pyRevit Documentation

Release 4.5

eirannejad

Jul 30, 2018

Getting Started

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.

CHAPTER 1

Anatomy of a pyRevit Script

pyRevit provides a few basic services to python scripts that use its engine. These fuctionalities are accessible through a few high level modules. This is a quick look at these services and their associated python modules.

1.1 Basic script parameters

- **Command Tooltips**

  Tooltips are shown similarly to the other buttons in Revit interface. You can define the tooltip for a script using the doscstring at the top of the script or by explicitly defining __doc__ parameter.

```
"""You can place the docstring (tooltip) at the top of the script file.
This serves both as python docstring and also button tooltip in pyRevit.
You should use triple quotes for standard python docstrings."""
```

```
__doc__ = 'This is the text for the button tooltip associated with this␣
↪script.'
```

- **Custom Command Title**

  When using the bundle name is not desired or you want to add a newline character to the command name for better display inside Revit UI Panel, you can define the __title__ variable in your script and set it to the desired button title.

```
__title__ = 'Sample\\nCommand'
```

- **Command Author**

  You can define the script author as shown below. This will show up on the button tooltip.

```
__author__ = 'Ehsan Iran-Nejad'
```

1.2 Command Availability

Revit commands use standard `IExternalCommandAvailability` class to let Revit know if they are available in different contexts. For example, if a command needs to work on a set of elements, it can tell Revit to deactivate the button unless the user has selected one or more elements.

In pyRevit, command availability is set through the `__context__` variable. Currently, pyRevit support three types of command availability types.

```
# Tool activates when at least one element is selected
__context__ = 'Selection'

# Tools are active even when there are no documents available/open in Revit
__context__ = 'zerodoc'

# Tool activates when all selected elements are of the given category or categories
__context__ = '<Element Category>'
__context__ = ['<Element Category>', '<Element Category>']
```

- **Element Categories**

  `<Element Category>` can be any of the standard Revit element categories. See *Appendix B: System Category Names* for a full list of system categories. You can use the `List` tool under `pyRevit > Spy` and list the standard categories.

  Here are a few examples:

  ```
  # Tool activates when all selected elements are of the given category

  __context__ = 'Doors'
  __context__ = 'Walls'
  __context__ = 'Floors'
  __context__ = ['Space Tags', 'Spaces']
  ```

1.3 pyrevit.script Module

All pyRevit scripts should use the *pyrevit.script* module to access pyRevit functionality unless listed otherwise. pyRevit internals are subject to changes and accessing them directly is not suggested.

Here is a list of supported modules for pyRevit scripts. Examples of using the functionality in these modules are provided on this page.

pyrevit.script

This module provides access to output window (*pyrevit.output*), logging (`pyrevit.coreutils.logger`), temporary files (`pyrevit.coreutils.appdata`), and other misc features. See the module page for usage examples and full documentation of all available functions.

1.3.1 Logging

You can get the default logger for the script using *pyrevit.script.get_logger()*.

```
from pyrevt import script

logger = script.get_logger()
```

(continues on next page)

(continued from previous page)

```
logger.info('Test Log Level :ok_hand_sign:')

logger.warning('Test Log Level')

logger.critical('Test Log Level')
```

Critical and warning messages are printed in color for clarity. Normally debug messages are not printed. you can hold CTRL and click on a command button to put that command in DEBUG mode and see all its debug messages

```
logger.debug('Yesss! Here is the debug message')
```

1.3.2 Controlling Output Window

Each script can control its own output window:

```
from pyrevit import script

output = script.get_output()

output.set_height(600)
output.get_title()
output.set_title('More control please!')
```

See *Effective Output/Input* for more info.

1.3.3 Script Config

Each script can save and load configuration pyRevit's user configuration file:

See *pyrevit.output* for more examples.

See *pyrevit.script.get_config()* and *pyrevit.script.save_config()* for the individual functions used here.

```
from pyrevit import script

config = script.get_config()

# set a new config parameter: firstparam
config.firstparam = True

# saving configurations
script.save_config()

# read the config parameter value
if config.firstparam:
    do_task_A()
```

1.3.4 Logging Results

pyRevit has a usage logging system that can record all tool usages to either a json file or to a web server. Scripts can return custom data to this logging system.

In example below, the script reports the amount of time it saved to the logging system:

```
from pyrevit import script

results = script.get_results()
results.timesaved = 10
```

1.3.5 Using Temporary Files

Scripts can create 3 different types of data files:

- **Universal files**

  These files are not marked by host Revit version and could be shared between all Revit versions and instances. These data files are saved in pyRevit's appdata directory and are NOT cleaned up at Revit restarts.

  See *pyrevit.script.get_universal_data_file()*

  Note: Script should take care of cleaning up these data files.

  ```
  # provide a unique file id and file extension
  # Method will return full path of the data file
  from pyrevit import script
  script.get_universal_data_file(file_id, file_ext)
  ```

- **Data files**

  These files are marked by host Revit version and could be shared between instances of host Revit version Data files are saved in pyRevit's appdata directory and are NOT cleaned up when Revit restarts.

  See *pyrevit.script.get_data_file()*

  Note: Script should take care of cleaning up these data files.

  ```
  # provide a unique file id and file extension
  # Method will return full path of the data file
  from pyrevit import script
  script.get_data_file(file_id, file_ext)
  ```

- **Instance Data files**

  These files are marked by host Revit version and process Id and are only available to current Revit instance. This avoids any conflicts between similar scripts running under two or more Revit instances. Data files are saved in pyRevit's appdata directory (with extension *.tmp*) and ARE cleaned up when Revit restarts.

  See *pyrevit.script.get_instance_data_file()*

  ```
  # provide a unique file id and file extension
  # Method will return full path of the data file
  from pyrevit import script
  script.get_instance_data_file(file_id)
  ```

- **Document Data files**

  (Shared only between instances of host Revit version): These files are marked by host Revit version and name of Active Project and could be shared between instances of host Revit version. Data files are saved in pyRevit's appdata directory and are NOT cleaned up when Revit restarts.

  See *pyrevit.script.get_document_data_file()*

  Note: Script should take care of cleaning up these data files.

```
# provide a unique file id and file extension
# Method will return full path of the data file
from pyrevit import script
script.get_document_data_file(file_id, file_ext)

# You can also pass a document object to get a data file for that
# document (use document name in file naming)
script.get_document_data_file(file_id, file_ext, doc)
```

1.4 Appendix A: Builtin Parameters Provided by pyRevit Engine

Variables listed below are provided for every script in pyRevit.

Note: It's strongly advised not to read or write values from these variables unless necessary. The *pyrevit* module provides wrappers around these variables that are safe to use.

```
# Revit UIApplication is accessible through:
__revit__

# Command data provided to this command by Revit is accessible through:
__commandData__

# selection of elements provided to this command by Revit
__elements__

# pyRevit engine manager that is managing this engine
__ipyenginemanager__

# This variable is True if command is being run in a cached engine
__cachedengine__

# pyRevit external command object wrapping the command being run
__externalcommand__

# information about the pyrevit command being run
__commandpath__              # main script path
__alternatecommandpath__     # alternate script path
__commandname__              # command name
__commandbundle__            # command bundle name
__commandextension__         # command extension name
__commanduniqueid__          # command unique id
```

(continues on next page)

(continued from previous page)

```
# This variable is True if user CTRL-Clicks the button
__forceddebugmode__

# This variable is True if user SHIFT-Clicks the button
__shiftclick__

# results dictionary
__result__
```

1.5 Appendix B: System Category Names

```
Adaptive Points
Air Terminal Tags
Air Terminals
Analysis Display Style
Analysis Results
Analytical Beam Tags
Analytical Beams
Analytical Brace Tags
Analytical Braces
Analytical Column Tags
Analytical Columns
Analytical Floor Tags
Analytical Floors
Analytical Foundation Slabs
Analytical Isolated Foundation Tags
Analytical Isolated Foundations
Analytical Link Tags
Analytical Links
Analytical Node Tags
Analytical Nodes
Analytical Slab Foundation Tags
Analytical Spaces
Analytical Surfaces
Analytical Wall Foundation Tags
Analytical Wall Foundations
Analytical Wall Tags
Analytical Walls
Annotation Crop Boundary
Area Load Tags
Area Tags
Areas
Assemblies
Assembly Tags
Boundary Conditions
Brace in Plan View Symbols
Cable Tray Fitting Tags
Cable Tray Fittings
Cable Tray Runs
Cable Tray Tags
Cable Trays
Callout Boundary
Callout Heads
Callouts
```

(continues on next page)

(continued from previous page)

```
Cameras
Casework
Casework Tags
Ceiling Tags
Ceilings
Color Fill Legends
Columns
Communication Device Tags
Communication Devices
Conduit Fitting Tags
Conduit Fittings
Conduit Runs
Conduit Tags
Conduits
Connection Symbols
Contour Labels
Crop Boundaries
Curtain Grids
Curtain Panel Tags
Curtain Panels
Curtain System Tags
Curtain Systems
Curtain Wall Mullions
Data Device Tags
Data Devices
Detail Item Tags
Detail Items
Dimensions
Displacement Path
Door Tags
Doors
Duct Accessories
Duct Accessory Tags
Duct Color Fill
Duct Color Fill Legends
Duct Fitting Tags
Duct Fittings
Duct Insulation Tags
Duct Insulations
Duct Lining Tags
Duct Linings
Duct Placeholders
Duct Systems
Duct Tags
Ducts
Electrical Circuits
Electrical Equipment
Electrical Equipment Tags
Electrical Fixture Tags
Electrical Fixtures
Electrical Spare/Space Circuits
Elevation Marks
Elevations
Entourage
Filled region
Fire Alarm Device Tags
Fire Alarm Devices
```

(continues on next page)

