# -\*- coding: utf-8 -\*-

# PyXBMCt framework module

#

# PyXBMCt is a mini-framework for creating Kodi (XBMC) Python addons

# with arbitrary UI made of Controls - decendants of xbmcgui.Control class.

# The framework uses image textures from Kodi Confluence skin.

#

# Licence: GPL v.3 http://www.gnu.org/licenses/gpl.html

"""

``pyxbmct.addonwindow`` module contains all classes and constants of PyXBMCt framework

"""

import os

import xbmc

import xbmcgui

from xbmcaddon import Addon

import uservar

from resources.libs import wizard as wiz

ADDON\_ID = uservar.ADDON\_ID

ADDONPATH = wiz.addonInfo(ADDON\_ID,'path')

\_images = os.path.join(ADDONPATH, 'resources', 'skins', 'DefaultSkin', 'media')

# Text alighnment constants. Mixed variants are obtained by bit OR (|)

ALIGN\_LEFT = 0

"""Align left"""

ALIGN\_RIGHT = 1

"""Align right"""

ALIGN\_CENTER\_X = 2

"""Align center horisontally"""

ALIGN\_CENTER\_Y = 4

"""Align center vertically"""

ALIGN\_CENTER = 6

"""Align center by both axis"""

ALIGN\_TRUNCATED = 8

"""Align truncated"""

ALIGN\_JUSTIFY = 10

"""Align justify"""

# Kodi key action codes.

# More codes available in xbmcgui module

ACTION\_PREVIOUS\_MENU = 10

"""ESC action"""

ACTION\_NAV\_BACK = 92

"""Backspace action"""

ACTION\_MOVE\_LEFT = 1

"""Left arrow key"""

ACTION\_MOVE\_RIGHT = 2

"""Right arrow key"""

ACTION\_MOVE\_UP = 3

"""Up arrow key"""

ACTION\_MOVE\_DOWN = 4

"""Down arrow key"""

ACTION\_MOUSE\_WHEEL\_UP = 104

"""Mouse wheel up"""

ACTION\_MOUSE\_WHEEL\_DOWN = 105

"""Mouse wheel down"""

ACTION\_MOUSE\_DRAG = 106

"""Mouse drag"""

ACTION\_MOUSE\_MOVE = 107

"""Mouse move"""

ACTION\_MOUSE\_LEFT\_CLICK = 100

"""Mouse click"""

def \_set\_textures(textures, kwargs):

"""Set texture arguments for controls."""

for texture in textures.keys():

if kwargs.get(texture) is None:

kwargs[texture] = textures[texture]

class AddonWindowError(Exception):

"""Custom exception"""

pass

class Label(xbmcgui.ControlLabel):

"""

Label(label, font=None, textColor=None, disabledColor=None, alignment=0,hasPath=False, angle=0)

ControlLabel class.

Implements a simple text label.

:param label: string or unicode - text string.

:param font: string - font used for label text. (e.g. 'font13')

:param textColor: hexstring - color of enabled label's label. (e.g. '0xFFFFFFFF')

:param disabledColor: hexstring - color of disabled label's label. (e.g. '0xFFFF3300')

:param alignment: integer - alignment of label - \*Note, see xbfont.h

:param hasPath: bool - True=stores a path / False=no path.

:param angle: integer - angle of control. (+ rotates CCW, - rotates CW)

.. note:: After you create the control, you need to add it to the window with placeControl().

Example::

self.label = Label('Status', angle=45)

"""

def \_\_new\_\_(cls, \*args, \*\*kwargs):

return super(Label, cls).\_\_new\_\_(cls, -10, -10, 1, 1, \*args, \*\*kwargs)

class FadeLabel(xbmcgui.ControlFadeLabel):

"""

FadeLabel(font=None, textColor=None, \_alignment=0)

Control that scrolls label text.

Implements a text label that can auto-scroll very long text.

