```
import os
from PyQt4 import QtCore as qc
from PyQt4 import QtGui as qg
import maya.cmds as mc
import maya.mel as mel

'''
07_08_16
This script was last amended on 14_07_14
To use the autoKeyframe UI make sure that there is a folder called "data"
in your project directory
before executing this code in Maya.
NB. time offset value is entered as a number between 0 and 1
'''

window = None

########## utility functions ##############
def getDataDir():
    projectDir = mc.workspace(query=True, rootDirectory=True)
    dataDirectory = projectDir + 'data'
    return dataDirectory

def getChannels(*args):
    getChannelBoxName = mel.eval('$temp=$gChannelBoxName')
    chList = mc.channelBox (getChannelBoxName, q=True,
selectedMainAttributes = True)
    if chList:
        for channel in chList:
            return channel
    else:
        print 'No channels selected; returning node only'
        return ''

def getNodeName():
    nodeList = mc.ls(selection=True)
    node = nodeList[0]
    channel = getChannels()
    name = "{node}.{channel}".format(node=node, channel=channel)
    return name

def createGroupBox(self, title='test', itemList=[]):
    #### create a Group Box
    groupBox = qg.QGroupBox(title)
    groupBox.setCheckable(True)
    groupBox.setFlat(True)
    # create the frame widget and set its style with now border
    frameWidget = qg.QFrame()
    frameWidget.setFrameStyle(0)
    #####connect group box checked signal to frame widget's display slot
    groupBox.toggled.connect(frameWidget.setShown)

    # Create a vertical box layout
    vLayout = qg.QVBoxLayout()
    ## add widgets and layouts from "item list" to the box layout
    i = 0
    for item in itemList:
```

```
        ###check if list item is a layout. assume that otherwise its a
widget
        if type(item) == qg.QVBoxLayout:
            # print 'item is a QVBoxLayout I think'
            vLayout.addLayout(itemList[i])

        elif type(item) == qg.QHBoxLayout:
            # print 'item is an QHBoxLayout I think'
            vLayout.addLayout(itemList[i])

        elif type(item) == qg.QBoxLayout:
            # print 'item is an QBoxLayout I think'
            vLayout.addLayout(itemList[i])

        else:
            # print 'item is a widget I think'
            vLayout.addWidget(itemList[i])
        i += 1

    #set the frame widget's layout
    frameWidget.setLayout(vLayout)
    ##Make a layout and add the frame widget
    framelayout = qg.QVBoxLayout()
    ## set margins of frameLayout
    framelayout.setContentsMargins(0,0,0,0)
    framelayout.addWidget(frameWidget)
    #### add frame to the group box
    groupBox.setLayout(framelayout)

    return groupBox

def splitList(listName, splitNumber):
    newList=[]
    for i in range(0, len(listName), splitNumber):
        newList.append(listName[i:i+splitNumber])
    return newList

def writeFile(directory, fileName, target, driver, numberOfDriverKeys,
numberOfTargetKeys, keyValuesList, curveTimeShift):
    dataFolderDirectory = directory
    outputFileName = fileName + '.txt'
    myTarget = str(target)
    myDriver = str(driver)
    # print 'myTarget is ', myTarget

    #make the folder if it doesnt exist
    if not os.path.exists(dataFolderDirectory):
        os.makedirs(dataFolderDirectory)
    # print 'making directory', dataFolderDirectory

    outputFile =
open('{dataFolderDirectory}/{fileName}'.format(dataFolderDirectory=dataFo
lderDirectory, \
        fileName=outputFileName), 'wb')
    outputFile.write("TARGET\r\n")
    outputFile.write(myTarget)
    outputFile.write('\r\n')
    outputFile.write("DRIVER\r\n")
```

```
    outputFile.write(myDriver)
    outputFile.write('\r\n')

    outputFile.write("DRIVER CYCLE LENGTH\r\n")
    outputFile.write(str(numberOfDriverKeys))
    outputFile.write('\r\n')

    outputFile.write("TARGET CYCLE LENGTH\r\n")
    outputFile.write(str(numberOfTargetKeys))
    outputFile.write('\r\n')

    #add curveTimeShift
    outputFile.write("CURVE TIME SHIFT\r\n")
    outputFile.write(str(curveTimeShift))
    outputFile.write('\r\n')

    outputFile.write("KEY VALUES\r\n")

    for key in range(numberOfTargetKeys):
        # outputFile.write("KEY {key} VALUES\r\n".format(key=key))
        for index in range(6):
            outText = (keyValuesList[key])[index]
            outputFile.write(outText)
            outputFile.write('\r\n')
            # print outText

    outputFile.close()

def readFile(fileDirectory):
    # myFile = fileDirectory + '.txt'
    myFile = fileDirectory
    inputFile = open(myFile, 'r')

    target = ''
    driver = ''
    numberOfDriverKeys = ''
    numberOfTargetKeys = ''
    keyData = []

    index = 0
    lineList = []
    for line in inputFile.readlines():
        newLine = line.rstrip('\r\n')

        if index == 1:
            target = newLine

        if index == 3:
            driver = newLine

        if index == 5:
            numberOfDriverKeys = newLine

        if index == 7:
            numberOfTargetKeys = newLine

        if index == 9:
            curveTimeShift = newLine
```

```
        lineList.append(newLine)
        index +=1

    #create an empty list to store the key data lists
    keyDataLists = []
    # key data starts on line 9 ### make that 11
    index2 = 11
    #this loop twice cause 2 keys
    for i in range(int(numberOfTargetKeys)):
        #make a temporary list
        myList = []

        index3 = 1
        remainderIndex3 = index3 % 6
        #this loop 6 times
        for ii in range(6):
            if remainderIndex3 > 0:
                data = lineList[index2]
                myList.append(data)

                index3 += 1
                index2 += 1

        keyDataLists.append(myList)

    # print 'curveTimeShift is >>', curveTimeShift
    return target, driver, numberOfDriverKeys, \
    numberOfTargetKeys, keyDataLists, curveTimeShift


class AutoAnimPage(qg.QWidget):
    def __init__(self, targetNodeEntry):
        qg.QWidget.__init__(self)
        self.setGeometry(300, 350, 300, 250)
        #Make list to store key column layouts
        self.keyColumnsBoxList = []
        # get the targetNodeEntry as passed in
        self.targetNodeEntry = targetNodeEntry
        # make an empty keydataList
        self.keyDataList = []
        # make variable for data dir path
        self.dataDir = getDataDir()

        self.__createWidgets__()
        self.__connectWidgets__()
        self.__layoutWidgets__()


    def __createWidgets__(self):
        buttonWidth = 80
        self.LEdKeyTableWidth = 35
        ##### driver and target node widgets
        self.addDriverTextInput = qg.QLineEdit(parent=self,
maximumWidth=200)

self.addDriverTextInput.setPlaceholderText('driverNode.driverChannel')
```

```
        self.addDriverButton = qg.QPushButton("add driver", minimumWidth
= buttonWidth, parent=self)

        self.addTargetTextInput = qg.QLineEdit(parent=self,
maximumWidth=200)
        ### get text for target node text field...if self.targetNodeEntry
=! '', then ...
