domdf-wxpython-tools

Release 0.3.0.post1

Tools and widgets for wxPython.

Dominic Davis-Foster

Mar 12, 2024

Contents

Chapter

ONE

Installation

1.1 from PyPI

```
$ python3 -m pip install domdf_wxpython_tools --user
```

1.2 from GitHub

```
$ python3 -m pip install git+https://github.com/domdfcoding/domdf_wxpython_tools@master --user
```

Chapter

TWO

API Reference

2.1 ColourPickerPanel

Based on StylePickerPanel, a Panel for selecting a list of colours, and their order

Classes:

ColourPickerPanel(parent[, id, pos, size, ...])	Based on StylePickerPanel, a Panel for selecting a list of colours, and their order.

class ColourPickerPanel (*parent*, *id=- 1*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=524288*, *name=b'panel'*, *label='Choose Colours: '*, *picker_choices=None*, *selection_choices=None*)

Bases: *StylePickerPanel*

Based on StylePickerPanel, a Panel for selecting a list of colours, and their order.

Parameters

- **parent** (Window) – The parent window.
- **id** (int) – An identifier for the panel. wx.ID_ANY is taken to mean a default. Default -1.
- **pos** (Point) – The panel position. The value wx.DefaultPosition indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform. Default (-1, -1).
- **size** (Size) – The panel size. The value wx.DefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform. Default (-1, -1).
- **style** (int) – The window style. See wxPanel. Default 524288.
- **name** (str) – The window name. Default b'panel'.
- **label** (str) – Label for the panel. Default 'Choose Colours: '.
- **picker_choices** (Optional[List[str]]) – A list of hex value choices to populate the 'picker' side of the panel with. Default None.
- **selection_choices** (Optional[List[str]]) – A list of hex value choices to populate the 'selection' side of the panel with. Default None.

Methods:

GetSelection()	Returns a list of the currently selected colours
add(event)	Event handler for adding the colour currently selected in the 'picker' to the 'selection'
get_selection()	Returns a list of the currently selected colours

continues on next page

Table 2 – continued from previous page

pick(*args)	Open a wx.ColourDialog to edit the colour currently selected in the picker.
remove(event)	Event handler for removing the colour currently selected in the 'selection'
update_preview(list_obj, axes)	Update the preview from the given list.

GetSelection()
Returns a list of the currently selected colours

Return type List[str]

add(*event*)
Event handler for adding the colour currently selected in the 'picker' to the 'selection'

get_selection()
Returns a list of the currently selected colours

Return type List[str]

pick(**args*)
Open a wx.ColourDialog to edit the colour currently selected in the picker.

remove(*event*)
Event handler for removing the colour currently selected in the 'selection'

update_preview(*list_obj*, *axes*)
Update the preview from the given list.

Parameters

- **list_obj** (ListBox) – The list to update the preview for.
- **axes** (Axes) – The preview axes to update.

2.2 StylePickerPanel

Classes:

StylePickerPanel(parent[, id, pos, size, ...])	Based on StylePickerPanel, a Panel for selecting a list of colours, and their order.

class StylePickerPanel(*parent*, *id=- 1*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=524288*, *name=b'panel'*, *label='Choose Styles: '*, *selection_choices=None*)
Bases: Panel

Based on StylePickerPanel, a Panel for selecting a list of colours, and their order.

Parameters

- **parent** (Window) – The parent window.
- **id** – An identifier for the panel. wx.ID_ANY is taken to mean a default. Default -1.

- **pos** – The panel position. The value `wx.DefaultPosition` indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform. Default `(-1, -1)`.
- **size** – The panel size. The value `wx.DefaultSize` indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform. Default `(-1, -1)`.
- **style** – The window style. See wxPanel. Default `524288`.
- **name** – The window name. Default `b'panel'`.
- **label** – Label for the panel. Default `'Choose Styles:  '`.
- **selection_choices** (`Optional`[`List`[`str`]]) – A list of hex value choices to populate the 'selection' side of the panel with. Default `None`.

Methods:

add(event)
do_layout()
get_selection()
move([direction])
move_down(event)
move_up(event)
remove(event)
set_properties()
update_picker_preview(*_)
update_preview(list_obj, axes)
update_selection_preview(*_)

add(*event*)

do_layout()

get_selection()

move(*direction=1*)

move_down(*event*)

move_up(*event*)

remove(*event*)

set_properties()

update_picker_preview(*_)

update_preview(*list_obj*, *axes*)

update_selection_preview(*_)

2.3 WebView

Set the emulation level for wxWidgets WebView purely in Python.

Notes:

- The highest emulation level may be used even when the corresponding browser version is not installed.
- Using the `*_FORCE` options is not recommended.
- The *wxWEBVIEWIE_EMU_DEFAULT* can be used to reset the emulation level to the system default.

The values of the constants were taken from https://msdn.microsoft.com/library/ee330730.aspx#browser_emulation and must not be changed.

Functions:

MSWSetEmulationLevel([level, program_name])	Sets the emulation level for wxWidgets WebView.

Data:

wxWEBVIEWIE_EMU_DEFAULT	The system default browser emulation level.
wxWEBVIEWIE_EMU_IE10	Emulate Internet Explorer 10
wxWEBVIEWIE_EMU_IE10_FORCE	Emulate Internet Explorer 10 (force)
wxWEBVIEWIE_EMU_IE11	Emulate Internet Explorer 11
wxWEBVIEWIE_EMU_IE11_FORCE	Emulate Internet Explorer 12 (force)
wxWEBVIEWIE_EMU_IE7	Emulate Internet Explorer 7
wxWEBVIEWIE_EMU_IE8	Emulate Internet Explorer 8
wxWEBVIEWIE_EMU_IE8_FORCE	Emulate Internet Explorer 8 (force)
wxWEBVIEWIE_EMU_IE9	Emulate Internet Explorer 9
wxWEBVIEWIE_EMU_IE9_FORCE	Emulate Internet Explorer 9 (force)

MSWSetEmulationLevel(*level=0*, *program_name=None*)

Sets the emulation level for wxWidgets WebView.

Parameters

- **level** – The emulation level to use. Default `0`.
- **program_name** – The name of the program to set the emulation level for. Defaults to the Python executable.

Returns Whether the operation completed successfully.

wxWEBVIEWIE_EMU_DEFAULT = 0

Type: `int`

The system default browser emulation level.

wxWEBVIEWIE_EMU_IE10 = 10000

Type: `int`

Emulate Internet Explorer 10

wxWEBVIEWIE_EMU_IE10_FORCE = 10001
Type: int

Emulate Internet Explorer 10 (force)

wxWEBVIEWIE_EMU_IE11 = 11000
Type: int

Emulate Internet Explorer 11

wxWEBVIEWIE_EMU_IE11_FORCE = 11001
Type: int

Emulate Internet Explorer 12 (force)

wxWEBVIEWIE_EMU_IE7 = 7000
Type: int

Emulate Internet Explorer 7

wxWEBVIEWIE_EMU_IE8 = 8000
Type: int

Emulate Internet Explorer 8

wxWEBVIEWIE_EMU_IE8_FORCE = 8888
Type: int

Emulate Internet Explorer 8 (force)

wxWEBVIEWIE_EMU_IE9 = 9000
Type: int

Emulate Internet Explorer 9

wxWEBVIEWIE_EMU_IE9_FORCE = 9999
Type: int

Emulate Internet Explorer 9 (force)

2.4 border_config

Dialog for configuring borders for charts.

Classes:

border_config(parent, chromatogram_figure, ...)	**type parent** Window

class border_config (*parent*, *chromatogram_figure*, **args*, ***kwds*)
Bases: Dialog

Parameters

- **parent** (Window)
- **chromatogram_figure**
- ***args**
- ****kwds**

Methods:

apply_tight_layout(_)	
close_dialog(_)	
update_borders(_)	

apply_tight_layout(_)

close_dialog(_)

update_borders(_)

2.5 chartpanel

A canvas for displaying a chart within a wxPython window

Classes:

ChartPanelBase(parent, fig, ax[, id, pos, ...])	Panel that contains a matplotlib plotting window, used for displaying an image.

class ChartPanelBase(*parent*, *fig*, *ax*, *id=- 1*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=0*, *name=b'panel'*)

Bases: Panel

Panel that contains a matplotlib plotting window, used for displaying an image. The image can be right clicked to bring up a context menu allowing copying, pasting and saving of the image. The image can be panned by holding the left mouse button and moving the mouse, and zoomed in and out using the scrollwheel on the mouse.

Parameters

- **parent** (Window) – The parent window.
- **fig** (Figure)
- **ax** (Axes)
- **id** (int) – An identifier for the panel. wx.ID_ANY is taken to mean a default. Default `-1`.
- **pos** (Point) – The panel position. The value `wx.DefaultPosition` indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform. Default `(-1, -1)`.
- **size** (Size) – The panel size. The value ::wxDefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform. Default `(-1, -1)`.
- **style** (int) – The window style. See wxPanel. Default `0`.

- **name** (str) – Window name. Default b'panel'.

Methods:

configure_borders([event])	Open the Configure Borders dialog.
constrain_zoom([key])	Constrain zoom to the x axis only.
on_size_change(_)	Event handler for size change events
pan([enable])	Enable the Pan tool.
previous_view(*_)	Go to the previous view of the chart.
reset_view(*_)	Reset the view of the chart.
setup_scrollwheel_zooming([scale])	Allow zooming of the chart with the scrollwheel.
setup_ylim_refresher(y_data, x_data)	Setup the function for updating the ylim whenever the xlim changes.
size_change()	Internal function that runs whenever the window is resized.
zoom([enable])	Enable the Zoom tool.

configure_borders(*event=None*)
Open the Configure Borders dialog.

constrain_zoom(*key='x'*)
Constrain zoom to the x axis only.

Parameters **key** (str) – Default 'x'.

on_size_change(_)
Event handler for size change events

pan(*enable=True*)
Enable the Pan tool.

previous_view(*_)
Go to the previous view of the chart.

reset_view(*_)
Reset the view of the chart.

setup_scrollwheel_zooming(*scale=1.1*)
Allow zooming of the chart with the scrollwheel.

Parameters **scale** (float) – Default 1.1.

setup_ylim_refresher(*y_data*, *x_data*)
Setup the function for updating the ylim whenever the xlim changes.

Parameters

- **y_data**
- **x_data**

size_change()
Internal function that runs whenever the window is resized.

zoom(*enable=True*)
Enable the Zoom tool.

2.6 clearable_textctrl

A TextCtrl with a button to clear its contents

Classes:

CTCWidget(parent, value, style, validator)	Text control used by *ClearableTextCtrl*.
ClearButton(parent, eventType, bmp)	Clear button for the *ClearableTextCtrl*.
ClearableTextCtrl(parent[, id, value, pos, ...])	TextCtrl with a button to clear its contents.

Data:

clear_btn	XPM button icon for clearing the text control.

class CTCWidget(*parent*, *value*, *style*, *validator*)
Bases: TextCtrl

Text control used by *ClearableTextCtrl*.

Parameters

- **parent** (*ClearableTextCtrl*) – The parent window.
- **value** (str) – The initial value of the text control
- **style** (int) – The style of the text control
- **validator**

Methods:

GetMainWindowOfCompositeControl()	Returns the parent object.
OnText(event)	Event handler for text being entered in the control.
OnTextEnter(_)	Event handler for the enter / return key being pressed

GetMainWindowOfCompositeControl()
Returns the parent object.

Return type *ClearableTextCtrl*

OnText(*event*)
Event handler for text being entered in the control.

Parameters **event** – The wxPython event.

OnTextEnter(_)
Event handler for the enter / return key being pressed

Parameters **event** – The wxPython event.

class ClearButton(*parent*, *eventType*, *bmp*)

Bases: Control

Clear button for the *ClearableTextCtrl*.

Parameters

- **parent** (*ClearableTextCtrl*) – The parent window.
- **eventType** (PyEventBinder)
- **bmp** (Bitmap)

Methods:

AcceptsFocusFromKeyboard()	Always returns False.
GetBestSize()	Returns the best size for the control.
GetMainWindowOfCompositeControl()	Returns the parent object.
OnLeftUp(_)	Event Handler for left mouse button being released.
OnPaint(_)	Event Handler for widget being painted.
SetBitmapLabel(label)	Set bitmap for the button.

AcceptsFocusFromKeyboard()

Always returns False.

Return type bool

GetBestSize()

Returns the best size for the control.

Return type Size

GetMainWindowOfCompositeControl()

Returns the parent object.

Return type *ClearableTextCtrl*

OnLeftUp(_)

Event Handler for left mouse button being released.