:param font: string - font used for label text. (e.g. 'font13')

:param textColor: hexstring - color of fadelabel's labels. (e.g. '0xFFFFFFFF')

:param \_alignment: integer - alignment of label - \*Note, see xbfont.h

.. note:: After you create the control, you need to add it to the window with placeControl().

Example::

self.fadelabel = FadeLabel(textColor='0xFFFFFFFF')

"""

def \_\_new\_\_(cls, \*args, \*\*kwargs):

return super(FadeLabel, cls).\_\_new\_\_(cls, -10, -10, 1, 1, \*args, \*\*kwargs)

class TextBox(xbmcgui.ControlTextBox):

"""

TextBox(font=None, textColor=None)

ControlTextBox class

Implements a box for displaying multi-line text.

Long text is truncated from below. Also supports auto-scrolling.

:param font: string - font used for text. (e.g. 'font13')

:param textColor: hexstring - color of textbox's text. (e.g. '0xFFFFFFFF')

.. note:: After you create the control, you need to add it to the window with placeControl().

Example::

self.textbox = TextBox(textColor='0xFFFFFFFF')

"""

def \_\_new\_\_(cls, \*args, \*\*kwargs):

return super(TextBox, cls).\_\_new\_\_(cls, -10, -10, 1, 1, \*args, \*\*kwargs)

class Image(xbmcgui.ControlImage):

"""

Image(filename, aspectRatio=0, colorDiffuse=None)

ControlImage class.

Implements a box for displaying .jpg, .png, and .gif images.

:param filename: string - image filename.

:param colorKey: hexString - (example, '0xFFFF3300')

:param aspectRatio: integer - (values 0 = stretch (default), 1 = scale up (crops), 2 = scale down (black bars)

:param colorDiffuse: hexString - (example, '0xC0FF0000' (red tint)).

.. note:: After you create the control, you need to add it to the window with placeControl().

Example::

self.image = Image('d:\images\picture.jpg', aspectRatio=2)

"""

def \_\_new\_\_(cls, \*args, \*\*kwargs):

return super(Image, cls).\_\_new\_\_(cls, -10, -10, 1, 1, \*args, \*\*kwargs)

class Button(xbmcgui.ControlButton):

"""

Button(label, focusTexture=None, noFocusTexture=None, textOffsetX=CONTROL\_TEXT\_OFFSET\_X, textOffsetY=CONTROL\_TEXT\_OFFSET\_Y, alignment=4, font=None, textColor=None, disabledColor=None, angle=0, shadowColor=None, focusedColor=None)

ControlButton class.

Implements a clickable button.

:param label: string or unicode - text string.

:param focusTexture: string - filename for focus texture.

:param noFocusTexture: string - filename for no focus texture.

:param textOffsetX: integer - x offset of label.

:param textOffsetY: integer - y offset of label.

:param alignment: integer - alignment of label - \*Note, see xbfont.h

:param font: string - font used for label text. (e.g. 'font13')

:param textColor: hexstring - color of enabled button's label. (e.g. '0xFFFFFFFF')

:param disabledColor: hexstring - color of disabled button's label. (e.g. '0xFFFF3300')

:param angle: integer - angle of control. (+ rotates CCW, - rotates CW)

:param shadowColor: hexstring - color of button's label's shadow. (e.g. '0xFF000000')

:param focusedColor: hexstring - color of focused button's label. (e.g. '0xFF00FFFF')

.. note:: After you create the control, you need to add it to the window with placeControl().

Example::

self.button = Button('Status', font='font14')

"""

def \_\_new\_\_(cls, \*args, \*\*kwargs):

textures = {'focusTexture': os.path.join(\_images, 'Button', 'KeyboardKey.png'),

'noFocusTexture': os.path.join(\_images, 'Button', 'KeyboardKeyNF.png')}

\_set\_textures(textures, kwargs)

if kwargs.get('alignment') is None:

kwargs['alignment'] = ALIGN\_CENTER

return super(Button, cls).\_\_new\_\_(cls, -10, -10, 1, 1, \*args, \*\*kwargs)

class RadioButton(xbmcgui.ControlRadioButton):

"""

RadioButton(label, focusTexture=None, noFocusTexture=None, textOffsetX=None, textOffsetY=None, \_alignment=None, font=None, textColor=None, disabledColor=None, angle=None, shadowColor=None, focusedColor=None, focusOnTexture=None, noFocusOnTexture=None, focusOffTexture=None, noFocusOffTexture=None)

ControlRadioButton class.