        if self.targetNodeEntry == '':

self.addTargetTextInput.setPlaceholderText('targetNode.targetChannel')
        else:
            self.addTargetTextInput.setText(self.targetNodeEntry)
        self.addTargetButton = qg.QPushButton("add target", minimumWidth
= buttonWidth, parent=self)

        ########key info widgets
        self.noKeysLabel = qg.QLabel("Target cycle length", parent=self)
        self.noKeysInput = qg.QLineEdit('2', parent=self,
maximumWidth=self.LEdKeyTableWidth)

        self.noKeysDriverLabel = qg.QLabel("Driver cycle length",
parent=self)
        self.noKeysDriverInput = qg.QLineEdit('2', parent=self,
maximumWidth=self.LEdKeyTableWidth)
                ########key shift widgets
        self.timeShiftCurveLabel = qg.QLabel("Time shift curve",
parent=self)
        self.timeShiftCurveInput = qg.QLineEdit('0', parent=self,
maximumWidth=self.LEdKeyTableWidth)


        #create key attribute label widgets
        self.refKeyLabel = qg.QLabel("Ref Key(s)", parent=self)
        self.timeOffsetLabel = qg.QLabel("Time Offset", parent=self)
        self.valueMultiplierLabel = qg.QLabel("Value Multiplier",
parent=self)
        self.valueOffsetLabel = qg.QLabel("Value Offset", parent=self)
        self.tangentInLabel = qg.QLabel("Tangent In", parent=self)
        self.tangentOutLabel = qg.QLabel("Tangent Out", parent=self)

        #### make CREATE KEYS BUTTON
        self.createKeysButton = qg.QPushButton("Create Keys",
minimumHeight=30, parent=self)
        self.createKeysButton.setStyleSheet('background-color:#554444;\
        border: 2px solid #222222')

        ########## make save and apply defaults widgets
        self.defPathLabel = qg.QLabel(self.dataDir)
        self.folderInput = qg.QLineEdit(parent=self, maximumWidth=150)
        self.folderInput.setPlaceholderText('enter folder name')
        self.fileInput = qg.QLineEdit(parent=self, maximumWidth=150)
        self.fileInput.setPlaceholderText('enter file name')

        self.applyDefaultsButton = qg.QPushButton('Apply default values',
minimumWidth=buttonWidth, parent=self)
        self.saveDefaultsButton = qg.QPushButton('Save default values',
minimumWidth=buttonWidth, parent=self)
```

```
    def __connectWidgets__(self):
        self.addDriverButton.pressed.connect(self.populateDriverText)
        self.addTargetButton.pressed.connect(self.populateTargetText)

self.addTargetButton.pressed.connect(self.populateDefaultFileText)

        self.noKeysInput.returnPressed.connect(self.updateKeysLayout)

        self.createKeysButton.pressed.connect(self.setKeys)
        # self.duplicateTabButton.pressed.connect(self.duplicateTab)
        ### connect defaults buttons
        self.applyDefaultsButton.pressed.connect(self.applyDefaults)
        self.saveDefaultsButton.pressed.connect(self.saveDefaults)
        self.folderInput.editingFinished.connect(self.updatePath)
        self.fileInput.editingFinished.connect(self.updatePath)

    def __layoutWidgets__(self):
        ##create 2xH layouts and 1 x QVlaout for target and driver
widgets
        self.addDriverLayout = qg.QHBoxLayout()
        self.addDriverLayout.addWidget(self.addDriverTextInput)
        self.addDriverLayout.addWidget(self.addDriverButton)

        self.addTargetLayout = qg.QHBoxLayout()
        self.addTargetLayout.addWidget(self.addTargetTextInput)
        self.addTargetLayout.addWidget(self.addTargetButton)

        self.driverTargetLayout = qg.QVBoxLayout()
        self.driverTargetLayout.addLayout(self.addDriverLayout)
        self.driverTargetLayout.addLayout(self.addTargetLayout)

        #create the first two key column layouts and add to
keyColumnsBoxList
        key0Column = self.createKeyColumnGroupBox(name = 'Key 0',
refKey='0')
        key1Column = self.createKeyColumnGroupBox(name = 'Key 1',
refKey='1')
        self.keyColumnsBoxList.append(key0Column)
        self.keyColumnsBoxList.append(key1Column)

        #create number of keys layout along the top
        self.noOfKeysLayout = qg.QHBoxLayout()
        self.noOfKeysLayout.addWidget(self.noKeysDriverLabel)
        self.noOfKeysLayout.addWidget(self.noKeysDriverInput)
        self.noOfKeysLayout.addWidget(self.noKeysLabel)
        self.noOfKeysLayout.addWidget(self.noKeysInput)
        # add timeshift curve to the layout
        self.noOfKeysLayout.addWidget(self.timeShiftCurveLabel)
        self.noOfKeysLayout.addWidget(self.timeShiftCurveInput)

        #create keys attribute labels layout down the side and add label
widgets
        self.attrLabelLayout = qg.QVBoxLayout()
        self.attrLabelLayout.addWidget(self.refKeyLabel)
        self.attrLabelLayout.addWidget(self.timeOffsetLabel)
        self.attrLabelLayout.addWidget(self.valueMultiplierLabel)
        self.attrLabelLayout.addWidget(self.valueOffsetLabel)
        self.attrLabelLayout.addWidget(self.tangentInLabel)
```

```
        self.attrLabelLayout.addWidget(self.tangentOutLabel)

        ########create group box and layout for attribute labels layout
down the side
        self.keyLabelGroupBox = qg.QGroupBox('Key Attribute')
        self.keyLabelGroupBox.setLayout(self.attrLabelLayout)

        # keys layout (bottom half of the window)
        self.keysLayout = qg.QHBoxLayout()
        self.keysLayout.addWidget(self.keyLabelGroupBox)
        #add the keys column Group Boxes
        self.keysLayout.addWidget(key0Column)
        self.keysLayout.addWidget(key1Column)