OnPaint(_)

Event Handler for widget being painted.

SetBitmapLabel(*label*)

Set bitmap for the button.

Parameters **label** (Bitmap) – Bitmap to set for the button.

class ClearableTextCtrl(*parent*, *id=-1*, *value=''*, *pos=(-1*, *-1)*, *size=(-1*, *-1)*, *style=0*, *validator=<wx.Validator object>*, *name='ClearableTextCtrl'*)

Bases: *TextCtrlWrapper*, Panel

TextCtrl with a button to clear its contents.

Parameters

- **parent** (Window) – The parent window.

- **id** (int) – An identifier for the control. wx.ID_ANY is taken to mean a default. Default -1.
- **value** (str) – Default text value. Default ''.
- **pos** (Point) – The control position. The value wx.DefaultPosition indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform. Default (-1, -1).
- **size** (Size) – The control size. The value wx.DefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform. Default (-1, -1).
- **style** (int) – The window style. See wx.TextCtrl. Default 0.
- **validator** (Validator) – Window validator. Default <wx.Validator object at 0x7f21e9c88f10>.
- **name** (str) – Window name. Default 'ClearableTextCtrl'.

Methods:

AutoComplete(completer)	Enable auto-completion using the provided completer object.
AutoCompleteDirectories()	Call this function to enable auto-completion of the text using the file system directories.
AutoCompleteFileNames()	Call this function to enable auto-completion of the text typed in a single-line text control using all valid file system paths.
ChangeValue(value)	Sets the new text control value.
DiscardEdits()	Resets the internal modified flag as if the current changes had been saved.
EmulateKeyPress(event)	Inserts into the control the character which would have been inserted if the given key event had occurred in the text control.
GetBestClientSize()	Returns the best size for the control.
GetCompositeWindowParts()	Returns a list of wx.Window objects that make up this control.
GetDefaultStyle()	Returns the style currently used for new text.
GetInsertionPoint()	Returns the insertion point, or cursor, position.
GetLineLength(lineNo)	Gets the length of the specified line, not including any trailing newline character(s).
GetLineText(lineNo)	Returns the contents of a given line in the text control, not including any trailing newline character(s).
GetNumberOfLines()	Returns the number of lines in the text control buffer.
GetRange(from_, to_)	Returns the string containing the text starting in the positions from and up to to in the control.
GetStyle(position, style)	Returns the style at this position in the text control.
HitTest(pt)	Finds the row and column of the character at the specified point.
HitTestPos(pt)	Finds the position of the character at the specified point.
IsClearButtonVisible()	Returns the clear button's visibility state.
IsModified()	Returns whether the text has been modified by user.
IsMultiLine()	Returns whether this is a multi line control.
IsSingleLine()	Returns whether this is a single line control.
LayoutControls()	Lays out the child controls.

continues on next page

Table 15 – continued from previous page

MarkDirty()	Mark the control as modified (dirty).
OnClearButton(event)	Event handler for clear button being pressed
OnSize(event)	Event handler for the size of the control being changed.
PositionToXY(pos)	Converts given position to a zero-based column, line number pair.
SetBackgroundColour(colour)	Sets the background colour of the control
SetClearBitmap(bitmap)	Sets the bitmap for the clear button.
SetDefaultStyle(style)	Changes the default style to use for the new text which is going to be added to the control.
SetEditable(editable)	Makes the text item editable or read-only, overriding the `wx.TE_READONLY` flag.
SetFont(font)	Sets the font for the control.
SetInsertionPoint(pos)	Sets the insertion point at the given position.
SetInsertionPointEnd()	Sets the insertion point at the end of the text control.
SetMaxLength(length)	This function sets the maximum number of characters the user can enter into the control.
SetModified(modified)	Marks the control as being modified by the user or not.
SetStyle(start, end, style)	Changes the style of the given range.
ShouldInheritColours()	**rtype** `bool`
ShowPosition(pos)	Makes the line containing the given position visible.
XYToPosition(x, y)	Converts the given zero-based column and line number to a position.

Attributes:

default_clear_bitmap	Returns the default clear button bitmap for the control.

AutoComplete(*completer*)

Enable auto-completion using the provided completer object.

The specified completer object will be used to retrieve the list of possible completions for the already entered text and will be deleted by `wx.TextEntry` itself when it's not needed any longer.

Parameters **completer** (`TextCompleter`) – The object to be used for generating completions if not `None`. If it is `None`, auto-completion is disabled. The `wx.TextEntry` object takes ownership of this pointer and will delete it in any case (i.e. even if this method return `False`).

Return type `bool`

Returns `True` if the auto-completion was enabled or `False` if the operation failed, typically because auto-completion is not supported by the current platform.

AutoCompleteDirectories()

Call this function to enable auto-completion of the text using the file system directories.

Unlike *AutoCompleteFileNames()*, which completes both file names and directories, this function only completes the directory names.

Note: This function is only implemented in wxMSW port and does nothing under the other platforms.

Return type bool

Returns True if the auto-completion was enabled or False if the operation failed, typically because auto-completion is not supported by the current platform.

AutoCompleteFileNames()
Call this function to enable auto-completion of the text typed in a single-line text control using all valid file system paths.

Note: This function is only implemented in wxMSW port and does nothing under the other platforms.

Return type bool

Returns True if the auto-completion was enabled or False if the operation failed, typically because auto-completion is not supported by the current platform.

ChangeValue(*value*)
Sets the new text control value.

It also marks the control as not-modified which means that *IsModified()* would return False immediately after the call to *ChangeValue()*.

The insertion point is set to the start of the control (i.e. position 0) by this function.

This functions does not generate the wx.wxEVT_TEXT event but otherwise is identical to SetValue().

Parameters **value** (str) – The new value to set. It may contain newline characters if the text control is multiline.

DiscardEdits()
Resets the internal modified flag as if the current changes had been saved.

EmulateKeyPress(*event*)
Inserts into the control the character which would have been inserted if the given key event had occurred in the text control.

The event object should be the same as the one passed to wx.EVT_KEY_DOWN handler previously by wxWidgets.

Note: This function doesn't currently work correctly for all keys under any platform but MSW.

Parameters **event** (KeyEvent)

Return type bool

Returns True if the event resulted in a change to the control, False otherwise.

GetBestClientSize()
Returns the best size for the control.

Return type Size

GetCompositeWindowParts()
Returns a list of wx.Window objects that make up this control.

Return type List[Window]

GetDefaultStyle()
Returns the style currently used for new text.

Return type TextAttr

GetInsertionPoint()
Returns the insertion point, or cursor, position.

This is defined as the zero based index of the character position to the right of the insertion point. For example, if the insertion point is at the end of the single-line text control, it is equal to GetLastPosition().

Return type int

GetLineLength(*lineNo*)
Gets the length of the specified line, not including any trailing newline character(s).

Parameters **lineNo** (int) – Line number (starting from zero).

Return type int

Returns The length of the line, or -1 if lineNo was invalid.

GetLineText(*lineNo*)
Returns the contents of a given line in the text control, not including any trailing newline character(s).

Parameters **lineNo** (int) – Line number (starting from zero).

Return type str

GetNumberOfLines()
Returns the number of lines in the text control buffer.

Return type int

GetRange(*from_*, *to_*)
Returns the string containing the text starting in the positions from and up to to in the control.

The positions must have been returned by another wx.TextCtrl method.

Note: The positions in a multiline wx.TextCtrl do not correspond to the indices in the string returned by GetValue because of the different new line representations (CR or CR LF) and so this method should be used to obtain the correct results instead of extracting parts of the entire value. It may also be more efficient, especially if the control contains a lot of data.

Parameters

- **from_** (int)
- **to_** (int)

Return type str

GetStyle(*position*, *style*)
Returns the style at this position in the text control.

Not all platforms support this function.

Parameters

- **position** (int)
- **style** (TextAttr)

Return type bool

Returns True on success, False if an error occurred (this may also mean that the styles are not supported under this platform).

HitTest(*pt*)
Finds the row and column of the character at the specified point.

If the return code is not wx.TE_HT_UNKNOWN the row and column of the character closest to this position are returned, otherwise the output parameters are not modified.

Note: This function is currently only implemented in Univ, wxMSW and wxGTK ports and always returns wx.TE_HT_UNKNOWN in the other ports.

NB: pt is in device coords (not adjusted for the client area origin nor scrolling).

Parameters **pt**

Return type TextCtrlHitTestResult

HitTestPos(*pt*)
Finds the position of the character at the specified point.

If the return code is not wx.TE_HT_UNKNOWN the position of the character closest to this position is returned, otherwise the output parameter is not modified.

Note: This function is currently only implemented in Univ, wxMSW and wxGTK ports and always returns wx.TE_HT_UNKNOWN in the other ports.

Parameters **pt**

Return type wxTextCtrlHitTestResult

IsClearButtonVisible()
Returns the clear button's visibility state.

Return type bool

IsModified()
Returns whether the text has been modified by user.

Note: Calling `SetValue()` doesn't make the control modified.

Return type bool

IsMultiLine()
Returns whether this is a multi line control.

Return type bool

IsSingleLine()
Returns whether this is a single line control.

Return type bool

LayoutControls()
Lays out the child controls.

MarkDirty()
Mark the control as modified (dirty).

OnClearButton(*event*)
Event handler for clear button being pressed

Parameters **event** (CommandEvent)

OnSize(*event*)
Event handler for the size of the control being changed.

Parameters **event** (SizeEvent)

PositionToXY(*pos*)
Converts given position to a zero-based column, line number pair.

Parameters **pos** (int) – Position

Return type Tuple[int, int]

SetBackgroundColour(*colour*)
Sets the background colour of the control

Parameters **colour** (Colour) – The colour to be used as the background colour; pass `wx.NullColour`. to reset to the default colour.

Note: You may want to use `wx.SystemSettings.GetColour` to retrieve a suitable colour to use rather than setting an hard-coded one.

Return type bool

Returns True if the operation completes successfully, False otherwise.

SetClearBitmap(*bitmap*)
Sets the bitmap for the clear button.

Parameters **bitmap** (Bitmap)

SetDefaultStyle(*style*)
Changes the default style to use for the new text which is going to be added to the control.

This applies both to the text added programmatically using WriteText() or AppendText() and to the text entered by the user interactively.

If either of the font, foreground, or background colour is not set in style, the values of the previous default style are used for them. If the previous default style didn't set them either then the global font or colours of the text control itself are used as fall back.

However, if the style parameter is the default wx.TextAttr, then the default style is just reset (instead of being combined with the new style which wouldn't change it at all).

Parameters **style** (TextAttr) – The style for the new text

Return type bool

Returns True on success, False if an error occurred (this may also mean that the styles are not supported under this platform).

SetEditable(*editable*)
Makes the text item editable or read-only, overriding the wx.TE_READONLY flag.

Parameters **editable** (bool) – If True, the control should be editable. If False, the control should be read-only.

SetFont(*font*)
Sets the font for the control.

Parameters **font** (Font) – Font to associate with this control. Pass wx.NullFont to reset to the default font.

Return type bool

Returns True if the operation completes successfully, False otherwise.

SetInsertionPoint(*pos*)
Sets the insertion point at the given position.

Parameters **pos** (int) – Position to set, in the range from 0 to GetLastPosition() inclusive.

SetInsertionPointEnd()
Sets the insertion point at the end of the text control.

This is equivalent to calling *SetInsertionPoint()* with GetLastPosition() as the argument.

SetMaxLength(*length*)
This function sets the maximum number of characters the user can enter into the control.

In other words, it allows limiting the text value length to len not counting the terminating NUL character.

If len is 0, the previously set max length limit, if any, is discarded, and the user may enter as much text as the underlying native text control widget supports (typically at least 32Kb).

If the user tries to enter more characters into the text control when it is already filled up to the maximal length, a `wx.wxEVT_TEXT_MAXLEN` event is sent to notify the program about it (giving it the possibility to show an explanatory message, for example) and the extra input is discarded.

Note that in wxGTK this function may only be used with single line text controls.

Parameters **length** (`int`)

SetModified(*modified*)
Marks the control as being modified by the user or not.

Parameters **modified** (`bool`)

SetStyle(*start*, *end*, *style*)
Changes the style of the given range.

If any attribute within style is not set, the corresponding attribute from GetDefaultStyle is used.

Parameters

- **start** (`int`) – The start of the range to change.
- **end** (`int`) – The end of the range to change.
- **style** (`TextAttr`) – The new style for the range.

Return type `bool`

Returns `True` on success, `False` if an error occurred (this may also mean the styles are not supported under this platform).

ShouldInheritColours()

Return type `bool`

ShowPosition(*pos*)
Makes the line containing the given position visible.