Implements a 2-state switch.

:param label: string or unicode - text string.

:param focusTexture: string - filename for focus texture.

:param noFocusTexture: string - filename for no focus texture.

:param textOffsetX: integer - x offset of label.

:param textOffsetY: integer - y offset of label.

:param \_alignment: integer - alignment of label - \*Note, see xbfont.h

:param font: string - font used for label text. (e.g. 'font13')

:param textColor: hexstring - color of enabled radio button's label. (e.g. '0xFFFFFFFF')

:param disabledColor: hexstring - color of disabled radio button's label. (e.g. '0xFFFF3300')

:param angle: integer - angle of control. (+ rotates CCW, - rotates CW)

:param shadowColor: hexstring - color of radio button's label's shadow. (e.g. '0xFF000000')

:param focusedColor: hexstring - color of focused radio button's label. (e.g. '0xFF00FFFF')

:param focusOnTexture: string - filename for radio focused/checked texture.

:param noFocusOnTexture: string - filename for radio not focused/checked texture.

:param focusOffTexture: string - filename for radio focused/unchecked texture.

:param noFocusOffTexture: string - filename for radio not focused/unchecked texture.

.. note:: To customize RadioButton all 4 abovementioned textures need to be provided.

.. note:: After you create the control, you need to add it to the window with placeControl().

Example::

self.radiobutton = RadioButton('Status', font='font14')

"""

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if int(xbmc.getInfoLabel('System.BuildVersion')[:2]) >= 13:

textures = {'focusTexture': os.path.join(\_images, 'RadioButton', 'MenuItemFO.png'),

'noFocusTexture': os.path.join(\_images, 'RadioButton', 'MenuItemNF.png'),

'focusOnTexture': os.path.join(\_images, 'RadioButton', 'radiobutton-focus.png'),

'noFocusOnTexture': os.path.join(\_images, 'RadioButton', 'radiobutton-focus.png'),

'focusOffTexture': os.path.join(\_images, 'RadioButton', 'radiobutton-nofocus.png'),

'noFocusOffTexture': os.path.join(\_images, 'RadioButton', 'radiobutton-nofocus.png')}

else: # This is for compatibility with Frodo and earlier versions.

textures = {'focusTexture': os.path.join(\_images, 'RadioButton', 'MenuItemFO.png'),

'noFocusTexture': os.path.join(\_images, 'RadioButton', 'MenuItemNF.png'),

'TextureRadioFocus': os.path.join(\_images, 'RadioButton', 'radiobutton-focus.png'),

'TextureRadioNoFocus': os.path.join(\_images, 'RadioButton', 'radiobutton-nofocus.png')}

\_set\_textures(textures, kwargs)

return super(RadioButton, cls).\_\_new\_\_(cls, -10, -10, 1, 1, \*args, \*\*kwargs)

class Edit(xbmcgui.ControlEdit):

"""

Edit(label, font=None, textColor=None, disabledColor=None, \_alignment=0, focusTexture=None, noFocusTexture=None, isPassword=False)

ControlEdit class.

Implements a clickable text entry field with an on-screen keyboard.

:param label: string or unicode - text string.

:param font: [opt] string - font used for label text. (e.g. 'font13')

:param textColor: [opt] hexstring - color of enabled label's label. (e.g. '0xFFFFFFFF')

:param disabledColor: [opt] hexstring - color of disabled label's label. (e.g. '0xFFFF3300')

:param \_alignment: [opt] integer - alignment of label - \*Note, see xbfont.h

:param focusTexture: [opt] string - filename for focus texture.

