pyRevit Documentation

Release 4.7.6

eirannejad

May 01, 2020

pyRevit Module

Note: This documentation is a work-in-progress. Thanks for your patience.

Getting Started

I suggest reading this section completely as it provides 99% of what you will need to know for developing scripts in pyRevit environment. Other sections dive deeper into pyRevit inner workings.

pyRevit Module

CHAPTER 1

pyrevit

pyRevit root level config for all pyrevit sub-modules.

Examples

```
>>> from pyrevit import DB, UI
>>> from pyrevit import PyRevitException, PyRevitIOError
```

```
>>> # pyrevit module has global instance of the
>>> # _HostAppPostableCommand and _ExecutorParams classes already created
>>> # import and use them like below
>>> from pyrevit import HOST_APP
>>> from pyrevit import EXEC_PARAMS
```

class pyrevit.**PyRevitException**
Common base class for all pyRevit exceptions.

Parameters args and message are derived from Exception class.

class pyrevit.**PyRevitIOError**
Common base class for all pyRevit io-related exceptions.

class pyrevit.**_HostAppPostableCommand**(*name*, *key*, *id*, *rvtobj*)
Private namedtuple for passing information about a PostableCommand

name
Postable command name

Type str

key
Postable command key string

Type str

id
Postable command id

Type int

rvtobj
Postable command Id Object

Type RevitCommandId

class pyrevit._HostApplication
Private Wrapper for Current Instance of Revit.

Provides version info and comparison functionality, alongside providing info on the active screen, active document and ui-document, available postable commands, and other functionality.

Parameters **host_uiapp** (UIApplication) – Instance of running host.

Example

```
>>> hostapp = _HostApplication()
>>> hostapp.is_newer_than(2017)
```

active_view
Return view that is active (UIDocument.ActiveView).

addin_id
Return active addin id.

app
Return Application provided to the running command.

available_servers
Return list of available Revit server names.

build
Return build number (e.g. '20170927_1515(x64)').

Type str

doc
Return active Document.

docs
Return list of open Document objects.

get_postable_commands()
Return list of postable commands.

Returns list of *_HostAppPostableCommand*

has_api_context
Determine if host application is in API context

is_demo
Determine if product is using demo license.

Type bool

is_exactly(*version*)
bool: Return True if host app is equal to provided version.

Parameters **version** (*str or int*) – version to check against.

is_newer_than(*version*, *or_equal=False*)
bool: Return True if host app is newer than provided version.

Parameters **version** (*str or int*) – version to check against.

is_older_than(*version*)
bool: Return True if host app is older than provided version.

Parameters **version** (*str or int*) – version to check against.

language
Return language type (e.g. 'LanguageType.English_USA').

Type str

pretty_name
Pretty name of the host (e.g. 'Autodesk Revit 2019.2 build: 20190808_0900(x64)')

Type str

proc
Return current process object.

Type System.Diagnostics.Process

proc_id
Return current process id.

Type int

proc_name
Return current process name.

Type str

proc_path
Return file path for the current process main module.

Type str

proc_screen
Return handle to screen hosting current process.

Type intptr

proc_screen_scalefactor
Return scaling for screen hosting current process.

Type float

proc_screen_workarea
Return screen working area.

Type System.Drawing.Rectangle

proc_window
Return handle to screen hosting current process.

Type intptr

serial_no
Return serial number number (e.g. '569-09704828').

Type str

subversion
Return subversion number (e.g. '2018.3').

Type str

uiapp
Return UIApplication provided to the running command.

uidoc
Return active UIDocument.

username
Return the username from Revit API (Application.Username).

Type str

version
Return version number (e.g. '2018').

Type str

version_name
Return version name (e.g. 'Autodesk Revit 2018').

Type str

class pyrevit.**_ExecutorParams**
Private Wrapper that provides runtime environment info.

cached_engine
Check whether pyrevit is running on a cached engine.

Type bool

command_bundle
Return current command bundle name.

Type str

command_config_path
Return current command config script path.

Type str

command_controlid
Return current command control id.

Type str

command_data
Return current command data.

Type ExternalCommandData

command_elements
Return elements passed to by Revit.

Type DB.ElementSet

command_extension
Return current command extension name.

Type str

command_mode
Check if pyrevit is running in pyrevit command context.

Type bool

command_name
Return current command name.

Type str

command_path
Return current command path.

Type str

command_uibutton
Return current command ui button.

Type str

command_uniqueid
Return current command unique id.

Type str

config_mode
Check if command is in config mode.

Type bool

debug_mode
Check if command is in debug mode.

Type bool

doc_mode
Check if pyrevit is running by doc generator.

Type bool

engine_cfgs
Return ScriptRuntime.ScriptRuntimeConfigs

engine_id
Return engine id

engine_ver
Return PyRevitLoader.ScriptExecutor hardcoded version.

Type str

event_args
Return event arguments object.

Type `DB.RevitAPIEventArgs`

event_sender
Return event sender object.

Type `Object`

exec_id
Return execution unique id

exec_timestamp
Return execution timestamp

executed_from_ui
Check if command was executed from ui.

Type bool

first_load
Check whether pyrevit is not running in pyrevit command.

Type bool

needs_clean_engine
Check if command needs a clean IronPython engine.

Type bool

needs_fullframe_engine
Check if command needs a full-frame IronPython engine.

Type bool

needs_persistent_engine
Check if command needs a persistent IronPython engine.

Type bool

needs_refreshed_engine
Check if command needs a newly refreshed IronPython engine.

Type bool

output_stream
Return ScriptIO

result_dict
Return results dict for logging.

Type `Dictionary<String, String>`

script_data
Return ScriptRuntime.ScriptData

script_runtime
Return command.

Type `PyRevitLabs.PyRevit.Runtime.ScriptRuntime`

script_runtime_cfgs
Return ScriptRuntime.ScriptRuntimeConfigs

window_handle
Return output window. handle

Type `PyRevitLabs.PyRevit.Runtime.ScriptConsole`

CHAPTER 2

pyrevit.api

Provide access to Revit API.

Example

```
>>> from pyrevit.api import AdWindows
```

pyrevit.api.**get_product_serial_number**()
Return serial number of running host instance.

pyrevit.api.**is_api_object**(*data_type*)
Check if given object belongs to Revit API

pyrevit.api.**is_product_demo**()
Determine if product is using demo license

CHAPTER 3

pyrevit.compat

python engine compatibility module.

Example

```
>>> from pyrevit.compat import IRONPY277
>>> from pyrevit.compat import safe_strtype
```

CHAPTER 4

pyrevit.coreutils

4.1 pyrevit.coreutils.appdata

Utility functions for creating data files within pyRevit environment.

Most times, scripts need to save some data to share between different scripts that work on a similar topic or between script executions. This module provides the necessary and consistent mechanism for creating and maintaining such files.

Example

```
>>> from pyrevit.coreutils import appdata
>>> appdata.list_data_files()
```

pyrevit.coreutils.appdata.**cleanup_appdata_folder**()
Cleanup appdata folder of all temporary appdata files.

pyrevit.coreutils.appdata.**find_data_files**(*file_ext*)
Find data files in all data files directories

Parameters **file_ext** (*str*) – data files with this extension will be listed only

Returns list of files

Return type list

pyrevit.coreutils.appdata.**find_instance_data_files**(*file_ext*, *instance_id*)
Find instance data files in all data files directories

Parameters

- **file_ext** (*str*) – data files with this extension will be listed only
- **instance_id** (*int*) – list data files for this instance id only

Returns list of files

Return type list

pyrevit.coreutils.appdata.**garbage_data_file**(*file_path*)
Mark and remove the given appdata file.

Current implementation removes the file immediately.

Parameters **file_path** (*str*) – path to the target file

pyrevit.coreutils.appdata.**get_data_file**(*file_id*, *file_ext*, *name_only=False*)
Get path to file that will not be cleaned up at Revit load.

e.g pyrevit_2016_eirannejad_file_id.file_ext

Parameters

- **file_id** (*str*) – Unique identifier for the file
- **file_ext** (*str*) – File extension
- **name_only** (*bool*) – If true, function returns file name only

Returns File name or full file path (depending on name_only)

Return type str

pyrevit.coreutils.appdata.**get_instance_data_file**(*file_id*, *file_ext='tmp'*, *name_only=False*)
Get path to file that should be used by current instance only.

These data files will be cleaned up at Revit restart. e.g pyrevit_2016_eirannejad_2353_file_id.file_ext

Parameters

- **file_id** (*str*) – Unique identifier for the file
- **file_ext** (*str*) – File extension
- **name_only** (*bool*) – If true, function returns file name only

Returns File name or full file path (depending on name_only)

Return type str

pyrevit.coreutils.appdata.**get_universal_data_file**(*file_id*, *file_ext*, *name_only=False*)
Get path to file that is shared between all host versions.

These data files are not cleaned up at Revit restart. e.g pyrevit_eirannejad_file_id.file_ext

Parameters

- **file_id** (*str*) – Unique identifier for the file
- **file_ext** (*str*) – File extension
- **name_only** (*bool*) – If true, function returns file name only

Returns File name or full file path (depending on name_only)

Return type str

pyrevit.coreutils.appdata.**is_data_file_available**(*file_id*, *file_ext*)
Check if given file is available within appdata directory.

Parameters

- **file_id** (*str*) – data file id
- **file_ext** (*str*) – file extension

Returns file path if file is available

Return type str

pyrevit.coreutils.appdata.**is_file_available**(*file_name*, *file_ext*, *universal=False*)
Check if given file is available within appdata directory.

Parameters

- **file_name** (*str*) – file name
- **file_ext** (*str*) – file extension
- **universal** (*bool*) – Check against universal data files

Returns file path if file is available

Return type str

pyrevit.coreutils.appdata.**is_pyrevit_data_file**(*file_name*)
Check if given file is a pyRevit data file.

Parameters **file_name** (*str*) – file name

Returns True if file is a pyRevit data file

Return type bool

pyrevit.coreutils.appdata.**list_data_files**(*file_ext*, *universal=False*)
List all data files with given extension.

Parameters

- **file_ext** (*str*) – file extension
- **universal** (*bool*) – Check against universal data files

Returns list of files

Return type list

pyrevit.coreutils.appdata.**list_instance_data_files**(*file_ext*)
List all data files associated with current session.

Parameters **file_ext** (*str*) – data files with this extension will be listed only.

Returns list of data files

Return type list

4.2 pyrevit.coreutils.charts

Charts engine for output window

class pyrevit.coreutils.charts.**PyRevitOutputChart**(*output*, *chart_type='line'*, *version=None*)
Bases: object

Chart wrapper object for output window.

output
output window wrapper object

Type *pyrevit.output.PyRevitOutputWindow*

chart_type
chart type name

Type str

draw()
Request chart to draw itself on output window.

randomize_colors()
Randomize chart datasets colors.

set_height(*height*)
Set chart height on output window.

set_style(*html_style*)
Set chart styling.

Parameters **html_style** (*str*) – inline html css styling string

Example

```
>>> chart.set_style('height:150px')
```

set_width(*width*)
Set chart width on output window.

class pyrevit.coreutils.charts.**PyRevitOutputChartData**
Bases: object

Chart data wrapper object.

new_dataset(*dataset_label*)
Create new data set.

Parameters **dataset_label** (*str*) – dataset label

Returns dataset wrapper object

Return type *PyRevitOutputChartDataset*

Example

```
>>> chart.data.new_dataset('set_a')
```

class pyrevit.coreutils.charts.**PyRevitOutputChartDataset**(*label*)
Bases: object

Chart dataset wrapper object.

set_color(**args*)
Set dataset color.

Arguments are expected to be R, G, B, A values.

Example

```
>>> dataset_obj.set_color(0xFF, 0x8C, 0x8D, 0.8)
```

class pyrevit.coreutils.charts.**PyRevitOutputChartOptions**
Bases: object

Chart options wrapper object.

4.3 pyrevit.coreutils.colors

Provide RGB color constants and a colors dictionary with elements formatted: COLORS[colorname] = CONSTANT

Example

```
>>> from pyrevit.coreutils import colors
>>> colors.COLORS['black']
... <RGB #000000>
>>> colors.BLACK
... <RGB #000000>
```

class pyrevit.coreutils.colors.**RGB**(*name='default'*, *red=0*, *green=0*, *blue=0*)
RGB named color object.

name
color name

Type str

red
value for red component (0-255)

Type int

green
value for green component (0-255)

Type int

blue
value for blue component (0-255)

Type int

hex_color
Return color in hex format

luminance
Return color luminance (preceived)

safe_text_color
Return text color that is safe to overlap this color.

4.3.1 Pre-Defined Colors

```
# color consts
ALICEBLUE = RGB(name='aliceblue', red=240, green=248, blue=255)
ANTIQUEWHITE = RGB(name='antiquewhite', red=250, green=235, blue=215)
ANTIQUEWHITE1 = RGB(name='antiquewhite1', red=255, green=239, blue=219)
ANTIQUEWHITE2 = RGB(name='antiquewhite2', red=238, green=223, blue=204)
```

(continues on next page)

(continued from previous page)

```
ANTIQUEWHITE3 = RGB(name='antiquewhite3', red=205, green=192, blue=176)
ANTIQUEWHITE4 = RGB(name='antiquewhite4', red=139, green=131, blue=120)
AQUA = RGB(name='aqua', red=0, green=255, blue=255)
AQUAMARINE1 = RGB(name='aquamarine1', red=127, green=255, blue=212)
AQUAMARINE2 = RGB(name='aquamarine2', red=118, green=238, blue=198)
AQUAMARINE3 = RGB(name='aquamarine3', red=102, green=205, blue=170)
AQUAMARINE4 = RGB(name='aquamarine4', red=69, green=139, blue=116)
AZURE1 = RGB(name='azure1', red=240, green=255, blue=255)
AZURE2 = RGB(name='azure2', red=224, green=238, blue=238)
AZURE3 = RGB(name='azure3', red=193, green=205, blue=205)
AZURE4 = RGB(name='azure4', red=131, green=139, blue=139)
BANANA = RGB(name='banana', red=227, green=207, blue=87)
BEIGE = RGB(name='beige', red=245, green=245, blue=220)
BISQUE1 = RGB(name='bisque1', red=255, green=228, blue=196)
BISQUE2 = RGB(name='bisque2', red=238, green=213, blue=183)
BISQUE3 = RGB(name='bisque3', red=205, green=183, blue=158)
BISQUE4 = RGB(name='bisque4', red=139, green=125, blue=107)
BLACK = RGB(name='black', red=0, green=0, blue=0)
BLANCHEDALMOND = RGB(name='blanchedalmond', red=255, green=235, blue=205)
BLUE = RGB(name='blue', red=0, green=0, blue=255)
BLUE2 = RGB(name='blue2', red=0, green=0, blue=238)
BLUE3 = RGB(name='blue3', red=0, green=0, blue=205)
BLUE4 = RGB(name='blue4', red=0, green=0, blue=139)
BLUEVIOLET = RGB(name='blueviolet', red=138, green=43, blue=226)
BRICK = RGB(name='brick', red=156, green=102, blue=31)
BROWN = RGB(name='brown', red=165, green=42, blue=42)
BROWN1 = RGB(name='brown1', red=255, green=64, blue=64)
BROWN2 = RGB(name='brown2', red=238, green=59, blue=59)
BROWN3 = RGB(name='brown3', red=205, green=51, blue=51)
BROWN4 = RGB(name='brown4', red=139, green=35, blue=35)
BURLYWOOD = RGB(name='burlywood', red=222, green=184, blue=135)
BURLYWOOD1 = RGB(name='burlywood1', red=255, green=211, blue=155)
BURLYWOOD2 = RGB(name='burlywood2', red=238, green=197, blue=145)
BURLYWOOD3 = RGB(name='burlywood3', red=205, green=170, blue=125)
BURLYWOOD4 = RGB(name='burlywood4', red=139, green=115, blue=85)
BURNTSIENNA = RGB(name='burntsienna', red=138, green=54, blue=15)
BURNTUMBER = RGB(name='burntumber', red=138, green=51, blue=36)
CADETBLUE = RGB(name='cadetblue', red=95, green=158, blue=160)
CADETBLUE1 = RGB(name='cadetblue1', red=152, green=245, blue=255)
CADETBLUE2 = RGB(name='cadetblue2', red=142, green=229, blue=238)
CADETBLUE3 = RGB(name='cadetblue3', red=122, green=197, blue=205)
CADETBLUE4 = RGB(name='cadetblue4', red=83, green=134, blue=139)
CADMIUMORANGE = RGB(name='cadmiumorange', red=255, green=97, blue=3)
CADMIUMYELLOW = RGB(name='cadmiumyellow', red=255, green=153, blue=18)
CARROT = RGB(name='carrot', red=237, green=145, blue=33)
CHARTREUSE1 = RGB(name='chartreuse1', red=127, green=255, blue=0)
CHARTREUSE2 = RGB(name='chartreuse2', red=118, green=238, blue=0)
CHARTREUSE3 = RGB(name='chartreuse3', red=102, green=205, blue=0)
CHARTREUSE4 = RGB(name='chartreuse4', red=69, green=139, blue=0)
CHOCOLATE = RGB(name='chocolate', red=210, green=105, blue=30)
CHOCOLATE1 = RGB(name='chocolate1', red=255, green=127, blue=36)
CHOCOLATE2 = RGB(name='chocolate2', red=238, green=118, blue=33)
CHOCOLATE3 = RGB(name='chocolate3', red=205, green=102, blue=29)
CHOCOLATE4 = RGB(name='chocolate4', red=139, green=69, blue=19)
COBALT = RGB(name='cobalt', red=61, green=89, blue=171)
COBALTGREEN = RGB(name='cobaltgreen', red=61, green=145, blue=64)
COLDGREY = RGB(name='coldgrey', red=128, green=138, blue=135)
```

(continues on next page)

(continued from previous page)

```
CORAL = RGB(name='coral', red=255, green=127, blue=80)
CORAL1 = RGB(name='coral1', red=255, green=114, blue=86)
CORAL2 = RGB(name='coral2', red=238, green=106, blue=80)
CORAL3 = RGB(name='coral3', red=205, green=91, blue=69)
CORAL4 = RGB(name='coral4', red=139, green=62, blue=47)
CORNFLOWERBLUE = RGB(name='cornflowerblue', red=100, green=149, blue=237)
CORNSILK1 = RGB(name='cornsilk1', red=255, green=248, blue=220)
CORNSILK2 = RGB(name='cornsilk2', red=238, green=232, blue=205)
CORNSILK3 = RGB(name='cornsilk3', red=205, green=200, blue=177)
CORNSILK4 = RGB(name='cornsilk4', red=139, green=136, blue=120)
CRIMSON = RGB(name='crimson', red=220, green=20, blue=60)
CYAN2 = RGB(name='cyan2', red=0, green=238, blue=238)
CYAN3 = RGB(name='cyan3', red=0, green=205, blue=205)
CYAN4 = RGB(name='cyan4', red=0, green=139, blue=139)
DARKGOLDENROD = RGB(name='darkgoldenrod', red=184, green=134, blue=11)
DARKGOLDENROD1 = RGB(name='darkgoldenrod1', red=255, green=185, blue=15)
DARKGOLDENROD2 = RGB(name='darkgoldenrod2', red=238, green=173, blue=14)
DARKGOLDENROD3 = RGB(name='darkgoldenrod3', red=205, green=149, blue=12)
DARKGOLDENROD4 = RGB(name='darkgoldenrod4', red=139, green=101, blue=8)
DARKGRAY = RGB(name='darkgray', red=169, green=169, blue=169)
DARKGREEN = RGB(name='darkgreen', red=0, green=100, blue=0)
DARKKHAKI = RGB(name='darkkhaki', red=189, green=183, blue=107)
DARKOLIVEGREEN = RGB(name='darkolivegreen', red=85, green=107, blue=47)
DARKOLIVEGREEN1 = RGB(name='darkolivegreen1', red=202, green=255, blue=112)
DARKOLIVEGREEN2 = RGB(name='darkolivegreen2', red=188, green=238, blue=104)
DARKOLIVEGREEN3 = RGB(name='darkolivegreen3', red=162, green=205, blue=90)
DARKOLIVEGREEN4 = RGB(name='darkolivegreen4', red=110, green=139, blue=61)
DARKORANGE = RGB(name='darkorange', red=255, green=140, blue=0)
DARKORANGE1 = RGB(name='darkorange1', red=255, green=127, blue=0)
DARKORANGE2 = RGB(name='darkorange2', red=238, green=118, blue=0)
DARKORANGE3 = RGB(name='darkorange3', red=205, green=102, blue=0)
DARKORANGE4 = RGB(name='darkorange4', red=139, green=69, blue=0)
DARKORCHID = RGB(name='darkorchid', red=153, green=50, blue=204)
DARKORCHID1 = RGB(name='darkorchid1', red=191, green=62, blue=255)
DARKORCHID2 = RGB(name='darkorchid2', red=178, green=58, blue=238)
DARKORCHID3 = RGB(name='darkorchid3', red=154, green=50, blue=205)
DARKORCHID4 = RGB(name='darkorchid4', red=104, green=34, blue=139)
DARKSALMON = RGB(name='darksalmon', red=233, green=150, blue=122)
DARKSEAGREEN = RGB(name='darkseagreen', red=143, green=188, blue=143)
DARKSEAGREEN1 = RGB(name='darkseagreen1', red=193, green=255, blue=193)
DARKSEAGREEN2 = RGB(name='darkseagreen2', red=180, green=238, blue=180)
DARKSEAGREEN3 = RGB(name='darkseagreen3', red=155, green=205, blue=155)
DARKSEAGREEN4 = RGB(name='darkseagreen4', red=105, green=139, blue=105)
DARKSLATEBLUE = RGB(name='darkslateblue', red=72, green=61, blue=139)
DARKSLATEGRAY = RGB(name='darkslategray', red=47, green=79, blue=79)
DARKSLATEGRAY1 = RGB(name='darkslategray1', red=151, green=255, blue=255)
DARKSLATEGRAY2 = RGB(name='darkslategray2', red=141, green=238, blue=238)
DARKSLATEGRAY3 = RGB(name='darkslategray3', red=121, green=205, blue=205)
DARKSLATEGRAY4 = RGB(name='darkslategray4', red=82, green=139, blue=139)
DARKTURQUOISE = RGB(name='darkturquoise', red=0, green=206, blue=209)
DARKVIOLET = RGB(name='darkviolet', red=148, green=0, blue=211)
DEEPPINK1 = RGB(name='deeppink1', red=255, green=20, blue=147)
DEEPPINK2 = RGB(name='deeppink2', red=238, green=18, blue=137)
DEEPPINK3 = RGB(name='deeppink3', red=205, green=16, blue=118)
DEEPPINK4 = RGB(name='deeppink4', red=139, green=10, blue=80)
DEEPSKYBLUE1 = RGB(name='deepskyblue1', red=0, green=191, blue=255)
DEEPSKYBLUE2 = RGB(name='deepskyblue2', red=0, green=178, blue=238)
```