Parameters **pos** (`int`) – The position that should be visible.

XYToPosition(*x*, *y*)
Converts the given zero-based column and line number to a position.

Parameters

- **x** (`int`) – The column number
- **y** (`int`) – The line number

Return type `int`

property default_clear_bitmap
Returns the default clear button bitmap for the control.

Return type `Bitmap`

textctrl
Type: TextCtrl

clear_btn = [b'25 19 2 1', b' c None', b'+ c #000000', b' ', b' ', b' ', b' ', b' ', b' ',
Type: List[bytes]

XPM button icon for clearing the text control.

2.7 dialogs

Several dialog classes and helper functions for file/folder dialogs

Classes:

FloatEntryDialog(*args, **kwargs)	Alternative to wx.NumberEntryDialog that provides a TextCtrl which only allows numbers and decimal points to be entered.
IntEntryDialog(*args, **kwargs)	Alternative to wx.NumberEntryDialog that provides a TextCtrl which only allows numbers to be entered.
Wildcards()	Class to generate glob wildcards for wx.FileDialog

Functions:

file_dialog(*args, **kwargs)	Create a wx.FileDialog with for the extension and file-typestring given, and return the filename selected.
file_dialog_multiple(parent, extension, ...)	Create a wx.FileDialog with the extension and file-typestring given, and return a list of the files selected.
file_dialog_wildcard(parent, title, wildcard)	Create a wx.FileDialog with the wildcard string given, and return a list of the files selected.

class FloatEntryDialog(**args*, ***kwargs*)
Bases: TextEntryDialog

Alternative to wx.NumberEntryDialog that provides a TextCtrl which only allows numbers and decimal points to be entered.

Based on http://wxpython-users.1045709.n5.nabble.com/Adding-Validation-to-wx-TextEntryDialog-td2371082.html

Methods:

GetValue()	

GetValue()

class IntEntryDialog(**args*, ***kwargs*)
Bases: TextEntryDialog

Alternative to wx.NumberEntryDialog that provides a TextCtrl which only allows numbers to be entered.

Based on http://wxpython-users.1045709.n5.nabble.com/Adding-Validation-to-wx-TextEntryDialog-td2371082.html

Methods:

GetValue()	

GetValue()

class Wildcards

Bases: object

Class to generate glob wildcards for wx.FileDialog

Methods:

__repr__()	Return repr(self).
__str__()	Return str(self).
add_all_files_wildcard([hint_format])	Add a wildcard for 'All Files'.
add_common_filetype(filetype[, hint_format, ...])	Add a common filetype.
add_filetype(description[, extensions, ...])	Add a filetype to the wildcards
add_image_wildcard([value_format])	Add a wildcard for all image filetypes.

Attributes:

wildcard	Returns a string representing the wildcards for use in wx.FileDialog or file_dialog_wildcards

__repr__()
Return repr(self).

__str__()
Return str(self).

add_all_files_wildcard(*hint_format=0*)
Add a wildcard for 'All Files'.

Parameters **hint_format** (int) – How the hints should be formatted. Valid values are None and *style_hidden*. Default 0.

add_common_filetype(*filetype*, *hint_format=8*, *value_format=12*)
Add a common filetype.

Parameters

- **filetype** (str) – The name of the filetype, Possible values are in common_filetypes
- **hint_format** (int) – How the hints should be formatted. Default 8.
- **value_format** (int) – How the values should be formatted. Default 12.

Valid values for *hint_format* and *value_format* are *style_uppercase*, *style_lowercase* and *style_hidden*, which can be combined using the | operator.

add_filetype(*description*, *extensions=None*, *hint_format=8*, *value_format=12*)
Add a filetype to the wildcards

Parameters

- **description** (str) – Description of the filetype
- **extensions** (Optional[Sequence[str]]) – A list of valid file extensions for the filetype. Default None.
- **hint_format** (int) – How the hints should be formatted. Default 8.
- **value_format** (int) – How the values should be formatted. Default 12.

Valid values for *hint_format* and *value_format* are *style_uppercase*, *style_lowercase* and *style_hidden*, which can be combined using the | operator.

add_image_wildcard(*value_format=12*)
Add a wildcard for all image filetypes.

Parameters **value_format** (int) – How the values should be formatted. Default 12.

Valid values for *value_format* are *style_uppercase*, *style_lowercase* and *style_hidden*, which can be combined using the | operator.

property wildcard
Returns a string representing the wildcards for use in wx.FileDialog or file_dialog_wildcards

Return type str

file_dialog(**args*, ***kwargs*)
Create a wx.FileDialog with for the extension and filetypestring given, and return the filename selected.

Parameters

- **parent**
- **extension**
- **title**
- **filetypestring**
- **style**
- ****kwargs**

Return type Optional[str]

Returns The filename selected in the dialog. If wx.FD_MULTIPLE is in the style, this will be the first filename selected.

file_dialog_multiple(*parent*, *extension*, *title*, *filetypestring*, *style=6*, ***kwargs*)
Create a wx.FileDialog with the extension and filetypestring given, and return a list of the files selected.

Parameters

- **parent** (Window) – Parent window. Should not be None.
- **extension** (str)
- **title** (str)
- **filetypestring** (str)
- **style** – Default 6.

- ****kwargs**

Return type Optional[List[str]]

Returns List of filenames for the selected files. If wx.FD_MULTIPLE is not in the style, the list will contain only one element.

file_dialog_wildcard(*parent*, *title*, *wildcard*, *style=6*, ***kwargs*)

Create a wx.FileDialog with the wildcard string given, and return a list of the files selected.

Parameters

- **parent** (Window) – Parent window. Should not be None.
- **wildcard** (str)
- **title** (str)
- **style** (int) – Default 6.
- **kwargs**

Returns List of filenames for the selected files. If wx.FD_MULTIPLE is not in the style, the list will contain only one element

Return type list of str

2.8 editable_listbox

A Python implementation of wx.EditableListBox, a ListBox with editable items.

Available in two flavours:

- Vanilla, that accepts any string value; and
- **Numerical, that only accepts numerical values.** Those could be str, int, float or decimal.Decimal, but decimal.Decimal is used internally and is the type that will be returned.

Classes:

CleverListCtrl(parent[, id, pos, size, ...])	list control with auto-resizable column:
EditableListBox(parent[, id, label, pos, ...])	This class provides a composite control that lets the user easily enter and edit a list of strings.
EditableNumericalListBox(parent[, id, ...])	**type parent** Window

class CleverListCtrl(*parent*, *id=-1*, *pos=(-1, -1)*, *size=(-1, -1)*, *style=4*, *validator=<wx.Validator object>*, *name=b'listCtrl'*)

Bases: ListCtrl

list control with auto-resizable column:

Parameters

- **parent** (Window) – Parent window. Should not be None.
- **id** (int) – Default -1.

- **pos** (Point) – Default (-1, -1).
- **size** (Size) – Default (-1, -1).
- **style** (int) – Default 4.
- **validator** (Validator) – Default <wx.Validator object at 0x7f21e9c88f10>.
- **name** (str) – Default b'listCtrl'.

Methods:

CreateColumns()
OnSize(event) **param event** The wxPython event.
SizeColumns()

CreateColumns()

OnSize(*event*)

Parameters **event** – The wxPython event.

SizeColumns()

class EditableListBox(*parent*, *id=- 1*, *label=''*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=1792*, *name=b'editableListBox'*)

Bases: Panel

This class provides a composite control that lets the user easily enter and edit a list of strings.

Styles supported:

- wx.adv.EL_ALLOW_NEW - Allow user to create new items.
- wx.adv.EL_ALLOW_EDIT - Allow user to edit text in the control.
- wx.adv.EL_ALLOW_DELETE - Allow user to delete text from the control.

Parameters

- **parent** (Window) – Parent window. Should not be None.
- **id** (int) – Default -1.
- **label** (str) – Default ''.
- **pos** (Point) – Default (-1, -1).
- **size** (Size) – Default (-1, -1).
- **style** (int) – Default 1792.
- **name** (str) – Default b'editableListBox'.

Methods:

GetDelButton()	Retrieves a reference to the BitmapButton that is used as the 'delete' button in the control.
GetDownButton()	Retrieves a reference to the BitmapButton that is used as the 'down' button in the control.
GetEditButton()	Retrieves a reference to the BitmapButton that is used as the 'edit' button in the control.
GetListCtrl()	Returns a reference to the actual list control portion of the custom control.
GetNewButton()	Retrieves a reference to the BitmapButton that is used as the 'new' button in the control.
GetStrings()	Returns a list of the current contents of the control.
GetUpButton()	Retrieves a reference to the BitmapButton that is used as the 'up' button in the control.
OnBeginLabelEdit(event)	**param event** The wxPython event.
OnDelItem(_)	
OnDownItem(_)	
OnEditItem(_)	
OnEndLabelEdit(event)	**param event** The wxPython event.
OnItemActivated(evt)	
OnItemSelected(event)	
OnNewItem(event)	**param event** The wxPython event.
OnUpItem(_)	
SetStrings(strings)	Replaces current contents with given strings.
SetupEditControl()	
SwapItems(i1, i2)	**type i1** `int`
on_value_changed(event)	

GetDelButton()
Retrieves a reference to the BitmapButton that is used as the 'delete' button in the control.

GetDownButton()
Retrieves a reference to the BitmapButton that is used as the 'down' button in the control.

GetEditButton()
Retrieves a reference to the BitmapButton that is used as the 'edit' button in the control.

GetListCtrl()
Returns a reference to the actual list control portion of the custom control.

Returns

Return type wx.ListCtrl

GetNewButton()
Retrieves a reference to the BitmapButton that is used as the 'new' button in the control.

GetStrings()
Returns a list of the current contents of the control.

Returns list of strings

Return type list of str

GetUpButton()
Retrieves a reference to the BitmapButton that is used as the 'up' button in the control.

OnBeginLabelEdit(*event*)

Parameters **event** – The wxPython event.

OnDelItem(*_*)

OnDownItem(*_*)

OnEditItem(*_*)

OnEndLabelEdit(*event*)

Parameters **event** – The wxPython event.

OnItemActivated(*evt*)

OnItemSelected(*event*)

OnNewItem(*event*)

Parameters **event** – The wxPython event.

OnUpItem(*_*)

SetStrings(*strings*)
Replaces current contents with given strings.

Parameters **strings** (List[str]) – list of strings.

SetupEditControl()

SwapItems(*i1*, *i2*)

Parameters

- **i1** (int)
- **i2** (int)

on_value_changed(*event*)

class EditableNumericalListBox(*parent*, *id=- 1*, *label=''*, *decimal_places=- 1*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=1792*, *name=b'editableListBox'*)

Bases: *EditableListBox*

Parameters

- **parent** (Window) – Parent window. Should not be None.
- **id** (int) – Default -1.
- **label** (str) – Default ''.
- **pos** (Point) – Default (-1, -1).
- **size** (Size) – Default (-1, -1).
- **style** (int) – Default 1792.
- **name** (str) – Default b'editableListBox'.

Methods:

GetDecimalPlaces()	
GetStrings()	Returns a list of the current contents of the control.
GetValues()	Returns a list of the current contents of the control.
SetDecimalPlaces(_decimal_places)	
SetStrings(strings)	Replaces current contents with given strings.
SetValues(values)	Replaces current contents with given values.
SetupEditControl()	
on_value_changed(event)	

Attributes:

decimal_places

GetDecimalPlaces()

GetStrings()

Returns a list of the current contents of the control.

Returns list of strings

Return type list of str

GetValues()

Returns a list of the current contents of the control.

Returns

Return type

SetDecimalPlaces(*_decimal_places*)

SetStrings(*strings*)

Replaces current contents with given strings.

Parameters **strings** – list of strings.

SetValues(*values*)
Replaces current contents with given values.

Parameters **values** (Sequence[Union[float, Decimal]]) – list of values

SetupEditControl()

property decimal_places

on_value_changed(*event*)

2.9 events

Reusable code for simple events

Usage:

```
>>> from domdf_wxpython_tools import events
```

```
>>> # Create the event outside of the class
>>> myEVT = events.SimpleEvent()
```

```
>>> # Set the receiver to the class you want.
>>> # This is usually done from within the receiver class
>>> class MyClass(object):
...     def __init__(self):
...         myEVT.set_receiver(self)
...
...         # Then bind the event to a handler
...         myEVT.Bind(self.handler)
...
...     def handler(self):
...         '''Handler for myEVT'''
...         pass
```

```
>>> # From within the thread, trigger the event with the following syntax:
```

```
>>> myEVT.trigger()
```

Classes:

PayloadEvent(etype, eid, value)	Event containing a message payload.
SimpleEvent([receiver, name])	**param receiver**

class PayloadEvent(*etype*, *eid*, *value*)
Bases: PyCommandEvent

Event containing a message payload.