:param noFocusTexture: [opt] string - filename for no focus texture.

:param isPassword: [opt] bool - if true, mask text value.

.. note:: You can use the above as keywords for arguments and skip certain optional arguments.

Once you use a keyword, all following arguments require the keyword.

After you create the control, you need to add it to the window with ``palceControl()``.

Example::

self.edit = Edit('Status')

"""

def \_\_new\_\_(cls, \*args, \*\*kwargs):

textures = {'focusTexture': os.path.join(\_images, 'Edit', 'button-focus.png'),

'noFocusTexture': os.path.join(\_images, 'Edit', 'black-back2.png')}

\_set\_textures(textures, kwargs)

return super(Edit, cls).\_\_new\_\_(cls, -10, -10, 1, 1, \*args, \*\*kwargs)

class List(xbmcgui.ControlList):

"""

List(font=None, textColor=None, buttonTexture=None, buttonFocusTexture=None, selectedColor=None, \_imageWidth=10, \_imageHeight=10, \_itemTextXOffset=10, \_itemTextYOffset=2, \_itemHeight=27, \_space=2, \_alignmentY=4)

ControlList class.

Implements a scrollable list of items.

:param font: string - font used for items label. (e.g. 'font13')

:param textColor: hexstring - color of items label. (e.g. '0xFFFFFFFF')

:param buttonTexture: string - filename for no focus texture.

:param buttonFocusTexture: string - filename for focus texture.

:param selectedColor: integer - x offset of label.

:param \_imageWidth: integer - width of items icon or thumbnail.

:param \_imageHeight: integer - height of items icon or thumbnail.

:param \_itemTextXOffset: integer - x offset of items label.

:param \_itemTextYOffset: integer - y offset of items label.

:param \_itemHeight: integer - height of items.

:param \_space: integer - space between items.

:param \_alignmentY: integer - Y-axis alignment of items label - \*Note, see xbfont.h

.. note:: After you create the control, you need to add it to the window with placeControl().

Example::

self.cList = List('font14', space=5)

"""

def \_\_new\_\_(cls, \*args, \*\*kwargs):

textures = {'buttonTexture': os.path.join(\_images, 'List', 'MenuItemNF1.png'),

'buttonFocusTexture': os.path.join(\_images, 'List', 'MenuItemFO1.png')}

\_set\_textures(textures, kwargs)

return super(List, cls).\_\_new\_\_(cls, -10, -10, 1, 1, \*args, \*\*kwargs)

class Slider(xbmcgui.ControlSlider):

"""

Slider(textureback=None, texture=None, texturefocus=None)

ControlSlider class.

Implements a movable slider for adjusting some value.

:param textureback: string - image filename.

:param texture: string - image filename.

:param texturefocus: string - image filename.

.. note:: After you create the control, you need to add it to the window with placeControl().

Example::

self.slider = Slider()

"""

def \_\_new\_\_(cls, \*args, \*\*kwargs):

textures = {'textureback': os.path.join(\_images, 'Slider', 'osd\_slider\_bg.png'),

'texture': os.path.join(\_images, 'Slider', 'osd\_slider\_nibNF.png'),

'texturefocus': os.path.join(\_images, 'Slider', 'osd\_slider\_nib.png')}

\_set\_textures(textures, kwargs)

return super(Slider, cls).\_\_new\_\_(cls, -10, -10, 1, 1, \*args, \*\*kwargs)

class AbstractWindow(object):

"""

Top-level control window.

The control windows serves as a parent widget for other XBMC UI controls

much like Tkinter.Tk or PyQt QWidget class.

This class is a basic "skeleton" for a control window.

.. warning:: This is an abstract class and is not supposed to be instantiated directly!

"""

def \_\_init\_\_(self):

self.actions\_connected = []

self.controls\_connected = []

def setGeometry(self, width\_, height\_, rows\_, columns\_, pos\_x=-1, pos\_y=-1):

"""

Set width, height, Grid layout, and coordinates (optional) for a new control window.