        #create key info layout
        self.keyInfolayout = qg.QVBoxLayout()
        self.keyInfolayout.addLayout(self.noOfKeysLayout)
        # self.keyInfolayout.addLayout(self.noOfDriverKeysLayout)
        self.keyInfolayout.addLayout(self.keysLayout)
        ############ create group Box for Key Info and pass in
self.noOfKeysLayout, self.keyInfolayout
        self.keyInfoGroupBox = createGroupBox(self, title='Key Info',
itemList=[self.noOfKeysLayout, self.keyInfolayout])

        ### Layout apply and save defaults
        self.fileFolderIn = qg.QHBoxLayout()
        self.fileFolderIn.addWidget(self.folderInput)
        self.fileFolderIn.addWidget(self.fileInput)

        self.applySaveDefLayout = qg.QHBoxLayout()
        self.applySaveDefLayout.addWidget(self.applyDefaultsButton)
        self.applySaveDefLayout.addWidget(self.saveDefaultsButton)

        ## createGroupBox(); pass in apply defaults layout and save
defaults layout
        self.defaultValuesGroupBox = createGroupBox(self, title='Default
Values', \
            itemList=[self.fileFolderIn, self.defPathLabel,
self.applySaveDefLayout])

        #create the main layout and add key info group box, test data
group box
        self.layout = qg.QBoxLayout(qg.QBoxLayout.TopToBottom, self)
        self.layout.addLayout(self.driverTargetLayout)
        self.layout.addWidget(self.defaultValuesGroupBox)
        self.layout.addWidget(self.keyInfoGroupBox)
        self.layout.addWidget(self.createKeysButton)


    def applyDefaults(self):
        fileDirectory = self.updatePath()[1] + '.txt'
        defaultValues = readFile(fileDirectory)
        target = defaultValues[0]
        driver = defaultValues[1]
        numberOfDriverKeys = defaultValues[2]
        numberOfTargetKeys = defaultValues[3]
        keyDataLists = defaultValues[4]
        #get curveTimeShift from readFile()
```

```
        curveTimeShift = defaultValues[5]
        # print 'applyingDefaults including curveTimeShift of ',
curveTimeShift

        ######## fill in target, driver, noKeysDriver, noKeysTarget,
curveTimeShift
        self.addTargetTextInput.setText(target)
        self.addDriverTextInput.setText(driver)
        self.noKeysDriverInput.setText(numberOfDriverKeys)
        self.noKeysInput.setText(numberOfTargetKeys)
        # fill in curve shift info
        self.timeShiftCurveInput.setText(curveTimeShift)

        #### update key rows
        self.updateKeysLayout()
        ## populate the key data
        self.populateDataLists(keyDataLists)

    def populateDataLists(self, keyDataLists):
        # print 'keyDataLists is ', keyDataLists
        # iterate over the keyColumn boxes
        keyColumns = self.keyColumnsBoxList
        dataListIndex = 0
        for keyColumn in keyColumns:
            dataList = keyDataLists[dataListIndex]
            dataIndex = 0

            #get all the line edits in this column
            lineEditList = keyColumn.findChildren(qg.QLineEdit)
            #get all the combo boxes in this column
            comboBoxList = keyColumn.findChildren(qg.QComboBox)

            # print 'lineEditList is ', lineEditList
            # print 'comboBoxList is ', comboBoxList

            while dataIndex < 4:
                for linEdit in lineEditList:
                    # linEdit = lineEditList[dataIndex]
                    data = dataList[dataIndex]
                    # fill current lineEdit with current data
                    linEdit.setText(data)
                    # print 'data for ', dataIndex, 'index is ', data

                    dataIndex += 1

            ######### make another loop for tangent combo boxes
            while (dataIndex > 3) and (dataIndex < 6):
                for comboBox in comboBoxList:
                    # linEdit = comboBoxList[dataIndex]
                    data = dataList[dataIndex]
                    # ############ fill current comboBox with current
data
                    comboIndex = comboBox.findText(data)
                    # print 'comboIndex is ', comboIndex
                    comboBox.setCurrentIndex(comboIndex)
                    # print 'data for comboBox ', dataIndex, 'index is ',
data
```

```
                    dataIndex += 1
            dataListIndex += 1

    def poulateTabFromFile(self, fileData, folderName, fileName):
        target = fileData[0]
        driver = fileData[1]
        numberOfDriverKeys = fileData[2]
        numberOfTargetKeys = fileData[3]
        # curveTimeShift is returned last
        curveTimeShift = fileData[5]

        keyDataLists = fileData[4]
        ######## fill in target, driver, noKeysDriver, noKeysTarget,
curveTimeShift
        self.addTargetTextInput.setText(target)
        self.addDriverTextInput.setText(driver)
        self.noKeysDriverInput.setText(numberOfDriverKeys)
        self.noKeysInput.setText(numberOfTargetKeys)
        self.timeShiftCurveInput.setText(curveTimeShift)

        ###### fill in default values folder name and file name
        ## if folderName is not = to None
        if folderName != None:
            self.folderInput.setText(folderName)
        self.fileInput.setText(fileName)

        #### update key rows
        self.updateKeysLayout()
        ## populate the key data
        self.populateDataLists(keyDataLists)
        # print 'folderName, filename ', folderName, fileName
        # print 'file data is ', fileData

    def saveDefaults(self):
        folderDirectory = self.updatePath()[0]