Methods:

GetValue()	Returns the value from the event.

GetValue()
Returns the value from the event. @return: the value of this event

class SimpleEvent(*receiver=None*, *name='Event'*)
Bases: object

Parameters

- **receiver** – Default None.
- **name** – Default 'Event'.

Methods:

Bind(handler[, receiver])	Bind the event to the handler.
Unbind([receiver])	Unbind the event from the handler
__repr__()	Return a nicely formatted representation string.
set_receiver(receiver)	Set the class that is to receive the event trigger.
trigger([value])	Trigger the event

Bind(*handler*, *receiver=None*, ***kwargs*)
Bind the event to the handler.

Parameters

- **handler** – handler to bind the event to.
- **kwargs** – keyword arguments to pass through to receiver's Bind method.

Unbind(*receiver=None*, ***kwargs*)
Unbind the event from the handler

Parameters **kwargs** – keyword arguments to pass through to receiver's Unbind method

__repr__()
Return a nicely formatted representation string.

set_receiver(*receiver*)
Set the class that is to receive the event trigger.

Parameters **receiver**

trigger(*value=None*)
Trigger the event

2.10 filebrowsectrl

Classes:

DirBrowseCtrl(parent[, id, pos, size, ...])	
FileBrowseCtrl(parent[, id, pos, size, ...])	A control to allow the user to type in a filename or browse with the standard file dialog to select file.
FileBrowseCtrlWithHistory(*arguments, ...)	with following additions:

class DirBrowseCtrl(*parent*, *id=- 1*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=537401408*, *labelText='Select a directory:'*, *buttonText='Browse'*, *toolTip='Type directory name or browse to select'*, *dialogTitle=''*, *initialValue=None*, *changeCallback=None*, *name='DirBrowseCtrl'*)

Bases: *FileBrowseCtrl*

Methods:

OnBrowse([ev])	Going to browse for file...

Attributes:

—

OnBrowse(*ev=None*)

Going to browse for file...

textctrl

Type: TextCtrl

class FileBrowseCtrl(*parent*, *id=-1*, *pos=(-1, -1)*, *size=(-1, -1)*, *style=524289*, *labelText='File Entry:'*, *buttonText='Browse'*, *toolTip='Type a filename or click the browse button to choose a file'*, *dialogTitle='Choose a file'*, *initialValue=''*, *changeCallback=<function '<lambda>'>*, *labelWidth=0*, *name='fileBrowseButton'*, *show_cancel_btn=True*, *fileMask='All files (*.*)|*.*'*, *dialog_title='File Picker'*, ***kwargs*)

Bases: *TextCtrlWrapper*, FileBrowseButton

A control to allow the user to type in a filename or browse with the standard file dialog to select file.

Based on and subclassed from wx.lib.filebrowsebutton.FileBrowseButton but with a wx.SearchCtrl in place of the wx.TextCtrl to provide the cancel/clear button and with an icon on the browse button.

Parameters

- **parent** (Window) – Parent window. Should not be None.
- **id** (int) – Control identifier. A value of -1 denotes a default value. Default -1.
- **pos** (Point) – Control position. Default (-1, -1).
- **size** (Size) – Control size. Default (-1, -1).
- **style** (int) – Window style. See wx.Window and ClearableTextCtrl for supported styles. Default 524289.
- **labelText** (str) – Text for label to left of text field. Default 'File Entry:'.

- **buttonText** (str) – Text for button which launches the file dialog. Default 'Browse'.
- **toolTip** (str) – Help text. Default 'Type a filename or click the browse button to choose a file'.
- **dialogTitle** (str) – Title used in file dialog. Default 'Choose a file'.
- **initialValue** (str) – The initial value of the TextCtrl. Default ''.
- **changeCallback** – Optional callback called for all changes in value of the control. Default domdf_wxpython_tools.filebrowsectrl..
- **labelWidth** – Width of the label. Default 0.
- **name** – Default 'fileBrowseButton'.
- **show_cancel_btn** (bool) – Whether to show or hide the cancel button. Default True.
- **dialog_title** (str) – The title of the FileDialog. Default 'File Picker'.
- **fileMask** (str) – File mask (glob pattern, such as *.*) to use in file dialog. See wx.FileDialog for more information. Default 'All files (*.*)|*.*'.

Methods:

ChangeValue(value)	Sets the new text control value.
GetLabel()	Retrieve the label's current text.
GetLineLength(lineNo)	Gets the length of the specified line, not including any trailing newline character(s).
GetLineText(lineNo)	Returns the contents of a given line in the text control, not including any trailing newline character(s).
GetNumberOfLines()	Returns the number of lines in the text control buffer.
GetRange(from_, to_)	Returns the string containing the text starting in the positions from and up to in the control.
IsEditable()	Returns True if the controls contents may be edited by user (note that it always can be changed by the program).
IsModified()	Returns True if the text has been modified by user.
IsMultiLine()	Returns True if this is a multi line edit control and False otherwise.
IsSingleLine()	Returns True if this is a single line edit control and False otherwise.
MarkDirty()	Mark text as modified (dirty).
OnBrowse([event])	Going to browse for file...
SetLabel(value)	Set the label's current text.
SetModified(modified)	Marks the control as being modified by the user or not.
SetValue(value[, callBack])	Sets the new text control value.
createBrowseButton()	Create the browse-button control.
createTextControl()	Create the text control.

Attributes:

—

ChangeValue(*value*)
Sets the new text control value.

It also marks the control as not-modified which means that IsModified() would return `False` immediately after the call to ChangeValue .

The insertion point is set to the start of the control (i.e. position 0) by this function.

This functions does not generate the wxEVT_TEXT event but otherwise is identical to SetValue .

Parameters **value** (`str`) – The new value to set. It may contain newline characters if the text control is multi-line.

GetLabel()
Retrieve the label's current text.

GetLineLength(*lineNo*)
Gets the length of the specified line, not including any trailing newline character(s).

Parameters **lineNo** (`int`) – Line number (starting from zero).

Return type `int`

Returns The length of the line, or `-1` if `lineNo` was invalid.

GetLineText(*lineNo*)
Returns the contents of a given line in the text control, not including any trailing newline character(s).

Parameters **lineNo** (`int`) – Line number (starting from zero).

Return type `str`

Returns The contents of the line.

GetNumberOfLines()
Returns the number of lines in the text control buffer.

Returns

Return type int

GetRange(*from_*, *to_*)
Returns the string containing the text starting in the positions from and up to in the control.

The positions must have been returned by another wx.TextCtrl method.

Parameters

- **from_** (`int`)
- **to_** (`int`)

Return type `str`

IsEditable()
Returns `True` if the controls contents may be edited by user (note that it always can be changed by the program).

In other words, this functions returns `True` if the control hasn't been put in read-only mode by a previous call to SetEditable .

Return type bool

IsModified()
Returns `True` if the text has been modified by user.

Note that calling SetValue doesn't make the control modified.

Returns

Return type bool

IsMultiLine()
Returns `True` if this is a multi line edit control and `False` otherwise.

Returns

Return type bool

IsSingleLine()
Returns `True` if this is a single line edit control and `False` otherwise.

Returns

Return type bool

MarkDirty()
Mark text as modified (dirty).

Returns

Return type

OnBrowse(*event=None*)
Going to browse for file...

SetLabel(*value*)
Set the label's current text.

SetModified(*modified*)
Marks the control as being modified by the user or not.

Parameters **modified** (`bool`)

SetValue(*value*, *callBack=1*)
Sets the new text control value.

It also marks the control as not-modified which means that IsModified() would return `False` immediately after the call to SetValue .

The insertion point is set to the start of the control (i.e. position 0) by this function unless the control value doesn't change at all, in which case the insertion point is left at its original position.

Note that, unlike most other functions changing the controls values, this function generates a wxEVT_TEXT event. To avoid this you can use ChangeValue instead.

Parameters: value (string) – The new value to set. It may contain newline characters if the text control is multi-line.

createBrowseButton()
Create the browse-button control.

createTextControl()
Create the text control.

textctrl
Type: TextCtrl

class FileBrowseCtrlWithHistory(**arguments*, ***namedarguments*)
Bases: *FileBrowseCtrl*

with following additions: __init__(..., history=None)

history – optional list of paths for initial history drop-down (must be passed by name, not a positional argument) If history is callable it will must return a list used for the history drop-down

changeCallback – as for FileBrowseCtrl, but with a work-around for win32 systems which don't appear to create wx.EVT_COMBOBOX events properly. There is a (slight) chance that this work-around will cause some systems to create two events for each Combobox selection. If you discover this condition, please report it!

As for a FileBrowseCtrl.__init__ otherwise.

GetHistoryControl() Return reference to the control which implements interfaces required for manipulating the history list. See GetHistoryControl documentation for description of what that interface is.

GetHistory() Return current history list

SetHistory(value=(), selectionIndex = None) Set current history list, if selectionIndex is not None, select that index

Methods:

GetHistory()	Return the current history list
GetHistoryControl()	Return a pointer to the control which provides (at least) the following methods for manipulating the history list:
OnSetFocus(event)	When the history scroll is selected, update the history
SetHistory([value, selectionIndex, control])	Set the current history list
createTextControl()	Create the text control.

Attributes:

—

GetHistory()
Return the current history list

GetHistoryControl()
Return a pointer to the control which provides (at least) the following methods for manipulating the history list:

Append(item) – add item Clear() – clear all items Delete(index) – 0-based index to delete from list SetSelection(index) – 0-based index to select in list

Semantics of the methods follow those for the wxComboBox control

OnSetFocus(*event*)
When the history scroll is selected, update the history

SetHistory(*value=()*, *selectionIndex=None*, *control=None*)
Set the current history list

createTextControl()
Create the text control.

textctrl
Type: TextCtrl

2.11 icons

Functions:

GetStockBitmap(art_id[, art_client])	Get a stock bitmap from its wx.ART_xxx ID
GetStockToolbarBitmap(art_id)	
get_button_icon(icon_name[, size])	
get_toolbar_icon(icon_name[, size])	

GetStockBitmap(*art_id*, *art_client=None*)
Get a stock bitmap from its wx.ART_xxx ID

GetStockToolbarBitmap(*art_id*)

get_button_icon(*icon_name*, *size=24*)

get_toolbar_icon(*icon_name*, *size=24*)

2.12 imagepanel

Based on ChartPanelBase, a canvas for displaying an image within a wxPython window using PIL and matplotlib, with a right click menu with some basic options

Classes:

EvtImgPanelChanged(windowID, obj)	Custom Event for an image in the ImagePanel being changed.
ImagePanel(parent[, image, id, pos, size, ...])	Panel that contains a matplotlib plotting window, used for displaying an image.

class EvtImgPanelChanged(*windowID*, *obj*)
Bases: PyCommandEvent

Custom Event for an image in the ImagePanel being changed.

Parameters

- **windowID** (int)
- **obj**

Attributes:

eventType

eventType = 0
Type: int

class ImagePanel(*parent*, *image=None*, *id=- 1*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=0*, *name=b'panel'*)
Bases: *ChartPanelBase*

Panel that contains a matplotlib plotting window, used for displaying an image. The image can be right clicked to bring up a context menu allowing copying, pasting and saving of the image. The image can be panned by holding the left mouse button and moving the mouse, and zoomed in and out using the scrollwheel on the mouse.

Parameters

- **parent** (Window) – The parent window.
- **image** – Default None.
- **id** (int) – An identifier for the panel. wx.ID_ANY is taken to mean a default. Default -1.
- **pos** (Point) – The panel position. The value wx.DefaultPosition indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform. Default (-1, -1).
- **size** (Size) – The panel size. The value ::wxDefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform. Default (-1, -1).
- **style** (int) – The window style. See wxPanel. Default 0.
- **name** (str) – Window name. Default b'panel'.

Methods:

clear([event])	Clear the image from the control
copy([_])	Copy the image to the clipboard.
load_image([new_image, suppress_event])	Load the 'new_image' into the control.
on_context_menu(event)	Event Handler for bringing up right click context menu
on_load([event])	Load the image into the dialog from the file selected in the dialog
on_save([event])	Save the image to the location selected in the dialog
paste([event])	Paste the image from the clipboard into the control.
reset_view(*_)	Reset the view of the image.