(continued from previous page)

```
Flex Duct Tags
Flex Ducts
Flex Pipe Tags
Flex Pipes
Floor Tags
Floors
Foundation Span Direction Symbol
Furniture
Furniture System Tags
Furniture Systems
Furniture Tags
Generic Annotations
Generic Model Tags
Generic Models
Grid Heads
Grids
Guide Grid
HVAC Zones
Imports in Families
Internal Area Load Tags
Internal Line Load Tags
Internal Point Load Tags
Keynote Tags
Level Heads
Levels
Lighting Device Tags
Lighting Devices
Lighting Fixture Tags
Lighting Fixtures
Line Load Tags
Lines
Masking Region
Mass
Mass Floor Tags
Mass Tags
Matchline
Material Tags
Materials
Mechanical Equipment
Mechanical Equipment Tags
MEP Fabrication Containment
MEP Fabrication Containment Tags
MEP Fabrication Ductwork
MEP Fabrication Ductwork Tags
MEP Fabrication Hanger Tags
MEP Fabrication Hangers
MEP Fabrication Pipework
MEP Fabrication Pipework Tags
Multi-Category Tags
Multi-Rebar Annotations
Nurse Call Device Tags
Nurse Call Devices
Panel Schedule Graphics
Parking
Parking Tags
Part Tags
Parts
```

(continues on next page)

(continued from previous page)

```
Pipe Accessories
Pipe Accessory Tags
Pipe Color Fill
Pipe Color Fill Legends
Pipe Fitting Tags
Pipe Fittings
Pipe Insulation Tags
Pipe Insulations
Pipe Placeholders
Pipe Segments
Pipe Tags
Pipes
Piping Systems
Plan Region
Planting
Planting Tags
Plumbing Fixture Tags
Plumbing Fixtures
Point Clouds
Point Load Tags
Project Information
Property Line Segment Tags
Property Tags
Railing Tags
Railings
Ramps
Raster Images
Rebar Cover References
Rebar Set Toggle
Rebar Shape
Reference Lines
Reference Planes
Reference Points
Render Regions
Revision Cloud Tags
Revision Clouds
Roads
Roof Tags
Roofs
Room Tags
Rooms
Routing Preferences
Schedule Graphics
Scope Boxes
Section Boxes
Section Line
Section Marks
Sections
Security Device Tags
Security Devices
Shaft Openings
Sheets
Site
Site Tags
Space Tags
Spaces
Span Direction Symbol
```

(continues on next page)

(continued from previous page)

```
Specialty Equipment
Specialty Equipment Tags
Spot Coordinates
Spot Elevation Symbols
Spot Elevations
Spot Slopes
Sprinkler Tags
Sprinklers
Stair Landing Tags
Stair Paths
Stair Run Tags
Stair Support Tags
Stair Tags
Stair Tread/Riser Numbers
Stairs
Structural Annotations
Structural Area Reinforcement
Structural Area Reinforcement Symbols
Structural Area Reinforcement Tags
Structural Beam System Tags
Structural Beam Systems
Structural Column Tags
Structural Columns
Structural Connection Tags
Structural Connections
Structural Fabric Areas
Structural Fabric Reinforcement
Structural Fabric Reinforcement Symbols
Structural Fabric Reinforcement Tags
Structural Foundation Tags
Structural Foundations
Structural Framing
Structural Framing Tags
Structural Internal Loads
Structural Load Cases
Structural Loads
Structural Path Reinforcement
Structural Path Reinforcement Symbols
Structural Path Reinforcement Tags
Structural Rebar
Structural Rebar Coupler Tags
Structural Rebar Couplers
Structural Rebar Tags
Structural Stiffener Tags
Structural Stiffeners
Structural Truss Tags
Structural Trusses
Switch System
Telephone Device Tags
Telephone Devices
Text Notes
Title Blocks
Topography
View Reference
View Titles
Viewports
Views
```

(continues on next page)

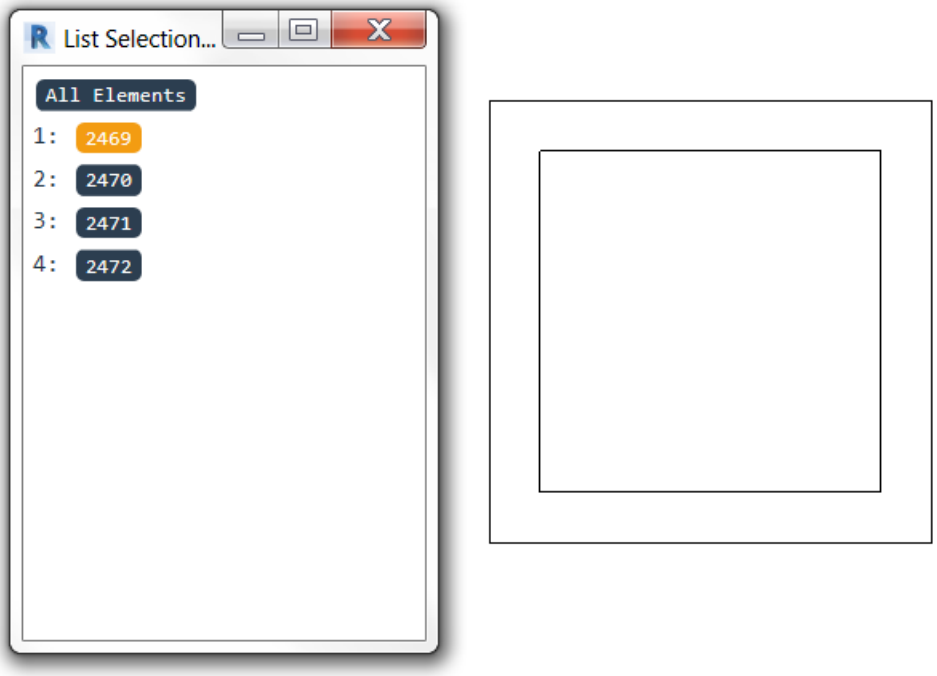
(continued from previous page)

```
Wall Tags
Walls
Window Tags
Windows
Wire Tags
Wires
Zone Tags
```