(continues on next page)

(continued from previous page)

```
DEEPSKYBLUE3 = RGB(name='deepskyblue3', red=0, green=154, blue=205)
DEEPSKYBLUE4 = RGB(name='deepskyblue4', red=0, green=104, blue=139)
DIMGRAY = RGB(name='dimgray', red=105, green=105, blue=105)
DIMGRAY = RGB(name='dimgray', red=105, green=105, blue=105)
DODGERBLUE1 = RGB(name='dodgerblue1', red=30, green=144, blue=255)
DODGERBLUE2 = RGB(name='dodgerblue2', red=28, green=134, blue=238)
DODGERBLUE3 = RGB(name='dodgerblue3', red=24, green=116, blue=205)
DODGERBLUE4 = RGB(name='dodgerblue4', red=16, green=78, blue=139)
EGGSHELL = RGB(name='eggshell', red=252, green=230, blue=201)
EMERALDGREEN = RGB(name='emeraldgreen', red=0, green=201, blue=87)
FIREBRICK = RGB(name='firebrick', red=178, green=34, blue=34)
FIREBRICK1 = RGB(name='firebrick1', red=255, green=48, blue=48)
FIREBRICK2 = RGB(name='firebrick2', red=238, green=44, blue=44)
FIREBRICK3 = RGB(name='firebrick3', red=205, green=38, blue=38)
FIREBRICK4 = RGB(name='firebrick4', red=139, green=26, blue=26)
FLESH = RGB(name='flesh', red=255, green=125, blue=64)
FLORALWHITE = RGB(name='floralwhite', red=255, green=250, blue=240)
FORESTGREEN = RGB(name='forestgreen', red=34, green=139, blue=34)
GAINSBORO = RGB(name='gainsboro', red=220, green=220, blue=220)
GHOSTWHITE = RGB(name='ghostwhite', red=248, green=248, blue=255)
GOLD1 = RGB(name='gold1', red=255, green=215, blue=0)
GOLD2 = RGB(name='gold2', red=238, green=201, blue=0)
GOLD3 = RGB(name='gold3', red=205, green=173, blue=0)
GOLD4 = RGB(name='gold4', red=139, green=117, blue=0)
GOLDENROD = RGB(name='goldenrod', red=218, green=165, blue=32)
GOLDENROD1 = RGB(name='goldenrod1', red=255, green=193, blue=37)
GOLDENROD2 = RGB(name='goldenrod2', red=238, green=180, blue=34)
GOLDENROD3 = RGB(name='goldenrod3', red=205, green=155, blue=29)
GOLDENROD4 = RGB(name='goldenrod4', red=139, green=105, blue=20)
GRAY = RGB(name='gray', red=128, green=128, blue=128)
GRAY1 = RGB(name='gray1', red=3, green=3, blue=3)
GRAY10 = RGB(name='gray10', red=26, green=26, blue=26)
GRAY11 = RGB(name='gray11', red=28, green=28, blue=28)
GRAY12 = RGB(name='gray12', red=31, green=31, blue=31)
GRAY13 = RGB(name='gray13', red=33, green=33, blue=33)
GRAY14 = RGB(name='gray14', red=36, green=36, blue=36)
GRAY15 = RGB(name='gray15', red=38, green=38, blue=38)
GRAY16 = RGB(name='gray16', red=41, green=41, blue=41)
GRAY17 = RGB(name='gray17', red=43, green=43, blue=43)
GRAY18 = RGB(name='gray18', red=46, green=46, blue=46)
GRAY19 = RGB(name='gray19', red=48, green=48, blue=48)
GRAY2 = RGB(name='gray2', red=5, green=5, blue=5)
GRAY20 = RGB(name='gray20', red=51, green=51, blue=51)
GRAY21 = RGB(name='gray21', red=54, green=54, blue=54)
GRAY22 = RGB(name='gray22', red=56, green=56, blue=56)
GRAY23 = RGB(name='gray23', red=59, green=59, blue=59)
GRAY24 = RGB(name='gray24', red=61, green=61, blue=61)
GRAY25 = RGB(name='gray25', red=64, green=64, blue=64)
GRAY26 = RGB(name='gray26', red=66, green=66, blue=66)
GRAY27 = RGB(name='gray27', red=69, green=69, blue=69)
GRAY28 = RGB(name='gray28', red=71, green=71, blue=71)
GRAY29 = RGB(name='gray29', red=74, green=74, blue=74)
GRAY3 = RGB(name='gray3', red=8, green=8, blue=8)
GRAY30 = RGB(name='gray30', red=77, green=77, blue=77)
GRAY31 = RGB(name='gray31', red=79, green=79, blue=79)
GRAY32 = RGB(name='gray32', red=82, green=82, blue=82)
GRAY33 = RGB(name='gray33', red=84, green=84, blue=84)
```

(continues on next page)

(continued from previous page)

```
GRAY34 = RGB(name='gray34', red=87, green=87, blue=87)
GRAY35 = RGB(name='gray35', red=89, green=89, blue=89)
GRAY36 = RGB(name='gray36', red=92, green=92, blue=92)
GRAY37 = RGB(name='gray37', red=94, green=94, blue=94)
GRAY38 = RGB(name='gray38', red=97, green=97, blue=97)
GRAY39 = RGB(name='gray39', red=99, green=99, blue=99)
GRAY4 = RGB(name='gray4', red=10, green=10, blue=10)
GRAY40 = RGB(name='gray40', red=102, green=102, blue=102)
GRAY42 = RGB(name='gray42', red=107, green=107, blue=107)
GRAY43 = RGB(name='gray43', red=110, green=110, blue=110)
GRAY44 = RGB(name='gray44', red=112, green=112, blue=112)
GRAY45 = RGB(name='gray45', red=115, green=115, blue=115)
GRAY46 = RGB(name='gray46', red=117, green=117, blue=117)
GRAY47 = RGB(name='gray47', red=120, green=120, blue=120)
GRAY48 = RGB(name='gray48', red=122, green=122, blue=122)
GRAY49 = RGB(name='gray49', red=125, green=125, blue=125)
GRAY5 = RGB(name='gray5', red=13, green=13, blue=13)
GRAY50 = RGB(name='gray50', red=127, green=127, blue=127)
GRAY51 = RGB(name='gray51', red=130, green=130, blue=130)
GRAY52 = RGB(name='gray52', red=133, green=133, blue=133)
GRAY53 = RGB(name='gray53', red=135, green=135, blue=135)
GRAY54 = RGB(name='gray54', red=138, green=138, blue=138)
GRAY55 = RGB(name='gray55', red=140, green=140, blue=140)
GRAY56 = RGB(name='gray56', red=143, green=143, blue=143)
GRAY57 = RGB(name='gray57', red=145, green=145, blue=145)
GRAY58 = RGB(name='gray58', red=148, green=148, blue=148)
GRAY59 = RGB(name='gray59', red=150, green=150, blue=150)
GRAY6 = RGB(name='gray6', red=15, green=15, blue=15)
GRAY60 = RGB(name='gray60', red=153, green=153, blue=153)
GRAY61 = RGB(name='gray61', red=156, green=156, blue=156)
GRAY62 = RGB(name='gray62', red=158, green=158, blue=158)
GRAY63 = RGB(name='gray63', red=161, green=161, blue=161)
GRAY64 = RGB(name='gray64', red=163, green=163, blue=163)
GRAY65 = RGB(name='gray65', red=166, green=166, blue=166)
GRAY66 = RGB(name='gray66', red=168, green=168, blue=168)
GRAY67 = RGB(name='gray67', red=171, green=171, blue=171)
GRAY68 = RGB(name='gray68', red=173, green=173, blue=173)
GRAY69 = RGB(name='gray69', red=176, green=176, blue=176)
GRAY7 = RGB(name='gray7', red=18, green=18, blue=18)
GRAY70 = RGB(name='gray70', red=179, green=179, blue=179)
GRAY71 = RGB(name='gray71', red=181, green=181, blue=181)
GRAY72 = RGB(name='gray72', red=184, green=184, blue=184)
GRAY73 = RGB(name='gray73', red=186, green=186, blue=186)
GRAY74 = RGB(name='gray74', red=189, green=189, blue=189)
GRAY75 = RGB(name='gray75', red=191, green=191, blue=191)
GRAY76 = RGB(name='gray76', red=194, green=194, blue=194)
GRAY77 = RGB(name='gray77', red=196, green=196, blue=196)
GRAY78 = RGB(name='gray78', red=199, green=199, blue=199)
GRAY79 = RGB(name='gray79', red=201, green=201, blue=201)
GRAY8 = RGB(name='gray8', red=20, green=20, blue=20)
GRAY80 = RGB(name='gray80', red=204, green=204, blue=204)
GRAY81 = RGB(name='gray81', red=207, green=207, blue=207)
GRAY82 = RGB(name='gray82', red=209, green=209, blue=209)
GRAY83 = RGB(name='gray83', red=212, green=212, blue=212)
GRAY84 = RGB(name='gray84', red=214, green=214, blue=214)
GRAY85 = RGB(name='gray85', red=217, green=217, blue=217)
GRAY86 = RGB(name='gray86', red=219, green=219, blue=219)
```

(continues on next page)

(continued from previous page)

```
GRAY87 = RGB(name='gray87', red=222, green=222, blue=222)
GRAY88 = RGB(name='gray88', red=224, green=224, blue=224)
GRAY89 = RGB(name='gray89', red=227, green=227, blue=227)
GRAY9 = RGB(name='gray9', red=23, green=23, blue=23)
GRAY90 = RGB(name='gray90', red=229, green=229, blue=229)
GRAY91 = RGB(name='gray91', red=232, green=232, blue=232)
GRAY92 = RGB(name='gray92', red=235, green=235, blue=235)
GRAY93 = RGB(name='gray93', red=237, green=237, blue=237)
GRAY94 = RGB(name='gray94', red=240, green=240, blue=240)
GRAY95 = RGB(name='gray95', red=242, green=242, blue=242)
GRAY97 = RGB(name='gray97', red=247, green=247, blue=247)
GRAY98 = RGB(name='gray98', red=250, green=250, blue=250)
GRAY99 = RGB(name='gray99', red=252, green=252, blue=252)
GREEN = RGB(name='green', red=0, green=128, blue=0)
GREEN1 = RGB(name='green1', red=0, green=255, blue=0)
GREEN2 = RGB(name='green2', red=0, green=238, blue=0)
GREEN3 = RGB(name='green3', red=0, green=205, blue=0)
GREEN4 = RGB(name='green4', red=0, green=139, blue=0)
GREENYELLOW = RGB(name='greenyellow', red=173, green=255, blue=47)
HONEYDEW1 = RGB(name='honeydew1', red=240, green=255, blue=240)
HONEYDEW2 = RGB(name='honeydew2', red=224, green=238, blue=224)
HONEYDEW3 = RGB(name='honeydew3', red=193, green=205, blue=193)
HONEYDEW4 = RGB(name='honeydew4', red=131, green=139, blue=131)
HOTPINK = RGB(name='hotpink', red=255, green=105, blue=180)
HOTPINK1 = RGB(name='hotpink1', red=255, green=110, blue=180)
HOTPINK2 = RGB(name='hotpink2', red=238, green=106, blue=167)
HOTPINK3 = RGB(name='hotpink3', red=205, green=96, blue=144)
HOTPINK4 = RGB(name='hotpink4', red=139, green=58, blue=98)
INDIANRED = RGB(name='indianred', red=176, green=23, blue=31)
INDIANRED = RGB(name='indianred', red=205, green=92, blue=92)
INDIANRED1 = RGB(name='indianred1', red=255, green=106, blue=106)
INDIANRED2 = RGB(name='indianred2', red=238, green=99, blue=99)
INDIANRED3 = RGB(name='indianred3', red=205, green=85, blue=85)
INDIANRED4 = RGB(name='indianred4', red=139, green=58, blue=58)
INDIGO = RGB(name='indigo', red=75, green=0, blue=130)
IVORY1 = RGB(name='ivory1', red=255, green=255, blue=240)
IVORY2 = RGB(name='ivory2', red=238, green=238, blue=224)
IVORY3 = RGB(name='ivory3', red=205, green=205, blue=193)
IVORY4 = RGB(name='ivory4', red=139, green=139, blue=131)
IVORYBLACK = RGB(name='ivoryblack', red=41, green=36, blue=33)
KHAKI = RGB(name='khaki', red=240, green=230, blue=140)
KHAKI1 = RGB(name='khaki1', red=255, green=246, blue=143)
KHAKI2 = RGB(name='khaki2', red=238, green=230, blue=133)
KHAKI3 = RGB(name='khaki3', red=205, green=198, blue=115)
KHAKI4 = RGB(name='khaki4', red=139, green=134, blue=78)
LAVENDER = RGB(name='lavender', red=230, green=230, blue=250)
LAVENDERBLUSH1 = RGB(name='lavenderblush1', red=255, green=240, blue=245)
LAVENDERBLUSH2 = RGB(name='lavenderblush2', red=238, green=224, blue=229)
LAVENDERBLUSH3 = RGB(name='lavenderblush3', red=205, green=193, blue=197)
LAVENDERBLUSH4 = RGB(name='lavenderblush4', red=139, green=131, blue=134)
LAWNGREEN = RGB(name='lawngreen', red=124, green=252, blue=0)
LEMONCHIFFON1 = RGB(name='lemonchiffon1', red=255, green=250, blue=205)
LEMONCHIFFON2 = RGB(name='lemonchiffon2', red=238, green=233, blue=191)
LEMONCHIFFON3 = RGB(name='lemonchiffon3', red=205, green=201, blue=165)
LEMONCHIFFON4 = RGB(name='lemonchiffon4', red=139, green=137, blue=112)
LIGHTBLUE = RGB(name='lightblue', red=173, green=216, blue=230)
LIGHTBLUE1 = RGB(name='lightblue1', red=191, green=239, blue=255)
```

(continues on next page)

(continued from previous page)

```
LIGHTBLUE2 = RGB(name='lightblue2', red=178, green=223, blue=238)
LIGHTBLUE3 = RGB(name='lightblue3', red=154, green=192, blue=205)
LIGHTBLUE4 = RGB(name='lightblue4', red=104, green=131, blue=139)
LIGHTCORAL = RGB(name='lightcoral', red=240, green=128, blue=128)
LIGHTCYAN1 = RGB(name='lightcyan1', red=224, green=255, blue=255)
LIGHTCYAN2 = RGB(name='lightcyan2', red=209, green=238, blue=238)
LIGHTCYAN3 = RGB(name='lightcyan3', red=180, green=205, blue=205)
LIGHTCYAN4 = RGB(name='lightcyan4', red=122, green=139, blue=139)
LIGHTGOLDENROD1 = RGB(name='lightgoldenrod1', red=255, green=236, blue=139)
LIGHTGOLDENROD2 = RGB(name='lightgoldenrod2', red=238, green=220, blue=130)
LIGHTGOLDENROD3 = RGB(name='lightgoldenrod3', red=205, green=190, blue=112)
LIGHTGOLDENROD4 = RGB(name='lightgoldenrod4', red=139, green=129, blue=76)
LIGHTGOLDENRODYELLOW = \
    RGB(name='lightgoldenrodyellow', red=250, green=250, blue=210)
LIGHTGREY = RGB(name='lightgrey', red=211, green=211, blue=211)
LIGHTPINK = RGB(name='lightpink', red=255, green=182, blue=193)
LIGHTPINK1 = RGB(name='lightpink1', red=255, green=174, blue=185)
LIGHTPINK2 = RGB(name='lightpink2', red=238, green=162, blue=173)
LIGHTPINK3 = RGB(name='lightpink3', red=205, green=140, blue=149)
LIGHTPINK4 = RGB(name='lightpink4', red=139, green=95, blue=101)
LIGHTSALMON1 = RGB(name='lightsalmon1', red=255, green=160, blue=122)
LIGHTSALMON2 = RGB(name='lightsalmon2', red=238, green=149, blue=114)
LIGHTSALMON3 = RGB(name='lightsalmon3', red=205, green=129, blue=98)
LIGHTSALMON4 = RGB(name='lightsalmon4', red=139, green=87, blue=66)
LIGHTSEAGREEN = RGB(name='lightseagreen', red=32, green=178, blue=170)
LIGHTSKYBLUE = RGB(name='lightskyblue', red=135, green=206, blue=250)
LIGHTSKYBLUE1 = RGB(name='lightskyblue1', red=176, green=226, blue=255)
LIGHTSKYBLUE2 = RGB(name='lightskyblue2', red=164, green=211, blue=238)
LIGHTSKYBLUE3 = RGB(name='lightskyblue3', red=141, green=182, blue=205)
LIGHTSKYBLUE4 = RGB(name='lightskyblue4', red=96, green=123, blue=139)
LIGHTSLATEBLUE = RGB(name='lightslateblue', red=132, green=112, blue=255)
LIGHTSLATEGRAY = RGB(name='lightslategray', red=119, green=136, blue=153)
LIGHTSTEELBLUE = RGB(name='lightsteelblue', red=176, green=196, blue=222)
LIGHTSTEELBLUE1 = RGB(name='lightsteelblue1', red=202, green=225, blue=255)
LIGHTSTEELBLUE2 = RGB(name='lightsteelblue2', red=188, green=210, blue=238)
LIGHTSTEELBLUE3 = RGB(name='lightsteelblue3', red=162, green=181, blue=205)
LIGHTSTEELBLUE4 = RGB(name='lightsteelblue4', red=110, green=123, blue=139)
LIGHTYELLOW1 = RGB(name='lightyellow1', red=255, green=255, blue=224)
LIGHTYELLOW2 = RGB(name='lightyellow2', red=238, green=238, blue=209)
LIGHTYELLOW3 = RGB(name='lightyellow3', red=205, green=205, blue=180)
LIGHTYELLOW4 = RGB(name='lightyellow4', red=139, green=139, blue=122)
LIMEGREEN = RGB(name='limegreen', red=50, green=205, blue=50)
LINEN = RGB(name='linen', red=250, green=240, blue=230)
MAGENTA = RGB(name='magenta', red=255, green=0, blue=255)
MAGENTA2 = RGB(name='magenta2', red=238, green=0, blue=238)
MAGENTA3 = RGB(name='magenta3', red=205, green=0, blue=205)
MAGENTA4 = RGB(name='magenta4', red=139, green=0, blue=139)
MANGANESEBLUE = RGB(name='manganeseblue', red=3, green=168, blue=158)
MAROON = RGB(name='maroon', red=128, green=0, blue=0)
MAROON1 = RGB(name='maroon1', red=255, green=52, blue=179)
MAROON2 = RGB(name='maroon2', red=238, green=48, blue=167)
MAROON3 = RGB(name='maroon3', red=205, green=41, blue=144)
MAROON4 = RGB(name='maroon4', red=139, green=28, blue=98)
MEDIUMORCHID = RGB(name='mediumorchid', red=186, green=85, blue=211)
MEDIUMORCHID1 = RGB(name='mediumorchid1', red=224, green=102, blue=255)
MEDIUMORCHID2 = RGB(name='mediumorchid2', red=209, green=95, blue=238)
MEDIUMORCHID3 = RGB(name='mediumorchid3', red=180, green=82, blue=205)
```

(continues on next page)

