous Region Example converted to Grid Layout**

```
def PopulateLayout(mainLyt):

    x = FBAddRegionParam(5,FBAttachType.kFBAttachLeft,"") # Create a label
that is left justify
    y = FBAddRegionParam(5,FBAttachType.kFBAttachTop,"") # ... and at the top
of the layout.
    w = FBAddRegionParam(5,FBAttachType.kFBAttachRight,"") # its width is
fixed at 300 pixels
    h = FBAddRegionParam(5,FBAttachType.kFBAttachBottom,"") # and its height
is fixed at 15 pixels

    mainLyt.AddRegion("main","main", x, y, w, h)
    grid = FBGridLayout()
    mainLyt.SetControl("main", grid)


    #1.  Creating the instruction title
    l = FBLabel()
    l.Caption = "Select the shot track definition file to use:"
    l.Style = FBTextStyle.kFBTextStyleBold
    l.WordWrap = True
    grid.Add(l,0,0, height = 40)

    #2.  Creating a temp button for a place holder
    e = FBEdit()
    e.ReadOnly = True
    e.Text = ""
    grid.Add(e,1,0, width = 200, height = 20)

    #3.  Creating a the browse button
    b2 = FBButton()
    b2.Caption = "Browse..."
    b2.Justify = FBTextJustify.kFBTextJustifyCenter
    grid.Add(b2,1,1, width = 57, height = 20)
```

```
#4. Creating the 'create backplate' checkbox
cb = FBButton()
cb.Style = FBButtonStyle.kFBCheckbox
cb.Caption = "Add backplates to each shot"
grid.Add(cb,2,0, width = 200, height = 20)


#5. Creating the button the automatically does everything
b3 = FBButton()
b3.Caption = "Setup Shots"
b3.Justify = FBTextJustify.kFBTextJustifyCenter
grid.Add(b3,3,0, width = 100, height = 20)
```

## 6.6 Additional Helper Classes

### *The 'FBTabControl' Class*

A real TabControl that improve the behavior of the FBTabPanel.

This class manage the Tabs and the 'middle region' used to display. It manages all the region 'swapping' necessary to implement a Tab behavior.

Use Add method to Add a Control to the TabControl. This will create a tab and ensure that when that tab is clicked the Control is shown.
Use SetContent with a tab index To specify which tab should be displayed.

### *The 'FBButtonGroup' Class*

Button group class used to manage multiple radio buttons. This class ensure that only one radio button is enabled (it does all the ClickState management)

Use the Add method to Add new radio button to the group.
Use AddCallback method to register a UNIQUE callback that will be called when ANY of the registered radio buttons is clicked.

## 6.7 Available Callback Classes for your UI

These are the available callbacks for working with UI:

- FBEventActivate()
- FBEventDblClick()
- FBEventDragAndDrop()

- FBEventExpose()
- FBEventInput()
- FBEventMenu()
- FBEventResize()
- FBEventSceneChange()
- FBEventShow()
- FBEventSpread()
- FBEventTakeChange()
- FBEventTransaction()
- FBEventTree()
- FBEventTreeSelect()

## 6.8 Tool Manager and startup scripts

When a tool is well-tested enough to be used in production, there are two ways to start it.

If it needs to be started each time you start MotionBuilder, add the tool creation script to the Python Startup script folder, located at C:\Program Files\Autodesk\MotionBuilder 2013\bin\config\PythonStartup If the tool is rarely used, start it by dragging it into the scene. Be sure not to drag it and execute it a second time. Generally, it is not advised to create a Python tool multiple times. The tool might contain some specific states, and each time you execute it the tool and its states are deleted and then recreated.

The Tool Manager script is itself a Python tool and is found in the Python script folder. It uses the function provided by pyfbsdk_additions module to know when a tool is destroyed or created. You can even write your own Tool Manager. See Tool Manager.py in the Python script folder for details on how to register your own Tool Manager.

## 6.9 Other possible UI tool kits and MotionBuilder

Every once in a while you will get the question, can I use other Python UI tool kits in MotionBuilder. The short answer is yes, but the long answer is, you may not want to. Let me elaborate, firstly, Autodesk will not help you integrate them as they are not supported so you are on your own, secondly they do not sit inside the MotionBuilder Event loop, so depending on what you want to do this may not matter.

The major issue you will run into is that to view the changes in MotionBuilder that your interface triggered you have to close the UI. It has to do with not having the UI in the live update thread (Event thread), there is no interactive update from the UI tool to the

Viewer in MotionBuilder.

For example if you want an interface to animate a character you can trigger all the actions by hitting the buttons in the UI, but visually you will not see the changes until you close the new UI and bring the focus back to MotionBuilder. Depending on what you want to do this could be minor.

Here are a few options, which you can download off the web, Tkinter, wxPython and pyQT.