:param width\_: widgh of the created window.

:param height\_: height of the created window.

:param rows\_: # rows of the Grid layout to place controls on.

:param columns\_: # colums of the Grid layout to place controls on.

:param pos\_x: (opt) x coordinate of the top left corner of the window.

:param pos\_y: (opt) y coordinates of the top left corner of the window.

If pos\_x and pos\_y are not privided, the window will be placed

at the center of the screen.

Example::

self.setGeometry(400, 500, 5, 4)

"""

self.width = width\_

self.height = height\_

self.rows = rows\_

self.columns = columns\_

if pos\_x > 0 and pos\_y > 0:

self.x = pos\_x

self.y = pos\_y

else:

self.x = 640 - self.width/2

self.y = 360 - self.height/2

self.\_setGrid()

def \_setGrid(self):

"""

Set window grid layout of rows x columns.

This is a helper method not to be called directly.

"""

self.grid\_x = self.x

self.grid\_y = self.y

self.tile\_width = self.width / self.columns

self.tile\_height = self.height / self.rows

def placeControl(self, control, row, column, rowspan=1, columnspan=1, pad\_x=5, pad\_y=5):

"""

Place a control within the window grid layout.

:param control: control instance to be placed in the grid.

:param row: row number where to place the control (starts from 0).

:param column: column number where to place the control (starts from 0).

:param rowspan: set when the control needs to occupy several rows.

:param columnspan: set when the control needs to occupy several columns.

:param pad\_x: horisontal padding.

:param pad\_y: vertical padding.

:raises: :class:`AddonWindowError` if a grid has not yet been set.

Use ``pad\_x`` and ``pad\_y`` to adjust control's aspect.

Negative padding values can be used to make a control overlap with grid cells next to it, if necessary.

Example::

self.placeControl(self.label, 0, 1)

"""

try:

control\_x = (self.grid\_x + self.tile\_width \* column) + pad\_x

control\_y = (self.grid\_y + self.tile\_height \* row) + pad\_y

control\_width = self.tile\_width \* columnspan - 2 \* pad\_x

control\_height = self.tile\_height \* rowspan - 2 \* pad\_y

except AttributeError:

raise AddonWindowError('Window geometry is not defined! Call setGeometry first.')

control.setPosition(control\_x, control\_y)

control.setWidth(control\_width)

control.setHeight(control\_height)

self.addControl(control)

self.setAnimation(control)

def getX(self):

"""Get X coordinate of the top-left corner of the window."""

try:

return self.x

except AttributeError:

raise AddonWindowError('Window geometry is not defined! Call setGeometry first.')

def getY(self):

"""Get Y coordinate of the top-left corner of the window."""

try:

return self.y

except AttributeError:

raise AddonWindowError('Window geometry is not defined! Call setGeometry first.')

def getWindowWidth(self):

"""Get window width."""

try:

return self.width

except AttributeError:

raise AddonWindowError('Window geometry is not defined! Call setGeometry first.')

def getWindowHeight(self):

"""Get window height."""

try:

return self.height

except AttributeError:

raise AddonWindowError('Window geometry is not defined! Call setGeometry first.')

def getRows(self):

"""

Get grid rows count.

:raises: :class:`AddonWindowError` if a grid has not yet been set.

"""

try:

return self.rows

except AttributeError:

raise AddonWindowError('Grid layot is not set! Call setGeometry first.')

def getColumns(self):

"""

Get grid columns count.

:raises: :class:`AddonWindowError` if a grid has not yet been set.

"""

try:

return self.columns

except AttributeError:

raise AddonWindowError('Grid layout is not set! Call setGeometry first.')

def connect(self, event, callable):

"""

Connect an event to a function.

:param event: event to be connected.

:param callable: callable object the event is connected to.

An event can be an inctance of a Control object or an integer key action code.

Several basic key action codes are provided by PyXBMCt. ``xbmcgui`` module

provides more action codes.