(continued from previous page)

```
MEDIUMORCHID4 = RGB(name='mediumorchid4', red=122, green=55, blue=139)
MEDIUMPURPLE = RGB(name='mediumpurple', red=147, green=112, blue=219)
MEDIUMPURPLE1 = RGB(name='mediumpurple1', red=171, green=130, blue=255)
MEDIUMPURPLE2 = RGB(name='mediumpurple2', red=159, green=121, blue=238)
MEDIUMPURPLE3 = RGB(name='mediumpurple3', red=137, green=104, blue=205)
MEDIUMPURPLE4 = RGB(name='mediumpurple4', red=93, green=71, blue=139)
MEDIUMSEAGREEN = RGB(name='mediumseagreen', red=60, green=179, blue=113)
MEDIUMSLATEBLUE = RGB(name='mediumslateblue', red=123, green=104, blue=238)
MEDIUMSPRINGGREEN = RGB(name='mediumspringgreen', red=0, green=250, blue=154)
MEDIUMTURQUOISE = RGB(name='mediumturquoise', red=72, green=209, blue=204)
MEDIUMVIOLETRED = RGB(name='mediumvioletred', red=199, green=21, blue=133)
MELON = RGB(name='melon', red=227, green=168, blue=105)
MIDNIGHTBLUE = RGB(name='midnightblue', red=25, green=25, blue=112)
MINT = RGB(name='mint', red=189, green=252, blue=201)
MINTCREAM = RGB(name='mintcream', red=245, green=255, blue=250)
MISTYROSE1 = RGB(name='mistyrose1', red=255, green=228, blue=225)
MISTYROSE2 = RGB(name='mistyrose2', red=238, green=213, blue=210)
MISTYROSE3 = RGB(name='mistyrose3', red=205, green=183, blue=181)
MISTYROSE4 = RGB(name='mistyrose4', red=139, green=125, blue=123)
MOCCASIN = RGB(name='moccasin', red=255, green=228, blue=181)
NAVAJOWHITE1 = RGB(name='navajowhite1', red=255, green=222, blue=173)
NAVAJOWHITE2 = RGB(name='navajowhite2', red=238, green=207, blue=161)
NAVAJOWHITE3 = RGB(name='navajowhite3', red=205, green=179, blue=139)
NAVAJOWHITE4 = RGB(name='navajowhite4', red=139, green=121, blue=94)
NAVY = RGB(name='navy', red=0, green=0, blue=128)
OLDLACE = RGB(name='oldlace', red=253, green=245, blue=230)
OLIVE = RGB(name='olive', red=128, green=128, blue=0)
OLIVEDRAB = RGB(name='olivedrab', red=107, green=142, blue=35)
OLIVEDRAB1 = RGB(name='olivedrab1', red=192, green=255, blue=62)
OLIVEDRAB2 = RGB(name='olivedrab2', red=179, green=238, blue=58)
OLIVEDRAB3 = RGB(name='olivedrab3', red=154, green=205, blue=50)
OLIVEDRAB4 = RGB(name='olivedrab4', red=105, green=139, blue=34)
ORANGE = RGB(name='orange', red=255, green=128, blue=0)
ORANGE1 = RGB(name='orange1', red=255, green=165, blue=0)
ORANGE2 = RGB(name='orange2', red=238, green=154, blue=0)
ORANGE3 = RGB(name='orange3', red=205, green=133, blue=0)
ORANGE4 = RGB(name='orange4', red=139, green=90, blue=0)
ORANGERED1 = RGB(name='orangered1', red=255, green=69, blue=0)
ORANGERED2 = RGB(name='orangered2', red=238, green=64, blue=0)
ORANGERED3 = RGB(name='orangered3', red=205, green=55, blue=0)
ORANGERED4 = RGB(name='orangered4', red=139, green=37, blue=0)
ORCHID = RGB(name='orchid', red=218, green=112, blue=214)
ORCHID1 = RGB(name='orchid1', red=255, green=131, blue=250)
ORCHID2 = RGB(name='orchid2', red=238, green=122, blue=233)
ORCHID3 = RGB(name='orchid3', red=205, green=105, blue=201)
ORCHID4 = RGB(name='orchid4', red=139, green=71, blue=137)
PALEGOLDENROD = RGB(name='palegoldenrod', red=238, green=232, blue=170)
PALEGREEN = RGB(name='palegreen', red=152, green=251, blue=152)
PALEGREEN1 = RGB(name='palegreen1', red=154, green=255, blue=154)
PALEGREEN2 = RGB(name='palegreen2', red=144, green=238, blue=144)
PALEGREEN3 = RGB(name='palegreen3', red=124, green=205, blue=124)
PALEGREEN4 = RGB(name='palegreen4', red=84, green=139, blue=84)
PALETURQUOISE1 = RGB(name='paleturquoise1', red=187, green=255, blue=255)
PALETURQUOISE2 = RGB(name='paleturquoise2', red=174, green=238, blue=238)
PALETURQUOISE3 = RGB(name='paleturquoise3', red=150, green=205, blue=205)
PALETURQUOISE4 = RGB(name='paleturquoise4', red=102, green=139, blue=139)
PALEVIOLETRED = RGB(name='palevioletred', red=219, green=112, blue=147)
```

(continues on next page)

(continued from previous page)

```
PALEVIOLETRED1 = RGB(name='palevioletred1', red=255, green=130, blue=171)
PALEVIOLETRED2 = RGB(name='palevioletred2', red=238, green=121, blue=159)
PALEVIOLETRED3 = RGB(name='palevioletred3', red=205, green=104, blue=137)
PALEVIOLETRED4 = RGB(name='palevioletred4', red=139, green=71, blue=93)
PAPAYAWHIP = RGB(name='papayawhip', red=255, green=239, blue=213)
PEACHPUFF1 = RGB(name='peachpuff1', red=255, green=218, blue=185)
PEACHPUFF2 = RGB(name='peachpuff2', red=238, green=203, blue=173)
PEACHPUFF3 = RGB(name='peachpuff3', red=205, green=175, blue=149)
PEACHPUFF4 = RGB(name='peachpuff4', red=139, green=119, blue=101)
PEACOCK = RGB(name='peacock', red=51, green=161, blue=201)
PINK = RGB(name='pink', red=255, green=192, blue=203)
PINK1 = RGB(name='pink1', red=255, green=181, blue=197)
PINK2 = RGB(name='pink2', red=238, green=169, blue=184)
PINK3 = RGB(name='pink3', red=205, green=145, blue=158)
PINK4 = RGB(name='pink4', red=139, green=99, blue=108)
PLUM = RGB(name='plum', red=221, green=160, blue=221)
PLUM1 = RGB(name='plum1', red=255, green=187, blue=255)
PLUM2 = RGB(name='plum2', red=238, green=174, blue=238)
PLUM3 = RGB(name='plum3', red=205, green=150, blue=205)
PLUM4 = RGB(name='plum4', red=139, green=102, blue=139)
POWDERBLUE = RGB(name='powderblue', red=176, green=224, blue=230)
PURPLE = RGB(name='purple', red=128, green=0, blue=128)
PURPLE1 = RGB(name='purple1', red=155, green=48, blue=255)
PURPLE2 = RGB(name='purple2', red=145, green=44, blue=238)
PURPLE3 = RGB(name='purple3', red=125, green=38, blue=205)
PURPLE4 = RGB(name='purple4', red=85, green=26, blue=139)
RASPBERRY = RGB(name='raspberry', red=135, green=38, blue=87)
RAWSIENNA = RGB(name='rawsienna', red=199, green=97, blue=20)
RED1 = RGB(name='red1', red=255, green=0, blue=0)
RED2 = RGB(name='red2', red=238, green=0, blue=0)
RED3 = RGB(name='red3', red=205, green=0, blue=0)
RED4 = RGB(name='red4', red=139, green=0, blue=0)
ROSYBROWN = RGB(name='rosybrown', red=188, green=143, blue=143)
ROSYBROWN1 = RGB(name='rosybrown1', red=255, green=193, blue=193)
ROSYBROWN2 = RGB(name='rosybrown2', red=238, green=180, blue=180)
ROSYBROWN3 = RGB(name='rosybrown3', red=205, green=155, blue=155)
ROSYBROWN4 = RGB(name='rosybrown4', red=139, green=105, blue=105)
ROYALBLUE = RGB(name='royalblue', red=65, green=105, blue=225)
ROYALBLUE1 = RGB(name='royalblue1', red=72, green=118, blue=255)
ROYALBLUE2 = RGB(name='royalblue2', red=67, green=110, blue=238)
ROYALBLUE3 = RGB(name='royalblue3', red=58, green=95, blue=205)
ROYALBLUE4 = RGB(name='royalblue4', red=39, green=64, blue=139)
SALMON = RGB(name='salmon', red=250, green=128, blue=114)
SALMON1 = RGB(name='salmon1', red=255, green=140, blue=105)
SALMON2 = RGB(name='salmon2', red=238, green=130, blue=98)
SALMON3 = RGB(name='salmon3', red=205, green=112, blue=84)
SALMON4 = RGB(name='salmon4', red=139, green=76, blue=57)
SANDYBROWN = RGB(name='sandybrown', red=244, green=164, blue=96)
SAPGREEN = RGB(name='sapgreen', red=48, green=128, blue=20)
SEAGREEN1 = RGB(name='seagreen1', red=84, green=255, blue=159)
SEAGREEN2 = RGB(name='seagreen2', red=78, green=238, blue=148)
SEAGREEN3 = RGB(name='seagreen3', red=67, green=205, blue=128)
SEAGREEN4 = RGB(name='seagreen4', red=46, green=139, blue=87)
SEASHELL1 = RGB(name='seashell1', red=255, green=245, blue=238)
SEASHELL2 = RGB(name='seashell2', red=238, green=229, blue=222)
SEASHELL3 = RGB(name='seashell3', red=205, green=197, blue=191)
SEASHELL4 = RGB(name='seashell4', red=139, green=134, blue=130)
```

(continues on next page)

(continued from previous page)

```
SEPIA = RGB(name='sepia', red=94, green=38, blue=18)
SGIBEET = RGB(name='sgibeet', red=142, green=56, blue=142)
SGIBRIGHTGRAY = RGB(name='sgibrightgray', red=197, green=193, blue=170)
SGICHARTREUSE = RGB(name='sgichartreuse', red=113, green=198, blue=113)
SGIDARKGRAY = RGB(name='sgidarkgray', red=85, green=85, blue=85)
SGIGRAY12 = RGB(name='sgigray12', red=30, green=30, blue=30)
SGIGRAY16 = RGB(name='sgigray16', red=40, green=40, blue=40)
SGIGRAY32 = RGB(name='sgigray32', red=81, green=81, blue=81)
SGIGRAY36 = RGB(name='sgigray36', red=91, green=91, blue=91)
SGIGRAY52 = RGB(name='sgigray52', red=132, green=132, blue=132)
SGIGRAY56 = RGB(name='sgigray56', red=142, green=142, blue=142)
SGIGRAY72 = RGB(name='sgigray72', red=183, green=183, blue=183)
SGIGRAY76 = RGB(name='sgigray76', red=193, green=193, blue=193)
SGIGRAY92 = RGB(name='sgigray92', red=234, green=234, blue=234)
SGIGRAY96 = RGB(name='sgigray96', red=244, green=244, blue=244)
SGILIGHTBLUE = RGB(name='sgilightblue', red=125, green=158, blue=192)
SGILIGHTGRAY = RGB(name='sgilightgray', red=170, green=170, blue=170)
SGIOLIVEDRAB = RGB(name='sgiolivedrab', red=142, green=142, blue=56)
SGISALMON = RGB(name='sgisalmon', red=198, green=113, blue=113)
SGISLATEBLUE = RGB(name='sgislateblue', red=113, green=113, blue=198)
SGITEAL = RGB(name='sgiteal', red=56, green=142, blue=142)
SIENNA = RGB(name='sienna', red=160, green=82, blue=45)
SIENNA1 = RGB(name='sienna1', red=255, green=130, blue=71)
SIENNA2 = RGB(name='sienna2', red=238, green=121, blue=66)
SIENNA3 = RGB(name='sienna3', red=205, green=104, blue=57)
SIENNA4 = RGB(name='sienna4', red=139, green=71, blue=38)
SILVER = RGB(name='silver', red=192, green=192, blue=192)
SKYBLUE = RGB(name='skyblue', red=135, green=206, blue=235)
SKYBLUE1 = RGB(name='skyblue1', red=135, green=206, blue=255)
SKYBLUE2 = RGB(name='skyblue2', red=126, green=192, blue=238)
SKYBLUE3 = RGB(name='skyblue3', red=108, green=166, blue=205)
SKYBLUE4 = RGB(name='skyblue4', red=74, green=112, blue=139)
SLATEBLUE = RGB(name='slateblue', red=106, green=90, blue=205)
SLATEBLUE1 = RGB(name='slateblue1', red=131, green=111, blue=255)
SLATEBLUE2 = RGB(name='slateblue2', red=122, green=103, blue=238)
SLATEBLUE3 = RGB(name='slateblue3', red=105, green=89, blue=205)
SLATEBLUE4 = RGB(name='slateblue4', red=71, green=60, blue=139)
SLATEGRAY = RGB(name='slategray', red=112, green=128, blue=144)
SLATEGRAY1 = RGB(name='slategray1', red=198, green=226, blue=255)
SLATEGRAY2 = RGB(name='slategray2', red=185, green=211, blue=238)
SLATEGRAY3 = RGB(name='slategray3', red=159, green=182, blue=205)
SLATEGRAY4 = RGB(name='slategray4', red=108, green=123, blue=139)
SNOW1 = RGB(name='snow1', red=255, green=250, blue=250)
SNOW2 = RGB(name='snow2', red=238, green=233, blue=233)
SNOW3 = RGB(name='snow3', red=205, green=201, blue=201)
SNOW4 = RGB(name='snow4', red=139, green=137, blue=137)
SPRINGGREEN = RGB(name='springgreen', red=0, green=255, blue=127)
SPRINGGREEN1 = RGB(name='springgreen1', red=0, green=238, blue=118)
SPRINGGREEN2 = RGB(name='springgreen2', red=0, green=205, blue=102)
SPRINGGREEN3 = RGB(name='springgreen3', red=0, green=139, blue=69)
STEELBLUE = RGB(name='steelblue', red=70, green=130, blue=180)
STEELBLUE1 = RGB(name='steelblue1', red=99, green=184, blue=255)
STEELBLUE2 = RGB(name='steelblue2', red=92, green=172, blue=238)
STEELBLUE3 = RGB(name='steelblue3', red=79, green=148, blue=205)
STEELBLUE4 = RGB(name='steelblue4', red=54, green=100, blue=139)
TAN = RGB(name='tan', red=210, green=180, blue=140)
TAN1 = RGB(name='tan1', red=255, green=165, blue=79)
```

(continues on next page)

(continued from previous page)

```
TAN2 = RGB(name='tan2', red=238, green=154, blue=73)
TAN3 = RGB(name='tan3', red=205, green=133, blue=63)
TAN4 = RGB(name='tan4', red=139, green=90, blue=43)
TEAL = RGB(name='teal', red=0, green=128, blue=128)
THISTLE = RGB(name='thistle', red=216, green=191, blue=216)
THISTLE1 = RGB(name='thistle1', red=255, green=225, blue=255)
THISTLE2 = RGB(name='thistle2', red=238, green=210, blue=238)
THISTLE3 = RGB(name='thistle3', red=205, green=181, blue=205)
THISTLE4 = RGB(name='thistle4', red=139, green=123, blue=139)
TOMATO1 = RGB(name='tomato1', red=255, green=99, blue=71)
TOMATO2 = RGB(name='tomato2', red=238, green=92, blue=66)
TOMATO3 = RGB(name='tomato3', red=205, green=79, blue=57)
TOMATO4 = RGB(name='tomato4', red=139, green=54, blue=38)
TURQUOISE = RGB(name='turquoise', red=64, green=224, blue=208)
TURQUOISE1 = RGB(name='turquoise1', red=0, green=245, blue=255)
TURQUOISE2 = RGB(name='turquoise2', red=0, green=229, blue=238)
TURQUOISE3 = RGB(name='turquoise3', red=0, green=197, blue=205)
TURQUOISE4 = RGB(name='turquoise4', red=0, green=134, blue=139)
TURQUOISEBLUE = RGB(name='turquoiseblue', red=0, green=199, blue=140)
VIOLET = RGB(name='violet', red=238, green=130, blue=238)
VIOLETRED = RGB(name='violetred', red=208, green=32, blue=144)
VIOLETRED1 = RGB(name='violetred1', red=255, green=62, blue=150)
VIOLETRED2 = RGB(name='violetred2', red=238, green=58, blue=140)
VIOLETRED3 = RGB(name='violetred3', red=205, green=50, blue=120)
VIOLETRED4 = RGB(name='violetred4', red=139, green=34, blue=82)
WARMGREY = RGB(name='warmgrey', red=128, green=128, blue=105)
WHEAT = RGB(name='wheat', red=245, green=222, blue=179)
WHEAT1 = RGB(name='wheat1', red=255, green=231, blue=186)
WHEAT2 = RGB(name='wheat2', red=238, green=216, blue=174)
WHEAT3 = RGB(name='wheat3', red=205, green=186, blue=150)
WHEAT4 = RGB(name='wheat4', red=139, green=126, blue=102)
WHITE = RGB(name='white', red=255, green=255, blue=255)
WHITESMOKE = RGB(name='whitesmoke', red=245, green=245, blue=245)
WHITESMOKE = RGB(name='whitesmoke', red=245, green=245, blue=245)
YELLOW1 = RGB(name='yellow1', red=255, green=255, blue=0)
YELLOW2 = RGB(name='yellow2', red=238, green=238, blue=0)
YELLOW3 = RGB(name='yellow3', red=205, green=205, blue=0)
YELLOW4 = RGB(name='yellow4', red=139, green=139, blue=0)
```

4.4 pyrevit.coreutils.configparser

Base module for pyRevit config parsing.

class pyrevit.coreutils.configparser.**PyRevitConfigParser**(*cfg_file_path=None*)

Bases: object

Config parser object. Handle config sections and io.

add_section(*section_name*)

Add section with given name to config.

get_config_file_hash()

Get calculated unique hash for this config.

get_section(*section_name*)
Get section with given name.

Raises `AttributeError` – if section is missing

has_section(*section_name*)
Check if config contains given section.

reload(*cfg_file_path=None*)
Reload config from original or given file.

remove_section(*section_name*)
Remove section from config.

save(*cfg_file_path=None*)
Save config to original or given file.

class pyrevit.coreutils.configparser.**PyRevitConfigSectionParser**(*config_parser*, *section_name*)

Bases: `object`

Config section parser object. Handle section options.

add_subsection(*section_name*)
Add subsection to section.

get_option(*op_name*, *default_value=None*)
Get option value or return default.

get_subsection(*section_name*)
Get subsection with given name.

get_subsections()
Get all subsections.

has_option(*option_name*)
Check if section contains given option.

has_subsection(*section_name*)
Check if section has any subsections.

header
Section header.

remove_option(*option_name*)
Remove given option from section.

set_option(*op_name*, *value*)
Set value of given option.

subheader
Section sub-header e.g. Section.SubSection

4.5 pyrevit.coreutils.envvars

pyRevit managed environment variables framework.

pyRevit provides the environment variables framework to the pyRevit core and all pyRevit tools so they can store arbitary data withing the running host session and share small data quickly between script runs.

Some settings needs to be set for the current session and might need to affect the behaviour of all individual scripts inside the extensions. (e.g. If user activates the DEBUG mode, all scripts should follow and log the debug entries.) The information is saved using AppDomain.GetData and SetData in a dictionary parameter. The dictionary is used to minimize the addition of named parameters to the AppDomain. The dictionary then includes all the internal parameters and their associated value. This way each script does not need to read the usersettings data which reduces io and saves time.

pyRevit uses environment variables extensively at its core and making changes to the core environment variables (starting with PYREVIT_) through scripts is strongly prohibited.

Example

```
>>> from pyrevit.coreutils import envvars
>>> envvars.set_pyrevit_env_var('MY_SCRIPT_STATUS', True)
>>> envvars.set_pyrevit_env_var('MY_SCRIPT_CONFIG', {'someconfig': True})
```

Then another script or same script when executed later within the same session can query the shared environment variable:

```
>>> envvars.get_pyrevit_env_vars('MY_SCRIPT_STATUS')
True
>>> envvars.get_pyrevit_env_vars('MY_SCRIPT_CONFIG')
{'someconfig': True}
```

pyrevit.coreutils.envvars.**get_pyrevit_env_var**(*param_name*)
Get value of a parameter shared between all scripts.

Parameters **param_name** (*str*) – name of environment variable

Returns any object stored as the environment variable value

Return type object

pyrevit.coreutils.envvars.**get_pyrevit_env_vars**()
Get the root dictionary, holding all environment variables.

pyrevit.coreutils.envvars.**set_pyrevit_env_var**(*param_name*, *param_value*)
Set value of a parameter shared between all scripts.

Parameters

- **param_name** (*str*) – name of environment variable
- **param_value** (*object*) – any python object

4.6 pyrevit.coreutils.git

Description: LibGit2Sharp wrapper module for pyRevit.

Documentation: https://github.com/libgit2/libgit2sharp/wiki

exception pyrevit.coreutils.git.**PyRevitGitAuthenticationError**
Bases: *pyrevit.PyRevitException*

Git authentication error.

class pyrevit.coreutils.git.**RepoInfo**(*repo*)
Bases: object

Repo wrapper for passing around repository information.

directory
repo directory

Type str

name
repo name

Type str

head_name
head branch name

Type str

last_commit_hash
hash of head commit

Type str

repo
`LibGit2Sharp.Repository` object

Type str

branch
current branch name

Type str

username
credentials - username

Type str

password
credentials - password

Type str

pyrevit.coreutils.git.**compare_branch_heads**(*repo_info*)
Compare local and remote branch heads and return ???

Parameters **repo_info** (*RepoInfo*) – target repo object

Returns desc

Return type type

pyrevit.coreutils.git.**get_all_new_commits**(*repo_info*)
Fetch and return new commits ahead of current head.

Parameters **repo_info** (*RepoInfo*) – target repo object

Returns str]: ordered dict of commit hash:message

Return type OrderedDict[str

pyrevit.coreutils.git.**get_repo**(*repo_dir*)
Return repo object for given git repo directory.

Parameters **repo_dir** (*str*) – full path of git repo directory

Returns repo object

Return type *RepoInfo*

pyrevit.coreutils.git.**git_clone**(*repo_url*, *clone_dir*, *username=None*, *password=None*)
Clone git repository to given location

Parameters

- **repo_url** (*str*) – repo .git url
- **clone_dir** (*str*) – destination path
- **username** (*str*) – credentials - username
- **password** (*str*) – credentials - password

pyrevit.coreutils.git.**git_fetch**(*repo_info*)
Fetch current branch of given repo.

Parameters **repo_info** (*RepoInfo*) – target repo object

Returns repo object with updated head

Return type *RepoInfo*

pyrevit.coreutils.git.**git_pull**(*repo_info*)
Pull the current head of given repo.

Parameters **repo_info** (*RepoInfo*) – target repo object

Returns repo object with updated head

Return type *RepoInfo*

4.7 pyrevit.coreutils.logger

Core logging module for pyRevit.

class pyrevit.coreutils.logger.**DispatchingFormatter**(*log_formatters*, *log_default_formatter*)
Bases: object

Dispatching formatter to format by log level.