CHAPTER 2

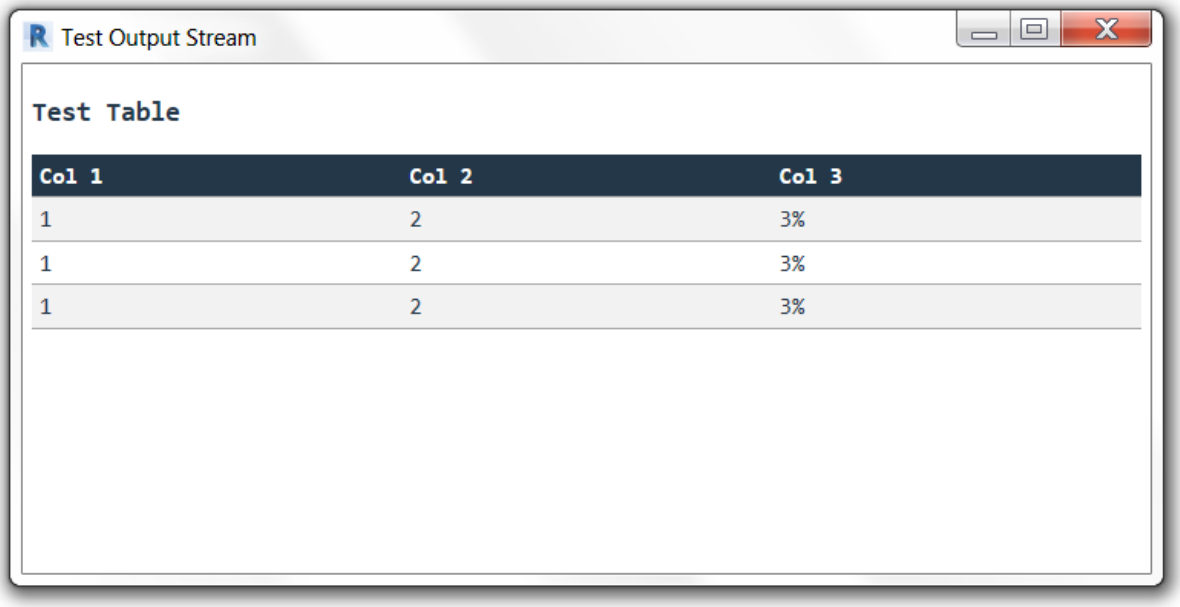
Effective Output/Input

2.1 Clickable Element Links



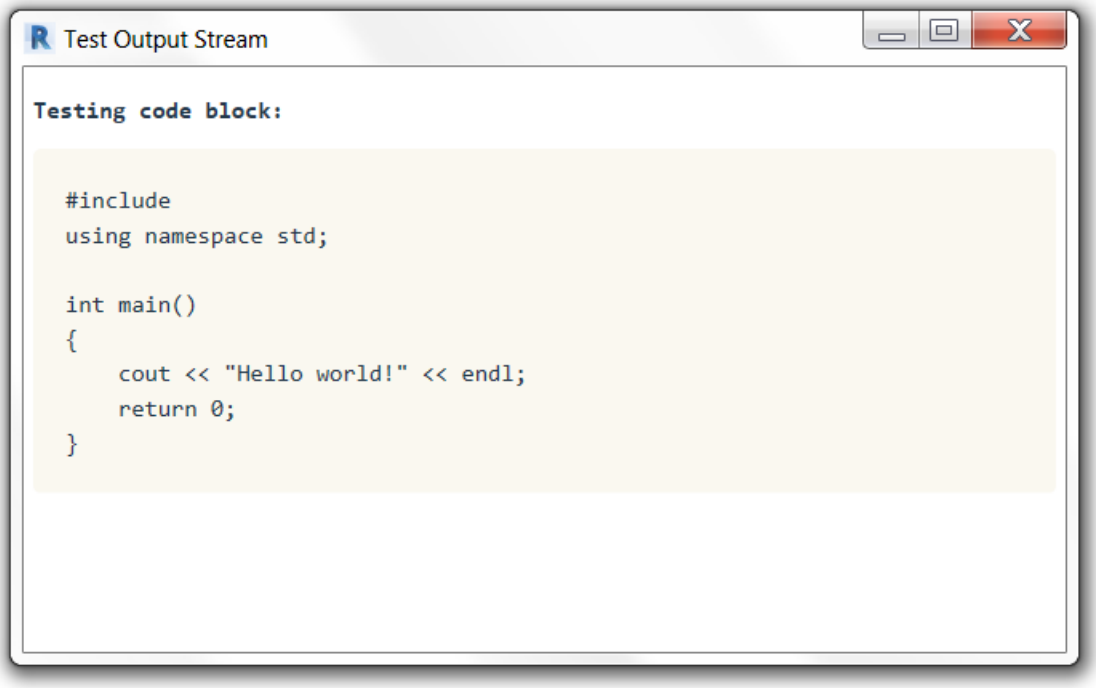
work in progress

2.2 Tables



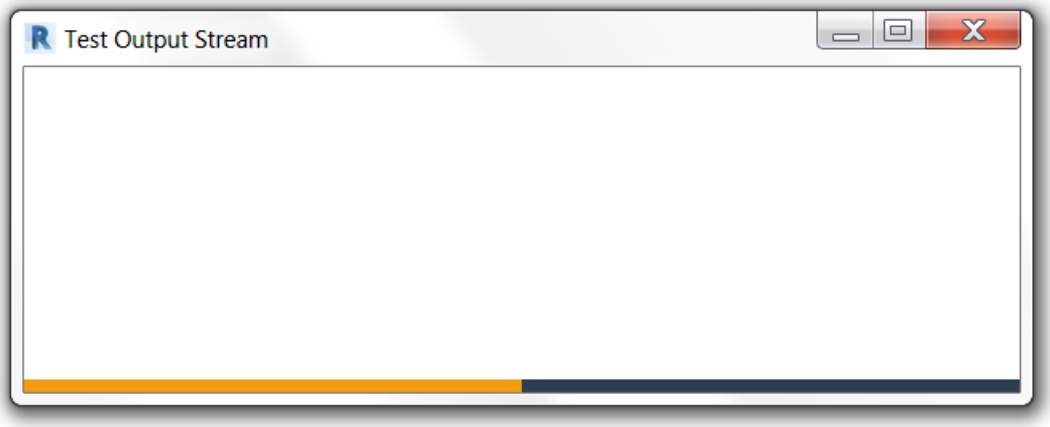
work in progress

2.3 Code Output



work in progress

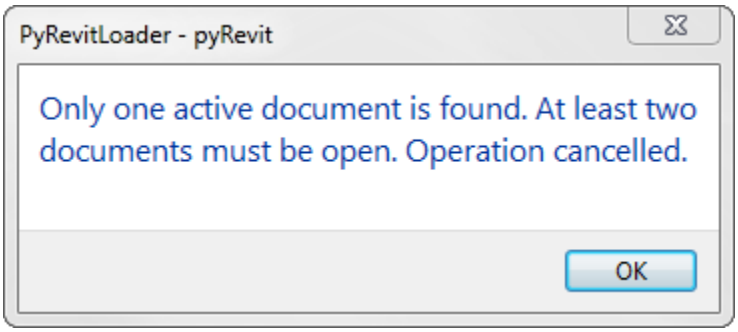
2.4 Progress bars



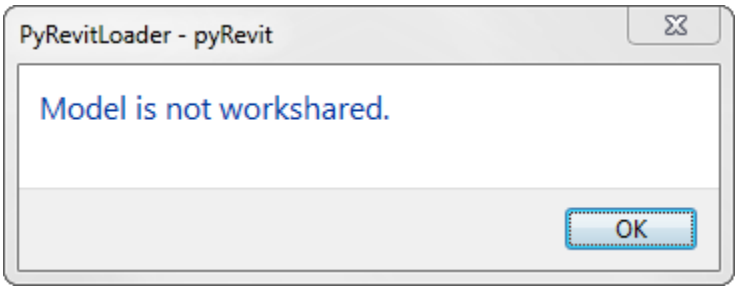
work in progress

2.5 Standard Prompts

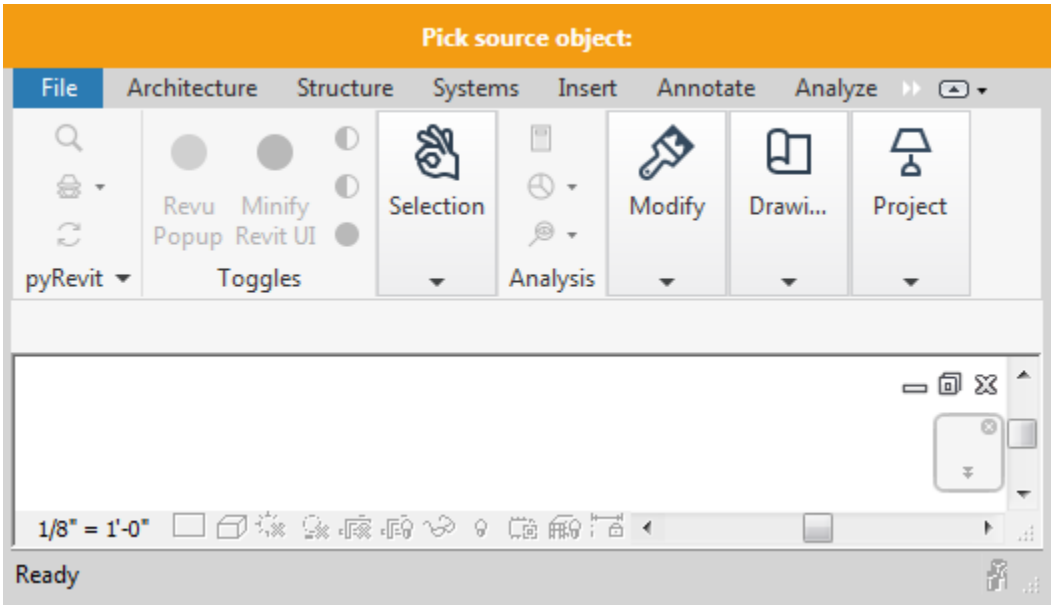
2.5.1 Alerts



pyrevit.forms.**alert**(*msg*, *title='pyRevit'*, *ok=True*, *cancel=False*, *yes=False*, *no=False*, *retry=False*, *exit=False*)



pyrevit.forms.**check_workshared**(*doc*)



class pyrevit.forms.**WarningBar**(*height=32*, ***kwargs*)

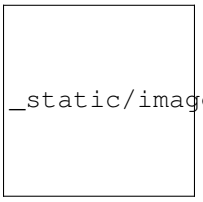
Show warning bar at the top of Revit window.

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

Example

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

2.5.2 Command Options



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

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:

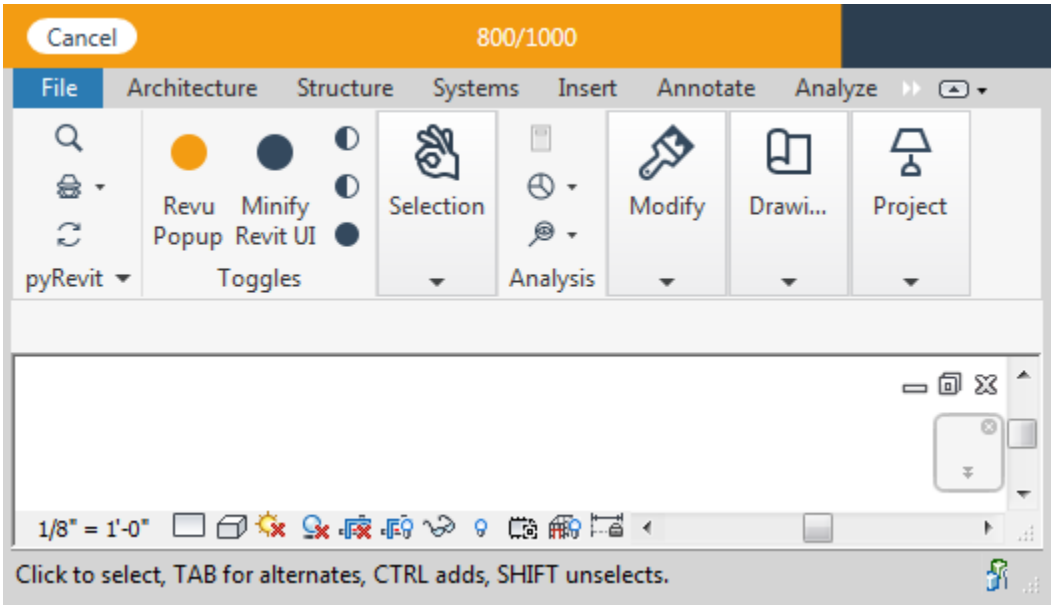
```
>>> from pyrevit import forms
>>> ops = ['option1', 'option2', 'option3', 'option4']
>>> switches = ['switch1', 'switch2']
>>> cfgs = {'option1': { 'background': '0xFF55FF'}}
>>> rops, rswitches = forms.CommandSwitchWindow.show(
...     ops,
...     switches=switches
...     message='Select Option',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

(continues on next page)

(continued from previous page)