Attributes:

default_image	
image	Returns the image being displayed in the control

clear(*event=None*)
Clear the image from the control

copy(*_=None*)
Copy the image to the clipboard.

default_image = ('RGB', (640, 480), (240, 240, 240))
Type: tuple

property image
Returns the image being displayed in the control

Return type PIL.Image.Image

load_image(*new_image=None*, *suppress_event=False*)
Load the 'new_image' into the control.

Parameters

- **new_image** (Union[Image, None, str, Path, PathLike]) – The image to load, or a string pointing to the image on a filesystem. Default None.
- **suppress_event** (bool) – Whether the event that the image has changed should be suppressed. Default False.

on_context_menu(*event*)
Event Handler for bringing up right click context menu

on_load(*event=None*)
Load the image into the dialog from the file selected in the dialog

on_save(*event=None*)
Save the image to the location selected in the dialog

paste(*event=None*)
Paste the image from the clipboard into the control.

reset_view(*_)
Reset the view of the image.

2.13 keyboard

Functions:

gen_keymap()

gen_keymap()

2.14 list_dialog

Classes:

list_dialog(parent[, id, title, label, ...])	A dialog containing a wx.ListBox.

class list_dialog (*parent*, *id=- 1*, *title='Choose'*, *label='Choose: '*, *choices=None*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=536877056*, *name=b'dialog'*)

Bases: Dialog

A dialog containing a wx.ListBox.

Parameters

- **parent** (Window) – The parent window. Can be None, a frame or another dialog box
- **id** (int) – An identifier for the dialog. wx.ID_ANY is taken to mean a default. Default -1.
- **title** (str) – The title of the dialog. Default 'Choose'.
- **label** (str) – The label for the wx.ListBox. Default 'Choose: '.
- **choices** (Optional[List[str]]) – A list of choices for the wx.ListBox. Default None.
- **pos** (Point) – The dialog position. The value wx.DefaultPosition indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform. Default (-1, -1).
- **size** (Size) – The dialog size. The value ::wxDefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform. Default (-1, -1).
- **style** (int) – The window style. See wxPanel. Default 536877056.
- **name** (str) – Window name. Default b'dialog'.

Methods:

do_cancel(_)	Event Handler for dialog being cancelled
do_select(_)	Event Handler for item in list being selected

do_cancel (_)
Event Handler for dialog being cancelled

do_select (_)
Event Handler for item in list being selected

2.15 logctrl

Log Control, supporting text copying and zoom.

Classes:

LogCtrl(parent[, id, pos, size, style, name])	Log Window based on StyledTextCtrl.

class LogCtrl (*parent*, *id=- 1*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=138412032*, *name='Log'*)

Bases: StyledTextCtrl

Log Window based on StyledTextCtrl.

Parameters

- **parent** (Window) – The parent window.
- **id** (int) – An identifier for the Log window. wx.ID_ANY is taken to mean a default. Default -1.
- **pos** (Point) – The Log window position. The value wx.DefaultPosition indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform. Default (-1, -1).
- **size** (Size) – The Log window size. The value ::wxDefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform. Default (-1, -1).
- **style** (int) – The window style. See wxPanel. Default 138412032.
- **name** (str) – Window name. Default 'Log'.

Methods:

Append(text[, c])	Add the text to the end of the control using colour c which should be suitable for feeding directly to wx.NamedColour.
AppendStderr(text)	Add the stderr text to the end of the control using colour "red"
CanCopy()	Returns True if text is selected and can be copied, False otherwise.
Copy()	Copy the selection and place it on the clipboard.
DoFindNext(findData[, findDlg])	
GetContextMenu()	Create and return a context menu for the log.
GetLastPosition()	
GetRange(start, end)	
GetSelection()	
OnContextMenu(_)	vent Handler for showing the context menu
OnFindClose(_)	
OnFindText(*_)	
OnZoomDefault(*_)	Event Handler for resetting the zoom
OnZoomIn(*_)	Event Handler for zooming in
OnZoomOut(*_)	Event Handler for zooming out
ShowPosition(pos)	
ToggleLineNumbers(*_)	
ToggleWrap(*_)	Toggle word wrap.

continues on next page

Table 47 – continued from previous page

bufferHasChanged()	
bufferSave()	
fixLineEndings(text)	Return text with line endings replaced by OS-specific endings.
getStyle([c])	Returns a style for a given colour if one exists.
onKeyPress(event)	Event Handler for key being pressed.
setDisplayLineNumbers(state)	
setStyles(faces)	Configure font size, typeface and color for lexer.
wrap([wrap])	Set whether text is word wrapped.
write(text)	Display text in the log.

Attributes:

—

Append(*text*, *c=None*)
Add the text to the end of the control using colour c which should be suitable for feeding directly to wx.NamedColour.

Parameters

- **text** – … Should be a unicode string or contain only ascii data.
- **c** – Default `None`.

AppendStderr(*text*)
Add the stderr text to the end of the control using colour "red"

Parameters **text** – … Should be a unicode string or contain only ascii data.

CanCopy()
Returns `True` if text is selected and can be copied, `False` otherwise.

Return type bool

Copy()
Copy the selection and place it on the clipboard.

DoFindNext(*findData*, *findDlg=None*)

GetContextMenu()
Create and return a context menu for the log. This is used instead of the scintilla default menu in order to correctly respect our immutable buffer.

GetLastPosition()

GetRange(*start*, *end*)

GetSelection()

OnContextMenu(_)
vent Handler for showing the context menu

OnFindClose(_)

OnFindText(*_)

OnZoomDefault(*_)
Event Handler for resetting the zoom

OnZoomIn(*_)
Event Handler for zooming in

OnZoomOut(*_)
Event Handler for zooming out

ShowPosition(*pos*)

ToggleLineNumbers(*_)

ToggleWrap(*_)
Toggle word wrap.

bufferHasChanged()

bufferSave()

findDlg
Type: Optional[FindReplaceDialog]

fixLineEndings(*text*)
Return text with line endings replaced by OS-specific endings.

Parameters **text**

getStyle(*c='black'*)
Returns a style for a given colour if one exists.

If no style exists for the colour, make a new style.

If we run out of styles, (only 32 allowed here) we go to the top of the list and reuse previous styles.

Parameters **c** (str) – Default 'black'.

onKeyPress(*event*)
Event Handler for key being pressed.

setDisplayLineNumbers(*state*)

setStyles(*faces*)
Configure font size, typeface and color for lexer.

Parameters **faces**

wrap(*wrap=True*)
Set whether text is word wrapped.

Parameters **wrap** (bool) – Whether the text should be word wrapped. Default True.

write(*text*)
Display text in the log.

Replace line endings with OS-specific endings.

Parameters **text**

2.16 picker

Classes:

dir_picker(parent[, id, value, pos, size, ...])	**type parent** Window
file_folder_picker(parent[, id, value, pos, ...])	**type parent** Window
file_picker(parent[, id, value, pos, size, ...])	**type parent** Window

class dir_picker(*parent*, *id=- 1*, *value=''*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=524288*, *name=b'dir_picker'*)
Bases: TextCtrlWrapper, Panel

Parameters

- **parent** (Window) – Parent window. Should not be None.
- **id** – Default -1.
- **value** – Default ''.
- **pos** – Default (-1, -1).
- **size** – Default (-1, -1).
- **style** – Default 524288.
- **name** – Default b'dir_picker'.

Methods:

Browse(*_)	
Clear(*_)	Clears the text in the control.
GetInsertionPoint()	Returns the insertion point, or cursor, position.
ResetValue()	Resets the text ctrl to the initial value, either specified in __init__ or set with SetInitialValue.
SetHeight(height)	Set the height of the widgets
SetInitialValue(value)	Sets the initial value for the text ctrl.

continues on next page

Table 50 – continued from previous page

SetInsertionPoint(pos)	Sets the insertion point at the given position.
SetInsertionPointEnd()	Sets the insertion point at the end of the text control.
SetTextWidth(width)	Sets the width of the TextCtrl.
get_value()	
reset_value()	
set_height(height)	
set_textctrl_width(width)	

Attributes:

—

Browse(*_)

Clear(*_)
Clears the text in the control.

Note that this function will generate a wxEVT_TEXT event, i.e. its effect is identical to calling SetValue (“”).

GetInsertionPoint()
Returns the insertion point, or cursor, position.

This is defined as the zero based index of the character position to the right of the insertion point. For example, if the insertion point is at the end of the single-line text control, it is equal to GetLastPosition .

Return type long

ResetValue()
Resets the text ctrl to the initial value, either specified in __init__ or set with `SetInitialValue`.

SetHeight(*height*)
Set the height of the widgets

Parameters **height** (`int`) – Height of the widgets

SetInitialValue(*value*)
Sets the initial value for the text ctrl.

Parameters **value** (`str`) – Initial value for the text ctrl.

SetInsertionPoint(*pos*)
Sets the insertion point at the given position.

Parameters **pos** (`int`) – Position to set, in the range from `0` to *GetLastPosition()* inclusive.

SetInsertionPointEnd()
Sets the insertion point at the end of the text control.

This is equivalent to calling wx.TextCtrl.SetInsertionPoint with wx.TextCtrl.GetLastPosition argument.

SetTextWidth(*width*)
Sets the width of the TextCtrl.

Parameters **width** (int) – The width of the TextCtrl.

get_value()

reset_value()

set_height(*height*)

set_textctrl_width(*width*)

textctrl
Type: TextCtrl

class file_folder_picker(*parent*, *id=- 1*, *value=''*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=524294*, *name=b'file_picker'*, *extension='*'*, *title='File Picker'*, *filetypestring='All Files'*, *start_as_files=True*, ***kwargs*)

Bases: *dir_picker*

Parameters

- **parent** (Window) – Parent window. Should not be None.
- **id** – Default -1.
- **value** – Default ''.
- **pos** – Default (-1, -1).
- **size** – Default (-1, -1).
- **style** – Default 524294.
- **name** – Default b'file_picker'.
- **extension** – Default '*'.
- **title** – Default 'File Picker'.
- **filetypestring** – Default 'All Files'.
- **kwargs**

Methods:

Browse(*_)
set_files_mode()
set_folders_mode()
toggle_mode()

Attributes:

—

Browse(*_)

set_files_mode()

set_folders_mode()

textctrl
Type: TextCtrl

toggle_mode()

class file_picker(*parent*, *id=- 1*, *value=''*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=524294*, *name=b'file_picker'*, *extension='*'*, *title='File Picker'*, *filetypestring='All Files'*, ***kwargs*)
Bases: *dir_picker*

Parameters

- **parent** (Window) – Parent window. Should not be None.
- **id** – Default -1.
- **value** – Default ''.
- **pos** – Default (-1, -1).
- **size** – Default (-1, -1).
- **style** – Default 524294.
- **name** – Default b'file_picker'.
- **extension** – Default '*'.
- **title** – Default 'File Picker'.
- **filetypestring** – Default 'All Files'.
- **kwargs**

Methods:

Browse(*_)	

Attributes:

—

Browse(*_)

textctrl
Type: TextCtrl

2.17 projections

Classes:

NoZoom(fig, *args[, facecolor, frameon, ...])	Prevent zooming in pan mode.
XPanAxes(fig, *args[, facecolor, frameon, ...])	Constrain pan to x-axis.
XPanAxes_NoZoom(fig, *args[, facecolor, ...])	Constrain pan to x-axis and prevent zooming.

class NoZoom(*fig*, **args*, *facecolor=None*, *frameon=True*, *sharex=None*, *sharey=None*, *label=''*, *xscale=None*, *yscale=None*, *box_aspect=None*, ***kwargs*)

Bases: Axes

Prevent zooming in pan mode.

Methods:

drag_pan(button, key, x, y)	**param button** The pressed mouse button.
set(*[, adjustable, agg_filter, alpha, ...])	Set multiple properties at once.