Parameters

- **(dict[int** (*log_formatters*) – logging.Formatter]): dict of level:formatter key pairs
- **log_default_formatter** (*logging.Formatter*) – default formatter

format(*record*)
Format given record by log level.

class pyrevit.coreutils.logger.**LoggerWrapper**(**args*)
Bases: logging.Logger

Custom logging object.

Parameters

- **val** (*type*) – desc
- **val** – desc

callHandlers(*record*)
Override logging.Logger.callHandlers

get_level()
Return current logging level.

has_errors()
Check if logger has reported any errors.

isEnabledFor(*level*)
Override logging.Logger.isEnabledFor

is_enabled_for(*level*)
Check if logger is enabled for level in pyRevit environment.

reset_level()
Reset logging level back to default.

set_debug_mode()
Activate debug mode. Log levels >= DEBUG are enabled.

set_level(*level*)
Set logging level to level.

set_quiet_mode()
Activate quiet mode. All log levels are disabled.

set_verbose_mode()
Activate verbose mode. Log levels >= INFO are enabled.

pyrevit.coreutils.logger.**get_file_hndlr**()
Return file logging handler object.

Returns configured instance of python's native stream handler

Return type logging.FileHandler

pyrevit.coreutils.logger.**get_logger**(*logger_name*)
Register and return a logger with given name.

Caches all registered loggers and returns the same logger object on second call with the same logger name.

Parameters

- **logger_name** (*str*) – logger name
- **val** (*type*) – desc

Returns logger object wrapper python's native logger

Return type *LoggerWrapper*

Example

```
>>> get_logger('my command')
... <LoggerWrapper ...>
```

pyrevit.coreutils.logger.**get_stdout_hndlr**()
Return stdout logging handler object.

Returns configured instance of python's native stream handler

Return type logging.StreamHandler

pyrevit.coreutils.logger.**loggers_have_errors**()
Check if any errors have been reported by any of registered loggers.

pyrevit.coreutils.logger.**set_file_logging**(*status*)
Set file logging status (enable/disable).

Parameters **status** (*bool*) – True to enable, False to disable

4.8 pyrevit.coreutils.mathnet

MathNet importer module.

See https://www.mathdotnet.com for documentation.

Example

```
>>> from pyrevit.coreutils.mathnet import MathNet
```

4.9 pyrevit.coreutils.moduleutils

Utility fuctions to support smart modules.

pyrevit.coreutils.moduleutils.**collect_marked**(*module_obj*, *prop_name*)
Collect module objects that are marked with given property

pyrevit.coreutils.moduleutils.**copy_func**(*func*, *func_name*, *doc_string=None*, *arg_list=None*)
Copy a function object to create a new function.

This is used inside smart modules that auto-generate functions based on context.

Parameters

- **func** (*object*) – python source function object
- **func_name** (*str*) – new function name
- **doc_string** (*str*) – new function docstring
- **arg_list** (*list*) – list of default values for function arguments

Returns new python function objects

Return type object

pyrevit.coreutils.moduleutils.**filter_kwargs**(*function_obj*, *kwargs*)
Filter given arguments dict for function_obj arguments

pyrevit.coreutils.moduleutils.**has_any_arguments**(*function_obj*, *arg_name_list*)
Check if given function object has any of given arguments

pyrevit.coreutils.moduleutils.**has_argument**(*function_obj*, *arg_name*)
Check if given function object has argument matching arg_name

pyrevit.coreutils.moduleutils.**mark**(*prop_name*)
Decorator function to add a marker property to the given type

4.10 pyrevit.coreutils.pyutils

Helper functions for python.

Example

```
>>> from pyrevit.coreutils import pyutils
>>> pyutils.safe_cast('string', int, 0)
```

class pyrevit.coreutils.pyutils.**DefaultOrderedDict**(*default_factory=None*, **a*, ***kw*)

Bases: collections.OrderedDict

Ordered dictionary with default type.

This is similar to defaultdict and maintains the order of items added to it so in that regards it functions similar to OrderedDict.

Example

```
>>> from pyrevit.coreutils import pyutils
>>> od = pyutils.DefaultOrderedDict(list)
>>> od['A'] = [1, 2, 3]
>>> od['B'] = [4, 5, 6]
>>> od['C'].extend([7, 8, 9])
>>> for k, v in od.items():
...     print(k, v)
('A', [1, 2, 3])
('B', [4, 5, 6])
('C', [7, 8, 9])
```

copy()

Copy the dictionary.

pyrevit.coreutils.pyutils.**almost_equal**(*a*, *b*, *rnd=5*)

Check if two numerical values almost equal

Parameters

- **a** (*float*) – value a
- **b** (*float*) – value b
- **rnd** (*int, optional*) – n digits after comma. Defaults to 5.

Returns True if almost equal

Return type bool

pyrevit.coreutils.pyutils.**compare_lists**(*x*, *y*)

Compare two lists.

See: https://stackoverflow.com/a/10872313/2350244

Parameters

- **x** (*list*) – first list
- **y** (*list*) – second list

pyrevit.coreutils.pyutils.**isnumber**(*token*)

Verify if given string token is int or float.

Parameters **token** (*str*) – string value

Returns True of token is int or float

Return type bool

Example

```
>>> isnumber('12.3')
True
```

pyrevit.coreutils.pyutils.**merge**(*d1*, *d2*)

Merge d2 into d1.

d2 dict values are recursively merged into d1 dict values other d2 values are added to d1 dict values with the same key new d2 values are added to d1 d2 values override other d1 values

Parameters

- **d1** (*dict*) – dict to be updated
- **d2** (*dict*) – dict to be merge into d1

Returns updated d1

Return type dict

Example

```
>>> d1 = {1: 1, 2: "B"    , 3: {1:"A", 2:"B"}, 4: "b"  , 5: ["a", "b"]}
>>> d2 = {1: 1, 2: {1:"A"}, 3: {1:"S", 3:"C"}, 4: ["a"], 5: ["c"]}
>>> merge(d1, d2)
... { 1:1,
...   2:{1:'A', 2:'B'},
...   3:{1:'S', 2:'B', 3:'C'},
...   4:['a','b'],
...   5: ['c', 'a', 'b']
... }
```

pyrevit.coreutils.pyutils.**pairwise**(*iterable*, *step=2*)

Iterate through items in pairs.

Parameters

- **iterable** (*iterable*) – any iterable object
- **step** (*int*) – number of steps to move when making pairs

Returns list of pairs

Return type iterable

Example

```
>>> pairwise([1, 2, 3, 4, 5])
[(1, 2), (3, 4)]    # 5 can not be paired
>>> pairwise([1, 2, 3, 4, 5, 6])
[(1, 2), (3, 4), (5, 6)]
>>> pairwise([1, 2, 3, 4, 5, 6], step=1)
[(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]
```

pyrevit.coreutils.pyutils.**safe_cast**(*val*, *to_type*, *default=None*)

Convert value to type gracefully.

This method basically calls to_type(value) and returns the default if exception occurs.

Parameters

- **val** (*any*) – value to be converted
- **to_type** (*type*) – target type
- **default** (*any*) – value to rerun on conversion exception

Example

```
>>> safe_cast('name', int, default=0)
0
```

4.11 pyrevit.coreutils.ribbon

Base module to interact with Revit ribbon.

class pyrevit.coreutils.ribbon.**ButtonIcons**(*image_file*)

Bases: object

pyRevit ui element icon.

Upon init, this type reads the given image file into an io stream and releases the os lock on the file.

Parameters **image_file** (*str*) – image file path to be used as icon

icon_file_path

icon image file path

Type str

filestream

io stream containing image binary data

Type IO.FileStream

check_icon_size()

Verify icon size is within acceptable range.

create_bitmap(*icon_size*)

Resamples image and creates bitmap for the given size.

Icons are assumed to be square.

Parameters **icon_size** (*int*) – icon size (width or height)

Returns object containing image data at given size

Return type Imaging.BitmapSource

large_bitmap
Resamples image and creates bitmap for size ICON_LARGE.

Returns object containing image data at given size

Return type Imaging.BitmapSource

medium_bitmap
Resamples image and creates bitmap for size ICON_MEDIUM.

Returns object containing image data at given size

Return type Imaging.BitmapSource

small_bitmap
Resamples image and creates bitmap for size ICON_SMALL.

Returns object containing image data at given size

Return type Imaging.BitmapSource

class pyrevit.coreutils.ribbon.**GenericPyRevitUIContainer**
Bases: object

Common type for all pyRevit ui containers.

name
container name

Type str

itemdata_mode
if container is wrapping UI.*ItemData

Type bool

activate()
Activate this container in ui.

contains(*pyrvt_cmp_name*)
Check if container contains a component with given name.

Parameters

- **pyrvt_cmp_name** (*str*) – target component name
- **val** (*type*) – desc

deactivate()
Deactivate this container in ui.

enabled
Is container enabled.

find_child(*child_name*)
Find a component with given name in children.

Parameters **child_name** (*str*) – target component name

Returns component object if found, otherwise None

Return type

-

get_adwindows_object()
Return underlying AdWindows API object for this container.

get_flagged_children(*state=True*)
Get all children with their flag equal to given state.

Flagging is a mechanism to mark certain containers. There are various reasons that container flagging might be used e.g. marking updated containers or the ones in need of an update or removal.

Parameters **state** (*bool*) – flag state to filter children

Returns list of filtered child objects

Return type list[*]

get_rvtapi_object()
Return underlying Revit API object for this container.

is_dirty()
Is dirty flag set.

static is_native()
Is this container generated by pyRevit or is native.

reorder_after(*item_name*, *ritem_name*)
Reorder and place item_name after ritem_name

Parameters

- **item_name** (*str*) – name of component to be moved
- **ritem_name** (*str*) – name of component that should be on the left

reorder_afterall(*item_name*)
Reorder and place item_name after all others.

Parameters **item_name** (*str*) – name of component to be moved

reorder_before(*item_name*, *ritem_name*)
Reorder and place item_name before ritem_name

Parameters

- **item_name** (*str*) – name of component to be moved
- **ritem_name** (*str*) – name of component that should be on the right

reorder_beforeall(*item_name*)
Reorder and place item_name before all others.

Parameters **item_name** (*str*) – name of component to be moved

set_dirty_flag(*state=True*)
Set dirty flag to given state.

See .get_flagged_children()

Parameters **state** (*bool*) – state to set flag

set_rvtapi_object(*rvtapi_obj*)
Set underlying Revit API object for this container.

Parameters **rvtapi_obj** (*obj*) – Revit API container object

visible
Is container visible.

class pyrevit.coreutils.ribbon.**GenericRevitNativeUIContainer**
Bases: *pyrevit.coreutils.ribbon.GenericPyRevitUIContainer*

Common base type for native Revit API UI containers.

activate()
Activate this container in ui.

Under current implementation, raises PyRevitUIError exception as native Revit API UI components should not be changed.

deactivate()
Deactivate this container in ui.

Under current implementation, raises PyRevitUIError exception as native Revit API UI components should not be changed.

static is_native()
Is this container generated by pyRevit or is native.

exception pyrevit.coreutils.ribbon.**PyRevitUIError**
Bases: *pyrevit.PyRevitException*

Common base class for all pyRevit ui-related exceptions.

class pyrevit.coreutils.ribbon.**RevitNativeRibbonButton**(*adwnd_ribbon_button*)
Bases: *pyrevit.coreutils.ribbon.GenericRevitNativeUIContainer*

Revit API UI native ribbon button.

class pyrevit.coreutils.ribbon.**RevitNativeRibbonGroupItem**(*adwnd_ribbon_item*)
Bases: *pyrevit.coreutils.ribbon.GenericRevitNativeUIContainer*

Revit API UI native ribbon button.

button(*name*)
Get button item with given name.

Parameters **name** (*str*) – name of button item to find

Returns button object if found

Return type *RevitNativeRibbonButton*

class pyrevit.coreutils.ribbon.**RevitNativeRibbonPanel**(*adwnd_ribbon_panel*)
Bases: *pyrevit.coreutils.ribbon.GenericRevitNativeUIContainer*

Revit API UI native ribbon button.

ribbon_item(*item_name*)
Get panel item with given name.

Parameters **item_name** (*str*) – name of panel item to find

Returns panel item if found, could be *RevitNativeRibbonButton* or *RevitNativeRibbonGroupItem*

Return type object

class pyrevit.coreutils.ribbon.**RevitNativeRibbonTab**(*adwnd_ribbon_tab*)
Bases: *pyrevit.coreutils.ribbon.GenericRevitNativeUIContainer*

Revit API UI native ribbon tab.

static is_pyrevit_tab()
Is this tab generated by pyRevit.

ribbon_panel (*panel_name*)
Get panel with given name.

Parameters **panel_name** (*str*) – name of panel to find

Returns panel if found

Return type *RevitNativeRibbonPanel*

pyrevit.coreutils.ribbon.**get_current_ui** (*all_native=False*)
Revit UI Wrapper class for interacting with current pyRevit UI.

Returned class provides min required functionality for user interaction

Example

```
>>> current_ui = pyrevit.session.current_ui()
>>> this_script = pyrevit.session.get_this_command()
>>> current_ui.update_button_icon(this_script, new_icon)
```

Returns wrapper around active ribbon gui

Return type _PyRevitUI

pyrevit.coreutils.ribbon.**get_uibutton** (*command_unique_name*)
Find and return ribbon ui button with given unique id.

Parameters **command_unique_name** (*str*) – unique id of pyRevit command

Returns ui button wrapper object

Return type _PyRevitRibbonButton

pyrevit.coreutils.ribbon.**load_bitmapimage** (*image_file*)
Load given png file.

Parameters **image_file** (*str*) – image file path

Returns bitmap image object

Return type Imaging.BitmapImage

Misc Helper functions for pyRevit.

Example

```
>>> from pyrevit import coreutils
>>> coreutils.cleanup_string('some string')
```

class pyrevit.coreutils.**FileWatcher** (*filepath*)
Bases: object

Simple file version watcher.

This is a simple utility class to look for changes in a file based on its timestamp.

Example

```
>>> watcher = FileWatcher('/path/to/file.ext')
>>> watcher.has_changed
True
```

has_changed
Compare current file timestamp to the cached timestamp.

update_tstamp()
Update the cached timestamp for later comparison.

class pyrevit.coreutils.**SafeDict**
Bases: dict

Dictionary that does not fail on any key.

This is a dictionary subclass to help with string formatting with unknown key values.

Example

```
>>> string = '{target} {attr} is {color}.'
>>> safedict = SafeDict({'target': 'Apple',
...                      'attr':   'Color'})
>>> string.format(safedict)  # will not fail with missing 'color' key
'Apple Color is {color}.'
```

class pyrevit.coreutils.**ScriptFileParser**(*file_address*)
Bases: object

Parse python script to extract variables and docstrings.

Primarily designed to assist pyRevit in determining script configurations but can work for any python script.

Example

```
>>> finder = ScriptFileParser('/path/to/coreutils/__init__.py')
>>> finder.docstring()
... "Misc Helper functions for pyRevit."
>>> finder.extract_param('SomeValue', [])
[]
```

extract_node_value(*node*)
Manual extraction of values from node

extract_param(*param_name*, *default_value=None*)
Find variable and extract its value.

Parameters

- **param_name** (*str*) – variable name
- **default_value** (*any*) – default value to be returned if variable does not exist

Returns value of the variable or None

Return type any

get_docstring()
Get global docstring.

class pyrevit.coreutils.**Timer**
Bases: object

Timer class using python native time module.

Example

```
>>> timer = Timer()
>>> timer.get_time()
12
```

get_time()
Get Elapsed Time.

restart()
Restart Timer.

pyrevit.coreutils.**calculate_dir_hash**(*dir_path*, *dir_filter*, *file_filter*)
Create a unique hash to represent state of directory.

Parameters

- **dir_path** (*str*) – target directory
- **dir_filter** (*str*) – exclude directories matching this regex
- **file_filter** (*str*) – exclude files matching this regex

Returns hash value as string

Return type str

Example

```
>>> calculate_dir_hash(source_path, '\.extension', '\.json')
"1a885a0cae99f53d6088b9f7cee3bf4d"
```

pyrevit.coreutils.**can_access_url**(*url_to_open*, *timeout=1000*)
Check if url is accessible within timeout.

Parameters

- **url_to_open** (*str*) – url to check access for
- **timeout** (*int*) – timeout in milliseconds

Returns true if accessible

Return type bool

pyrevit.coreutils.**check_internet_connection**(*timeout=1000*)
Check if internet connection is available.

Pings a few well-known websites to check if internet connection is present.

Parameters **timeout** (*int*) – timeout in milliseconds

Returns url if internet connection is present, None if no internet.

pyrevit.coreutils.**check_revittxt_encoding**(*filename*)
Check if given file is in UTF-16 (UCS-2 LE) encoding.

Parameters **filename** (*str*) – file path

pyrevit.coreutils.**check_utf8bom_encoding**(*filename*)
Check if given file is in UTF-8 encoding.

Parameters **filename** (*str*) – file path

pyrevit.coreutils.**cleanup_filename**(*file_name*, *windows_safe=False*)
Cleanup file name from special characters.

Parameters **file_name** (*str*) – file name

Returns cleaned up file name

Return type str

Example

```
>>> cleanup_filename('Myfile-(3).txt')
"Myfile(3).txt"
```

```
>>> cleanup_filename('Perforations 1/8" (New)')
"Perforations 18 (New).txt"
```

pyrevit.coreutils.**cleanup_string**(*input_str*, *skip=None*)
Replace special characters in string with another string.

This function was created to help cleanup pyRevit command unique names from any special characters so C# class names can be created based on those unique names.

`coreutils.SPECIAL_CHARS` is the conversion table for this function.

Parameters **input_str** (*str*) – input string to be cleaned

Example

```
>>> src_str = 'TEST@Some*<value>'
>>> cleanup_string(src_str)
"TESTATSomeSTARvalue"
```

pyrevit.coreutils.**correct_revittxt_encoding**(*filename*)
Convert encoding of text file generated by Revit to UTF-8.

Parameters **filename** (*str*) – file path

pyrevit.coreutils.**current_date**()
Return formatted current date.

Current implementation uses %Y-%m-%d to format date.

Returns formatted current date.

Return type str

Example

```
>>> current_date()
'2018-01-03'
```

pyrevit.coreutils.**current_time**()
Return formatted current time.

Current implementation uses %H:%M:%S to format time.

Returns formatted current time.

Return type str

Example

```
>>> current_time()
'07:50:53'
```

pyrevit.coreutils.**decrement_str**(*input_str*, *step=1*, *shrink=False*)
Decrement identifier.

Parameters

- **input_str** (*str*) – identifier e.g. A310a
- **step** (*int*) – number of steps to change the identifier

Returns modified identifier

Return type str

Example

```
>>> decrement_str('A310a')
'A309z'
```

pyrevit.coreutils.**dletter_to_unc**(*dletter_path*)
Convert drive letter path into UNC path of that drive.

Parameters **dletter_path** (*str*) – drive letter path

Returns UNC path

Return type str

Example

```
>>> # assuming J: is mapped to //filestore/server/jdrive
>>> dletter_to_unc('J:/somefile.txt')
'//filestore/server/jdrive/somefile.txt'
```

pyrevit.coreutils.**extend_counter**(*input_str*, *upper=True*, *use_zero=False*)
Add a new level to identifier. e.g. A310 -> A310A

Parameters

- **input_str** (*str*) – identifier e.g. A310

- **upper** (*bool*) – use UPPERCASE characters for extension
- **use_zero** (*bool*) – start from 0 for numeric extension

Returns extended identifier

Return type str

Example

```
>>> extend_counter('A310')
'A310A'
>>> extend_counter('A310A', use_zero=True)
'A310A0'
```

pyrevit.coreutils.**extract_guid**(*source_str*)
Extract GUID number from a string.

pyrevit.coreutils.**extract_range**(*formatted_str*, *max_range=500*)
Extract range from formatted string.

String must be formatted as below A103 No range A103-A106 A103 to A106 A103:A106 A103 to A106 A103,A105a A103 and A105a A103;A105a A103 and A105a

Parameters **formatted_str** (*str*) – string specifying range

Returns list of names in the specified range

Return type list

Example

```
>>> exract_range('A103:A106')
['A103', 'A104', 'A105', 'A106']
>>> exract_range('S203-S206')
['S203', 'S204', 'S205', 'S206']
>>> exract_range('M00A,M00B')
['M00A', 'M00B']
```

pyrevit.coreutils.**filter_null_items**(*src_list*)
Remove None items in the given list.

Parameters **src_list** (list) – list of any items

Returns cleaned list

Return type list

pyrevit.coreutils.**format_hex_rgb**(*rgb_value*)
Formats rgb value as #RGB value string.

pyrevit.coreutils.**fully_remove_dir**(*dir_path*)
Remove directory recursively.

Parameters **dir_path** (*str*) – directory path

pyrevit.coreutils.**fuzzy_search_ratio**(*target_string*, *sfilter*, *regex=False*)
Match target string against the filter and return a match ratio.

Parameters

- **target_string** (*str*) – target string
- **sfilter** (*str*) – search term
- **regex** (*bool*) – treat the sfilter as regular expression pattern

Returns integer between 0 to 100, with 100 being the exact match

Return type int

pyrevit.coreutils.**get_all_subclasses**(*parent_classes*)
Return all subclasses of a python class.

Parameters **parent_classes** (*list*) – list of python classes

Returns list of python subclasses

Return type list

pyrevit.coreutils.**get_canonical_parts**(*canonical_string*)
Splots argument using dot, returning all composing parts.