```
...     )
>>> rops
'option2'
>>> rswitches
{'switch1': False, 'switch2': True}
```

2.5.3 Showing Progress



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:

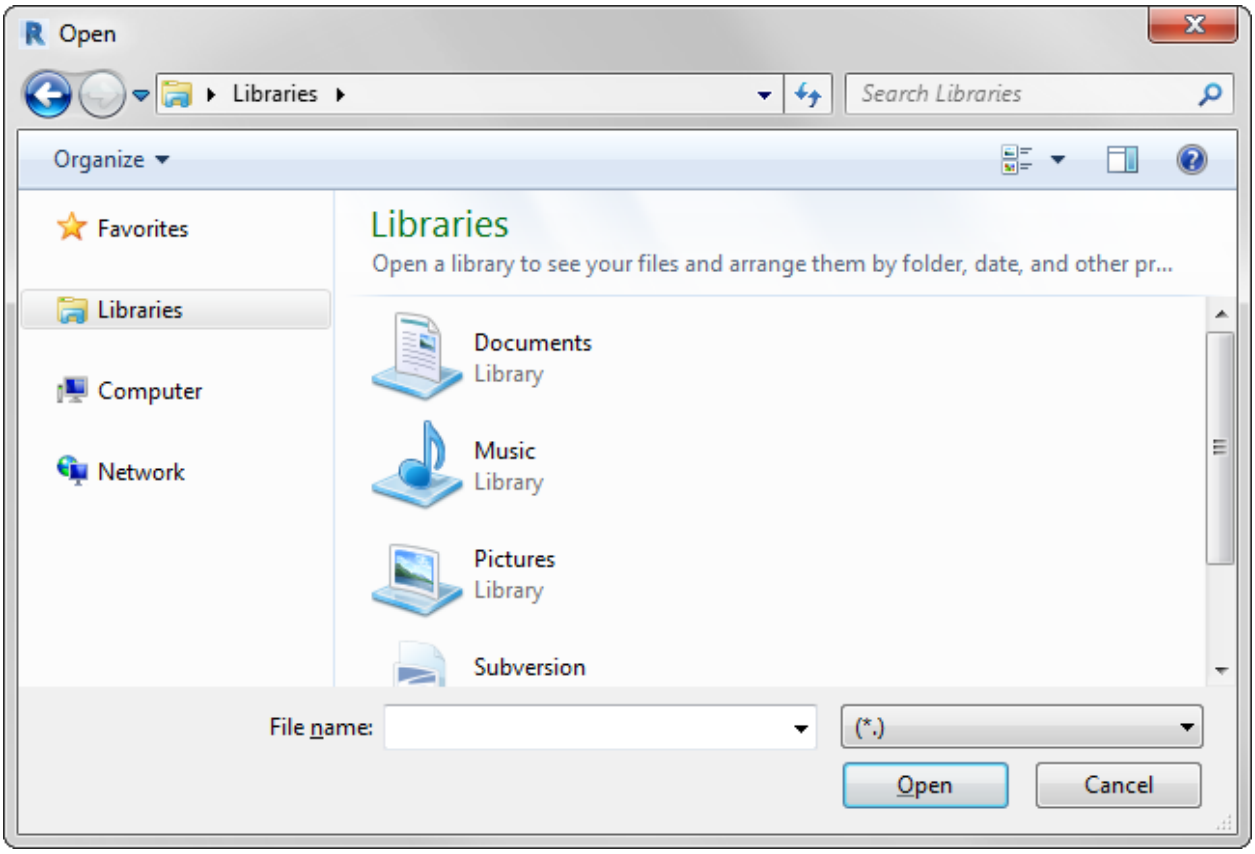
```
>>> with forms.ProgressBar(title='message', indeterminate=True):
```

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
```

2.6 Standard Dialogs

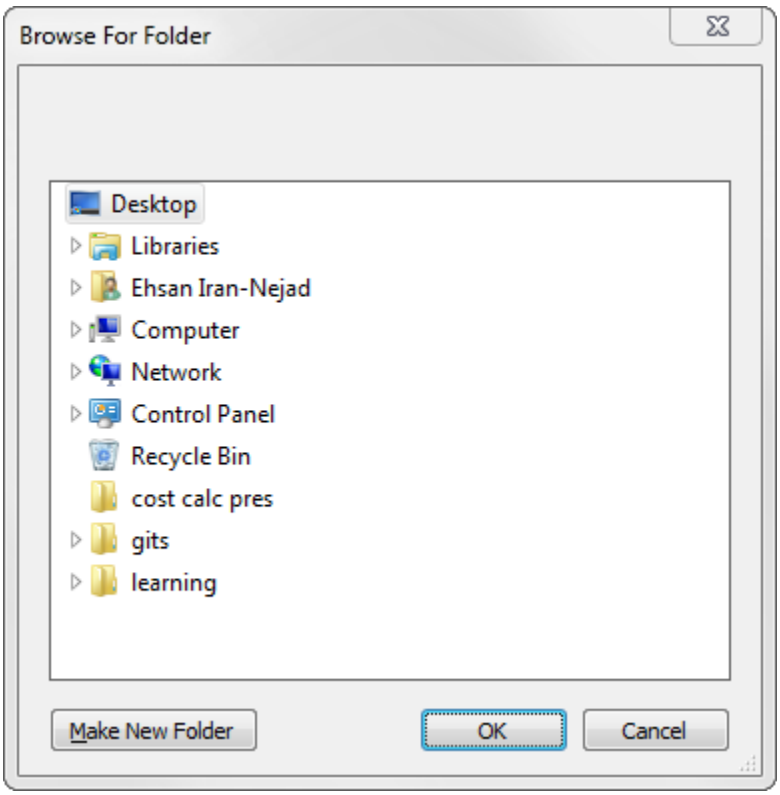
2.6.1 Pick File



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

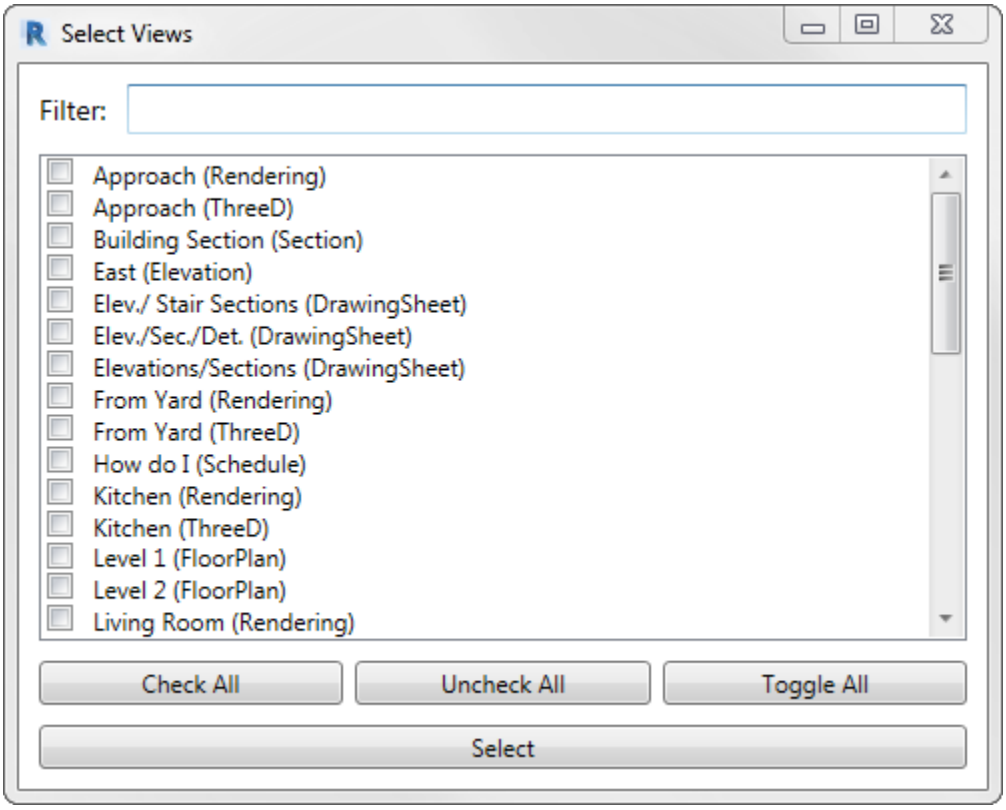
pyrevit.forms.**save_file**(*file_ext=''*, *files_filter=''*, *init_dir=''*, *default_name=''*, *restore_dir=True*, *unc_paths=False*)

2.6.2 Pick Folder



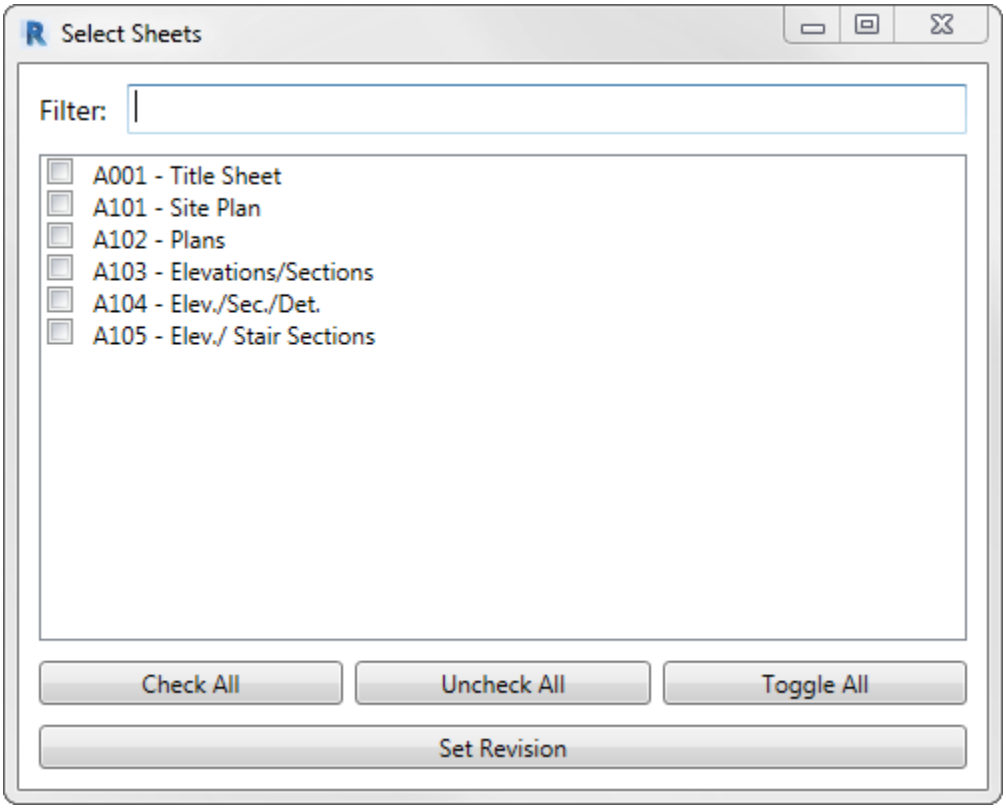
pyrevit.forms.**pick_folder**()