        fileName = self.fileInput.text()
        target =self.addTargetTextInput.text()
        driver = self.addDriverTextInput.text()
        numberOfDriverKeys = int(self.noKeysDriverInput.text())
        numberOfTargetKeys = int(self.noKeysInput.text())
        ### get curveTimeShift
        curveTimeShift = self.timeShiftCurveInput.text()

        keyValuesList = self.makeAllDataLists()
        writeFile(folderDirectory, fileName, target, driver,
numberOfDriverKeys, \
            numberOfTargetKeys, keyValuesList, curveTimeShift)
        print folderDirectory, fileName

    def getTabData(self):
        target = str(self.addTargetTextInput.text())
        driver = str(self.addDriverTextInput.text())
        numberOfDriverKeys = str(self.noKeysDriverInput.text())
        numberOfTargetKeys = str(self.noKeysInput.text())
        keyValuesList = self.makeAllDataLists()
        data = [target, driver, numberOfDriverKeys, numberOfTargetKeys,
keyValuesList]
        # print data
```

```
        return data

    def updatePath(self):
        self.dataDir = getDataDir()
        folderName = str(self.folderInput.text())
        fileName = str(self.fileInput.text())

        folderDirectory = self.dataDir + '/' + folderName
        fileDirectory = folderDirectory + '/' + fileName
        self.defPathLabel.setText(fileDirectory)

        return folderDirectory, fileDirectory
    #ACCESSORS
    def getNumberOfKeys(self):
        numberOfKeys = int(self.noKeysInput.text())
        return numberOfKeys

    def getFileFolderNames(self):
        fileName = str(self.fileInput.text())
        folderName = str(self.folderInput.text())
        return fileName, folderName

    def compareLength(self):
        numberOfKeys = self.getNumberOfKeys()
        # keyColumns = (self.keysLayout.count()) - 1

        keyColumnsLayoutListLength = len(self.keyColumnsBoxList)
        keyColumns = keyColumnsLayoutListLength

        # print 'number of keyColumns is:
{keyColumns}'.format(keyColumns=keyColumns)
        # print keyColumnsLayoutListLength

        if numberOfKeys > keyColumns:
            columnsToAdd = numberOfKeys - keyColumns
            columnsToDelete = 0

        elif numberOfKeys < keyColumns:
            columnsToDelete = keyColumns - numberOfKeys
            columnsToAdd = 0

        else:
            columnsToAdd = 0
            columnsToDelete = 0

        return columnsToDelete, columnsToAdd

    #MUTATORS
    def updateKeysLayout(self):
        columnsToDelete = self.compareLength()[0]
        columnsToAdd = self.compareLength()[1]

        if columnsToAdd > 0:
            self.addKeyColumnGroupBox()

        if columnsToDelete > 0:
            self.removeKeyColumns()
```

```
    def removeKeyColumns(self):
        columnsToDelete = self.compareLength()[0]

        for i in range(columnsToDelete):
            columnNumber = len(self.keyColumnsBoxList)-1
            #get the keyColumnGroupBox
            keyColumnGroupBoxToDelete =
self.keyColumnsBoxList[columnNumber]
            # remove from the list of key column layouts before its
deleted
            self.keyColumnsBoxList.remove(keyColumnGroupBoxToDelete)
            # delete the group box widget (I'm not sure if this also
deletes the layout and the LineEdit widgets)
            keyColumnGroupBoxToDelete.deleteLater()
            keyColumnGroupBoxToDelete = None

    def populateDriverText(self):
        driverNodeText = getNodeName()
        # print driverNodeText
        self.addDriverTextInput.setText(driverNodeText)

    def populateTargetText(self):
        targetNodeText = getNodeName()
        # print targetNodeText
        self.addTargetTextInput.setText(targetNodeText)

    def populateDefaultFileText(self):
        targetNodeText = getNodeName()
        ## replace old character with new one; str.replace(old, new[,
max])
        targetNodeText2 = targetNodeText.replace('.', '_')
        targetNodeText3 = targetNodeText2.replace(':', '')

        self.fileInput.setText(targetNodeText3)
        ########## use the following when I dont always want to update
default file name
        ## get the target node text entry
        # currentDefaultFileText = self.fileInput.text()
        # if currentDefaultFileText:
        #     print currentDefaultFileText
        #     # pass
        # else:
        #     self.fileInput.setText(targetNodeText3)

    def populateDefaultFolderText(self, folderName):
        ## apply default folder to current tab
        self.folderInput.setText(folderName)


    #CREATORS
    def createKeyColumnGroupBox(self, name = "Key number", refKey = '1'):
        # refKeysIn = key number (ie. key index)
        refKey = refKey
        #this creates a group box and a keyColumnLayout. it returns the
group box widget
        refKeysIn = qg.QLineEdit(refKey, parent=self,
maximumWidth=self.LEdKeyTableWidth, objectName='refKeys')
```

```
        timeOffsetIn = qg.QLineEdit("0", parent=self,
maximumWidth=self.LEdKeyTableWidth, objectName='timeOffset')
        valueMultIn = qg.QLineEdit("1", parent=self,
maximumWidth=self.LEdKeyTableWidth, objectName='valueMult')
        valueOffsetIn = qg.QLineEdit("0", parent=self,
maximumWidth=self.LEdKeyTableWidth, objectName='valueOffset')
        #try in and out tangents as a combo box
        tangentItems = ["spline", "linear", "fast", "slow", "flat",
"stepped", "step next", "fixed", "clamped", "plateau"]