Attributes:

name

drag_pan(*button*, *key*, *x*, *y*)

Parameters

- **button** – The pressed mouse button.
- **key** (Optional[str]) – The pressed key, if any.
- **x** (float) – The mouse coordinates in display coords.
- **y** (float) – The mouse coordinates in display coords.

name = 'NoZoom'

Type: str

set(*, *adjustable=<UNSET>*, *agg_filter=<UNSET>*, *alpha=<UNSET>*, *anchor=<UNSET>*, *animated=<UNSET>*, *aspect=<UNSET>*, *autoscale_on=<UNSET>*, *autoscalex_on=<UNSET>*, *autoscaley_on=<UNSET>*, *axes_locator=<UNSET>*, *axisbelow=<UNSET>*, *box_aspect=<UNSET>*, *clip_box=<UNSET>*, *clip_on=<UNSET>*, *clip_path=<UNSET>*, *facecolor=<UNSET>*, *frame_on=<UNSET>*, *gid=<UNSET>*, *in_layout=<UNSET>*, *label=<UNSET>*, *mouseover=<UNSET>*, *navigate=<UNSET>*, *path_effects=<UNSET>*, *picker=<UNSET>*, *position=<UNSET>*, *prop_cycle=<UNSET>*, *rasterization_zorder=<UNSET>*, *rasterized=<UNSET>*, *sketch_params=<UNSET>*, *snap=<UNSET>*, *subplotspec=<UNSET>*, *title=<UNSET>*, *transform=<UNSET>*, *url=<UNSET>*, *visible=<UNSET>*, *xbound=<UNSET>*, *xlabel=<UNSET>*, *xlim=<UNSET>*, *xmargin=<UNSET>*, *xscale=<UNSET>*, *xticklabels=<UNSET>*, *xticks=<UNSET>*, *ybound=<UNSET>*, *ylabel=<UNSET>*, *ylim=<UNSET>*, *ymargin=<UNSET>*, *yscale=<UNSET>*, *yticklabels=<UNSET>*, *yticks=<UNSET>*, *zorder=<UNSET>*)

Set multiple properties at once.

Supported properties are

Properties: adjustable: {'box', 'datalim'} agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array and two offsets from the bottom left corner of the image alpha: scalar or None anchor: (float, float) or {'C', 'SW', 'S', 'SE', 'E', 'NE', ...} animated: bool aspect: {'auto', 'equal'} or float autoscale_on: bool autoscalex_on: unknown autoscaley_on: unknown axes_locator: Callable[[Axes, Renderer], Bbox] axisbelow: bool or 'line' box_aspect: float or None clip_box: *~matplotlib.transforms.BboxBase* or None clip_on: bool clip_path: Patch

or (Path, Transform) or None facecolor or fc: color figure: *~matplotlib.figure.Figure* frame_on: bool gid: str in_layout: bool label: object mouseover: bool navigate: bool navigate_mode: unknown path_effects: list of *.AbstractPathEffect* picker: None or bool or float or callable position: [left, bottom, width, height] or *~matplotlib.transforms.Bbox* prop_cycle: *~cycler.Cycler* rasterization_zorder: float or None rasterized: bool sketch_params: (scale: float, length: float, randomness: float) snap: bool or None subplotspec: unknown title: str transform: *~matplotlib.transforms.Transform* url: str visible: bool xbound: (lower: float, upper: float) xlabel: str xlim: (left: float, right: float) xmargin: float greater than -0.5 xscale: unknown xticklabels: unknown xticks: unknown ybound: (lower: float, upper: float) ylabel: str ylim: (bottom: float, top: float) ymargin: float greater than -0.5 yscale: unknown yticklabels: unknown yticks: unknown zorder: float

class XPanAxes(*fig*, **args*, *facecolor=None*, *frameon=True*, *sharex=None*, *sharey=None*, *label=''*, *xscale=None*, *yscale=None*, *box_aspect=None*, ***kwargs*)

Bases: `Axes`

Constrain pan to x-axis.

Methods:

drag_pan(button, key, x, y)	**param button** The pressed mouse button.
set(*[, adjustable, agg_filter, alpha, ...])	Set multiple properties at once.

Attributes:

name

drag_pan(*button*, *key*, *x*, *y*)

Parameters

- **button** – The pressed mouse button.
- **key** (`Optional[str]`) – The pressed key, if any.
- **x** (`float`) – The mouse coordinates in display coords.
- **y** (`float`) – The mouse coordinates in display coords.

name = 'XPanAxes'

Type: `str`

set (*, *adjustable=<UNSET>*, *agg_filter=<UNSET>*, *alpha=<UNSET>*, *anchor=<UNSET>*, *animated=<UNSET>*, *aspect=<UNSET>*, *autoscale_on=<UNSET>*, *autoscalex_on=<UNSET>*, *autoscaley_on=<UNSET>*, *axes_locator=<UNSET>*, *axisbelow=<UNSET>*, *box_aspect=<UNSET>*, *clip_box=<UNSET>*, *clip_on=<UNSET>*, *clip_path=<UNSET>*, *facecolor=<UNSET>*, *frame_on=<UNSET>*, *gid=<UNSET>*, *in_layout=<UNSET>*, *label=<UNSET>*, *mouseover=<UNSET>*, *navigate=<UNSET>*, *path_effects=<UNSET>*, *picker=<UNSET>*, *position=<UNSET>*, *prop_cycle=<UNSET>*, *rasterization_zorder=<UNSET>*, *rasterized=<UNSET>*, *sketch_params=<UNSET>*, *snap=<UNSET>*, *subplotspec=<UNSET>*, *title=<UNSET>*, *transform=<UNSET>*, *url=<UNSET>*, *visible=<UNSET>*, *xbound=<UNSET>*, *xlabel=<UNSET>*, *xlim=<UNSET>*, *xmargin=<UNSET>*, *xscale=<UNSET>*, *xticklabels=<UNSET>*, *xticks=<UNSET>*, *ybound=<UNSET>*, *ylabel=<UNSET>*, *ylim=<UNSET>*, *ymargin=<UNSET>*, *yscale=<UNSET>*, *yticklabels=<UNSET>*, *yticks=<UNSET>*, *zorder=<UNSET>*)

Set multiple properties at once.

Supported properties are

Properties: adjustable: {'box', 'datalim'} agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array and two offsets from the bottom left corner of the image alpha: scalar or None anchor: (float, float) or {'C', 'SW', 'S', 'SE', 'E', 'NE', ...} animated: bool aspect: {'auto', 'equal'} or float autoscale_on: bool autoscalex_on: unknown autoscaley_on: unknown axes_locator: Callable[[Axes, Renderer], Bbox] axisbelow: bool or 'line' box_aspect: float or None clip_box: *~matplotlib.transforms.BboxBase* or None clip_on: bool clip_path: Patch or (Path, Transform) or None facecolor or fc: color figure: *~matplotlib.figure.Figure* frame_on: bool gid: str in_layout: bool label: object mouseover: bool navigate: bool navigate_mode: unknown path_effects: list of *.AbstractPathEffect* picker: None or bool or float or callable position: [left, bottom, width, height] or *~matplotlib.transforms.Bbox* prop_cycle: *~cycler.Cycler* rasterization_zorder: float or None rasterized: bool sketch_params: (scale: float, length: float, randomness: float) snap: bool or None subplotspec: unknown title: str transform: *~matplotlib.transforms.Transform* url: str visible: bool xbound: (lower: float, upper: float) xlabel: str xlim: (left: float, right: float) xmargin: float greater than -0.5 xscale: unknown xticklabels: unknown xticks: unknown ybound: (lower: float, upper: float) ylabel: str ylim: (bottom: float, top: float) ymargin: float greater than -0.5 yscale: unknown yticklabels: unknown yticks: unknown zorder: float

class XPanAxes_NoZoom (*fig*, **args*, *facecolor=None*, *frameon=True*, *sharex=None*, *sharey=None*, *label=''*, *xscale=None*, *yscale=None*, *box_aspect=None*, ***kwargs*)

Bases: Axes

Constrain pan to x-axis and prevent zooming.

Methods:

drag_pan(button, key, x, y)	**param button** The pressed mouse button.
set(*[, adjustable, agg_filter, alpha, ...])	Set multiple properties at once.

Attributes:

name

drag_pan (*button*, *key*, *x*, *y*)

Parameters

- **button** – The pressed mouse button.
- **key** (Optional[str]) – The pressed key, if any.
- **x** (float) – The mouse coordinates in display coords.
- **y** (float) – The mouse coordinates in display coords.

name = 'XPanAxes_NoZoom'
Type: str

set(*, *adjustable=<UNSET>*, *agg_filter=<UNSET>*, *alpha=<UNSET>*, *anchor=<UNSET>*, *animated=<UNSET>*, *aspect=<UNSET>*, *autoscale_on=<UNSET>*, *autoscalex_on=<UNSET>*, *autoscaley_on=<UNSET>*, *axes_locator=<UNSET>*, *axisbelow=<UNSET>*, *box_aspect=<UNSET>*, *clip_box=<UNSET>*, *clip_on=<UNSET>*, *clip_path=<UNSET>*, *facecolor=<UNSET>*, *frame_on=<UNSET>*, *gid=<UNSET>*, *in_layout=<UNSET>*, *label=<UNSET>*, *mouseover=<UNSET>*, *navigate=<UNSET>*, *path_effects=<UNSET>*, *picker=<UNSET>*, *position=<UNSET>*, *prop_cycle=<UNSET>*, *rasterization_zorder=<UNSET>*, *rasterized=<UNSET>*, *sketch_params=<UNSET>*, *snap=<UNSET>*, *subplotspec=<UNSET>*, *title=<UNSET>*, *transform=<UNSET>*, *url=<UNSET>*, *visible=<UNSET>*, *xbound=<UNSET>*, *xlabel=<UNSET>*, *xlim=<UNSET>*, *xmargin=<UNSET>*, *xscale=<UNSET>*, *xticklabels=<UNSET>*, *xticks=<UNSET>*, *ybound=<UNSET>*, *ylabel=<UNSET>*, *ylim=<UNSET>*, *ymargin=<UNSET>*, *yscale=<UNSET>*, *yticklabels=<UNSET>*, *yticks=<UNSET>*, *zorder=<UNSET>*)
Set multiple properties at once.

Supported properties are

Properties: adjustable: {'box', 'datalim'} agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array and two offsets from the bottom left corner of the image alpha: scalar or None anchor: (float, float) or {'C', 'SW', 'S', 'SE', 'E', 'NE', ...} animated: bool aspect: {'auto', 'equal'} or float autoscale_on: bool autoscalex_on: unknown autoscaley_on: unknown axes_locator: Callable[[Axes, Renderer], Bbox] axisbelow: bool or 'line' box_aspect: float or None clip_box: *~matplotlib.transforms.BboxBase* or None clip_on: bool clip_path: Patch or (Path, Transform) or None facecolor or fc: color figure: *~matplotlib.figure.Figure* frame_on: bool gid: str in_layout: bool label: object mouseover: bool navigate: bool navigate_mode: unknown path_effects: list of *.AbstractPathEffect* picker: None or bool or float or callable position: [left, bottom, width, height] or *~matplotlib.transforms.Bbox* prop_cycle: *~cycler.Cycler* rasterization_zorder: float or None rasterized: bool sketch_params: (scale: float, length: float, randomness: float) snap: bool or None subplotspec: unknown title: str transform: *~matplotlib.transforms.Transform* url: str visible: bool xbound: (lower: float, upper: float) xlabel: str xlim: (left: float, right: float) xmargin: float greater than -0.5 xscale: unknown xticklabels: unknown xticks: unknown ybound: (lower: float, upper: float) ylabel: str ylim: (bottom: float, top: float) ymargin: float greater than -0.5 yscale: unknown yticklabels: unknown yticks: unknown zorder: float

2.18 style_picker

Dialogs for choosing Matplotlib colours and styles.

Classes:

ColourPicker(parent[, title, label, ...])	Dialog for choosing Matplotlib colours.
StylePicker(parent[, title, label, ...])	Dialog for choosing Matplotlib marker styles.

class ColourPicker(*parent*, *title='Choose Colours'*, *label='Choose Colours: '*, *picker_choices=None*, *selection_choices=None*, **args*, ***kwds*)

Bases: *StylePicker*

Dialog for choosing Matplotlib colours.

Parameters

- **parent** (Window) – Parent window. Should not be None.
- **title** (str) – The dialog title. Default 'Choose Colours'.
- **label** (str) – The dialog label. Default 'Choose Colours: '.
- **selection_choices** (Optional[List]) – Default None.
- ***args** – Additional arguments passed to wx.Dialog.
- ****kwds** – Additional keyword arguments passed to wx.Dialog.

Methods:

apply(event)	Called when the user presses the *Apply* button.

apply(*event*)

Called when the user presses the *Apply* button.

Parameters **event** – The wxPython event.

class StylePicker(*parent*, *title='Choose Styles'*, *label='Choose Styles: '*, *selection_choices=None*, **args*, ***kwds*)

Bases: Dialog

Dialog for choosing Matplotlib marker styles.

Parameters

- **parent** (Window) – Parent window. Should not be None.
- **title** (str) – The dialog title. Default 'Choose Styles'.
- **label** (str) – The dialog label. Default 'Choose Styles: '.
- **selection_choices** (Optional[List]) – Default None.
- ***args** – Additional arguments passed to wx.Dialog.
- ****kwds** – Additional keyword arguments passed to wx.Dialog.

Methods:

apply(event)	Called when the user presses the *Apply* button.
cancel(_)	Called when the user presses the *Cancel* button.

apply(*event*)
Called when the user presses the *Apply* button.