2.6.3 Select Views



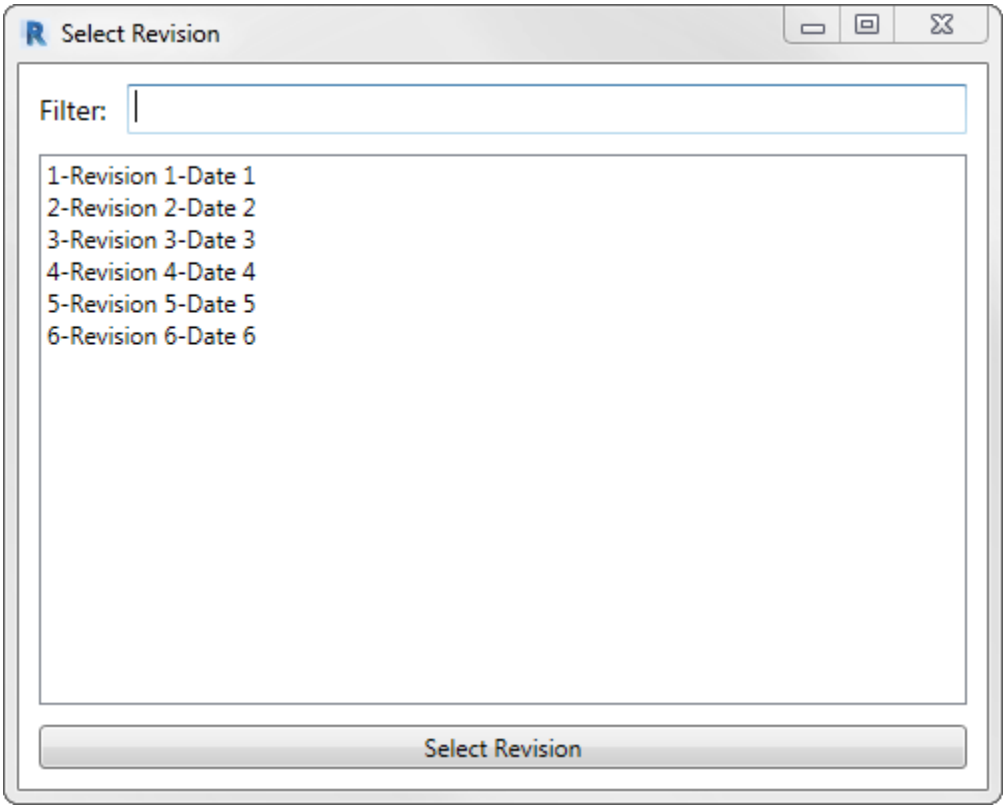
pyrevit.forms.**select_views**(*title='Select Views', button_name='Select', width=500, multiple=True, filterfunc=None, doc=None*)

2.6.4 Select Sheets



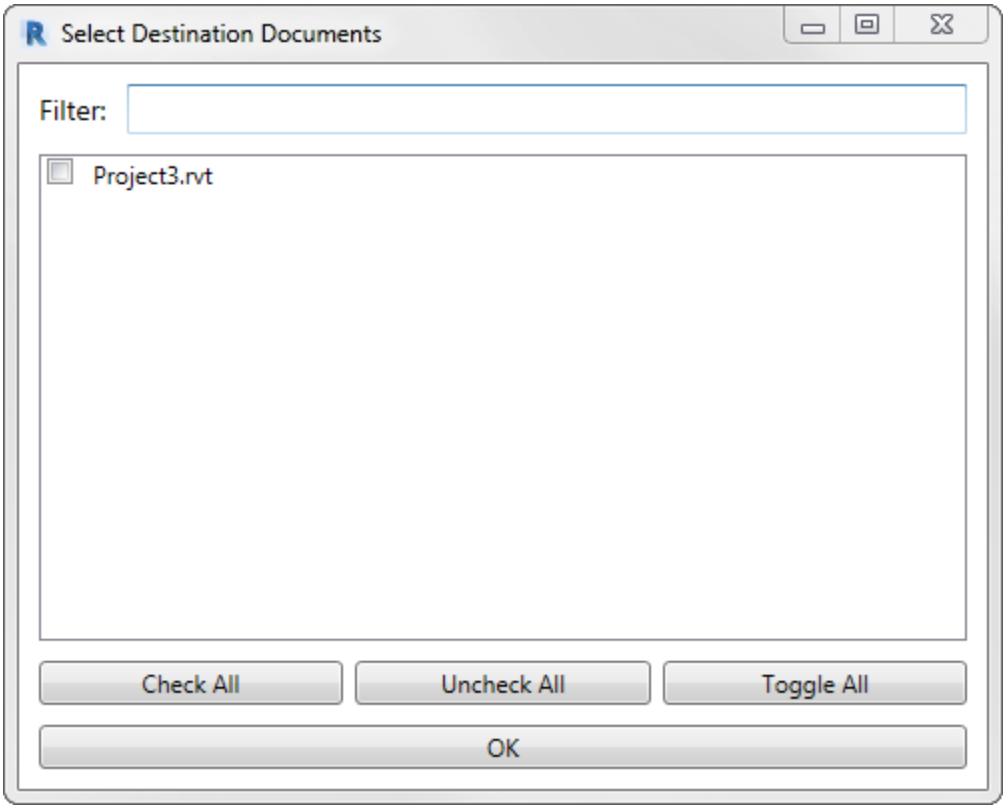
pyrevit.forms.**select_sheets**(*title='Select Sheets', button_name='Select', width=500, multiple=True, filterfunc=None, doc=None*)

2.6.5 Select Revisions



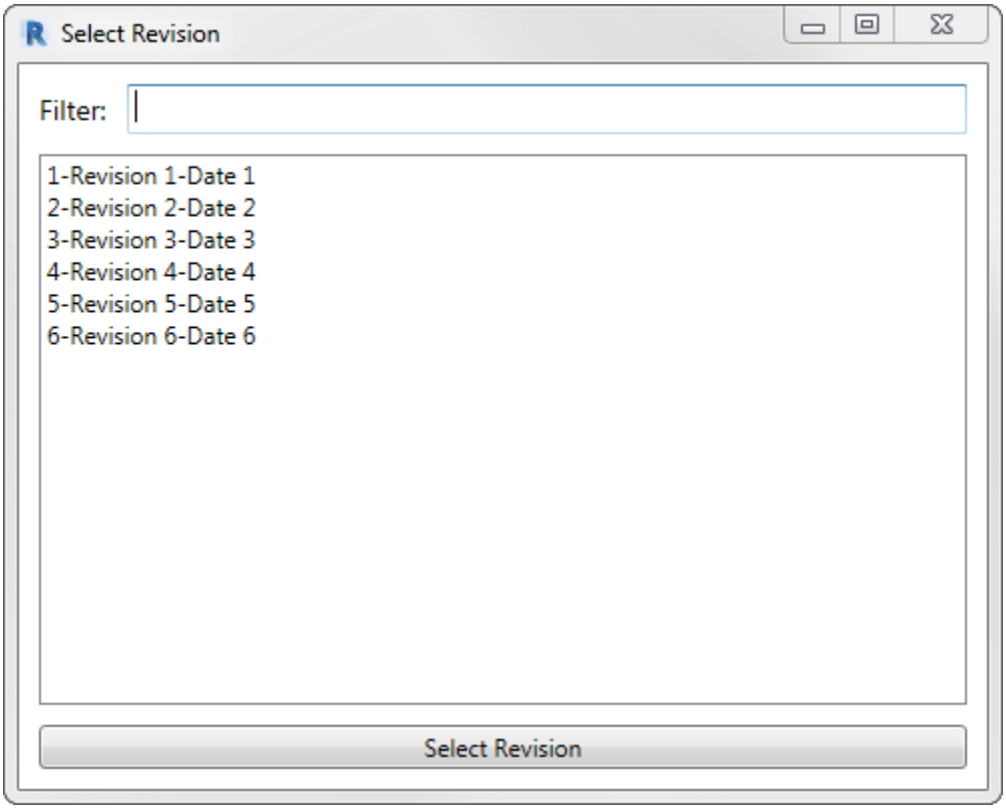
pyrevit.forms.**select_revisions**(*title='Select Revision'*, *button_name='Select'*, *width=500*, *multiselect=True*, *filterfunc=None*, *doc=None*)

2.6.6 Select From Open Documents



pyrevit.forms.**select_dest_docs**()

2.6.7 Select From List



class pyrevit.forms.**SelectFromList**(*context*, *title*, *width*, *height*, ***kwargs*)

Standard form to select from a list of items.