You can connect the following Controls: :class:`Button`, :class:`RadioButton`

and :class:`List`. Other Controls do not generate any control events when activated

so their connections won't work.

To catch :class:`Slider` events you need to connect the following key actions:

``ACTION\_MOVE\_LEFT``, ``ACTION\_MOVE\_RIGHT`` and ``ACTION\_MOUSE\_DRAG``, and do a check

whether the ``Slider`` instance is focused.

``callable`` parameter is a function or a method to be executed on when the event is fired.

.. warning:: For connection you must provide a function object without brackets ``()``,

not a function call!

``lambda`` can be used as to call another function or method with parameters known at runtime.

Examples::

self.connect(self.exit\_button, self.close)

or::

self.connect(ACTION\_NAV\_BACK, self.close)

"""

try:

self.disconnect(event)

except AddonWindowError:

if type(event) == int:

self.actions\_connected.append([event, callable])

else:

self.controls\_connected.append([event, callable])

def connectEventList(self, events, function):

"""

Connect a list of controls/action codes to a function.

See :meth:`connect` docstring for more info.

"""

[self.connect(event, function) for event in events]

def disconnect(self, event):

"""

Disconnect an event from a function.

An event can be an inctance of a Control object or an integer key action code

which has previously been connected to a function or a method.

:param event: event to be disconnected.

:raises: :class:`AddonWindowError` if an event is not connected to any function.

Examples::

self.disconnect(self.exit\_button)

or::

self.disconnect(ACTION\_NAV\_BACK)

"""

if type(event) == int:

event\_list = self.actions\_connected

else:

event\_list = self.controls\_connected

for index in range(len(event\_list)):

if event == event\_list[index][0]:

event\_list.pop(index)

break

else:

raise AddonWindowError('The action or control %s is not connected!' % event)

def disconnectEventList(self, events):

"""

Disconnect a list of controls/action codes from functions.

See :func:`disconnect` docstring for more info.

:param events: the list of events to be disconnected.

:raises: :class:`AddonWindowError` if at least one event in the list

is not connected to any function.

"""

[self.disconnect(event) for event in events]

def \_executeConnected(self, event, connected\_list):

"""

Execute a connected event (an action or a control).

This is a helper method not to be called directly.

"""

for item in connected\_list:

if event == item[0]:

item[1]()

break

def setAnimation(self, control):

"""

Set animation for control

:param control: control for which animation is set.

This method is called automatically to set animation properties for all controls

added to the current addon window instance -- both for built-in controls

(window background, title bar etc.) and for controls added with :meth:`placeControl`.

It receives a control instance as the 2nd positional argument (besides ``self``).

By default the method does nothing, i.e. no animation is set for controls.

To add animation you need to re-implement this method in your child class.

E.g::

def setAnimation(self, control):

control.setAnimations([('WindowOpen', 'effect=fade start=0 end=100 time=1000',),

('WindowClose', 'effect=fade start=100 end=0 time=1000',)])

"""

pass

class AddonWindow(AbstractWindow):

"""

Top-level control window.

The control windows serves as a parent widget for other XBMC UI controls

much like ``Tkinter.Tk`` or PyQt ``QWidget`` class.

This is an abstract class which is not supposed to be instantiated directly

and will raise exeptions. It is designed to be implemented in a grand-child class

with the second inheritance from ``xbmcgui.Window`` or ``xbmcgui.WindowDialog``

in a direct child class.

This class provides a control window with a background and a header

similar to top-level widgets of desktop UI frameworks.

.. warning:: This is an abstract class and is not supposed to be instantiated directly!

"""

def \_\_init\_\_(self, title=''):

"""Constructor method."""

super(AddonWindow, self).\_\_init\_\_()

self.\_setFrame(title)

def \_setFrame(self, title):

"""

Set window frame

Define paths to images for window background and title background textures,

and set control position adjustment constants used in setGrid.

This is a helper method not to be called directly.