Parameters **event** – The wxPython event.

cancel(_)
Called when the user presses the *Cancel* button.

2.19 tabbable_textctrl

Multiline wx.TextCtrl that allows tabbing to the next or previous control.

Classes:

TabbableTextCtrl(parent[, id, value, pos, ...])	Multiline `wx.TextCtrl` that allows tabbing to the next or previous control.

class TabbableTextCtrl(*parent*, *id=-1*, *value=''*, *pos=(-1, -1)*, *size=(-1, -1)*, *style=0*, *validator=<wx.Validator object>*, *name=b'text'*)
Bases: `TextCtrl`

Multiline `wx.TextCtrl` that allows tabbing to the next or previous control.

Parameters

- **parent** (`Window`) – Parent window. Should not be `None`.
- **id** (`int`) – Control identifier. A value of `-1` denotes a default value. Default `-1`.
- **value** (`str`) – Default text value. Default `''`.
- **pos** (`Point`) – Text control position. Default `(-1, -1)`.
- **size** (`Size`) – Text control size. Default `(-1, -1)`.
- **style** (`int`) – Window style. See `wx.TextCtrl`. Default `0`.
- **validator** (`Validator`) – Window validator. Default `wx.DefaultValidator`.
- **name** (`AnyStr`) – Window name. Default `b'text'`.

Methods:

on_char(event)	Event handler for key being pressed, to allow for navigating between controls with `TAB`.

static on_char(*event*)
Event handler for key being pressed, to allow for navigating between controls with `TAB`.

2.20 textctrlwrapper

Classes:

TextCtrlWrapper()	Base class for wrappers around wx.TextCtrl.

class TextCtrlWrapper

Bases: object

Base class for wrappers around wx.TextCtrl.

Subclasses must set the value of *textctrl*.

Methods:

AppendText(text)	Appends the given text to the end of the text control.
CanCopy()	Returns True if the selection can be copied to the clipboard.
CanCut()	Returns True if the selection can be cut to the clipboard.
CanPaste()	Returns True if the contents of the clipboard can be pasted into the text control.
CanRedo()	Returns True if there is a redo facility available, and the last operation can be redone.
CanUndo()	Returns True if there is an undo facility available, and the last operation can be undone.
Clear()	Clears the text in the control.
Copy()	Copies the selected text to the clipboard.
Cut()	Copies the selected text to the clipboard and removes it from the control.
GetLastPosition()	Returns the zero based index of the last position in the text control, which is equal to the number of characters in the control.
GetSelection()	Gets the current selection span.
GetStringSelection()	Returns the text currently selected in the control.
GetValue()	Gets the contents of the control.
IsEditable()	Returns True if the controls contents may be edited by user (note that it always can be changed by the program).
IsEmpty()	Returns whether the control is currently empty.
Paste()	Pastes the clipboard contents into the control.
Redo()	If there is a redo facility and the last operation can be redone, redoes the last operation.
Remove(from_, to_)	Removes the text starting at the first given position up to (but not including) the character at the last position.
Replace(from_, to_, value)	Replaces the text starting at the first position up to (but not including) the character at the last position with the given text.
SelectAll()	Selects all text in the control.
SelectNone()	Deselects selected text in the control.

continues on next page

Table 69 – continued from previous page

SetSelection(from_, to_)	Selects the text starting at the first position up to (but not including) the character at the last position.
SetValue(value)	Sets the new text control value.
Undo()	If there is an undo facility, and the last operation can be undone, undoes the last operation.
WriteText(text)	Writes the text into the text control at the current insertion position.

Attributes:

textctrl	The wx.TextCtrl being wrapped.

AppendText(*text*)

Appends the given text to the end of the text control.

Note: After the text is appended, the insertion point will be at the end of the text control. If this behaviour is not desired, the programmer should use GetInsertionPoint and SetInsertionPoint.

Parameters **text** (str)

CanCopy()

Returns True if the selection can be copied to the clipboard.

Return type bool

CanCut()

Returns True if the selection can be cut to the clipboard.

Return type bool

CanPaste()

Returns True if the contents of the clipboard can be pasted into the text control.

On some platforms (Motif, GTK) this is an approximation and returns True if the control is editable, False otherwise.

Return type bool

CanRedo()

Returns True if there is a redo facility available, and the last operation can be redone.

Return type bool

CanUndo()

Returns True if there is an undo facility available, and the last operation can be undone.

Return type bool

Clear()

Clears the text in the control.

Note that this function will generate a `wx.wxEVT_TEXT` event, i.e. its effect is identical to calling *`SetValue('')`*.

Copy()
Copies the selected text to the clipboard.

Cut()
Copies the selected text to the clipboard and removes it from the control.

GetLastPosition()
Returns the zero based index of the last position in the text control, which is equal to the number of characters in the control.

Return type wx.TextPos

GetSelection()
Gets the current selection span.

If the returned values are equal, there was no selection.

Note: The indices returned may be used with the other `wx.TextCtrl` methods but don't necessarily represent the correct indices into the string returned by GetValue for multiline controls under Windows (at least,) you should use GetStringSelection to get the selected text.

Return type `Tuple`[`int`, `int`]

GetStringSelection()
Returns the text currently selected in the control.

If there is no selection, the returned string is empty.

Return type `str`

GetValue()
Gets the contents of the control.

Note: For a multiline text control, the lines will be separated by (Unix-style) `\n` characters, even under Windows where they are separated by a `\r\n` sequence in the native control.

Return type `str`

IsEditable()
Returns `True` if the controls contents may be edited by user (note that it always can be changed by the program).

In other words, this functions returns `True` if the control hasn't been put in read-only mode by a previous call to *`SetEditable()`*.

Return type `bool`

IsEmpty()
Returns whether the control is currently empty.

Return type bool

Paste()
Pastes the clipboard contents into the control.

Redo()
If there is a redo facility and the last operation can be redone, redoes the last operation.

Does nothing if there is no redo facility.

Remove(*from_*, *to_*)
Removes the text starting at the first given position up to (but not including) the character at the last position.

This function puts the current insertion point position at to as a side effect.

Parameters

- **from_** (int) – The first position
- **to_** (int) – The last position

Replace(*from_*, *to_*, *value*)
Replaces the text starting at the first position up to (but not including) the character at the last position with the given text.

This function puts the current insertion point position at to as a side effect.

Parameters

- **from_** (int) – The first position
- **to_** (int) – The last position
- **value** – The value to replace the existing text with.

Return type str

SelectAll()
Selects all text in the control.

SelectNone()
Deselects selected text in the control.

SetSelection(*from_*, *to_*)
Selects the text starting at the first position up to (but not including) the character at the last position.

If both parameters are equal to -1 all text in the control is selected.

Notice that the insertion point will be moved to from by this function.

Parameters

- **from_** (int) – The first position
- **to_** (int) – The last position

SetValue(*value*)
Sets the new text control value.

It also marks the control as not-modified which means that `IsModified()` would return `False` immediately after the call to `SetValue()`.

The insertion point is set to the start of the control (i.e. position 0) by this function unless the control value doesn't change at all, in which case the insertion point is left at its original position.

Note: Unlike most other functions changing the control's values, this function generates a `wx.wxEVT_TEXT` event. To avoid this you can use `ChangeValue()` instead.

Parameters **value** (`str`) – The new value to set. It may contain newline characters if the text control is multiline.

Undo()
If there is an undo facility, and the last operation can be undone, undoes the last operation.

Does nothing if there is no undo facility.

WriteText(*text*)
Writes the text into the text control at the current insertion position.

Parameters **text** (`str`) – Text to write to the text control

textctrl
Type: `TextCtrl`

The `wx.TextCtrl` being wrapped.

2.21 timer_thread

Background thread that sends an event after the specified interval.

Useful for timeouts or updating timers, clocks etc.

Classes:

Timer(parent[, interval])	Background Timer Class.

Data:

timer_event	An instance of `domdf_python_tools.events.SimpleEvent` called **Timer**.

class Timer(*parent*, *interval=1.0*)
Bases: `Thread`

Background Timer Class.

Parameters

- **parent** (Window) – Class to send event updates to.
- **interval** (float) – Interval to trigger events at, in seconds. Default 1.0.

Methods:

join([timeout])	Stop the thread and wait for it to end.
run()	Run the timer thread.

join (*timeout=None*)
Stop the thread and wait for it to end.

Parameters **timeout** – Default None.

run ()
Run the timer thread.

timer_event = SimpleEvent(name=Timer)
Type: *SimpleEvent*

An instance of domdf_python_tools.events.SimpleEvent called **Timer**.

This event is triggered when the timer has expired.

2.22 utils

General utility functions.

Functions:

collapse_label(text[, collapsed])	Constructs the label to display on a collapsible section.
coming_soon([msg])	Displays a message box informing the user that the desired feature has not been implemented.
generate_faces()	Returns a platform dependent set of typefaces.
toggle(control)	Toggle value of the given control.

collapse_label (*text*, *collapsed=True*)
Constructs the label to display on a collapsible section.

Parameters

- **text** (str) – The text of the label.
- **collapsed** (bool) – Whether the section is collapsed. Default True.

Return type str

coming_soon (*msg='This feature has not been implemented yet'*)
Displays a message box informing the user that the desired feature has not been implemented.

Parameters **msg** (str) – The message to display. Default 'This feature has not been implemented yet'.

generate_faces()
Returns a platform dependent set of typefaces.

Return type Dict[str, Any]

toggle(*control*)
Toggle value of the given control.

Usually used for checkboxes.

Parameters **control** (CheckBox)

Return type bool

2.23 validators

Various validator classes

Classes:

CharValidator(flag)	A Validator that only allows the type of characters selected to be entered.
FloatValidator(flag)	A Validator that only allows numbers and decimal points to be entered.
ValidatorBase()	Base class for Validators.

class CharValidator(*flag*)
Bases: *ValidatorBase*

A Validator that only allows the type of characters selected to be entered.

The possible flags are:

- int-only - only the numbers 0123456789 can be entered.
- float-only - only numbers and decimal points can be entered.

Methods:

Clone()	Clones the *CharValidator*.
OnChar(event)	Event handler for text being entered in the control.
Validate(win)	Validate the control.

Clone()
Clones the *CharValidator*.

OnChar(*event*)
Event handler for text being entered in the control.

Parameters **event** – The wxPython event.

Validate(*win*)
Validate the control.

Parameters **win**

Return type bool

class FloatValidator(*flag*)

Bases: *CharValidator*

A Validator that only allows numbers and decimal points to be entered. If a decimal point has already been entered, a second one cannot be entered. The argument *flag* is used to limit the number of decimal places that can be entered.

Methods:

OnChar(event)	Event handler for text being entered in the control.

OnChar(*event*)

Event handler for text being entered in the control.

Parameters **event** – The wxPython event.

class ValidatorBase

Bases: Validator

Base class for Validators.

Methods:

Clone()	Clones the wx.Validator.
TransferFromWindow()	Transfer data from window to validator.
TransferToWindow()	Transfer data from validator to window.
reset_ctrl()	Resets the control's background colour.
set_warning()	Set the control's background colour to pink.

Clone()

Clones the wx.Validator.

TransferFromWindow()

Transfer data from window to validator.

The default implementation returns False, indicating that an error occurred. We simply return True, as we don't do any data transfer.

Return type bool

TransferToWindow()

Transfer data from validator to window.

The default implementation returns False, indicating that an error occurred. We simply return True, as we don't do any data transfer.

Return type bool

reset_ctrl()

Resets the control's background colour.

Return type bool

set_warning()
Set the control's background colour to pink.

Return type bool

2.24 panel_listctrl

2.24.1 constants

Constants for *panel_listctrl*.

2.24.2 css_parser

Functions:

parse_css(css_data)	Parse the stylesheet from the given string
parse_css_file(filename)	Parse the stylesheet in the given file.

parse_css(*css_data*)
Parse the stylesheet from the given string

Parameters **css_data** (str) – A string representing a CSS stylesheel

Return type Dict

Returns Parsed CSS stylesheet

parse_css_file(*filename*)
Parse the stylesheet in the given file.

Parameters **filename** (Union[str, Path, PathLike]) – The filename of the stylesheet to parse.

Return type Dict

Returns Parsed CSS stylesheet.

2.24.3 font_parser

Functions:

freezeargs(func)	Makes a mutable dictionary immutable, so it can be used as an argument for slru_cache().
parse_font(style_dict)	Parse the font from the style_dict.

freezeargs(*func*)
Makes a mutable dictionary immutable, so it can be used as an argument for slru_cache().