        inTangentIn = qg.QComboBox(parent=self,
maximumWidth=self.LEdKeyTableWidth + 16)
        inTangentIn.addItems(tangentItems)
        outTangent = qg.QComboBox(parent=self,
maximumWidth=self.LEdKeyTableWidth + 16)
        outTangent.addItems(tangentItems)

        keyColumnLayout = qg.QVBoxLayout()
        keyColumnLayout.addWidget(refKeysIn)
        keyColumnLayout.addWidget(timeOffsetIn)
        keyColumnLayout.addWidget(valueMultIn)
        keyColumnLayout.addWidget(valueOffsetIn)
        keyColumnLayout.addWidget(inTangentIn)
        keyColumnLayout.addWidget(outTangent)

        keyColumnGroupBox = qg.QGroupBox(name)
        keyColumnGroupBox.setLayout(keyColumnLayout)

        return keyColumnGroupBox

    def addKeyColumnGroupBox(self):
            # add the column (key number) here to name the Group Box
            columnsToAdd = self.compareLength()[1]
            numberExistingColumns = len(self.keyColumnsBoxList)
            #work out the name of the first added column
            columnNumber = numberExistingColumns

            for column in range(columnsToAdd):
                columnName = 'Key
{columnNumber}'.format(columnNumber=columnNumber)
                refKey = str(column + 2)
                keyColumnGroupBox =
self.createKeyColumnGroupBox(name=columnName, refKey=refKey)
                self.keysLayout.addWidget(keyColumnGroupBox)
                #increase name by 1
                columnNumber += 1
                # add to the list of key column layouts
                self.keyColumnsBoxList.append(keyColumnGroupBox)

    # ################## Key data calculation functions - START
############
    def makeAllDataLists(self):
        self.keyDataList = []
        # iterate over the keyColumn boxes
        keyColumns = self.keyColumnsBoxList
        for keyColumn in keyColumns:
            #get all the line edits in this column
            lineEditList = keyColumn.findChildren(qg.QLineEdit)
            comboBoxList = keyColumn.findChildren(qg.QComboBox)
```

```
            #make empty temporary keyDataList
            tempDataList = []
            # iterate over the list of line edits
            for lineEdit in lineEditList:
                keyData = str(lineEdit.text())
                #add data to temporary list for this key
                tempDataList.append(keyData)
            # a second loop for combo boxes (ie tangent in and tangent
out)
            for comboBox in comboBoxList:
                keyData = str(comboBox.currentText())
                #add data to temporary list for this key
                tempDataList.append(keyData)

            # add temp dataList to dataLists
            self.keyDataList.append(tempDataList)

        # print self.keyDataList
        return self.keyDataList

    def getSpecificDataList(self, dataTypeIndex=0):
        # make empty Ref keys list
        dataList = []
        #freshly make and access key data dictionary
        allKeyDataList = self.makeAllDataLists()
        for i in range(len(allKeyDataList)):
            currentKeyDataList = allKeyDataList[i]
            data = currentKeyDataList[dataTypeIndex]
            dataList.append(data)
        # print dataList
        return dataList

    def getDriverKeyInfo(self):
        # get the driver text - NB. I get this in a lot of places
        driver = str(self.addDriverTextInput.text())
        # print driver
        #get a list of the driver keys using cmds
        driverKeyList = mc.keyframe(driver, query=True)
        driverKeyListLen = len(driverKeyList)
        return driverKeyList, driverKeyListLen

    def calculateNumKeysToLay(self):
        driverKeyLoopLen = int(self.noKeysDriverInput.text())
        targetKeyLoopLen = int(self.noKeysInput.text())
        driverKeyListLen = self.getDriverKeyInfo()[1]
        #get no. of loops to lay; driverKeyListLen /driverKeyLoopLen
        loopsToLay = driverKeyListLen / driverKeyLoopLen
        #get no. of keys to lay
        numKeysToLay = loopsToLay * targetKeyLoopLen
        return numKeysToLay

    def createNewDriverKeyList(self):
        # get driver key list, driver key split list, and driver key list
length
        driverKeyInfoList = self.getDriverKeyInfo()[0]
        driverKeyLoopLen = int(self.noKeysDriverInput.text())
        targetKeyLoopLen = int(self.noKeysInput.text())
```

```
        driverKeyListSplit = splitList(driverKeyInfoList,
driverKeyLoopLen)
        #get no. of keys to lay
        numKeysToLay = self.calculateNumKeysToLay()
        #get the position in the loop of the driver key to refer to
        driverIndexNumList = self.getSpecificDataList(0)

        #create empty driver key list
        newDriverKeyList = []
        nextDriverKeyList = []

        loopCounter = 1
        loopIndex = 0
        for key in range(numKeysToLay):
            # print 'key ==', key
            ##get the current drive key position in the loop
            currentDriverIndexStr = driverIndexNumList[key %
targetKeyLoopLen]
            currentDriverIndexNum = int(currentDriverIndexStr)

            # calculate current driver key
            currentDriverKey =
(driverKeyListSplit[loopIndex])[currentDriverIndexNum]
            newDriverKeyList.append(currentDriverKey)

             #add to loopIndex only every third time through
            if loopCounter == targetKeyLoopLen:
                loopIndex += 1
                loopCounter = 0
            loopCounter += 1