"""

# Window background image

self.background\_img = os.path.join(\_images, 'AddonWindow', 'ContentPanel.png')

# Background for a window header

self.title\_background\_img = os.path.join(\_images, 'AddonWindow', 'dialogheader.png')

# Horisontal adjustment for a header background if the main background has transparent edges.

self.X\_MARGIN = 5

# Vertical adjustment for a header background if the main background has transparent edges

self.Y\_MARGIN = 5

# Header position adjustment if the main backround has visible borders.

self.Y\_SHIFT = 4

# The height of a window header (for the title background and the title label).

self.HEADER\_HEIGHT = 35

self.background = xbmcgui.ControlImage(-10, -10, 1, 1, self.background\_img)

self.addControl(self.background)

self.setAnimation(self.background)

self.title\_background = xbmcgui.ControlImage(-10, -10, 1, 1, self.title\_background\_img)

self.addControl(self.title\_background)

self.setAnimation(self.title\_background)

self.title\_bar = xbmcgui.ControlLabel(-10, -10, 1, 1, title, alignment=ALIGN\_CENTER, textColor='0xFFFFA500',

font='font13\_title')

self.addControl(self.title\_bar)

self.setAnimation(self.title\_bar)

self.window\_close\_button = xbmcgui.ControlButton(-100, -100, 60, 30, '',

focusTexture=os.path.join(\_images, 'AddonWindow', 'DialogCloseButton-focus.png'),

noFocusTexture=os.path.join(\_images, 'AddonWindow', 'DialogCloseButton.png'))

self.addControl(self.window\_close\_button)

self.setAnimation(self.window\_close\_button)

def setGeometry(self, width\_, height\_, rows\_, columns\_, pos\_x=-1, pos\_y=-1, padding=5):

"""

Set width, height, Grid layout, and coordinates (optional) for a new control window.

:param width\_: new window width in pixels.

:param height\_: new window height in pixels.

:param rows\_: # of rows in the Grid layout to place controls on.

:param columns\_: # of colums in the Grid layout to place controls on.

:param pos\_x: (optional) x coordinate of the top left corner of the window.

:param pos\_y: (optional) y coordinate of the top left corner of the window.

:param padding: (optional) padding between outer edges of the window

and controls placed on it.

If ``pos\_x`` and ``pos\_y`` are not privided, the window will be placed

at the center of the screen.

Example::

self.setGeometry(400, 500, 5, 4)

"""

self.win\_padding = padding

super(AddonWindow, self).setGeometry(width\_, height\_, rows\_, columns\_, pos\_x, pos\_y)

self.background.setPosition(self.x, self.y)

self.background.setWidth(self.width)

self.background.setHeight(self.height)

self.title\_background.setPosition(self.x + self.X\_MARGIN, self.y + self.Y\_MARGIN + self.Y\_SHIFT)

self.title\_background.setWidth(self.width - 2 \* self.X\_MARGIN)

self.title\_background.setHeight(self.HEADER\_HEIGHT)

self.title\_bar.setPosition(self.x + self.X\_MARGIN, self.y + self.Y\_MARGIN + self.Y\_SHIFT)

self.title\_bar.setWidth(self.width - 2 \* self.X\_MARGIN)

self.title\_bar.setHeight(self.HEADER\_HEIGHT)

self.window\_close\_button.setPosition(self.x + self.width - 70, self.y + self.Y\_MARGIN + self.Y\_SHIFT)

def \_setGrid(self):

"""

Set window grid layout of rows \* columns.

This is a helper method not to be called directly.

"""

self.grid\_x = self.x + self.X\_MARGIN + self.win\_padding

self.grid\_y = self.y + self.Y\_MARGIN + self.Y\_SHIFT + self.HEADER\_HEIGHT + self.win\_padding

self.tile\_width = (self.width - 2 \* (self.X\_MARGIN + self.win\_padding))/self.columns

self.tile\_height = (

self.height - self.HEADER\_HEIGHT - self.Y\_SHIFT - 2 \* (self.Y\_MARGIN + self.win\_padding))/self.rows

def setWindowTitle(self, title=''):

"""

Set window title.