Parameters **canonical_string** (str) – Source string e.g. "Config.SubConfig"

Returns list of composing parts

Return type list[str]

Example

```
>>> get_canonical_parts("Config.SubConfig")
['Config', 'SubConfig']
```

pyrevit.coreutils.**get_enum_none**(*enum_type*)
Returns the None value in given Enum.

pyrevit.coreutils.**get_enum_value**(*enum_type*, *value_string*)
Return enum value matching given value string (case insensitive)

pyrevit.coreutils.**get_enum_values**(*enum_type*)
Returns enum values.

pyrevit.coreutils.**get_exe_version**(*exepath*)
Extract Product Version value from EXE file.

pyrevit.coreutils.**get_file_name**(*file_path*)
Return file basename of the given file.

Parameters **file_path** (*str*) – file path

pyrevit.coreutils.**get_integer_length**(*number*)
Return digit length of given number.

pyrevit.coreutils.**get_mapped_drives_dict**()
Return a dictionary of currently mapped network drives.

pyrevit.coreutils.**get_my_ip**()
Return local ip address of this machine

pyrevit.coreutils.**get_paper_sizes**(*printer_name=None*)
Get paper sizes defined on this system

Returns list of papersize instances

Return type list[]

pyrevit.coreutils.**get_reg_key**(*key*, *subkey*)
Get value of the given Windows registry key and subkey.

Parameters

- **key** (*PyHKEY*) – parent registry key
- **subkey** (*str*) – subkey path

Returns registry key if found, None if not found

Return type PyHKEY

Example

```
>>> get_reg_key(wr.HKEY_CURRENT_USER, 'Control Panel/International')
... <PyHKEY at 0x...>
```

pyrevit.coreutils.**get_revit_instance_count**()
Return number of open host app instances.

Returns number of open host app instances.

Return type int

pyrevit.coreutils.**get_str_hash**(*source_str*)
Calculate hash value of given string.

Current implementation uses `hashlib.md5()` hash function.

Parameters **source_str** (*str*) – source str

Returns hash value as string

Return type str

pyrevit.coreutils.**get_sub_folders**(*search_folder*)
Get a list of all subfolders directly inside provided folder.

Parameters **search_folder** (*str*) – folder path

Returns list of subfolder names

Return type list

pyrevit.coreutils.**has_nonprintable**(*input_str*)
Check input string for non-printable characters.

Parameters **input_str** (*str*) – input string

Returns True if contains non-printable characters

Return type bool

pyrevit.coreutils.**hex2int_long**(*hex_string*)
Hexadecimal string to Integer.

pyrevit.coreutils.**increment_str**(*input_str*, *step=1*, *expand=False*)
Incremenet identifier.

Parameters

- **input_str** (*str*) – identifier e.g. A310a
- **step** (*int*) – number of steps to change the identifier

Returns modified identifier

Return type str

Example

```
>>> increment_str('A319z')
'A320a'
```

pyrevit.coreutils.**inspect_calling_scope_global_var**(*variable_name*)
Trace back the stack to find the variable in the caller global stack.

Parameters **variable_name** (*str*) – variable name to look up in caller global scope

pyrevit.coreutils.**inspect_calling_scope_local_var**(*variable_name*)
Trace back the stack to find the variable in the caller local stack.

PyRevitLoader defines __revit__ in builtins and __window__ in locals. Thus, modules have access to __revit__ but not to __window__. This function is used to find __window__ in the caller stack.

Parameters **variable_name** (*str*) – variable name to look up in caller local scope

pyrevit.coreutils.**int2hex_long**(*number*)
Integer to hexadecimal string.

pyrevit.coreutils.**is_blank**(*input_string*)
Check if input string is blank (multiple white spaces is blank).

Parameters **input_string** (*str*) – input string

Returns True if string is blank

Return type bool

Example

```
>>> is_blank('    ')
True
```

pyrevit.coreutils.**is_box_visible_on_screens**(*left*, *top*, *width*, *height*)
Check if given box is visible on any screen.

pyrevit.coreutils.**is_url_valid**(*url_string*)
Check if given URL is in valid format.

Parameters **url_string** (*str*) – URL string

Returns True if URL is in valid format

Return type bool

Example

```
>>> is_url_valid('https://www.google.com')
True
```

pyrevit.coreutils.**join_strings**(*str_list*, *separator=';'*)
Join strings using provided separator.

Parameters

- **str_list** (*list*) – list of string values
- **separator** (*str*) – single separator character, defaults to DEFAULT_SEPARATOR

Returns joined string

Return type str

pyrevit.coreutils.**kill_tasks**(*task_name*)
Kill running tasks matching task_name

Parameters **task_name** (*str*) – task name

Example

```
>>> kill_tasks('Revit.exe')
```

pyrevit.coreutils.**make_canonical_name**(**args*)
Join arguments with dot creating a unique id.

Parameters ***args** – Variable length argument list of type str

Returns dot separated unique name

Return type str

Example

```
>>> make_canonical_name('somename', 'someid', 'txt')
"somename.someid.txt"
```

pyrevit.coreutils.**new_uuid**()
Create a new UUID (using dotnet Guid.NewGuid)

pyrevit.coreutils.**open_folder_in_explorer**(*folder_path*)
Open given folder in Windows Explorer.

Parameters **folder_path** (*str*) – directory path

pyrevit.coreutils.**prepare_html_str**(*input_string*)
Reformat html string and prepare for pyRevit output window.

pyRevit output window renders html content. But this means that < and > characters in outputs from python (e.g. <class at xxx>) will be treated as html tags. To avoid this, all <> characters that are defining html content need to be replaced with special phrases. pyRevit output later translates these phrases back in to < and >. That is how pyRevit distinquishes between <> printed from python and <> that define html.

Parameters **input_string** (*str*) – input html string

Example

```
>>> prepare_html_str('<p>Some text</p>')
"&clt;p&cgt;Some text&clt;/p&cgt;"
```

pyrevit.coreutils.**random_alpha**()
Return a random alpha value (between 0 and 1.00).

pyrevit.coreutils.**random_color**()
Return a random color channel value (between 0 and 255).

pyrevit.coreutils.**random_hex_color**()
Return a random color in hex format.

Example

```
>>> random_hex_color()
'#FF0000'
```

pyrevit.coreutils.**random_rgb_color**()
Return a random color in rgb format.

Example

```
>>> random_rgb_color()
'rgb(255, 0, 0)'
```

pyrevit.coreutils.**random_rgba_color**()
Return a random color in rgba format.

Example

```
>>> random_rgba_color()
'rgba(255, 0, 0, 0.5)'
```

pyrevit.coreutils.**read_source_file**(*source_file_path*)
Read text file and return contents.

Parameters **source_file_path** (*str*) – target file path

Returns file contents

Return type str

Raises PyRevitException on read error

pyrevit.coreutils.**read_url**(*url_to_open*)
Get the url and return response.

Parameters **url_to_open** (*str*) – url to check access for

pyrevit.coreutils.**reformat_string**(*orig_str*, *orig_format*, *new_format*)
Reformat a string into a new format.

Extracts information from a string based on a given pattern, and recreates a new string based on the given new pattern.

Parameters

- **orig_str** (*str*) – Original string to be reformatted
- **orig_format** (*str*) – Pattern of the original str (data to be extracted)
- **new_format** (*str*) – New pattern (how to recompose the data)

Returns Reformatted string

Return type str

Example

```
>>> reformat_string('150 - FLOOR/CEILING - WD - 1 HR - FLOOR ASSEMBLY',
                    '{section} - {loc} - {mat} - {rating} - {name}',
                    '{section}:{mat}:{rating} - {name} ({loc})'))
'150:WD:1 HR - FLOOR ASSEMBLY (FLOOR/CEILING)'
```

pyrevit.coreutils.**reverse_dict**(*input_dict*)

Reverse the key, value pairs.

Parameters **input_dict** (dict) – source ordered dict

Returns reversed dictionary

Return type defaultdict

Example

```
>>> reverse_dict({1: 2, 3: 4})
defaultdict(<type 'list'>, {2: [1], 4: [3]})
```

pyrevit.coreutils.**reverse_html**(*input_html*)

Reformat codified pyRevit output html string back to normal html.

pyRevit output window renders html content. But this means that < and > characters in outputs from python (e.g. <class at xxx>) will be treated as html tags. To avoid this, all <> characters that are defining html content need to be replaced with special phrases. pyRevit output later translates these phrases back in to < and >. That is how pyRevit distinquishes between <> printed from python and <> that define html.

Parameters **input_html** (*str*) – input codified html string

Example

```
>>> prepare_html_str('&clt;p&cgt;Some text&clt;/p&cgt;')
"<p>Some text</p>"
```

pyrevit.coreutils.**run_process**(*proc*, *cwd='C:'*)

Run shell process silently.

Parameters

- **proc** (*str*) – process executive name
- **cwd** (*str*) – current working directory

Exmaple:

```
>>> run_process('notepad.exe', 'c:/')
```

pyrevit.coreutils.**show_entry_in_explorer**(*entry_path*)

Show given entry in Windows Explorer.

Parameters **entry_path** (*str*) – directory or file path

pyrevit.coreutils.**split_words**(*input_string*)
Splits given string by uppercase characters

Parameters **input_string** (*str*) – input string

Returns split string

Return type list[str]

Example

```
>>> split_words("UIApplication_ApplicationClosing")
... ['UIApplication', 'Application', 'Closing']
```

pyrevit.coreutils.**timestamp**()
Return timestamp for current time.

Returns timestamp in string format

Return type str

Example

```
>>> timestamp()
'01003075032506808'
```

pyrevit.coreutils.**touch**(*fname*, *times=None*)
Update the timestamp on the given file.

Parameters

- **fname** (*str*) – target file path
- **times** (*int*) – number of times to touch the file

pyrevit.coreutils.**unc_to_dletter**(*unc_path*)
Convert UNC path into drive letter path.

Parameters **unc_path** (*str*) – UNC path

Returns drive letter path

Return type str

Example

```
>>> # assuming J: is mapped to //filestore/server/jdrive
>>> unc_to_dletter('//filestore/server/jdrive/somefile.txt')
'J:/somefile.txt'
```

pyrevit.coreutils.**verify_directory**(*folder*)
Check if the folder exists and if not create the folder.

Parameters **folder** (*str*) – path of folder to verify

Returns path of verified folder, equals to provided folder

Return type str

Raises OSError on folder creation error.

CHAPTER 5

pyrevit.forms

5.1 pyrevit.forms.toaster

Base module for pushing toast messages on Win 10.

This module is a wrapper for a cli utility that provides toast message functionality. See *https://github.com/go-toast/toast*

pyrevit.forms.toaster.**get_toaster**()
Return full file path of the toast binary utility.

pyrevit.forms.toaster.**send_toast**(*message*, *title=None*, *appid=None*, *icon=None*, *click=None*, *actions=None*)
Send toast notificaton.

Parameters

- **message** (*str*) – notification message
- **title** (*str*) – notification title
- **appid** (*str*) – application unique id (see *–app-id* cli option)
- **icon** (*str*) – notification icon (see *–icon* cli option)
- **click** (*str*) – click action (see *–activation-arg* cli option)
- **(dict[str** (*actions*) – str]): list of actions (see *–action* and *–action-arg* cli options)

5.2 pyrevit.forms.utils

Utility functions to support forms module.

pyrevit.forms.utils.**bitmap_from_file**(*bitmap_file*)
Create BitmapImage from a bitmap file.

Parameters **bitmap_file** (*str*) – path to bitmap file

Returns bitmap image object

Return type BitmapImage

pyrevit.forms.utils.**load_component**(*xaml_file*, *comp_type*)
Load WPF component from xaml file.

Parameters

- **xaml_file** (*str*) – xaml file path
- **comp_type** (*System.Windows.Controls*) – WPF control type

Returns loaded WPF control

Return type System.Windows.Controls

pyrevit.forms.utils.**load_ctrl_template**(*xaml_file*)
Load System.Windows.Controls.ControlTemplate from xaml file.

Parameters **xaml_file** (*str*) – xaml file path

Returns loaded control template

Return type System.Windows.Controls.ControlTemplate

pyrevit.forms.utils.**load_itemspanel_template**(*xaml_file*)
Load System.Windows.Controls.ItemsPanelTemplate from xaml file.

Parameters **xaml_file** (*str*) – xaml file path

Returns loaded items-panel template

Return type System.Windows.Controls.ControlTemplate

Reusable WPF forms for pyRevit.

Example

```
>>> from pyrevit.forms import WPFWindow
```

class pyrevit.forms.**CommandSwitchWindow**(*context*, *title*, *width*, *height*, ***kwargs*)
Standard form to select from a list of command options.

Parameters

- **context** (*list[str]*) – list of command options to choose from
- **switches** (*list[str]*) – list of on/off switches
- **message** (*str*) – window title message
- **config** (*dict*) – dictionary of config dicts for options or switches
- **recognize_access_key** (*bool*) – recognize '_' as mark of access key

Returns name of selected option

Return type str

Returns if `switches` option is used, returns a tuple of selection option name and dict of switches

Return type tuple(str, dict)

Example

This is an example with series of command options:

```
>>> from pyrevit import forms
>>> ops = ['option1', 'option2', 'option3', 'option4']
>>> forms.CommandSwitchWindow.show(ops, message='Select Option')
'option2'
```

A more advanced example of combining command options, on/off switches, and option or switch configuration options:

```
>>> from pyrevit import forms
>>> ops = ['option1', 'option2', 'option3', 'option4']
>>> switches = ['switch1', 'switch2']
>>> cfgs = {'option1': { 'background': '0xFF55FF'}}
>>> rops, rswitches = forms.CommandSwitchWindow.show(
...     ops,
...     switches=switches
...     message='Select Option',
...     config=cfgs,
...     recognize_access_key=False
...     )
>>> rops
'option2'
>>> rswitches
{'switch1': False, 'switch2': True}
```

handle_click(*sender*, *args*)
Handle mouse click.

handle_input_key(*sender*, *args*)
Handle keyboard inputs.

process_option(*sender*, *args*)
Handle click on command option button.

search_txt_changed(*sender*, *args*)
Handle text change in search box.

class pyrevit.forms.**FamilyParamOption**(*fparam*, *builtin=False*, *labeled=False*)
Level wrapper for *select_family_parameters()*.

istype
Is type parameter.

name
Family Parameter name.

class pyrevit.forms.**GetValueWindow**(*context*, *title*, *width*, *height*, ***kwargs*)
Standard form to get simple values from user.

Args:

Example

```
>>> from pyrevit import forms
>>> items = ['item1', 'item2', 'item3']
```

(continues on next page)

(continued from previous page)

```
>>> forms.SelectFromList.show(items, button_name='Select Item')
>>> ['item1']
```

select (*sender*, *args*)
Process input data and set the response.

string_value_changed (*sender*, *args*)
Handle string vlaue update event.

class pyrevit.forms.**LevelOption** (*level_element*)
Level wrapper for *select_levels()*.

name
Level name.

class pyrevit.forms.**ParamDef** (*name*, *istype*)
Parameter definition tuple.

name
parameter name

Type str

istype
true if type parameter, otherwise false

Type bool

istype
Alias for field number 1

name
Alias for field number 0

class pyrevit.forms.**ProgressBar** (*height=32*, ***kwargs*)
Show progress bar at the top of Revit window.

Parameters

- **title** (*string*) – progress bar text, defaults to 0/100 progress format
- **indeterminate** (*bool*) – create indeterminate progress bar
- **cancellable** (*bool*) – add cancel button to progress bar
- **step** (*int*) – update progress intervals

Example

```
>>> from pyrevit import forms
>>> count = 1
>>> with forms.ProgressBar(title='my command progress message') as pb:
...     # do stuff
...     pb.update_progress(count, 100)
...     count += 1
```

Progress bar title could also be customized to show the current and total progress values. In example below, the progress bar message will be in format "0 of 100"

```
>>> with forms.ProgressBar(title='{value} of {max_value}') as pb:
```

By default progress bar updates the progress every time the .update_progress method is called. For operations with a large number of max steps, the gui update process time will have a significate effect on the overall execution time of the command. In these cases, set the value of step argument to something larger than 1. In example below, the progress bar updates once per every 10 units of progress.

```
>>> with forms.ProgressBar(title='message', steps=10):
```

Progress bar could also be set to indeterminate for operations of unknown length. In this case, the progress bar will show an infinitely running ribbon:

```
>>> with forms.ProgressBar(title='message', indeterminate=True):
```

if cancellable is set on the object, a cancel button will show on the progress bar and .cancelled attribute will be set on the ProgressBar instance if users clicks on cancel button:

```
>>> with forms.ProgressBar(title='message',
...                        cancellable=True) as pb:
...     # do stuff
...     if pb.cancelled:
...         # wrap up and cancel operation
```

clicked_cancel (*sender*, *args*)
Handler for cancel button clicked event.

indeterminate
Progress bar indeterminate state.

reset ()
Reset progress value to 0.

title
Progress bar title.

update_progress (*new_value*, *max_value=1*)
Update progress bar state with given min, max values.

Parameters

- **new_value** (*float*) – current progress value
- **max_value** (*float*) – total progress value

class pyrevit.forms.**RevisionOption** (*revision_element*)
Revision wrapper for *select_revisions()*.

name
Revision name (description).

class pyrevit.forms.**SearchPrompt** (*search_db*, *width*, *height*, ***kwargs*)
Standard prompt for pyRevit search.

Parameters

- **search_db** (*list*) – list of possible search targets
- **search_tip** (*str*) – text to show in grayscale when search box is empty
- **switches** (*str*) – list of switches
- **width** (*int*) – width of search prompt window
- **height** (*int*) – height of search prompt window

Returns matched strings, and dict of switches if provided str: matched string if switches are not provided.

Return type str, dict

Example

```
>>> from pyrevit import forms
>>> # assume search input of '/switch1 target1'
>>> matched_str, args, switches = forms.SearchPrompt.show(
...     search_db=['target1', 'target2', 'target3', 'target4'],
...     switches=['/switch1', '/switch2'],
...     search_tip='pyRevit Search'
...     )
... matched_str
'target1'
... args
['--help', '--branch', 'branchname']
... switches
{'/switch1': True, '/switch2': False}
```

find_direct_match(*input_text*)
Find direct text matches in search term.

find_word_match(*input_text*)
Find direct word matches in search term.

handle_kb_key(*sender*, *args*)
Handle keyboard input event.

search_input
Current search input.

search_input_parts
Current cleaned up search term.

search_matches
List of matches for the given search term.

search_term
Current cleaned up search term.

search_term_args
Find arguments in search term.

search_term_main
Current cleaned up search term without the listed switches.

search_term_switches
Find matching switches in search term.

search_txt_changed(*sender*, *args*)
Handle text changed event.

set_search_results(**args*)
Set search results for returning.

classmethod show(*search_db*, *width=600*, *height=100*, ***kwargs*)
Show search prompt.

update_results_display (*fill_match=False*)
Update search prompt results based on current input text.

class pyrevit.forms.**SelectFromList** (*context*, *title*, *width*, *height*, ***kwargs*)
Standard form to select from a list of items.

Any object can be passed in a list to the `context` argument. This class wraps the objects passed to context, in *`TemplateListItem`*. This class provides the necessary mechanism to make this form work both for selecting items from a list, and from a list of checkboxes. See the list of arguments below for additional options and features.

Parameters

- **context** (*list[str] or dict[list[str]]*) – list of items to be selected from OR dict of list of items to be selected from. use dict when input items need to be grouped e.g. List of sheets grouped by sheet set.
- **title** (*str, optional*) – window title. see super class for defaults.
- **width** (*int, optional*) – window width. see super class for defaults.
- **height** (*int, optional*) – window height. see super class for defaults.
- **button_name** (*str, optional*) – name of select button. defaults to 'Select'
- **name_attr** (*str, optional*) – object attribute that should be read as item name.
- **multiselect** (*bool, optional*) – allow multi-selection (uses check boxes). defaults to False
- **return_all** (*bool, optional*) – return all items. This is handly when some input items have states and the script needs to check the state changes on all items. This options works in multiselect mode only. defaults to False
- **filterfunc** (*function*) – filter function to be applied to context items.
- **group_selector_title** (*str*) – title for list group selector. defaults to 'List Group'
- **default_group** (*str*) – name of defautl group to be selected

Example

```
>>> from pyrevit import forms
>>> items = ['item1', 'item2', 'item3']
>>> forms.SelectFromList.show(items, button_name='Select Item')
>>> ['item1']
```

```
>>> from pyrevit import forms
>>> ops = [viewsheet1, viewsheet2, viewsheet3]
>>> res = forms.SelectFromList.show(ops,
...                                 multiselect=False,
...                                 name_attr='Name',
...                                 button_name='Select Sheet')
```

```
>>> from pyrevit import forms
>>> ops = {'Sheet Set A': [viewsheet1, viewsheet2, viewsheet3],
...        'Sheet Set B': [viewsheet4, viewsheet5, viewsheet6]}
>>> res = forms.SelectFromList.show(ops,
...                                 multiselect=True,
...                                 name_attr='Name',
```

(continues on next page)

(continued from previous page)

```
...                                     group_selector_title='Sheet Sets',
...                                     button_name='Select Sheets')
```

This module also provides a wrapper base class *TemplateListItem* for when the checkbox option is wrapping another element, e.g. a Revit ViewSheet. Derive from this base class and define the name property to customize how the checkbox is named on the dialog.

```
>>> from pyrevit import forms
>>> class MyOption(forms.TemplateListItem):
...     @property
...     def name(self):
...         return '{} - {}{}'.format(self.item.SheetNumber,
...                                   self.item.SheetNumber)
>>> ops = [MyOption('op1'), MyOption('op2', True), MyOption('op3')]
>>> res = forms.SelectFromList.show(ops,
...                                 multiselect=True,
...                                 button_name='Select Item')
>>> [bool(x) for x in res]  # or [x.state for x in res]
[True, False, True]
```

button_select(*sender*, *args*)
Handle select button click.

check_all(*sender*, *args*)
Handle check all button to mark all check boxes as checked.

check_selected(*sender*, *args*)
Mark selected checkboxes as checked.

clear_search(*sender*, *args*)
Clear search box.

search_txt_changed(*sender*, *args*)
Handle text change in search box.

toggle_all(*sender*, *args*)
Handle toggle all button to toggle state of all check boxes.

uncheck_all(*sender*, *args*)
Handle uncheck all button to mark all check boxes as un-checked.

uncheck_selected(*sender*, *args*)
Mark selected checkboxes as unchecked.

class pyrevit.forms.**SheetOption**(*sheet_element*)
Sheet wrapper for *select_sheets()*.

name
Sheet name.

number
Sheet number.

class pyrevit.forms.**TemplateListItem**(*orig_item*, *checked=False*, *checkable=True*, *name_attr=None*)
Base class for checkbox option wrapping another object.

checkable
List Item CheckBox Visibility.

classmethod is_checkbox(*item*)
Check if the object has all necessary attribs for a checkbox.

name
Name property.

unwrap()
Unwrap and return wrapped object.

class pyrevit.forms.**TemplatePromptBar**(*height=32*, ***kwargs*)
Template context-manager class for creating prompt bars.