            ### from driverkeyList can I calculate next key along?
        NEXTDriverKeyList = []
        # testList = [1.0, 20.0, 40.0, 60.0, 80.0, 100.0, 120.0, 140.0]
        testList = newDriverKeyList
        # print 'newDriverKeyList is ', newDriverKeyList

        driver = str(self.addDriverTextInput.text())

        loopNum = 1
        for time in testList:

            keyIndex = mc.keyframe(driver, time=(time,time),
indexValue=True, query=True)
            nextIndex = keyIndex[0] +1

            if nextIndex < len(testList):
                # print 'nextIndex ', nextIndex
                nextKeyTime = (mc.keyframe(driver,
index=(nextIndex,nextIndex), timeChange=True, query=True))[0]
                # print 'nextKeyTime ', nextKeyTime
                NEXTDriverKeyList.append(nextKeyTime)

            else:
                # print 'else loop'
                #its teh last key in the list use projected value
                projectedTimeGap = testList[-1] - testList[-2]
                # print 'projectedTimeGap ', projectedTimeGap
```

```
                projectedTime = testList[-1] + projectedTimeGap
                #### not sure about this
                nextKeyTime = projectedTime
                # print print 'nextKeyTime ', nextKeyTime
                NEXTDriverKeyList.append(nextKeyTime)

            loopNum +=1

        # print 'NEXT DriverKeyList is ', NEXTDriverKeyList
        return newDriverKeyList, NEXTDriverKeyList


    def calculateNextKeyTime(self, currentKeyIndex):
        ###### newDriverKey returns a list of times of the driver keys
        NEXTDriverKeyList = self.createNewDriverKeyList()[1]
        nextKeyTime = NEXTDriverKeyList[currentKeyIndex]
        # print 'nextKeyTime == ', nextKeyTime, 'type == ',
type(nextKeyTime)
        return nextKeyTime


    def calculateKeyTime(self, timeOffset,keyIndex, keyRef):
        curveTimeShift = float(self.timeShiftCurveInput.text())
        # print 'curveTimeShift == ', curveTimeShift

        #convert timeOffset to a float
        timeOffset = float(timeOffset)
        newDriverKeyList = self.createNewDriverKeyList()[0]

        thisKey = newDriverKeyList[keyIndex]
        nextKey = self.calculateNextKeyTime(keyIndex)
        # print 'thisKey == ', thisKey, 'nextKey == ', nextKey
        # print 'types == ', type(thisKey), type(nextKey)

        amountToAdd = (nextKey - thisKey) * timeOffset

        keyTime = thisKey + amountToAdd
        keyTimeWithCurveShift = keyTime + curveTimeShift

        return keyTimeWithCurveShift

    def calculateKeyValue(self, valueMult, valueOff, keyIndex):
        valueMultiplier = float(valueMult)
        valueOffset = float(valueOff)
        # get the driver node and channel
        driver = str(self.addDriverTextInput.text())
        # get new driver key list
        newDriverKeyList = self.createNewDriverKeyList()[0]
        driverKeyTime = newDriverKeyList[keyIndex]

        ### actually I want to query the ref key value not the current
time value ## might be doing this...
        driverKeyValue = mc.keyframe(driver, query=True, eval=True,
time=(driverKeyTime,driverKeyTime))[0]
        keyValue = (driverKeyValue * valueMultiplier) + valueOffset
        # keyValue = driverKeyValue * valueMultiplier

        return keyValue
```

```
    def setKeys(self):
        #get target node and channel
        target = str(self.addTargetTextInput.text())
        #get the target loop length
        targetKeyLoopLen = int(self.noKeysInput.text())
        # get time offset list = length of target loop
        timeOffsetList = self.getSpecificDataList(dataTypeIndex=1)
        # get value multiplier list and valueOffsetList- used in
calculate key Value
        valueMultList = self.getSpecificDataList(dataTypeIndex=2)
        valueOffsetList = self.getSpecificDataList(dataTypeIndex=3)

        inTangentList = self.getSpecificDataList(dataTypeIndex=4)
        outTangentList = self.getSpecificDataList(dataTypeIndex=5)

        # check if the curves exist and delete if it does#######
        mc.cutKey(target, option="keys")
##########################calculate number of keys to lay
        numKeysToLay = self.calculateNumKeysToLay()
        #get the position in the loop of the driver key to refer to
        keyRefList = self.getSpecificDataList(0)
        # print ' keyRefList is >>', keyRefList

        keyIndex = 0
        loopCounter = 1
        loopIndex = 0
        for key in range(numKeysToLay):
            ##############get the current drive key position in the loop
            # currentDriverIndexNum = driverIndexNum[key %
targetKeyLoopLen]
            currentDriverIndexNum = keyRefList[key % targetKeyLoopLen]
            keyRef = int(currentDriverIndexNum)
            # print ' keyRef is >> ', keyRef
            # print 'type is >>', type(keyRef)
            ### get the current keyRef
            # ####keyRef = keyRefList[keyIndex]

            ##get the current time offset
            currentTimeOffset = timeOffsetList[key % targetKeyLoopLen]
            ##get the current tangent type
            inTangent = inTangentList[key % targetKeyLoopLen]
            outTangent = outTangentList[key % targetKeyLoopLen]
            ## get the current value multiplier
            currentValueMult = valueMultList[key % targetKeyLoopLen]
            ## get the current value offset
            currentValueOffset = valueOffsetList[key % targetKeyLoopLen]
            # calculate target key time
            keyTime = self.calculateKeyTime(currentTimeOffset, keyIndex,
keyRef)
            # #calculate target key value