.. warning:: This method must be called \*\*AFTER\*\* (!!!) :meth:`setGeometry`,

otherwise there is some werid bug with all skin text labels set to the ``title`` text.

Example::

self.setWindowTitle('My Cool Addon')

"""

self.title\_bar.setLabel(title)

def getWindowTitle(self):

"""Get window title."""

return self.title\_bar.getLabel()

class FullWindowMixin(xbmcgui.Window):

"""An abstract class to define window event processing."""

def onAction(self, action):

"""

Catch button actions.

``action`` is an instance of :class:`xbmcgui.Action` class.

"""

if action == ACTION\_PREVIOUS\_MENU:

self.close()

else:

self.\_executeConnected(action, self.actions\_connected)

def onControl(self, control):

"""

Catch activated controls.

``control`` is an instance of :class:`xbmcgui.Control` class.

"""

if control == self.window\_close\_button:

self.close()

else:

self.\_executeConnected(control, self.controls\_connected)

class DialogWindowMixin(xbmcgui.WindowDialog):

"""An abstract class to define window event processing."""

def onAction(self, action):

"""

Catch button actions.

``action`` is an instance of class:`xbmcgui.Action` class.

"""

if action == ACTION\_PREVIOUS\_MENU:

self.close()

else:

self.\_executeConnected(action, self.actions\_connected)

def onControl(self, control):

"""

Catch activated controls.

``control`` is an instance of :class:`xbmcgui.Control` class.

"""

if control == self.window\_close\_button:

self.close()

else:

self.\_executeConnected(control, self.controls\_connected)

class BlankFullWindow(FullWindowMixin, AbstractWindow):

"""

BlankFullWindow()

Addon UI container with a solid background.

This is a blank window with a black background and without any elements whatsoever.

The decoration and layout are completely up to an addon developer.

The window controls can hide under video or music visualization.

"""

pass

class BlankDialogWindow(DialogWindowMixin, AbstractWindow):

"""

BlankDialogWindow()

Addon UI container with a transparent background.

This is a blank window with a transparent background and without any elements whatsoever.

The decoration and layout are completely up to an addon developer.

The window controls are always displayed over video or music visualization.

"""

pass

class AddonFullWindow(FullWindowMixin, AddonWindow):

"""

AddonFullWindow(title='')

Addon UI container with a solid background.

``AddonFullWindow`` instance is displayed on top of the main background image --

``self.main\_bg`` -- and can hide behind a fullscreen video or music viaualisation.

Minimal example::

addon = AddonFullWindow('My Cool Addon')

addon.setGeometry(400, 300, 4, 3)

addon.doModal()

"""

def \_\_new\_\_(cls, title='', \*args, \*\*kwargs):

return super(AddonFullWindow, cls).\_\_new\_\_(cls, \*args, \*\*kwargs)

def \_setFrame(self, title):

"""

Set the image for for the fullscreen background.

"""

# Image for the fullscreen background.

self.main\_bg\_img = os.path.join(\_images, 'AddonWindow', 'SKINDEFAULT.jpg')

# Fullscreen background image control.

self.main\_bg = xbmcgui.ControlImage(1, 1, 1280, 720, self.main\_bg\_img)

self.addControl(self.main\_bg)

super(AddonFullWindow, self).\_setFrame(title)

def setBackground(self, image=''):

"""

Set the main bacground to an image file.

:param image: path to an image file as str.

Example::

self.setBackground('/images/bacground.png')

"""

self.main\_bg.setImage(image)

class AddonDialogWindow(DialogWindowMixin, AddonWindow):

"""

AddonDialogWindow(title='')

Addon UI container with a transparent background.

.. note:: ``AddonDialogWindow`` instance is displayed on top of XBMC UI,

including fullscreen video and music visualization.

Minimal example::

addon = AddonDialogWindow('My Cool Addon')

addon.setGeometry(400, 300, 4, 3)

addon.doModal()

"""

pass