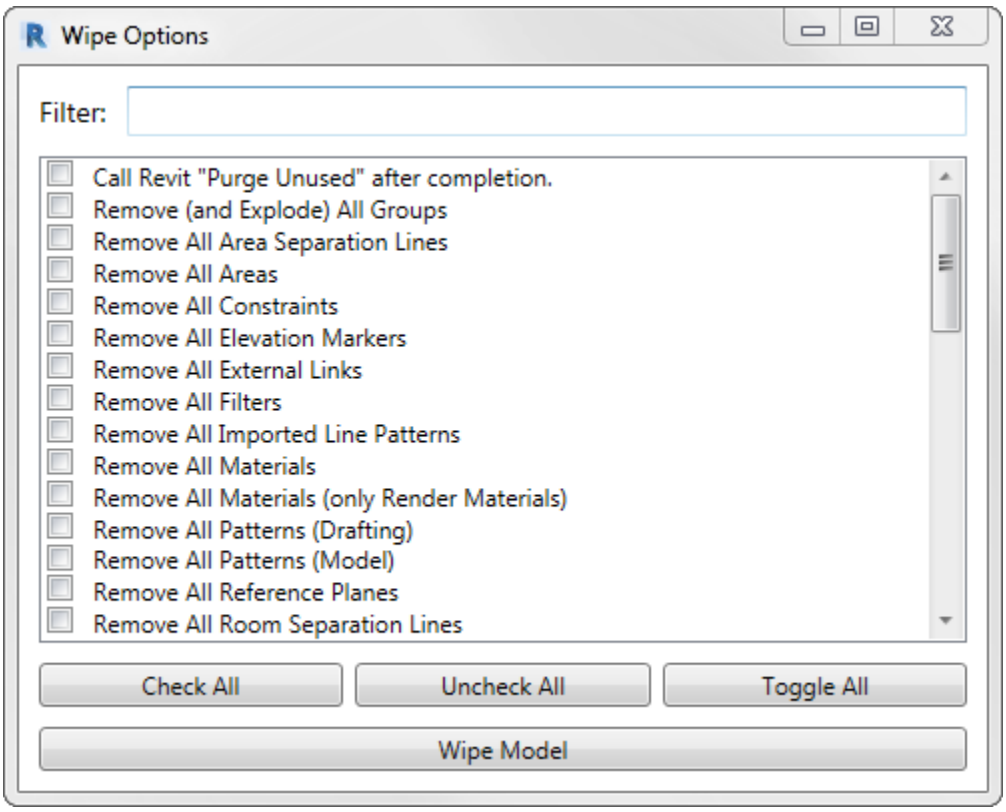
Parameters

- **context** (*list[str]*) – list of items to be selected from
- **title** (*str*) – window title
- **width** (*int*) – window width
- **height** (*int*) – window height
- **button_name** (*str*) – name of select button
- **multiselect** (*bool*) – allow multi-selection

Example

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

2.6.8 Select From Checkbox List



class pyrevit.forms.**SelectFromCheckBoxes**(*context*, *title*, *width*, *height*, ***kwargs*)

Standard form to select from a list of check boxes.

Check box items passed in context to this standard form, must implement `name` and `state` parameter and `__nonzero__` method for truth value testing.

Parameters

- **context** (*list[object]*) – list of items to be selected from
- **title** (*str*) – window title
- **width** (*int*) – window width
- **height** (*int*) – window height
- **button_name** (*str*) – name of select button

Example

```
>>> from pyrevit import forms
>>> class MyOption(object):
...     def __init__(self, name, state=False):
...         self.state = state
...         self.name = name
...
...     def __nonzero__(self):
...         return self.state
```

(continues on next page)

(continued from previous page)

```
...
...     def __str__(self):
...         return self.name
>>> ops = [MyOption('op1'), MyOption('op2', True), MyOption('op3')]
>>> res = forms.SelectFromCheckBoxes.show(ops,
...                                       button_name='Select Item')
>>> [bool(x) for x in res]  # or [x.state for x in res]
[True, False, True]
```

This module also provides a wrapper base class *BaseCheckBoxItem* 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.BaseCheckBoxItem)
...     @property
...     def name(self):
...         return '{} - {}{}'.format(self.item.SheetNumber,
...                                   self.item.SheetNumber)
>>> ops = [MyOption('op1'), MyOption('op2', True), MyOption('op3')]
>>> res = forms.SelectFromCheckBoxes.show(ops,
...                                       button_name='Select Item')
>>> [bool(x) for x in res]  # or [x.state for x in res]
[True, False, True]
```

class pyrevit.forms.**BaseCheckBoxItem**(*orig_item*)

Base class for checkbox option wrapping another object.

2.7 Base Forms

2.7.1 Input Dialog for pyRevit Search

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, switches = forms.SearchPrompt.show(
...     search_db=['target1', 'target2', 'target3', 'target4'],
...     switches=['/switch1', '/switch2'],
...     search_tip='pyRevit Search'
...     )
... matched_str
'target1'
... switches
{'/switch1': True, '/switch2': False}
```

2.7.2 Generic Forms

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()*

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()*

class pyrevit.forms.**WPFWindow**(*xaml_source*, *literal_string=False*)
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

Example

```
>>> from pyrevit import forms
>>> layout = '<Window ShowInTaskbar="False" ResizeMode="NoResize" ' \
>>>          'WindowStartupLocation="CenterScreen" ' \
>>>          'HorizontalContentAlignment="Center">' \
>>>          '</Window>'
```

(continues on next page)

(continued from previous page)

```
>>> w = forms.WPFWindow(layout, literal_string=True)
>>> w.show()
```

2.8 Graphs

Step 1: Create a chart object for the chart type that you want. We'll add data to this later...

```
from pyrevit import script

output = script.get_output()

# Line chart
chart = output.make_line_chart()
# Bar chart
chart = output.make_bar_chart()
# Bubble chart
chart = output.make_bubble_chart()
# Radar chart
chart = output.make_radar_chart()
# Polar chart
chart = output.make_polar_chart()
# Pie chart
chart = output.make_pie_chart()
# Doughnut chart
chart = output.make_doughnut_chart()
```

Step 1-a: Optional: Setup the chart title, and other options. the full list of options for every chart is available on Charts.js Documentation page. Some of the properties have their own sub-properties, for example the `title` option for the charts has multiple sub-properties as shown below. The value for these type of properties should be a dictionary of the sub-properties you'd like to set. All this is explained clearly in the Charts.js Documentation

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

chart.options.title = {'display': True,
                       'text':'Chart Title',
                       'fontSize': 18,
                       'fontColor': '#000',
                       'fontStyle': 'bold'}
```

Step 2: Now let's add data to the chart. Every chart object has a data property `chart.data` that we can interact with to add datasets to the chart. Different types of charts need different types of data sets in terms of how data is organized, so the chart can present multiple data sets correctly. I'm providing two examples here, one for a simple line chart (showing 3 different data sets) and another for a radial chart (also showing 3 different data sets within the same chart). They're all very similar to each other though.

```
# setting the charts x line data labels
chart.data.labels = ['Monday', 'Tuesday',
                     'Wednesday', 'Thursday',
                     'Friday', 'Saturday', 'Sunday']

# Let's add the first dataset to the chart object
# we'll give it a name: set_a
set_a = chart.data.new_dataset('set_a')
```

(continues on next page)

(continued from previous page)

```
# And let's add data to it.
# These are the data for the Y axis of the graph
# The data length should match the length of data for the X axis
set_a.data = [12, 19, 3, 17, 6, 3, 7]
# Set the color for this graph
set_a.set_color(0xFF, 0x8C, 0x8D, 0.8)
```

Step 3: The last step is to ask the chart object to draw itself.

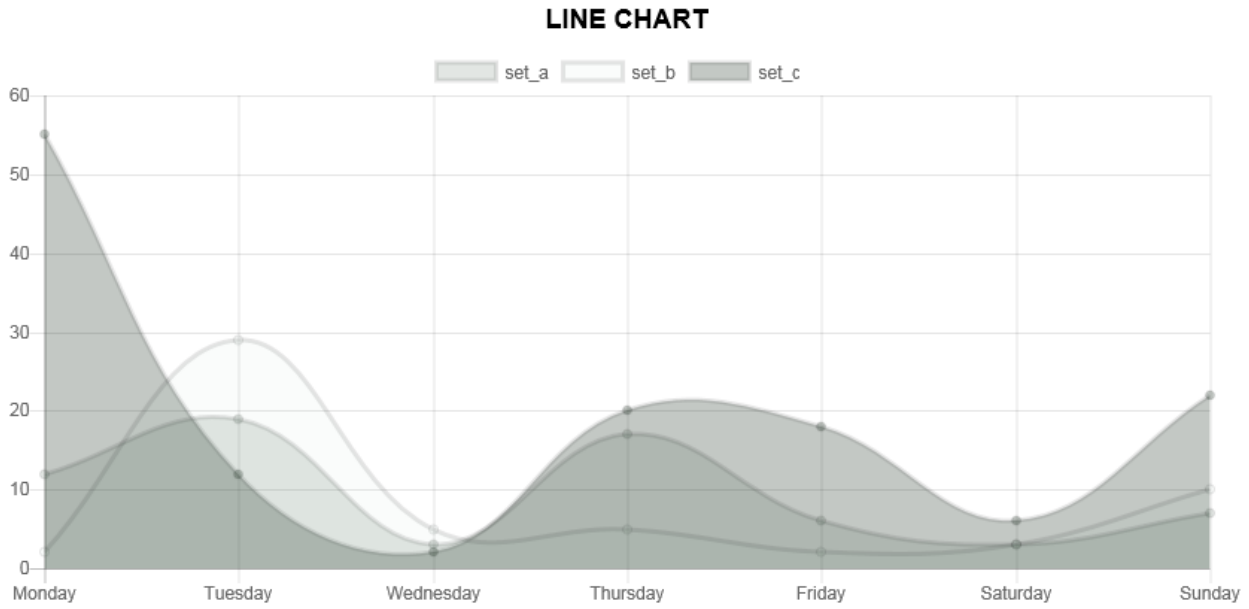
```
# Before drawing the chart you can randomize the colors
# if you have not added any color to the datasets.
chart.randomize_colors()