Return type Callable

parse_font(*style_dict*)
Parse the font from the `style_dict`.

Parameters **style_dict** (`Dict`) – Dictionary containing styling information for the font

Return type `Tuple[str, Dict]`

Returns Tuple containing the colour of the font, and the font properties

The font properties dictionary returned will contain the following keys:

- family (`wx.FontFamily`) – The font family: a generic portable way of referring to fonts without specifying a facename. This parameter must be one of the wx.FontFamily enumeration values. If the faceName argument is provided, then it overrides the font family.
- style (`wx.FontStyle`) – One of `wx.FONTSTYLE_NORMAL`, `wx.FONTSTYLE_SLANT` and `wx.FONTSTYLE_ITALIC`.
- weight (`wx.FontWeight`) – Font weight, sometimes also referred to as font boldness. One of the wx.FontWeight enumeration values.
- underline (`wx.bool`) – The value can be `True` or `False`. At present this has an effect on Windows and Motif 2.x only.
- faceName (`str`) – An optional string specifying the face name to be used. If it is an empty string, a default face name will be chosen based on the family.
- encoding (`wx.FontEncoding`) – An encoding which may be one of the enumeration values of wx.FontEncoding. If the specified encoding isn't available, no font is created (see also Font Encodings).

and one of:

- pointSize (`int`) – Size in points. See SetPointSize for more info. Notice that, for historical reasons, the value 70 here is interpreted at DEFAULT and results in creation of the font with the default size and not of a font with the size of 70pt. If you really need the latter, please use SetPointSize(70). Note that this constructor and the matching Create() method overload are the only places in wx.Font API handling DEFAULT specially: neither SetPointSize nor the constructor taking wx.FontInfo handle this value in this way.
- pixelSize (`wx.Size`) – Size in pixels. See SetPixelSize for more info.

depending on the font size specified in 'style_dict'

2.24.4 panel_listctrl

A custom Panel that acts as a ListCtrl for other wx.Panel objects.

An example ListItem exists that provides two StaticText fields and can be used as the basis for custom list items

Classes:

PanelListCtrl(parent[, id, pos, size, ...])	
PanelListItem(parent, text_dict, style_data)	
	type parent *PanelListCtrl*

class PanelListCtrl(*parent*, *id=- 1*, *pos=(- 1, - 1)*, *size=(- 1, - 1)*, *style=524288*, *name=b'panel'*, *left_padding=32*)
Bases: `ScrolledWindow`

Methods:

AcceptsFocus()	
AcceptsFocusFromKeyboard()	
Append(panel_list_item)	Append a 'PanelListItem' object, or an instance of a custom subclass, to the control.
AppendNewItem(text_dict, style_data)	Append a new 'PanelListItem' object to the control, passing the 'text_dict' and 'style_data' parameters to the new object.
Clear()	Removes all items from the control
DeleteItem(item)	Deletes the specified item from the control.
DeselectAll()	Deselect all items.
Focus(idx)	Set Focus to the the given item.
GetColumnCount()	Returns the number of columns.
GetFirstSelected(*_)	Returns the first selected item, or -1 when none is selected.
GetFocusedItem()	Gets the currently focused item or -1 if none is focused.
GetItem(itemIdx, *_)	Returns information about the item.
GetItemBackgroundColour(item)	Returns the colour for this item.
GetItemCount()	Returns the number of items in the list control.
GetItemPosition(item)	Returns the position of the item, or `-1` if it is not found.
GetNextSelected(item)	Returns subsequent selected items, or -1 when no more are selected.
GetSelectedItemCount()	Returns the number of selected items in the list control.
IsEmpty()	Returns true if the control doesn't currently contain any items.
IsSelected(idx)	Returns `:py:obj:`True`` if the item is selected.
RefreshItem(item)	Redraws the given item.
RefreshItems(itemFrom, itemTo)	Redraws the items between itemFrom and itemTo.
Select(idx[, on])	Selects/deselects an item.
SetSelection(idx)	Set the current selection to the item at the given index.

Attributes:

ColumnCount	Returns the number of columns.
FocusedItem	Gets the currently focused item or -1 if none is focused.
ItemCount	Returns the number of items in the list control.

AcceptsFocus()

AcceptsFocusFromKeyboard()

Append(*panel_list_item*)
Append a 'PanelListItem' object, or an instance of a custom subclass, to the control.

AppendNewItem(*text_dict*, *style_data*)
Append a new 'PanelListItem' object to the control, passing the 'text_dict' and 'style_data' parameters to

the new object.

Parameters

- **text_dict** (Dict)
- **style_data**

Return type *PanelListItem*

Returns The new PanelListItem object that was added to the control

Clear()
Removes all items from the control

property ColumnCount
Returns the number of columns.

Return type int

DeleteItem(*item*)
Deletes the specified item from the control.

Parameters **item**

Return type bool

Returns True if the item was removed, False otherwise (usually because the item wasn't in the control)

DeselectAll()
Deselect all items.

Focus(*idx*)
Set Focus to the the given item.

Parameters **idx**

property FocusedItem
Gets the currently focused item or -1 if none is focused.

Returns

Return type

GetColumnCount()
Returns the number of columns.

GetFirstSelected(*_)
Returns the first selected item, or -1 when none is selected.

Return type int

GetFocusedItem()
Gets the currently focused item or -1 if none is focused.

Returns

Return type

GetItem(*itemIdx*, **_*)
Returns information about the item. See `SetItem()` for more information.

Parameters **itemIdx**

GetItemBackgroundColour(*item*)
Returns the colour for this item.

Return type `Colour`

GetItemCount()
Returns the number of items in the list control.

Return type `int`

GetItemPosition(*item*)
Returns the position of the item, or `-1` if it is not found.

Return type `int`

GetNextSelected(*item*)
Returns subsequent selected items, or -1 when no more are selected.

Parameters **item**

GetSelectedItemCount()
Returns the number of selected items in the list control.

Return type `int`

IsEmpty()
Returns true if the control doesn't currently contain any items.

Returns

Return type

IsSelected(*idx*)
Returns `:py:obj:`True`` if the item is selected.

Parameters **idx**

property ItemCount
Returns the number of items in the list control.

Return type int

RefreshItem(*item*)
Redraws the given item.

Parameters **item**

RefreshItems(*itemFrom*, *itemTo*)
Redraws the items between itemFrom and itemTo.

Parameters

- **itemFrom**
- **itemTo**

Select(*idx*, *on=1*)
Selects/deselects an item.

Parameters

- **idx**
- **on** (int) – Default 1.

SetSelection(*idx*)
Set the current selection to the item at the given index.

Parameters **idx** (int) – index of the item to select.

class PanelListItem(*parent*, *text_dict*, *style_data*, *id=- 1*, *style=0*, *name=b'panel'*, *left_padding=32*)
Bases: Panel

Parameters

- **parent** (*PanelListCtrl*) – The PanelListCtrl the item is to go into
- **text_dict** (Dict)
- **style_data**
- **id** (int) – An identifier for the panel. ID_ANY is taken to mean a default. Default -1.
- **style** (int) – The window style. See wx.Panel. Default 0.
- **name** (str) – Window name. Default b'panel'.
- **left_padding** (int) – the spacing to the left of the text in the control. Default 32.

Methods:

DeselectItem()	
GetBackgroundColour()	
GetContents()	
GetCurrentBackgroundColour()	Returns the current background colour of the :class`wx.Panel` class.
IsSelected()	Returns whether the *PanelListItem* is selected.
OnClick(_)	
OnDoubleClick(_)	
OnKeyDown(event)	**param event** The wxPython event.
OnMiddleClick(_)	
OnRightClick(_)	
Refresh(**kwargs)	
SelectItem([select])	Select (or deselect) the given item.
SetBackgroundColour(colour)	Set the background colour for the item.
SetSelectedBackgroundColour(colour)	Set the background colour for the item when it is selected.

DeselectItem()

GetBackgroundColour()

GetContents()

GetCurrentBackgroundColour()
Returns the current background colour of the :class`wx.Panel` class.

IsSelected()
Returns whether the *PanelListItem* is selected.

Return type bool

OnClick(_)

OnDoubleClick(_)

OnKeyDown(*event*)

Parameters **event** – The wxPython event.

OnMiddleClick(_)

OnRightClick(_)

Refresh(***kwargs*)

SelectItem(*select=True*)
Select (or deselect) the given item.

Parameters **select** (bool) – If False the item is deselected. Default True.

SetBackgroundColour(*colour*)
Set the background colour for the item.

Parameters **colour**

SetSelectedBackgroundColour(*colour*)
Set the background colour for the item when it is selected.

Parameters **colour**

Chapter

THREE

Contributing

3.1 Overview

`domdf_wxpython_tools` uses tox to automate testing and packaging, and pre-commit to maintain code quality.

Install `pre-commit` with `pip` and install the git hook:

```
$ python -m pip install pre-commit
$ pre-commit install
```

3.2 Coding style

formate is used for code formatting.

It can be run manually via `pre-commit`:

```
$ pre-commit run formate -a
```

Or, to run the complete autoformatting suite:

```
$ pre-commit run -a
```

3.3 Automated tests

Tests are run with `tox` and `pytest`. To run tests for a specific Python version, such as Python 3.6:

```
$ tox -e py36
```

To run tests for all Python versions, simply run:

```
$ tox
```

3.4 Type Annotations

Type annotations are checked using `mypy`. Run `mypy` using `tox`:

```
$ tox -e mypy
```

3.5 Build documentation locally

The documentation is powered by Sphinx. A local copy of the documentation can be built with `tox`:

```
$ tox -e docs
```

3.6 Downloading source code

The `domdf_wxpython_tools` source code is available on GitHub, and can be accessed from the following URL: https://github.com/domdfcoding/domdf_wxpython_tools

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/domdfcoding/domdf_wxpython_tools
```

```
Cloning into 'domdf_wxpython_tools'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

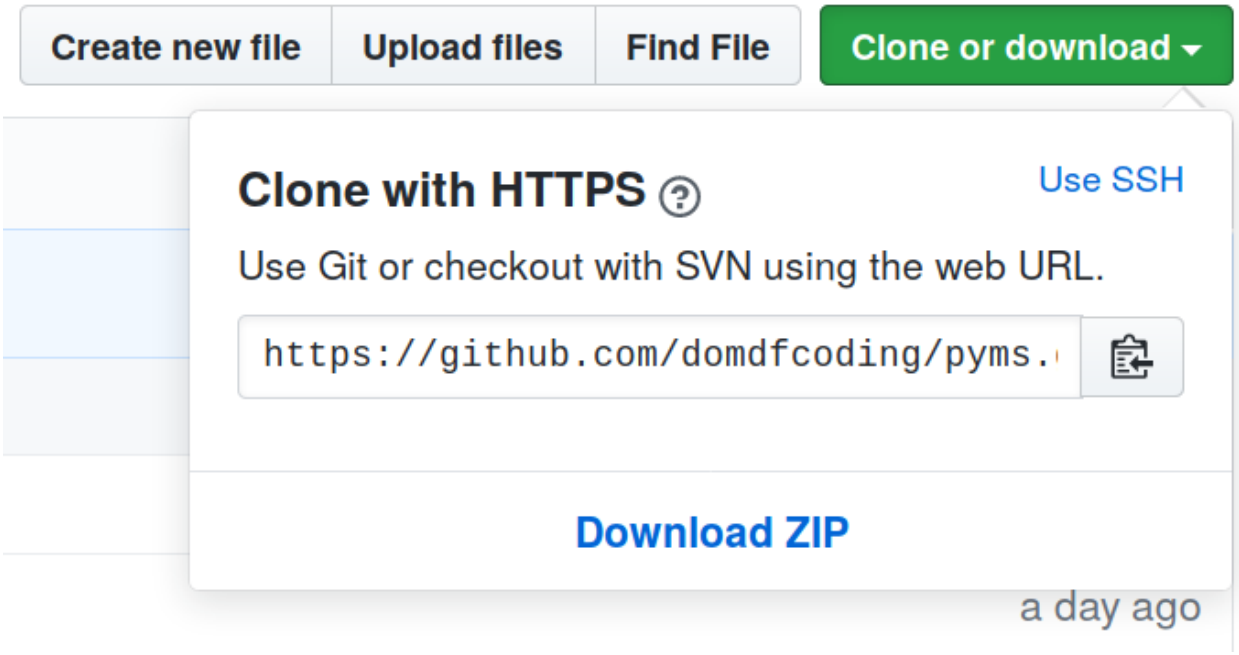

Fig. 1: Downloading a 'zip' file of the source code

3.6.1 Building from source

The recommended way to build `domdf_wxpython_tools` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

Python Module Index

d

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Z