Prompt bars are show at the top of the active Revit window and are designed for better prompt visibility.

Parameters

- **height** (*int*) – window height
- ****kwargs** – other arguments to be passed to _setup()

update_window()
Update the prompt bar to match Revit window.

class pyrevit.forms.**TemplateUserInputWindow**(*context*, *title*, *width*, *height*, ***kwargs*)
Base class for pyRevit user input standard forms.

Parameters

- **context** (*any*) – window context element(s)
- **title** (*str*) – window title
- **width** (*int*) – window width
- **height** (*int*) – window height
- ****kwargs** – other arguments to be passed to _setup()

classmethod show(*context*, *title='User Input'*, *width=500*, *height=600*, ***kwargs*)
Show user input window.

Parameters

- **context** (*any*) – window context element(s)
- **title** (*str*) – window title
- **width** (*int*) – window width
- **height** (*int*) – window height
- ****kwargs** (*any*) – other arguments to be passed to window

class pyrevit.forms.**ViewOption**(*view_element*)
View wrapper for *select_views()*.

name
View name.

class pyrevit.forms.**WPFWindow**(*xaml_source*, *literal_string=False*, *handle_esc=True*, *set_owner=True*)
WPF Window base class for all pyRevit forms.

Parameters

- **xaml_source** (*str*) – xaml source filepath or xaml content
- **literal_string** (*bool*) – xaml_source contains xaml content, not filepath

- **handle_esc** (*bool*) – handle Escape button and close the window
- **set_owner** (*bool*) – set the owner of window to host app window

Example

```
>>> from pyrevit import forms
>>> layout = '<Window ' \
>>>          'xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" '␣
↪\
>>>          'xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" ' \
>>>          'ShowInTaskbar="False" ResizeMode="NoResize" ' \
>>>          'WindowStartupLocation="CenterScreen" ' \
>>>          'HorizontalContentAlignment="Center">' \
>>>          '</Window>'
>>> w = forms.WPFWindow(layout, literal_string=True)
>>> w.show()
```

static disable_element(**wpf_elements*)
Enable elements.

Parameters ***wpf_elements** – WPF framework elements to be enabled

static enable_element(**wpf_elements*)
Enable elements.

Parameters ***wpf_elements** – WPF framework elements to be enabled

handle_input_key(*sender*, *args*)
Handle keyboard input and close the window on Escape.

handle_url_click(*sender*, *args*)
Callback for handling click on package website url

static hide_element(**wpf_elements*)
Collapse elements.

Parameters ***wpf_elements** – WPF framework elements to be collaped

pyrevit_version
Active pyRevit formatted version e.g. '4.9-beta'

set_icon(*icon_path*)
Set window icon to given icon path.

set_image_source(*wpf_element*, *image_file*)
Set source file for image element.

Parameters

- **element_name** (*System.Windows.Controls.Image*) – xaml image element
- **image_file** (*str*) – image file path

setup_icon()
Setup default window icon.

show(*modal=False*)
Show window.

show_dialog()
Show modal window.

static show_element(**wpf_elements*)
Show collapsed elements.

Parameters ***wpf_elements** – WPF framework elements to be set to visible.

static toggle_element(**wpf_elements*)
Toggle visibility of elements.

Parameters ***wpf_elements** – WPF framework elements to be toggled.

class pyrevit.forms.**WarningBar**(*height=32*, ***kwargs*)
Show warning bar at the top of Revit window.

Parameters **title** (*string*) – warning bar text

Example

```
>>> with WarningBar(title='my warning'):
...     # do stuff
```

pyrevit.forms.**alert**(*msg*, *title=None*, *sub_msg=None*, *expanded=None*, *footer=''*, *ok=True*, *cancel=False*, *yes=False*, *no=False*, *retry=False*, *warn_icon=True*, *options=None*, *exitscript=False*)
Show a task dialog with given message.

Parameters

- **msg** (*str*) – message to be displayed
- **title** (*str, optional*) – task dialog title
- **sub_msg** (*str, optional*) – sub message
- **expanded** (*str, optional*) – expanded area message
- **ok** (*bool, optional*) – show OK button, defaults to True
- **cancel** (*bool, optional*) – show Cancel button, defaults to False
- **yes** (*bool, optional*) – show Yes button, defaults to False
- **no** (*bool, optional*) – show NO button, defaults to False
- **retry** (*bool, optional*) – show Retry button, defaults to False
- **options** (*list[str], optional*) – list of command link titles in order
- **exitscript** (*bool, optional*) – exit if cancel or no, defaults to False

Returns True if okay, yes, or retry, otherwise False

Return type bool

Example

```
>>> from pyrevit import forms
>>> forms.alert('Are you sure?',
...             ok=False, yes=True, no=True, exitscript=True)
```

pyrevit.forms.**alert_ifnot**(*condition*, *msg*, **args*, ***kwargs*)
Show a task dialog with given message if condition is NOT met.

Parameters

- **condition** (*bool*) – condition to test
- **msg** (*str*) – message to be displayed
- **title** (*str, optional*) – task dialog title
- **ok** (*bool, optional*) – show OK button, defaults to True
- **cancel** (*bool, optional*) – show Cancel button, defaults to False
- **yes** (*bool, optional*) – show Yes button, defaults to False
- **no** (*bool, optional*) – show NO button, defaults to False
- **retry** (*bool, optional*) – show Retry button, defaults to False
- **exitscript** (*bool, optional*) – exit if cancel or no, defaults to False

Returns True if okay, yes, or retry, otherwise False

Return type bool

Example

```
>>> from pyrevit import forms
>>> forms.alert_ifnot(value > 12,
...                   'Are you sure?',
...                    ok=False, yes=True, no=True, exitscript=True)
```

pyrevit.forms.**ask_for_date**(*default=None*, *prompt=None*, *title=None*, ***kwargs*)

Ask user to select a date value.

This is a shortcut function that configures *GetValueWindow* for date data types. kwargs can be used to pass on other arguments.

Parameters

- **default** (*datetime.datetime*) – default selected date value
- **prompt** (*str*) – prompt message
- **title** (*str*) – title message
- **kwargs** (*type*) – other arguments to be passed to *GetValueWindow*

Returns selected date

Return type datetime.datetime

Example

```
>>> forms.ask_for_date(default="", title="Enter deadline:")
... datetime.datetime(2019, 5, 17, 0, 0)
```

pyrevit.forms.**ask_for_one_item**(*items*, *default=None*, *prompt=None*, *title=None*, ***kwargs*)

Ask user to select an item from a list of items.

This is a shortcut function that configures *GetValueWindow* for 'single-select' data types. kwargs can be used to pass on other arguments.

Parameters

- **items** (*list[str]*) – list of items to choose from

- **default** (*str*) – default selected item
- **prompt** (*str*) – prompt message
- **title** (*str*) – title message
- **kwargs** (*type*) – other arguments to be passed to *GetValueWindow*

Returns selected item

Return type str

Example

```
>>> forms.ask_for_one_item(
...     ['test item 1', 'test item 2', 'test item 3'],
...     default='test item 2',
...     prompt='test prompt',
...     title='test title'
... )
... 'test item 1'
```

pyrevit.forms.**ask_for_string**(*default=None*, *prompt=None*, *title=None*, ***kwargs*)

Ask user to select a string value.

This is a shortcut function that configures *GetValueWindow* for string data types. kwargs can be used to pass on other arguments.

Parameters

- **default** (*str*) – default unique string. must not be in reserved_values
- **prompt** (*str*) – prompt message
- **title** (*str*) – title message
- **kwargs** (*type*) – other arguments to be passed to *GetValueWindow*

Returns selected string value

Return type str

Example

```
>>> forms.ask_for_string(
...     default='some-tag',
...     prompt='Enter new tag name:',
...     title='Tag Manager')
... 'new-tag'
```

pyrevit.forms.**ask_for_unique_string**(*reserved_values*, *default=None*, *prompt=None*, *title=None*, ***kwargs*)

Ask user to select a unique string value.

This is a shortcut function that configures *GetValueWindow* for unique string data types. kwargs can be used to pass on other arguments.

Parameters

- **reserved_values** (*list[str]*) – list of reserved (forbidden) values
- **default** (*str*) – default unique string. must not be in reserved_values

- **prompt** (*str*) – prompt message
- **title** (*str*) – title message
- **kwargs** (*type*) – other arguments to be passed to *GetValueWindow*

Returns selected unique string

Return type str

Example

```
>>> forms.ask_for_unique_string(
...     prompt='Enter a Unique Name',
...     title=self.Title,
...     reserved_values=['Ehsan', 'Gui', 'Guido'],
...     owner=self)
... 'unique string'
```

In example above, owner argument is provided to be passed to underlying *GetValueWindow*.

pyrevit.forms.**ask_to_use_selected**(*type_name*, *count=None*, *multiple=True*)

Ask user if wants to use currently selected elements.

Parameters

- **type_name** (*str*) – Element type of expected selected elements
- **count** (*int*) – Number of selected items
- **multiple** (*bool*) – Whether multiple selected items are allowed

pyrevit.forms.**check_familydoc**(*doc=None*, *family_cat=None*, *exitscript=False*)

Verify document is a Family and notify user if not.

Parameters

- **doc** (*DB.Document*) – target document, current of not provided
- **family_cat** (*str*) – family category name
- **exitscript** (*bool*) – exit script if returning False

Returns True if doc is a Family and of provided category

Return type bool

Example

```
>>> from pyrevit import forms
>>> forms.check_familydoc(doc=revit.doc, family_cat='Data Devices')
... True
```

pyrevit.forms.**check_graphicalview**(*view*, *exitscript=False*)

Verify target view is a graphical view

Parameters

- **view** (*DB.View*) – target view
- **exitscript** (*bool*) – exit script if returning False

Returns True if view is a graphical view

Return type bool

Example

```
>>> from pyrevit import forms
>>> forms.check_graphicalview(revit.active_view)
... True
```

pyrevit.forms.**check_modeldoc**(*doc=None*, *exitscript=False*)

Verify document is a not a Model and notify user if not.

Parameters

- **doc** (*DB.Document*) – target document, current of not provided
- **exitscript** (*bool*) – exit script if returning False

Returns True if doc is a Model

Return type bool

Example

```
>>> from pyrevit import forms
>>> forms.check_modeldoc(doc=revit.doc)
... True
```

pyrevit.forms.**check_modelview**(*view*, *exitscript=False*)

Verify target view is a model view.

Parameters

- **view** (*DB.View*) – target view
- **exitscript** (*bool*) – exit script if returning False

Returns True if view is model view

Return type bool

Example

```
>>> from pyrevit import forms
>>> forms.check_modelview(view=revit.active_view)
... True
```

pyrevit.forms.**check_selection**(*exitscript=False*, *message='At least one element must be selected.'*)

Verify if selection is not empty notify user if it is.

Parameters

- **exitscript** (*bool*) – exit script if returning False
- **message** (*str*) – prompt message if returning False

Returns True if selection has at least one item

Return type bool

pyrevit.forms.**check_viewtype**(*view*, *view_type*, *exitscript=False*)
Verify target view is of given type

Parameters

- **view** (*DB.View*) – target view
- **view_type** (*DB.ViewType*) – type of view
- **exitscript** (*bool*) – exit script if returning False

Returns True if view is of given type

Return type bool

Example

```
>>> from pyrevit import forms
>>> forms.check_viewtype(revit.active_view, DB.ViewType.DrawingSheet)
... True
```

pyrevit.forms.**check_workshared**(*doc=None*, *message='Model is not workshared.'*)
Verify if model is workshared and notify user if not.

Parameters

- **doc** (*DB.Document*) – target document, current of not provided
- **message** (*str*) – prompt message if returning False

Returns True if doc is workshared

Return type bool

pyrevit.forms.**inform_wip**()
Show work-in-progress prompt to user and exit script.

Example

```
>>> forms.inform_wip()
```

pyrevit.forms.**pick_excel_file**(*save=False*)
File pick/save dialog for an excel file.

Parameters **save** (*bool*) – show file save dialog, instead of file pick dialog

Returns file path

Return type str

pyrevit.forms.**pick_file**(*file_ext='*'*, *files_filter=''*, *init_dir=''*, *restore_dir=True*, *multi_file=False*, *unc_paths=False*)
Pick file dialog to select a destination file.

Parameters

- **file_ext** (*str*) – file extension
- **files_filter** (*str*) – file filter
- **init_dir** (*str*) – initial directory
- **restore_dir** (*bool*) – restore last directory

- **multi_file** (*bool*) – allow select multiple files
- **unc_paths** (*bool*) – return unc paths

Returns file path or list of file paths if multi_file=True

Return type str or list[str]

Example

```
>>> from pyrevit import forms
>>> forms.pick_file(file_ext='csv')
... r'C:\output\somefile.csv'
```

```
>>> forms.pick_file(file_ext='csv', multi_file=True)
... [r'C:\output\somefile1.csv', r'C:\output\somefile2.csv']
```

```
>>> forms.pick_file(files_filter='All Files (*.*)|*.*|'
                                 'Excel Workbook (*.xlsx)|*.xlsx|'
                                 'Excel 97-2003 Workbook|*.xls',
                    multi_file=True)
... [r'C:\output\somefile1.xlsx', r'C:\output\somefile2.xls']
```

pyrevit.forms.**pick_folder**(*title=None*)

Show standard windows pick folder dialog.

Parameters **title** (*str, optional*) – title for the window

Returns folder path

Return type str

class pyrevit.forms.**reactive**(*getter*)

Decorator for WPF bound properties

setter(*setter*)

Descriptor to change the setter on a property.

pyrevit.forms.**save_excel_file**()

File save dialog for an excel file.

Returns file path

Return type str

pyrevit.forms.**save_file**(*file_ext=''*, *files_filter=''*, *init_dir=''*, *default_name=''*, *restore_dir=True*, *unc_paths=False*)

Save file dialog to select a destination file for data.

Parameters

- **file_ext** (*str*) – file extension
- **files_filter** (*str*) – file filter
- **init_dir** (*str*) – initial directory
- **default_name** (*str*) – default file name
- **restore_dir** (*bool*) – restore last directory
- **unc_paths** (*bool*) – return unc paths

Returns file path

Return type str

Example

```
>>> from pyrevit import forms
>>> forms.save_file(file_ext='csv')
... r'C:\output\somefile.csv'
```

pyrevit.forms.**select_family_parameters**(*family_doc*, *title='Select Parameters'*, *button_name='Select'*, *multiple=True*, *filterfunc=None*, *include_instance=True*, *include_type=True*, *include_builtin=True*, *include_labeled=True*)

Standard form for selecting parameters from given family document.

Parameters

- **family_doc** (*DB.Document*) – source family document
- **title** (*str, optional*) – list window title
- **button_name** (*str, optional*) – list window button caption
- **multiselect** (*bool, optional*) – allow multi-selection (uses check boxes). defaults to True
- **filterfunc** (*function*) – filter function to be applied to context items.
- **include_instance** (*bool, optional*) – list instance parameters
- **include_type** (*bool, optional*) – list type parameters
- **include_builtin** (*bool, optional*) – list builtin parameters
- **include_labeled** (*bool, optional*) – list parameters used as labels

Returns list of family parameter objects

Return type list[DB.FamilyParameter]

Example

```
>>> forms.select_family_parameters(
...     family_doc,
...     title='Select Parameters',
...     multiple=True,
...     include_instance=True,
...     include_type=True
... )
... [<DB.FamilyParameter >, <DB.FamilyParameter >]
```

pyrevit.forms.**select_image**(*images*, *title='Select Image'*, *button_name='Select'*)

Standard form for selecting an image.

Parameters

- **images** (*list[str] | list[framework.Imaging.BitmapImage]*) – list of image file paths or bitmaps
- **title** (*str, optional*) – swatch list window title

- **button_name** (*str, optional*) – swatch list window button caption

Returns path of the selected image

Return type str

Example

```
>>> from pyrevit import forms
>>> forms.select_image(['C:/path/to/image1.png',
                        'C:/path/to/image2.png'],
                        title="Select Variation")
... 'C:/path/to/image1.png'
```

pyrevit.forms.**select_levels**(*title='Select Levels', button_name='Select', width=500, multiple=True, filterfunc=None, doc=None, use_selection=False*)

Standard form for selecting levels.

Parameters

- **title** (*str, optional*) – list window title
- **button_name** (*str, optional*) – list window button caption
- **width** (*int, optional*) – width of list window
- **multiple** (*bool, optional*) – allow multi-selection (uses check boxes). defaults to True
- **filterfunc** (*function*) – filter function to be applied to context items.
- **doc** (*DB.Document, optional*) – source document for levels; defaults to active document
- **use_selection** (*bool, optional*) – ask if user wants to use currently selected levels.

Returns list of selected levels

Return type list[DB.Level]

Example

```
>>> from pyrevit import forms
>>> forms.select_levels()
... [<Autodesk.Revit.DB.Level object>,
...  <Autodesk.Revit.DB.Level object>]
```

pyrevit.forms.**select_open_docs**(*title='Select Open Documents', button_name='OK', width=500, multiple=True, filterfunc=None*)

Standard form for selecting open documents.

Parameters

- **title** (*str, optional*) – list window title
- **button_name** (*str, optional*) – list window button caption
- **width** (*int, optional*) – width of list window
- **multiselect** (*bool, optional*) – allow multi-selection (uses check boxes). defaults to True

- **filterfunc** (*function*) – filter function to be applied to context items.

Returns list of selected documents

Return type list[DB.Document]

Example

```
>>> from pyrevit import forms
>>> forms.select_open_docs()
... [<Autodesk.Revit.DB.Document object>,
...  <Autodesk.Revit.DB.Document object>]
```

pyrevit.forms.**select_parameters**(*src_element*, *title='Select Parameters'*, *button_name='Select'*, *multiple=True*, *filterfunc=None*, *include_instance=True*, *include_type=True*)

Standard form for selecting parameters from given element.

Parameters

- **src_element** (*DB.Element*) – source element
- **title** (*str, optional*) – list window title
- **button_name** (*str, optional*) – list window button caption
- **multiselect** (*bool, optional*) – allow multi-selection (uses check boxes). defaults to True
- **filterfunc** (*function*) – filter function to be applied to context items.
- **include_instance** (*bool, optional*) – list instance parameters
- **include_type** (*bool, optional*) – list type parameters

Returns list of paramdef objects

Return type list[*ParamDef*]

Example

```
>>> forms.select_parameter(
...     src_element,
...     title='Select Parameters',
...     multiple=True,
...     include_instance=True,
...     include_type=True
... )
... [<ParamDef >, <ParamDef >]
```

pyrevit.forms.**select_revisions**(*title='Select Revision'*, *button_name='Select'*, *width=500*, *multiple=True*, *filterfunc=None*, *doc=None*)

Standard form for selecting revisions.

Parameters

- **title** (*str, optional*) – list window title
- **button_name** (*str, optional*) – list window button caption
- **width** (*int, optional*) – width of list window

- **multiselect** (*bool, optional*) – allow multi-selection (uses check boxes). defaults to True
- **filterfunc** (*function*) – filter function to be applied to context items.
- **doc** (*DB.Document, optional*) – source document for revisions; defaults to active document

Returns list of selected revisions

Return type list[DB.Revision]

Example

```
>>> from pyrevit import forms
>>> forms.select_revisions()
... [<Autodesk.Revit.DB.Revision object>,
...  <Autodesk.Revit.DB.Revision object>]
```

pyrevit.forms.**select_schedules**(*title='Select Schedules', button_name='Select', width=500, multiple=True, filterfunc=None, doc=None*)

Standard form for selecting schedules.

Parameters

- **title** (*str, optional*) – list window title
- **button_name** (*str, optional*) – list window button caption
- **width** (*int, optional*) – width of list window
- **multiselect** (*bool, optional*) – allow multi-selection (uses check boxes). defaults to True
- **filterfunc** (*function*) – filter function to be applied to context items.
- **doc** (*DB.Document, optional*) – source document for views; defaults to active document

Returns list of selected schedules

Return type list[DB.ViewSchedule]

Example

```
>>> from pyrevit import forms
>>> forms.select_schedules()
... [<Autodesk.Revit.DB.ViewSchedule object>,
...  <Autodesk.Revit.DB.ViewSchedule object>]
```

pyrevit.forms.**select_sheets**(*title='Select Sheets', button_name='Select', width=500, multiple=True, filterfunc=None, doc=None, include_placeholder=True, use_selection=False*)

Standard form for selecting sheets.

Sheets are grouped into sheet sets and sheet set can be selected from a drop down box at the top of window.

Parameters

- **title** (*str, optional*) – list window title
- **button_name** (*str, optional*) – list window button caption

- **width** (*int, optional*) – width of list window
- **multiple** (*bool, optional*) – allow multi-selection (uses check boxes). defaults to True
- **filterfunc** (*function*) – filter function to be applied to context items.
- **doc** (*DB.Document, optional*) – source document for sheets; defaults to active document
- **use_selection** (*bool, optional*) – ask if user wants to use currently selected sheets.