# Finally let's draw the chart
chart.draw()
```

2.8.1 Line charts

See the comments in the script for more info

```
# get line chart object
chart = output.make_line_chart()

# this is a list of labels for the X axis of the line graph
chart.data.labels = ['Monday', 'Tuesday',
                     'Wednesday', 'Thursday',
                     'Friday', 'Saturday', 'Sunday']

# Let's add the first dataset to the chart object
# we'll give it a name: set_a
set_a = chart.data.new_dataset('set_a')
# And let's add data to it.
# These are the data for the Y axis of the graph
# The data length should match the length of data for the X axis
set_a.data = [12, 19, 3, 17, 6, 3, 7]
# Set the color for this graph
set_a.set_color(0xFF, 0x8C, 0x8D, 0.8)
# You can also set custom options for this graph
# See the Charts.js documentation for all the options
set_b.fill = False

# Same as above for a new data set: set_b
set_b = chart.data.new_dataset('set_b')
# Obviously a different set of data and a different color
set_b.data = [2, 29, 5, 5, 2, 3, 10]
set_b.set_color(0xFF, 0xCE, 0x56, 0.8)

# Same as above for a new data set: set_c
set_c = chart.data.new_dataset('set_c')
# Obviously a different set of data and a different colorset_c.data = [55, 12, 2, 20,␣
↪18, 6, 22]
set_c.set_color(0x36, 0xA2, 0xEB, 0.8)
```

And here is the result:

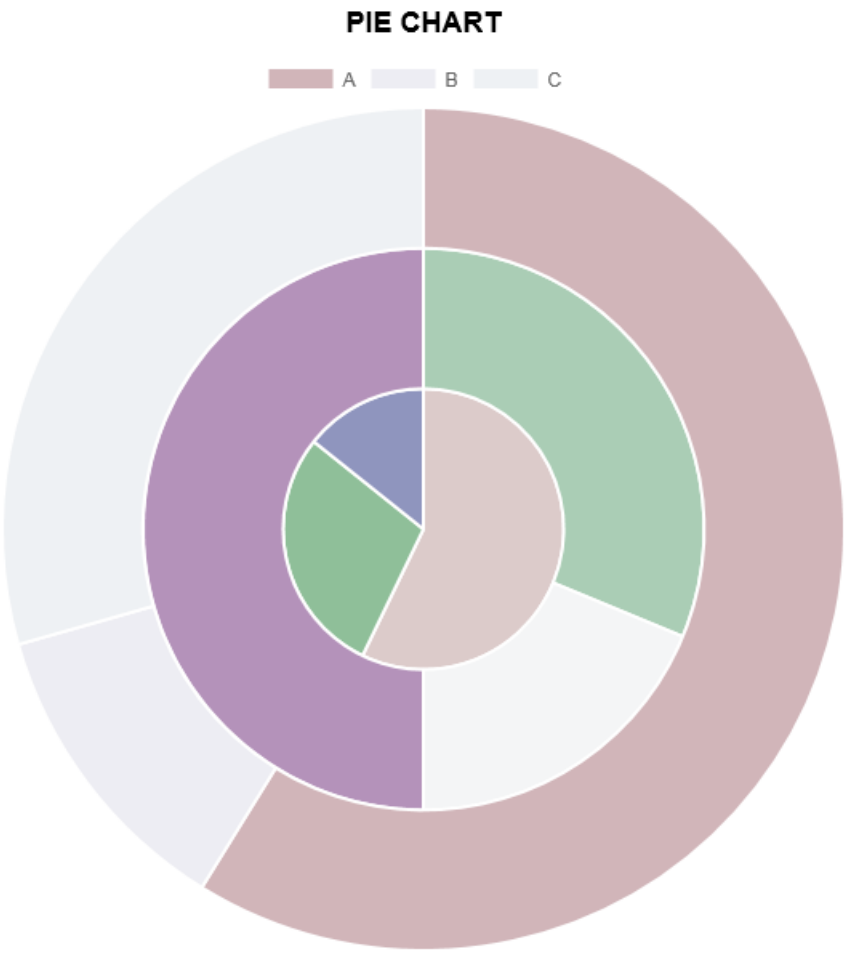


2.8.2 Pie charts

See the comments in the script for more info

```
# get pie chart object
chart = output.make_pie_chart()

# Set the labels for the circumference axis
chart.data.labels = ['A', 'B', 'C']

# Create new data sets
set_a = chart.data.new_dataset('set_a')
set_a.data = [100, 20, 50]
# You can set a different color for each pie of the chart
set_a.backgroundColor = ["#560764", "#1F6CB0", "#F98B60"]

set_b = chart.data.new_dataset('set_b')
set_b.data = [50, 30, 80]
set_b.backgroundColor = ["#913175", "#70A3C4", "#FFC057"]

set_c = chart.data.new_dataset('set_c')
set_c.data = [40, 20, 10]
set_c.backgroundColor = ["#DD5B82", "#E7E8F5", "#FFE084"]
```

And here is the result:

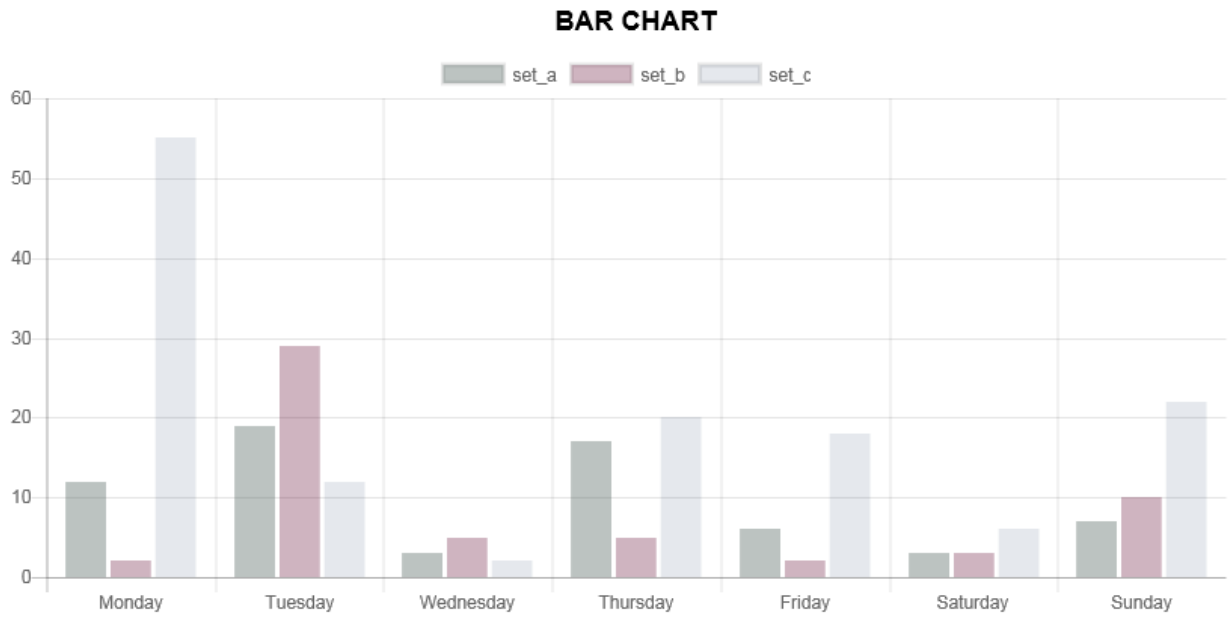


2.8.3 Bar charts

See the comments in the script for more info

```
# get bar chart object
chart = output.make_bar_chart()
```

And here is the result:

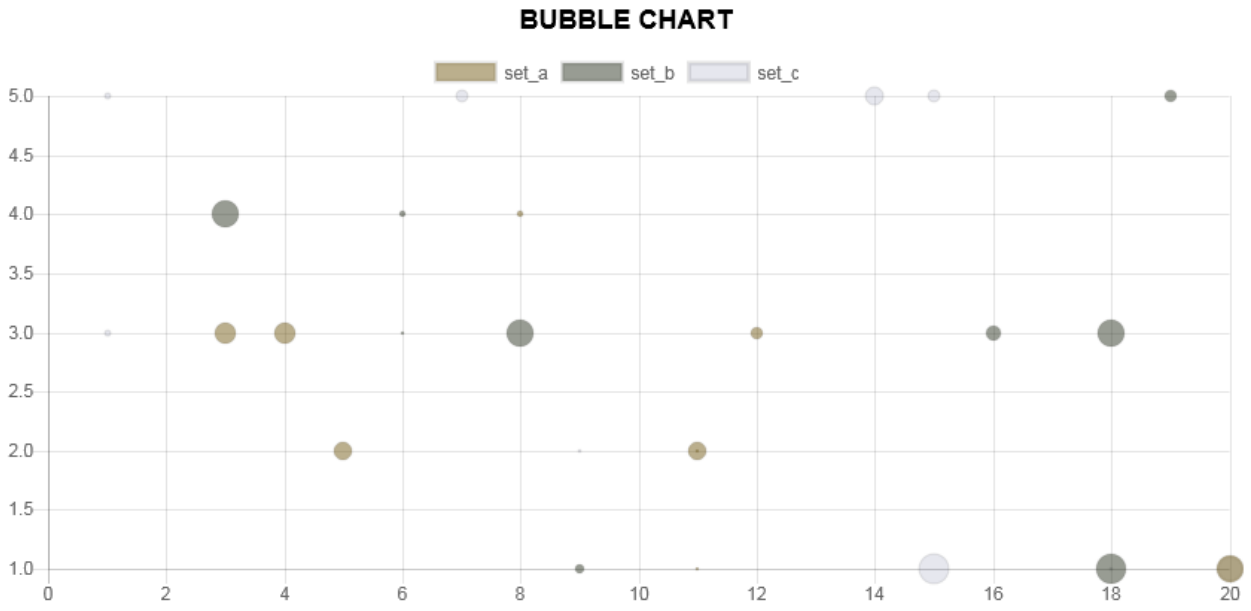


2.8.4 Bubble charts

See the comments in the script for more info