            keyValue = self.calculateKeyValue(currentValueMult,
currentValueOffset, keyIndex)
            # ### ########################create keyframe
            mc.setKeyframe(target, time=keyTime, inTangentType=inTangent,
outTangentType=outTangent,\
                value=keyValue)
```

```
            #add to loopIndex only every third time through
            if loopCounter == targetKeyLoopLen:
                loopIndex += 1
                loopCounter = 0
            loopCounter += 1
            keyIndex+=1

        print '>>>> Keyframes set on ', target


class AnimToolsWindow(qg.QWidget):
    def __init__(self):
        qg.QWidget.__init__(self)
        # set geometry
        self.setGeometry(300, 500, 350, 250)
        #set window title
        self.setWindowTitle('Animation Tools')

        self.dataDir = getDataDir()

        self.__createWidgets__()
        self.__connectWidgets__()
        self.__layoutWidgets__()

        # create a list of page widgets
        self.pageList = []
        self.getTabName()
        self.populateComboBox()

    def __createWidgets__(self):
        buttonWidth = 80

        self.tabWidget = qg.QTabWidget()

        self.refreshAllKeysButton = qg.QPushButton('Refresh all keys',
minimumHeight=30, parent=self)
        self.refreshAllKeysButton.setStyleSheet('background-
color:#554444;\
        border: 2px solid #222222')
        self.saveAllDefaultsButton = qg.QPushButton('Save defaults',
minimumHeight=30, maximumWidth=buttonWidth, parent=self)
        ##create tab control widgets and group box
        self.addTabButton = qg.QPushButton('Add tab',
minimumWidth=buttonWidth, parent=self)
        self.tabNameInput = qg.QLineEdit(parent=self)
        self.tabNameInput.setPlaceholderText('select node or enter tab
name')
        ###
        self.renameTabButton = qg.QPushButton('Rename tab',
minimumWidth=buttonWidth, parent=self)
        self.deleteTabButton = qg.QPushButton('Delete tab',
minimumWidth=buttonWidth, parent=self)
        self.duplicateTabButton = qg.QPushButton('Duplicate tab',
minimumWidth=buttonWidth, parent=self)

        self.defaultsPathLabel = qg.QLabel(self.dataDir)
```

```
        self.createTabsFromFiles = qg.QPushButton('Create tabs from
files', minimumWidth=buttonWidth, parent=self)
        # self.tabsFromFilesInput = qg.QLineEdit(maximumWidth=200,
parent=self)
        self.tabsFromFilesCombo = qg.QComboBox(maximumWidth=200,
parent=self, duplicatesEnabled=False)
        # self.tabsFromFilesInput.setPlaceholderText('enter folder name')
        self.refreshComboButton = qg.QPushButton('Refresh list',
parent=self)

    def __connectWidgets__(self):
        self.addTabButton.pressed.connect(self.addAutoAnimPage)
        self.deleteTabButton.pressed.connect(self.deletePage)
        self.duplicateTabButton.pressed.connect(self.duplicateTab)

        self.refreshAllKeysButton.pressed.connect(self.refreshAllKeys)
        self.saveAllDefaultsButton.pressed.connect(self.saveAllDefaults)

        self.renameTabButton.pressed.connect(self.renameCurrentTab)
        # connect default buttons and widgets

self.tabsFromFilesCombo.currentIndexChanged.connect(self.updatePath)
        self.createTabsFromFiles.pressed.connect(self.tabsFromFiles)
        ### needs to refresh not add incrementally
        self.refreshComboButton.pressed.connect(self.refreshComboBox)

    def __layoutWidgets__(self):

        ## create tab control; 2 x HLayouts and 1 x Vlayout and add
widgets
        self.addTabLayout = qg.QHBoxLayout()
        self.addTabLayout.addWidget(self.tabNameInput)
        self.addTabLayout.addWidget(self.addTabButton)


        self.renameTabLayout = qg.QHBoxLayout()
        self.renameTabLayout.addWidget(self.renameTabButton)
        self.renameTabLayout.addWidget(self.deleteTabButton)
        self.renameTabLayout.addWidget(self.duplicateTabButton)

        self.tabsFromFilesLayout = qg.QHBoxLayout()
        self.tabsFromFilesLayout.addWidget(self.refreshComboButton)
        self.tabsFromFilesLayout.addWidget(self.tabsFromFilesCombo)
        self.tabsFromFilesLayout.addWidget(self.createTabsFromFiles)


        self.tabCtrlList = [self.addTabLayout, self.renameTabLayout,
self.defaultsPathLabel, self.tabsFromFilesLayout]
        self.tabControlGB = createGroupBox(self, title='Add, delete and
rename tabs', itemList=self.tabCtrlList)

        ###create an HBoxLayout for save all defaults and refresh all
keys
        self.refreshAllLayout = qg.QHBoxLayout()
        self.refreshAllLayout.addWidget(self.refreshAllKeysButton)
        self.refreshAllLayout.addWidget(self.saveAllDefaultsButton)
        # Create main layout
        ##I think the following line can also be; self.setLayout(vBox)
```

```
        self.vBox = qg.QVBoxLayout(self)
        self.vBox.addWidget(self.tabWidget)
        self.vBox.addLayout(self.refreshAllLayout)
        self.vBox.addWidget(self.tabControlGB)

        ##################### tabs from files

    def populateComboBox(self):
        #get folder directory for combo box
        folderDirectory = getDataDir()
        #get folders
        foldersList = []
        folders = os.listdir(folderDirectory)
        for myFolder in folders:
            foldersList.append(myFolder)
            self.tabsFromFilesCombo.addItem(myFolder)

    def refreshComboBox(self):
        self.tabsFromFilesCombo.clear()
        self.populateComboBox()
        # print 'refreshing combo box'

    def tabsFromFiles(self):
        # if a txt file is chosen create one tab
        currentText = str(self.tabsFromFilesCombo.currentText())
        currentTextList = currentText.split('.')