Returns list of selected sheets

Return type list[DB.ViewSheet]

Example

```
>>> from pyrevit import forms
>>> forms.select_sheets()
... [<Autodesk.Revit.DB.ViewSheet object>,
...  <Autodesk.Revit.DB.ViewSheet object>]
```

pyrevit.forms.**select_swatch**(*title='Select Color Swatch'*, *button_name='Select'*)

Standard form for selecting a color swatch.

Parameters

- **title** (*str, optional*) – swatch list window title
- **button_name** (*str, optional*) – swatch list window button caption

Returns rgb color

Return type *pyrevit.coreutils.colors.RGB*

Example

```
>>> from pyrevit import forms
>>> forms.select_swatch(title="Select Text Color")
... <RGB #CD8800>
```

pyrevit.forms.**select_titleblocks**(*title='Select Titleblock'*, *button_name='Select'*, *no_tb_option='No Title Block'*, *width=500*, *multiple=False*, *filterfunc=None*, *doc=None*)

Standard form for selecting a titleblock.

Parameters

- **title** (*str, optional*) – list window title
- **button_name** (*str, optional*) – list window button caption
- **no_tb_option** (*str, optional*) – name of option for no title block
- **width** (*int, optional*) – width of list window
- **multiselect** (*bool, optional*) – allow multi-selection (uses check boxes). defaults to True
- **filterfunc** (*function*) – filter function to be applied to context items.

- **doc** (*DB.Document, optional*) – source document for titleblocks; defaults to active document

Returns selected titleblock id.

Return type DB.ElementId

Example

```
>>> from pyrevit import forms
>>> forms.select_titleblocks()
... <Autodesk.Revit.DB.ElementId object>
```

pyrevit.forms.**select_views**(*title='Select Views', button_name='Select', width=500, multiple=True, filterfunc=None, doc=None, use_selection=False*)

Standard form for selecting views.

Parameters

- **title** (*str, optional*) – list window title
- **button_name** (*str, optional*) – list window button caption
- **width** (*int, optional*) – width of list window
- **multiple** (*bool, optional*) – allow multi-selection (uses check boxes). defaults to True
- **filterfunc** (*function*) – filter function to be applied to context items.
- **doc** (*DB.Document, optional*) – source document for views; defaults to active document
- **use_selection** (*bool, optional*) – ask if user wants to use currently selected views.

Returns list of selected views

Return type list[DB.View]

Example

```
>>> from pyrevit import forms
>>> forms.select_views()
... [<Autodesk.Revit.DB.View object>,
...  <Autodesk.Revit.DB.View object>]
```

pyrevit.forms.**select_viewtemplates**(*title='Select View Templates', button_name='Select', width=500, multiple=True, filterfunc=None, doc=None*)

Standard form for selecting view templates.

Parameters

- **title** (*str, optional*) – list window title
- **button_name** (*str, optional*) – list window button caption
- **width** (*int, optional*) – width of list window
- **multiselect** (*bool, optional*) – allow multi-selection (uses check boxes). defaults to True

- **filterfunc** (*function*) – filter function to be applied to context items.
- **doc** (*DB.Document, optional*) – source document for views; defaults to active document

Returns list of selected view templates

Return type list[DB.View]

Example

```
>>> from pyrevit import forms
>>> forms.select_viewtemplates()
... [<Autodesk.Revit.DB.View object>,
...  <Autodesk.Revit.DB.View object>]
```

pyrevit.forms.**toast**(*message*, *title='pyRevit'*, *appid='pyRevit'*, *icon=None*, *click=None*, *actions=None*)

Show a Windows 10 notification.

Parameters

- **message** (*str*) – notification message
- **title** (*str*) – notification title
- **appid** (*str*) – app name (will show under message)
- **icon** (*str*) – file path to icon .ico file (defaults to pyRevit icon)
- **click** (*str*) – click action commands string
- **actions** (*dict*) – dictionary of button names and action strings

Example

```
>>> script.toast("Hello World!",
...              title="My Script",
...              appid="MyAPP",
...              click="https://eirannejad.github.io/pyRevit/",
...              actions={
...                  "Open Google":"https://google.com",
...                  "Open Toast64":"https://github.com/go-toast/toast"
...                  })
```

CHAPTER 6

pyrevit.framework

Provide access to DotNet Framework.

Example

```
>>> from pyrevit.framework import Assembly, Windows
```

pyrevit.framework.**get_current_thread_id**()
Return manageed thread id of current thread.

pyrevit.framework.**get_dll_file**(*assembly_name*)
Return path to given assembly name.

pyrevit.framework.**get_type**(*fw_object*)
Return CLR type of an object.

CHAPTER 7

pyrevit.loader

7.1 pyrevit.loader.asmmaker

Assembly maker module.

class pyrevit.loader.asmmaker.**ExtensionAssemblyInfo**(*name*, *location*, *reloading*)

location
Alias for field number 1

name
Alias for field number 0

reloading
Alias for field number 2

pyrevit.loader.asmmaker.**create_assembly**(*extension*)

Parameters **extension** (*pyrevit.extensions.components.Extension*) –

Returns:

7.2 pyrevit.loader.sessioninfo

Manage information about pyRevit sessions.

class pyrevit.loader.sessioninfo.**RuntimeInfo**(*pyrevit_version*, *engine_version*, *host_version*)

Session runtime information tuple.

Parameters

- **pyrevit_version** (*str*) – formatted pyRevit version
- **engine_version** (*int*) – active IronPython engine version

- **host_version** (*str*) – Current Revit version

engine_version
Alias for field number 1

host_version
Alias for field number 2

pyrevit_version
Alias for field number 0

pyrevit.loader.sessioninfo.**get_loaded_pyrevit_assemblies**()
Return list of loaded pyRevit assemblies from environment variable.

Returns list of loaded assemblies

Return type list[str]

pyrevit.loader.sessioninfo.**get_runtime_info**()
Return runtime information tuple.

Returns runtime info tuple

Return type *RuntimeInfo*

Example

```
>>> sessioninfo.get_runtime_info()
```

pyrevit.loader.sessioninfo.**get_session_uuid**()
Read session uuid from environment variable.

Returns session uuid string

Return type str

pyrevit.loader.sessioninfo.**new_session_uuid**()
Create a new uuid for a pyRevit session.

Returns session uuid string

Return type str

pyrevit.loader.sessioninfo.**report_env**()
Report python version, home directory, config file, etc.

pyrevit.loader.sessioninfo.**set_loaded_pyrevit_assemblies**(*loaded_assm_name_list*)
Set the environment variable with list of loaded assemblies.

Parameters

- **loaded_assm_name_list** (*list[str]*) – list of assembly names
- **val** (*type*) – desc

pyrevit.loader.sessioninfo.**set_session_uuid**(*uuid_str*)
Set session uuid on environment variable.

Parameters **uuid_str** (*str*) – session uuid string

pyrevit.loader.sessioninfo.**setup_runtime_vars**()
Setup runtime environment variables with session information.

7.3 pyrevit.loader.systemdiag

Session diagnostics.

pyrevit.loader.systemdiag.**system_diag**()
Verifies system status is appropriate for a pyRevit session.

7.4 pyrevit.loader.sessionmgr

The loader module manages the workflow of loading a new pyRevit session. It's main purpose is to orchestrate the process of finding pyRevit extensions, creating dll assemblies for them, and creating a user interface in the host application.

Everything starts from `sessionmgr.load_session()` function...

The only public function is `load_session()` that loads a new session. Everything else is private.

class pyrevit.loader.sessionmgr.**AssembledExtension**(*ext*, *assm*)

assm
Alias for field number 1

ext
Alias for field number 0

pyrevit.loader.sessionmgr.**execute_command**(*pyrevitcmd_unique_id*)
Executes a pyRevit command.

Parameters **pyrevitcmd_unique_id** (*str*) – Unique/Class Name of the pyRevit command

Returns results from the executed command

pyrevit.loader.sessionmgr.**execute_extension_startup_script**(*script_path*, *ext_name*, *sys_paths=None*)
Executes a script using pyRevit script executor.

Parameters **script_path** (*str*) – Address of the script file

Returns results dictionary from the executed script

pyrevit.loader.sessionmgr.**find_pyrevitcmd**(*pyrevitcmd_unique_id*)
Searches the pyRevit-generated assemblies under current session for the command with the matching unique name (class name) and returns the command type. Notice that this returned value is a 'type' and should be instantiated before use.

Example

```
>>> cmd = find_pyrevitcmd('pyRevitCorepyRevitpyRevittoolsReload')
>>> command_instance = cmd()
>>> command_instance.Execute() # Provide commandData, message, elements
```

Parameters **pyrevitcmd_unique_id** (*str*) – Unique name for the command

Returns Type for the command with matching unique name

pyrevit.loader.sessionmgr.**load_session**()

Handles loading/reloading of the pyRevit addin and extensions. To create a proper ui, pyRevit extensions needs to be properly parsed and a dll assembly needs to be created. This function handles these tasks through interactions with .extensions, .loader.asmmaker, and .loader.uimaker

Example

```
>>> from pyrevit.loader.sessionmgr import load_session
>>> load_session()      # start loading a new pyRevit session
```

Returns None

7.5 pyrevit.loader.uimaker

UI maker.

pyrevit.loader.uimaker.**update_pyrevit_ui**(*ui_ext*, *ext_asm_info*, *create_beta=False*)

Updates/Creates pyRevit ui for the given extension and provided assembly dll address.

Loader base module.

CHAPTER 8

pyrevit.output

8.1 pyrevit.output.linkmaker

Handle creation of output window helper links.

pyrevit.output.linkmaker.**make_link**(*element_ids*, *contents=None*)

Create link for given element ids.

This link is a special format link with revit:// scheme that is handled by the output window to select the provided element ids in current project. Scripts should not call this function directly. Creating clickable element links is handled by the output wrapper object through the `linkify()` method.

Example

```
>>> output = pyrevit.output.get_output()
>>> for idx, elid in enumerate(element_ids):
>>>     print('{}: {}'.format(idx+1, output.linkify(elid)))
```

Provide access to output window and its functionality.

This module provides access to the output window for the currently running pyRevit command. The proper way to access this wrapper object is through the *get_output()* of *pyrevit.script* module. This method, in return uses the *pyrevit.output* module to get access to the output wrapper.

Example

```
>>> from pyrevit import script
>>> output = script.get_output()
```

Here is the source of *pyrevit.script.get_output()*. As you can see this functions calls the *pyrevit.output.get_output()* to receive the output wrapper.

```
def get_output():
    """Return object wrapping output window for current script.

    Returns:
        :obj:`pyrevit.output.PyRevitOutputWindow`: Output wrapper object
    """
    return output.get_output()
```

class pyrevit.output.**PyRevitOutputWindow**

Wrapper to interact with the output window.

add_style(*style_code*, *attribs=None*)

Inject style tag into current html head of the output window.

Parameters

- **style_code** (*str*) – css styling code
- **attribs** (dict) – dictionary of attribute names and value

Example

```
>>> output = pyrevit.output.get_output()
>>> output.add_style('body { color: blue; }')
```

center()

Center the output window on the screen

close()

Close the window.

close_others(*all_open_outputs=False*)

Close all other windows that belong to the current command.

Parameters **all_open_outputs** (*bool*) – Close all any other windows if True

debug_mode

Set debug mode on output window and stream.

This will cause the output window to print information about the buffer stream and other aspects of the output window mechanism.

freeze()

Freeze output content update.

get_head_html()

str: Return inner code of html head element.

get_height()

int: Return current window height.

get_title()

str: Return current window title.

get_width()

int: Return current window width.

hide()

Hide the window.

hide_logpanel()
Hide output window logging panel.

hide_progress()
Hide output window progress bar.

indeterminate_progress(*state*)
Show or hide indeterminate progress bar.

inject_script(*script_code*, *attribs=None*, *body=False*)
Inject script tag into current head (or body) of the output window.

Parameters

- **script_code** (*str*) – javascript code
- **attribs** (dict) – dictionary of attribute names and value
- **body** (*bool, optional*) – injects script into body instead of head

Example

```
>>> output = pyrevit.output.get_output()
>>> output.inject_script('',   # no script since it's a link
                         {'src': js_script_file_path})
```

inject_to_body(*element_tag*, *element_contents*, *attribs=None*)
Inject html element to current html body of the output window.

Parameters

- **element_tag** (*str*) – html tag of the element e.g. 'div'
- **element_contents** (*str*) – html code of the element contents
- **attribs** (dict) – dictionary of attribute names and value

Example

```
>>> output = pyrevit.output.get_output()
>>> output.inject_to_body('script',
                          '',   # no script since it's a link
                          {'src': js_script_file_path})
```

inject_to_head(*element_tag*, *element_contents*, *attribs=None*)
Inject html element to current html head of the output window.

Parameters

- **element_tag** (*str*) – html tag of the element e.g. 'div'
- **element_contents** (*str*) – html code of the element contents
- **attribs** (dict) – dictionary of attribute names and value

Example

```
>>> output = pyrevit.output.get_output()
>>> output.inject_to_head('script',
                          '',   # no script since it's a link
                          {'src': js_script_file_path})
```

insert_divider(*level=''*)
Add horizontal rule to the output window.

static linkify(*element_ids*, *title=None*)
Create clickable link for the provided element ids.

This method, creates the link but does not print it directly.

Parameters

- **element_ids** (`list` of *ElementId*) –
- **element_ids** – single or multiple ids
- **title** (*str*) – tile of the link. defaults to list of element ids

Example

```
>>> output = pyrevit.output.get_output()
>>> for idx, elid in enumerate(element_ids):
>>>     print('{}: {}'.format(idx+1, output.linkify(elid)))
```

lock_size()
Lock window size.

log_debug(*message*)
Report DEBUG message into output logging panel.

log_error(*message*)
Report ERROR message into output logging panel.

log_info(*message*)
Report INFO message into output logging panel.

log_success(*message*)
Report SUCCESS message into output logging panel.

log_warning(*message*)
Report WARNING message into output logging panel.

make_bar_chart(*version=None*)
`PyRevitOutputChart`: Return bar chart object.

make_bubble_chart(*version=None*)
`PyRevitOutputChart`: Return bubble chart object.

make_chart(*version=None*)
`PyRevitOutputChart`: Return chart object.

make_doughnut_chart(*version=None*)
`PyRevitOutputChart`: Return dougnut chart object.

make_line_chart(*version=None*)
`PyRevitOutputChart`: Return line chart object.

make_pie_chart(*version=None*)
PyRevitOutputChart: Return pie chart object.

make_polar_chart(*version=None*)
PyRevitOutputChart: Return polar chart object.

make_radar_chart(*version=None*)
PyRevitOutputChart: Return radar chart object.

make_stacked_chart(*version=None*)
PyRevitOutputChart: Return stacked chart object.

next_page()
Add hidden next page tag to the output window.

This is helpful to silently separate the output to multiple pages for better printing.

open_page(*dest_file*)
Open html page in output window.

Parameters **dest_file** (*str*) – full path of the target html file

open_url(*dest_url*)
Open url page in output window.

Parameters **dest_url** (*str*) – web url of the target page

output_id
Return id of the output window.

In current implementation, Id of output window is equal to the unique id of the pyRevit command it belongs to. This means that all output windows belonging to the same pyRevit command, will have identical output_id values.

Type str

output_uniqueid
Return unique id of the output window.

In current implementation, unique id of output window is a GUID string generated when the output window is opened. This id is unique to the instance of output window.

Type str

static print_code(*code_str*)
Print code to the output window with special formatting.

Example

```
>>> output = pyrevit.output.get_output()
>>> output.print_code('value = 12')
```

static print_html(*html_str*)
Add the html code to the output window.

Example

```
>>> output = pyrevit.output.get_output()
>>> output.print_html('<strong>Title</strong>')
```

print_image(*image_path*)
Prints given image to the output.

Example

```
>>> output = pyrevit.output.get_output()
>>> output.print_image(r'C:\image.gif')
```

static print_md(*md_str*)
Process markdown code and print to output window.

Example

```
>>> output = pyrevit.output.get_output()
>>> output.print_md('### Title')
```

print_table(*table_data*, *columns=None*, *formats=None*, *title=''*, *last_line_style=''*)
Print provided data in a table in output window.

Parameters

- **table_data** (`list` of iterables) – 2D array of data
- **title** (*str*) – table title
- **columns** (`list` str) – list of column names
- **formats** (`list` str) – column data formats
- **last_line_style** (*str*) – css style of last row

Example

```
>>> data = [
... ['row1', 'data', 'data', 80 ],
... ['row2', 'data', 'data', 45 ],
... ]
>>> output.print_table(
... table_data=data,
... title="Example Table",
... columns=["Row Name", "Column 1", "Column 2", "Percentage"],
... formats=['', '', '', '{}%'],
... last_line_style='color:red;'
... )
```

renderer
Return html renderer inside output window.

Returns `System.Windows.Forms.WebBrowser` (In current implementation)

reset_icon()
Sets icon on the output window.

reset_progress()
Reset output window progress bar to zero.

resize(*width*, *height*)
Resize window to the new width and height.

save_contents(*dest_file*)
Save html code of the window.

Parameters **dest_file** (*str*) – full path of the destination html file

self_destruct(*seconds*)
Set self-destruct (close window) timer.

Parameters **seconds** (*int*) – number of seconds after which window is closed.

set_font(*font_family*, *font_size*)
Set window font family to the new font family and size.

Parameters

- **font_family** (*str*) – font family name e.g. 'Courier New'
- **font_size** (*int*) – font size e.g. 16

set_height(*height*)
Set window height to the new height.

set_icon(*iconpath*)
Sets icon on the output window.

set_title(*new_title*)
Set window title to the new title.

set_width(*width*)
Set window width to the new width.

show()
Show the window.

show_logpanel()
Show output window logging panel.

unfreeze()
Unfreeze output content update.

unhide_progress()
Unhide output window progress bar.

unlock_size()
Unock window size.

update_progress(*cur_value*, *max_value*)
Activate and update the output window progress bar.

Parameters

- **cur_value** (*float*) – current progress value e.g. 50
- **max_value** (*float*) – total value e.g. 100

Example

```
>>> output = pyrevit.output.get_output()
>>> for i in range(100):
>>>     output.update_progress(i, 100)
```

window
Return output window object.

Type PyRevitLabs.PyRevit.Runtime.ScriptConsole

pyrevit.output.**docclosing_eventhandler**(*sender*, *args*)
Close all output window on document closing.

pyrevit.output.**get_default_stylesheet**()
Return default css stylesheet used by output window.

pyrevit.output.**get_output**()
pyrevit.output.PyRevitOutputWindow : Return output window.

pyrevit.output.**get_stylesheet**()
Return active css stylesheet used by output window.

pyrevit.output.**reset_stylesheet**()
Reset active stylesheet to default.

pyrevit.output.**set_stylesheet**(*stylesheet*)
Set active css stylesheet used by output window.

Parameters **stylesheet** (*str*) – full path to stylesheet file

pyrevit.output.**setup_output_closer**()
Setup document closing event listener.

CHAPTER 9

pyrevit.revit

9.1 pyrevit.revit.files

Helper functions for working with revit files.

pyrevit.revit.files.**cleanup_backups**(*main_revitfile*)
Remove all incremental saves of the given Revit file.

9.2 pyrevit.revit.geom

Geometric utilities for Revit.

pyrevit.revit.geom.**convert_point_coord_system**(*rvt_point*, *rvt_transform*)
Return coordinates of point in another coordinate system.

Parameters

- **rvt_point** (*DB.XYZ*) – Revit point
- **rvt_transform** (*DB.Transform*) – Revit transform for target coord system

Returns Point coordinates in new coordinate system.

Return type DB.XYZ

pyrevit.revit.geom.**convert_point_to_metric**(*rvt_point*)
Convert given point coordinates to metric.

9.3 pyrevit.revit.serverutils

Helper functions for working with revit server.

class pyrevit.revit.serverutils.**SyncHistory**(*index*, *userid*, *timestamp*)
namedtuple for model sync history data in revit server

index
row index in history db

Type int

userid
user identifier

Type str

timestamp
time stamp string (e.g. "2017-12-13 19:56:20")

Type str

index
Alias for field number 0

timestamp
Alias for field number 2

userid
Alias for field number 1

pyrevit.revit.serverutils.**get_model_sync_history**(*server_path*)
Read model sync history from revit server sqlite history file.

Parameters **server_path** (*str*) – directory path of revit server filestore

Returns list of SyncHistory instances

Return type list``(``SyncHistory``)

Example

```
>>> get_model_sync_history("//servername/path/to/model.rvt")
... [SyncHistory(index=498, userid="user",
...              timestamp="2017-12-13 19:56:20")]
```

pyrevit.revit.serverutils.**get_server_path**(*doc*, *path_dict*)
Return file path of a model hosted on revit server.

Parameters

- **doc** (*Document*) – revit document object
- **path_dict** (*dict*) – dict of RSN paths and their directory paths

Example

```
>>> rsn_paths = {'RSN://SERVERNAME': '//servername/filestore'}
>>> get_server_path(doc, rsn_paths)
... "//servername/filestore/path/to/model.rvt"
```

9.4 pyrevit.revit.units

Unit conversion utilities for Revit.

pyrevit.revit.units.**format_area**(*area_value*, *doc=None*)
Return formatted area value in document units.

Parameters

- **area_value** (*float*) – area value
- **doc** (*DB.Document, optional*) – Revit document, defaults to current

Returns formatted value

Return type str

pyrevit.revit.units.**project_to_viewport**(*xyz*, *view*)
Project a point to viewport coordinates

Parameters

- **xyz** (*DB.XYZ*) – point to project
- **view** (*DB.View*) – target view

Returns [description]

Return type DB.UV

pyrevit.revit.units.**project_to_world**(*uv*, *view*)
Get view-based point (UV) back to model coordinates.

Parameters

- **uv** (*DB.UV*) – point on a view
- **view** (*DB.View*) – view to get coordinates from

Returns point in world coordinates

Return type DB.XYZ

class pyrevit.revit.**ErrorSwallower**(*log_errors=True*)
Suppresses warnings during script execution

Example

```
>>> with ErrorSwallower() as swallower:
>>>     for fam in families:
>>>         revit.doc.EditFamily(fam)
>>>         if swallower.get_swallowed():
>>>             logger.warn("Warnings swallowed")
```

get_swallowed_errors()
Return swallowed errors

on_failure_processing(*_*, *event_args*)
Failure processing event handler

reset()
Reset swallowed errors

class pyrevit.revit.**RevitWrapper**

static close_doc(*doc*)
Close given document.

Parameters **doc** (*DB.Document*) – document

static open_doc(*doc_path*)
Open document at given path.

Parameters **doc_path** (*str*) – document file path

Returns opened document

Return type DB.Document

CHAPTER 10

pyrevit.routes

10.1 pyrevit.routes.server

10.1.1 pyrevit.routes.server.base

Utility functions and types

class pyrevit.routes.server.base.**Request**(*path='/'*, *method='GET'*, *data=None*, *params=None*)

Request wrapper object

add_header(*key*, *value*)

Add new header key:value

callback_url

Request callback url, if provided in payload

headers

Request headers dict

params

Request parameters

class pyrevit.routes.server.base.**Response**(*status=200*, *data=None*, *headers=None*)

Response wrapper object

add_header(*key*, *value*)

Add new header key:value

headers

Response headers dict

Handles http api routing and serving with usage similar to flask.

class pyrevit.routes.server.**Request**(*path='/'*, *method='GET'*, *data=None*, *params=None*)

Request wrapper object

add_header(*key*, *value*)
Add new header key:value

callback_url
Request callback url, if provided in payload

headers
Request headers dict

params
Request parameters

class pyrevit.routes.server.**Response**(*status=200*, *data=None*, *headers=None*)
Response wrapper object

add_header(*key*, *value*)
Add new header key:value

headers
Response headers dict

pyrevit.routes.server.**init**()
Initialize routes. Reset all registered routes and shutdown servers

pyrevit.routes.server.**activate_server**()
Activate routes server for this host instance

pyrevit.routes.server.**deactivate_server**()
Deactivate the active routes server for this host instance

pyrevit.routes.server.**get_active_server**()
Get active routes server for this host instance

pyrevit.routes.server.**make_response**(*data*, *status=200*, *headers=None*)
Create Reponse object with

pyrevit.routes.server.**get_routes**(*api_name*)
Get all registered routes for given API name

Parameters **api_name** (*str*) – unique name of the api

pyrevit.routes.server.**add_route**(*api_name*, *pattern*, *method*, *handler_func*)
Add new route for given API name

Parameters

- **api_name** (*str*) – unique name of the api
- **pattern** (*str*) – route pattern
- **method** (*str*) – method name
- **handler_func** (*function*) – route handler function

pyrevit.routes.server.**remove_route**(*api_name*, *pattern*, *method*)
Remove previously registered route for given API name

Parameters

- **api_name** (*str*) – unique name of the api
- **pattern** (*str*) – route pattern
- **method** (*str*) – method name

HTTP API framework similar to flask.

class pyrevit.routes.**API**(*name*)
API root object

Parameters **name** (*str*) – URL-safe unique root name of the API

Example

```
>>> from pyrevit import routes
>>> api = routes.API("pyrevit-core")
>>> @api.route('/sessions/', methods=['POST'])
>>> def reload_pyrevit(uiapp):
...     new_session_id = sessionmgr.reload_pyrevit()
...     return {"session_id": new_session_id}
```

route(*pattern, methods=['GET']*)
Define a new route on this API.

pyrevit.routes.**active_routes_api**()
Activates routes API

CHAPTER 11

pyrevit.script

Provide basic utilities for pyRevit scripts.

Example

```
>>> from pyrevit import script
>>> script.clipboard_copy('some text')
>>> data = script.journal_read('data-key')
>>> script.exit()
```

pyrevit.script.**clipboard_copy**(*string_to_copy*)
Copy string to Windows Clipboard.

pyrevit.script.**dump_csv**(*data*, *filepath*)
Dumps given data into given csv file.

Parameters

- **data** (*list[list[str]]*) – data to be dumped
- **filepath** (*str*) – csv file path

pyrevit.script.**dump_json**(*data*, *filepath*)
Dumps given data into given json file.

Parameters

- **data** (*object*) – serializable data to be dumped
- **filepath** (*str*) – json file path

pyrevit.script.**exit**()
Stop the script execution and exit.

pyrevit.script.**get_all_buttons**()
Find and return all ui buttons matching current script command name.

Sometimes tools are duplicated across extensions for user access control so this would help smart buttons to find all the loaded buttons and make icon adjustments.

Returns list of ui button objects

Return type list(pyrevit.coreutils.ribbon._PyRevitRibbonButton)

pyrevit.script.**get_alt_script_path**()
Return config script path of the current pyRevit command.

Returns config script path

Return type str

pyrevit.script.**get_bundle_file**(*file_name*)
Return full path to file under current script bundle.

Parameters **file_name** (*str*) – bundle file name

Returns full bundle file path

Return type str

pyrevit.script.**get_bundle_files**(*sub_path=None*)
Return full path to all file under current script bundle.

Returns list of bundle file paths

Return type list[str]

pyrevit.script.**get_bundle_name**()
Return bundle name of the current pyRevit command.

Returns bundle name (e.g. MyButton.pushbutton)

Return type str

pyrevit.script.**get_button**()
Find and return current script ui button.

Returns ui button object

Return type pyrevit.coreutils.ribbon._PyRevitRibbonButton

pyrevit.script.**get_config**(*section=None*)
Create and return config section parser object for current script.

Parameters **section** (*str, optional*) – config section name

Returns Config section parser object

Return type *pyrevit.coreutils.configparser.PyRevitConfigSectionParser*

pyrevit.script.**get_data_file**(*file_id*, *file_ext*, *add_cmd_name=False*)
Return filename to be used by a user script to store data.

File name is generated in this format: pyRevit_{Revit Version}_{file_id}.{file_ext}

Example

```
>>> script.get_data_file('mydata', 'data')
'.../pyRevit_2018_mydata.data'
>>> script.get_data_file('mydata', 'data', add_cmd_name=True)
'.../pyRevit_2018_Command Name_mydata.data'
```

Data files are not cleaned up at pyRevit startup. Script should manage cleaning up these files.

Parameters

- **file_id** (*str*) – unique id for the filename
- **file_ext** (*str*) – file extension
- **add_cmd_name** (*bool, optional*) – add command name to file name

Returns full file path

Return type str

pyrevit.script.**get_document_data_file**(*file_id*, *file_ext*, *add_cmd_name=False*)

Return filename to be used by a user script to store data.

File name is generated in this format: `pyRevit_{Revit Version}_{file_id}_{Project Name}.{file_ext}`

Example

```
>>> script.get_document_data_file('mydata', 'data')
'.../pyRevit_2018_mydata_Project1.data'
>>> script.get_document_data_file('mydata', 'data', add_cmd_name=True)
'.../pyRevit_2018_Command Name_mydata_Project1.data'
```

Document data files are not cleaned up at pyRevit startup. Script should manage cleaning up these files.

Parameters

- **file_id** (*str*) – unique id for the filename
- **file_ext** (*str*) – file extension
- **add_cmd_name** (*bool, optional*) – add command name to file name

Returns full file path

Return type str

pyrevit.script.**get_envvar**(*envvar*)

Return value of give pyRevit environment variable.

The environment variable system is used to retain small values in memory between script runs (e.g. active/inactive state for toggle tools). Do not store large objects in memory using this method. List of currently set environment variables could be sees in pyRevit settings window.

Parameters **envvar** (*str*) – name of environment variable

Returns type of object stored in environment variable

Return type any

Example

```
>>> script.get_envvar('ToolActiveState')
True
```

pyrevit.script.**get_extension_name**()

Return extension name of the current pyRevit command.

Returns extension name (e.g. MyExtension.extension)

Return type str

pyrevit.script.**get_info**()
Return info on current pyRevit command.

Returns Command info object

Return type pyrevit.extensions.genericcomps.GenericUICommand

pyrevit.script.**get_instance_data_file**(*file_id*, *add_cmd_name=False*)
Return filename to be used by a user script to store data.

File name is generated in this format: pyRevit_{Revit Version}_{Process Id}_{file_id}.{file_ext}

Example

```
>>> script.get_instance_data_file('mydata')
'.../pyRevit_2018_6684_mydata.tmp'
>>> script.get_instance_data_file('mydata', add_cmd_name=True)
'.../pyRevit_2018_6684_Command Name_mydata.tmp'
```

Instance data files are cleaned up at pyRevit startup.

Parameters

- **file_id** (*str*) – unique id for the filename
- **add_cmd_name** (*bool, optional*) – add command name to file name

Returns full file path

Return type str

pyrevit.script.**get_logger**()
Create and return logger named for current script.

Returns Logger object

Return type *pyrevit.coreutils.logger.LoggerWrapper*

pyrevit.script.**get_output**()
Return object wrapping output window for current script.

Returns Output wrapper object

Return type *pyrevit.output.PyRevitOutputWindow*

pyrevit.script.**get_pyrevit_version**()
Return pyRevit version.

Returns pyRevit version provider

Return type *pyrevit.versionmgr._PyRevitVersion*

pyrevit.script.**get_results**()
Return command results dictionary for logging.

Returns Command results dict

Return type pyrevit.telemetry.record.CommandCustomResults

pyrevit.script.**get_script_path**()
Return script path of the current pyRevit command.

Returns script path

Return type str

pyrevit.script.**get_unique_id**()
Return unique id of the current pyRevit command.

Returns command unique id

Return type str

pyrevit.script.**get_universal_data_file**(*file_id*, *file_ext*, *add_cmd_name=False*)
Return filename to be used by a user script to store data.

File name is generated in this format: `pyRevit_{file_id}.{file_ext}`

Example

```
>>> script.get_universal_data_file('mydata', 'data')
'.../pyRevit_mydata.data'
>>> script.get_universal_data_file('mydata', 'data', add_cmd_name=True)
'.../pyRevit_Command Name_mydata.data'
```

Universal data files are not cleaned up at pyRevit startup. Script should manage cleaning up these files.

Parameters

- **file_id** (*str*) – unique id for the filename
- **file_ext** (*str*) – file extension
- **add_cmd_name** (*bool, optional*) – add command name to file name

Returns full file path

Return type str

pyrevit.script.**journal_read**(*data_key*)
Read value for provided key from active Revit journal.

Parameters **data_key** (*str*) – data key

Returns data value string

Return type str

pyrevit.script.**journal_write**(*data_key*, *msg*)
Write key and value to active Revit journal for current command.

Parameters

- **data_key** (*str*) – data key
- **msg** (*str*) – data value string

pyrevit.script.**load_csv**(*filepath*)
Read lines from given csv file

Parameters **filepath** (*str*) – csv file path

Returns csv data

Return type list[list[str]]

pyrevit.script.**load_data**(*slot_name*, *this_project=True*)

Wraps python pickle.load() to easily load data from pyRevit data files

To recover native Revit objects, use revit.deserialize(). See Example

Similar to pickle module, the custom data types must be defined in the main scope so the loader can create an instance and return original stored data

Parameters

- **slot_name** (*type*) – desc
- **this_project** (*bool*) – data belongs to this project only

Returns stored data

Return type obj

Example

```
>>> from pyrevit import revit
... from pyrevit import script
...
...
... class CustomData(object):
...     def __init__(self, count, element_ids):
...         self._count = count
...         # serializes the Revit native objects
...         self._elmnt_ids = [revit.serialize(x) for x in element_ids]
...
...     @property
...     def count(self):
...         return self._count
...
...     @property
...     def element_ids(self):
...         # de-serializes the Revit native objects
...         return [x.deserialize() for x in self._elmnt_ids]
...
...
... mydata = script.load_data("Selected Elements", element_ids)
... mydata.element_ids
[<DB.ElementId>, <DB.ElementId>, <DB.ElementId>]
```

pyrevit.script.**load_index**(*index_file='index.html'*)

Load html file into output window.

This method expects index.html file in the current command bundle, unless full path to an html file is provided.

Parameters **index_file** (*str, optional*) – full path of html file.

pyrevit.script.**load_json**(*filepath*)

Loads data from given json file.

Parameters **filepath** (*str*) – json file path

Returns deserialized data

Return type object

pyrevit.script.**open_url**(*url*)

Open url in a new tab in default webbrowser.

pyrevit.script.**reset_config**(*section=None*)
Reset pyRevit config.

Script should call this to reset any save configuration by removing section related to current script.

Parameters **section** (*str, optional*) – config section name

pyrevit.script.**save_config**()
Save pyRevit config.

Scripts should call this to save any changes they have done to their config section object received from script.get_config() method.

pyrevit.script.**set_envvar**(*envvar*, *value*)
Set value of give pyRevit environment variable.

The environment variable system is used to retain small values in memory between script runs (e.g. active/inactive state for toggle tools). Do not store large objects in memory using this method. List of currently set environment variables could be sees in pyRevit settings window.

Parameters

- **envvar** (*str*) – name of environment variable
- **value** (*any*) – value of environment variable

Example

```
>>> script.set_envvar('ToolActiveState', False)
>>> script.get_envvar('ToolActiveState')
False
```

pyrevit.script.**show_file_in_explorer**(*file_path*)
Show file in Windows Explorer.

pyrevit.script.**show_folder_in_explorer**(*folder_path*)
Show folder in Windows Explorer.

pyrevit.script.**store_data**(*slot_name*, *data*, *this_project=True*)
Wraps python pickle.dump() to easily store data to pyRevit data files

To store native Revit objects, use revit.serialize(). See Example

Parameters

- **slot_name** (*type*) – desc
- **data** (*obj*) – any pickalable data
- **this_project** (*bool*) – data belongs to this project only

Example

```
>>> from pyrevit import revit
... from pyrevit import script
...
...
... class CustomData(object):
...     def __init__(self, count, element_ids):
...         self._count = count
```

(continues on next page)

(continued from previous page)

```
...         # serializes the Revit native objects
...         self._elmnt_ids = [revit.serialize(x) for x in element_ids]
...
...     @property
...     def count(self):
...         return self._count
...
...     @property
...     def element_ids(self):
...         # de-serializes the Revit native objects
...         return [x.deserialize() for x in self._elmnt_ids]
...
...
... mydata = CustomData(
...     count=3,
...     element_ids=[<DB.ElementId>, <DB.ElementId>, <DB.ElementId>]
... )
...
... script.store_data("Selected Elements", mydata)
```

pyrevit.script.**toggle_icon**(*new_state*, *on_icon_path=None*, *off_icon_path=None*)

Set the state of button icon (on or off).

This method expects on.png and off.png in command bundle for on and off icon states, unless full path of icon states are provided.

Parameters

- **new_state** (*bool*) – state of the ui button icon.
- **on_icon_path** (*str, optional*) – full path of icon for on state. default='on.png'
- **off_icon_path** (*str, optional*) – full path of icon for off state. default='off.png'

CHAPTER 12

pyrevit.userconfig

Handle reading and parsing, writin and saving of all user configurations.

This module handles the reading and writing of the pyRevit configuration files. It's been used extensively by pyRevit sub-modules. `user_config` is set up automatically in the global scope by this module and can be imported into scripts and other modules to access the default configurations.

All other modules use this module to query user config.

Example

```
>>> from pyrevit.userconfig import user_config
>>> user_config.add_section('newsection')
>>> user_config.newsection.property = value
>>> user_config.newsection.get('property', default_value)
>>> user_config.save_changes()
```

The `user_config` object is also the destination for reading and writing configuration by pyRevit scripts through `get_config()` of *`pyrevit.script`* module. Here is the function source:

```
def get_config(section=None):
    """Create and return config section parser object for current script.

    Args:
        section (str, optional): config section name

    Returns:
        :obj:`pyrevit.coreutils.configparser.PyRevitConfigSectionParser`:
            Config section parser object
    """
    from pyrevit.userconfig import user_config
    if not section:
        script_cfg_postfix = 'config'
        section = EXEC_PARAMS.command_name + script_cfg_postfix
```

(continues on next page)

(continued from previous page)

```
    try:
        return user_config.get_section(section)
    except Exception:
        return user_config.add_section(section)
```

Example

```
>>> from pyrevit import script
>>> cfg = script.get_config()
>>> cfg.property = value
>>> cfg.get('property', default_value)
>>> script.save_config()
```

class pyrevit.userconfig.**PyRevitConfig**(*cfg_file_path=None*, *config_type='Unknown'*)

Provide read/write access to pyRevit configuration.

Parameters

- **cfg_file_path** (*str*) – full path to config file to be used.
- **config_type** (*str*) – type of config file

Example

```
>>> cfg = PyRevitConfig(cfg_file_path)
>>> cfg.add_section('sectionname')
>>> cfg.sectionname.property = value
>>> cfg.sectionname.get('property', default_value)
>>> cfg.save_changes()
```

config_file

Current config file path.

get_active_cpython_engine()

Return active cpython engine.

get_config_version()

Return version of config file used for change detection.

get_current_attachment()

Return current pyRevit attachment.

get_ext_root_dirs()

Return a list of all extension directories.

Returns list of strings. user extension directories.

Return type list

static get_list_separator()

Get list separator defined in user os regional settings.

get_thirdparty_ext_root_dirs(*include_default=True*)

Return a list of external extension directories set by the user.

Returns list of strings. External user extension directories.

Return type list

save_changes()
Save user config into associated config file.

set_thirdparty_ext_root_dirs(*path_list*)
Updates list of external extension directories in config file

Parameters **path_list** (*list[str]*) – list of external extension paths

pyrevit.userconfig.**find_config_file**(*target_path*)
Find config file in target path.

pyrevit.userconfig.**verify_configs**(*config_file_path=None*)
Create a user settings file.

if config_file_path is not provided, configs will be in memory only

Parameters **config_file_path** (*str, optional*) – config file full name and path

Returns pyRevit config file handler

Return type *pyrevit.userconfig.PyRevitConfig*

CHAPTER 13

pyrevit.versionmgr

13.1 pyrevit.versionmgr.about

Utility module for pyRevit project information.

Example

```
>>> from pyrevit.versionmgr import about
>>> a = about.get_pyrevit_about()
>>> a.subtitle
... 'python RAD Environment for Autodesk Revit®'
>>> a.copyright
... '© 2014-2020 Ehsan Iran-Nejad'
```

class pyrevit.versionmgr.about.**PyRevitAbout**(*subtitle*, *madein*, *copyright*)

pyRevit project info tuple.

subtitle

project subtitle

Type str

madein

project made-in info

Type str

copyright

project copyright info

Type str

copyright

Alias for field number 2

madein
Alias for field number 1

subtitle
Alias for field number 0

pyrevit.versionmgr.about.**get_pyrevit_about**()
Return information about pyRevit project.

Returns pyRevit project info tuple

Return type *PyRevitAbout*

13.2 pyrevit.versionmgr.updater

Handle updating pyRevit repository and its extensions.

pyrevit.versionmgr.updater.**check_for_updates**()
Check whether any available repo has pending updates.

pyrevit.versionmgr.updater.**get_all_extension_repos**()
Return a list of repos for all installed extensions.

pyrevit.versionmgr.updater.**get_thirdparty_ext_repos**()
Return a list of repos for installed third-party extensions.

pyrevit.versionmgr.updater.**get_updates**(*repo_info*)
Fetch updates on repository.

Parameters **repo_info** (*pyrevit.coreutils.git.RepoInfo*) – repository info wrapper object

pyrevit.versionmgr.updater.**has_core_updates**()
Check whether pyRevit repo has core updates.

This would require host application to be closed to release the file lock of core DLLs so they can be updated separately.

pyrevit.versionmgr.updater.**has_pending_updates**(*repo_info*)
Check for updates on repository.

Parameters **repo_info** (*pyrevit.coreutils.git.RepoInfo*) – repository info wrapper object

pyrevit.versionmgr.updater.**update_pyrevit**()
Update pyrevit and its extension repositories.

pyrevit.versionmgr.updater.**update_repo**(*repo_info*)
Update repository.

Parameters **repo_info** (*pyrevit.coreutils.git.RepoInfo*) – repository info wrapper object

13.3 pyrevit.versionmgr.upgrade

Perform upgrades between version, e.g. adding a new config parameter

pyrevit.versionmgr.upgrade.**upgrade_existing_pyrevit**()
Upgrade existing pyRevit deployment.

pyrevit.versionmgr.upgrade.**upgrade_user_config**(*user_config*)
Upgarde user configurations.

Parameters

- **user_config** (*pyrevit.userconfig.PyRevitConfig*) – config object
- **val** (*type*) – desc

Utility functions for managing pyRevit versions.

Example

```
>>> from pyrevit import versionmgr
>>> v = versionmgr.get_pyrevit_version()
>>> v.get_formatted()
... '4.10-beta2'
```

class pyrevit.versionmgr.**_PyRevitVersion**(*patch_number*)
pyRevit version wrapper.

Parameters **patch_number** (*str*) – patch value

as_int_tuple()
Returns version as an int tuple (major, minor, patch)

as_str_tuple()
Returns version as an string tuple ('major', 'minor', 'patch')

get_formatted(*nopatch=False*)
Returns 'major.minor:patch' in string

pyrevit.versionmgr.**get_pyrevit_cli_version**()
Return version of shipped pyRevit CLI utility.

Returns version string of pyRevit CLI utility binary

Return type str

pyrevit.versionmgr.**get_pyrevit_repo**()
Return pyRevit repository.

Returns repo wrapper object

Return type *pyrevit.coreutils.git.RepoInfo*

pyrevit.versionmgr.**get_pyrevit_version**()
Return information about active pyRevit version.

Returns version wrapper object

Return type *_PyRevitVersion*

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W