        if len(currentTextList) > 1:

            #get the file path
            fileDirectory = self.defaultsPathLabel.text()
            # open file and read it
            fileData = readFile(fileDirectory)
            #make one tab page
            tabPage = AutoAnimPage(targetNodeEntry='')
            tabName = currentTextList[0]
            # # add tabPage to the tabWidget
            self.tabWidget.addTab(tabPage, tabName)
            # make new page widget the cuurent one
            self.tabWidget.setCurrentWidget(tabPage)
            #add tabPage to pageList
            self.pageList.append(tabPage)
            ## get folder name and file name to populate tab
            folderName = None
            filename = tabName
            #### make function in tab class to populate itself
            tabPage.poulateTabFromFile(fileData, folderName, filename)
        else:
            # get folder directory
            folderDirectory = self.defaultsPathLabel.text()
            # get the number of files in that folder
            files = os.listdir(folderDirectory)
            for myFile in files:
                # get file directory
                fileDirectory = str(folderDirectory) + '/' + str(myFile)
                # open file and read it
                fileData = readFile(fileDirectory)
                # add tab using fileName as name
                tabPage = AutoAnimPage(targetNodeEntry='')
```

```
                tabName = (myFile.split('.'))[0]
                # # add tabPage to the tabWidget
                self.tabWidget.addTab(tabPage, tabName)
                # make new page widget the cuurent one
                self.tabWidget.setCurrentWidget(tabPage)
                #add tabPage to pageList
                self.pageList.append(tabPage)

                ################## get folder name and file name to
populate tab
                folderPathSplitList = folderDirectory.split('/')
                listLen = len(folderPathSplitList)
                folderName = folderPathSplitList[listLen - 1]
                filename = tabName
                #### make function in tab class to populate itself
                tabPage.poulateTabFromFile(fileData, folderName,
filename)

    def updatePath(self):
        self.dataDir = getDataDir()
        folderName = self.tabsFromFilesCombo.currentText()
        dataFolderDirectory = self.dataDir + '/' + folderName
        self.defaultsPathLabel.setText(dataFolderDirectory)

    def getTabName(self):
        #get tab number for default name
        pageListLength = len(self.pageList)
        tabNumber = pageListLength + 1
        ## get Maya selection
        selectedNode = mc.ls(selection=True)

        ## get the current input from the "add tab text input"
        tabName = self.tabNameInput.text()
        targetNodeEntry = ''

        ## if there is a node selected
        if selectedNode:
            # tabName = node.channel; get node and channel
            tabName = getNodeName()
            targetNodeEntry = tabName

        ## if a node is not selected
        else:
            # if tabname text has default text in it
            if tabName == 'select node or enter tab name':
                tabName = 'tab {tabNumber}'.format(tabNumber=tabNumber)
            # if tabname text has no text in it
            elif tabName == '':
                tabName = 'tab {tabNumber}'.format(tabNumber=tabNumber)
            else:
                tabName = self.tabNameInput.text()

        return tabName, targetNodeEntry

    def addAutoAnimPage(self):
```

```
        tabName = self.getTabName()[0]
        targetNodeEntry = self.getTabName()[1]

        #create the new tab page; instance of AutoAnimPage class. pass in
the name of the page to
        #be used as the target field if there is a node selected.
        tabPage = AutoAnimPage(targetNodeEntry=targetNodeEntry)
        # add tabPage to the tabWidget
        self.tabWidget.addTab(tabPage, tabName)
        # make the newly created page the current one
        self.tabWidget.setCurrentWidget(tabPage)

                #get previous tab index
        priorTabIndex = len(self.pageList) - 1
        # print 'priorTabIndex is ', priorTabIndex
        ## get previous tab object
        if priorTabIndex >= 0:
            priorTab = self.tabWidget.widget(priorTabIndex)
            # print 'priorTab is ', priorTab
            # get priorTab folderName
            folderName = priorTab.getFileFolderNames()[1]
            #### set the folder name
            tabPage.populateDefaultFolderText(folderName)

        #add tabPage to pageList
        self.pageList.append(tabPage)


        #### can I do the same for the file name?
        tabPage.populateDefaultFileText()

    def duplicateTab(self):
        ### select the current tab object
        currentIndex = self.tabWidget.currentIndex()
        #### seems that the ".widget(index)" is inherited from QWidget
        currentTab = self.tabWidget.widget(currentIndex)
                ##get current tab name
        tabName = self.tabWidget.tabText(currentIndex) + '_duplicate'
        #### get current tab folderName and fileName
        fileName = currentTab.getFileFolderNames()[0]
        folderName = currentTab.getFileFolderNames()[1]
        # print 'fileName is ',fileName
        # print 'folderName is ',folderName

        ###get current tab data
        tabData = currentTab.getTabData()
        #make one tab page
        tabPage = AutoAnimPage(targetNodeEntry='')

        # # add tabPage to the tabWidget
        self.tabWidget.addTab(tabPage, tabName)
        # make new page widget the cuurent one
        self.tabWidget.setCurrentWidget(tabPage)
        #add tabPage to pageList
        self.pageList.append(tabPage)

        #### make function in tab class to populate itself
        tabPage.poulateTabFromFile(tabData, folderName, fileName)
```

```
    def renameCurrentTab(self):
        # get new name
        tabName = self.getTabName()[0]
        # get the current tab
        currentIndex = self.tabWidget.currentIndex()
        #rename current tab
        self.tabWidget.setTabText(currentIndex, tabName)

    def deletePage(self):
        currentIndex = self.tabWidget.currentIndex()
        # print currentIndex
        # remove current page from the page list
        currentPage = self.pageList[currentIndex]
        self.pageList.remove(currentPage)
        #delete the current from tab widget
        self.tabWidget.removeTab(currentIndex)

    def refreshAllKeys(self):
        print 'Refreshing all keyframes;'
        #loop over the list of page widgets
        for page in self.pageList:
            page.setKeys()

    def saveAllDefaults(self):
        print 'Saving all default values;'
        #loop over the list of page widgets
        for page in self.pageList:
            page.saveDefaults()

def showWindow():
    global window
    if not window:
        window = AnimToolsWindow()
        window.show()

showWindow()
```