```
# get bubble chart object
chart = output.make_bubble_chart()
```

And here is the result:

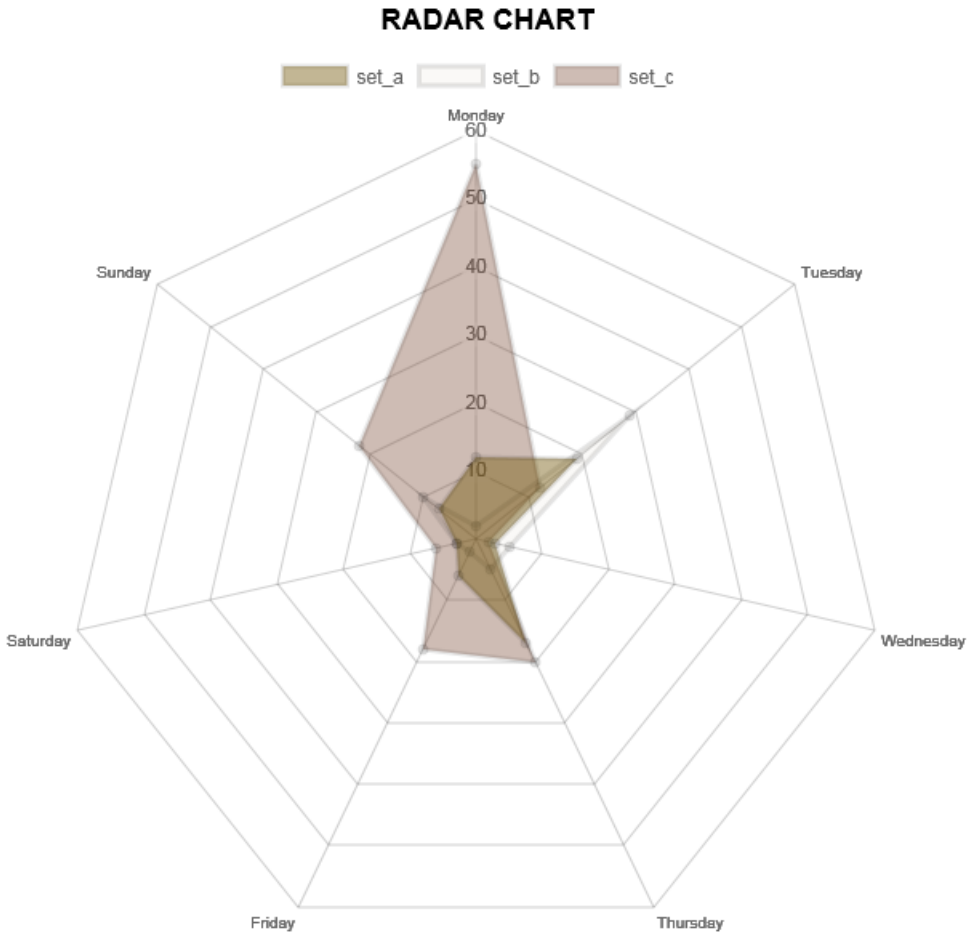


2.8.5 Radar charts

See the comments in the script for more info

```
# get radar chart object
chart = output.make_radar_chart()
```

And here is the result:

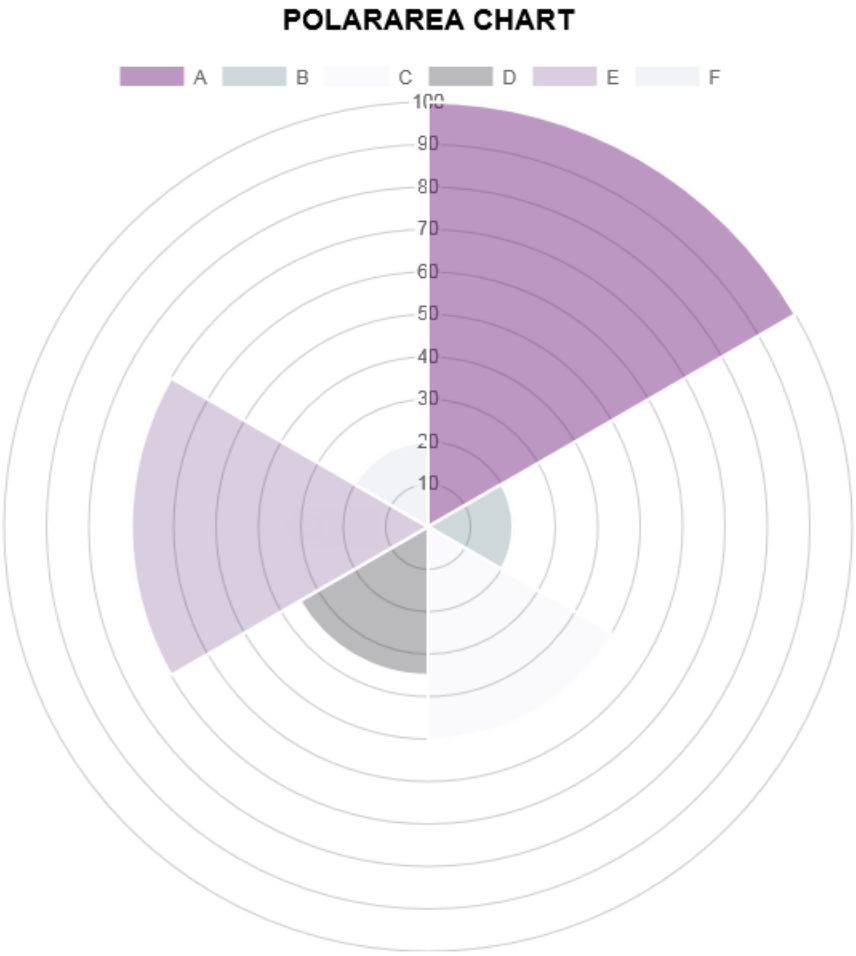


2.8.6 Polar Area charts

See the comments in the script for more info

```
# get polar chart object
chart = output.make_polar_chart()
```

And here is the result:

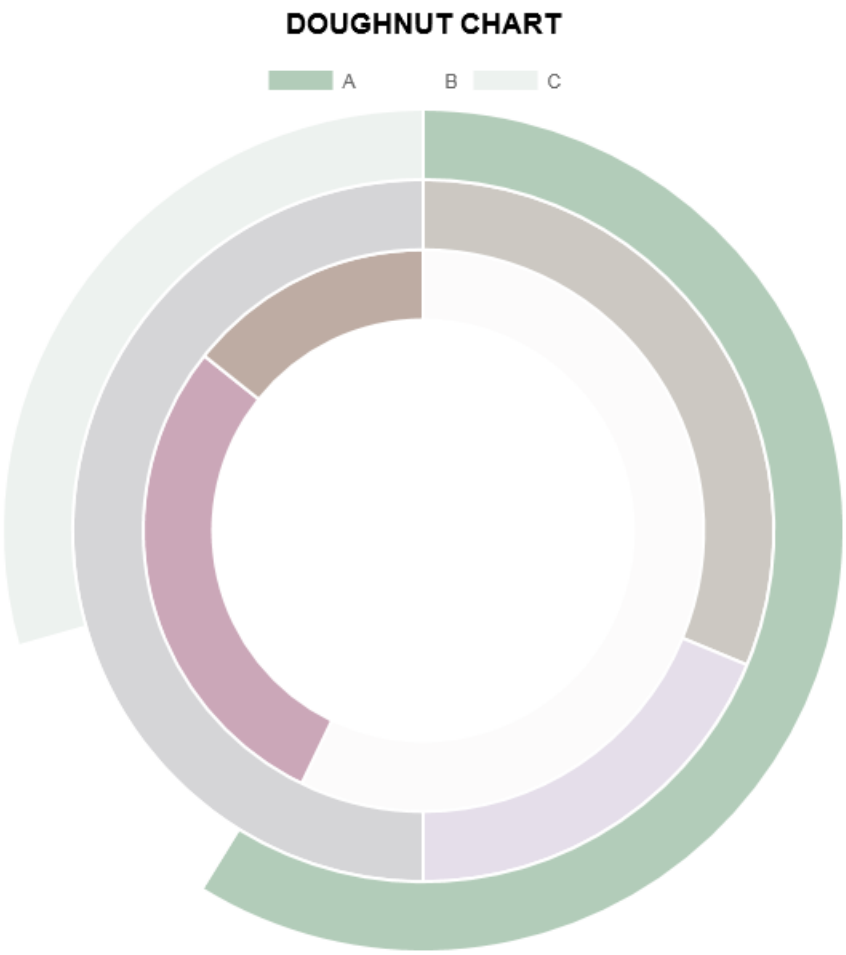


2.8.7 Doghnut charts

See the comments in the script for more info

```
# get doughnut chart object
chart = output.make_doughnut_chart()
```

And here is the result:



2.8.8 Charts engine

Here is a little info on how the charts engine works: the pyRevit charts module is `pyrevit.coreutils.charts`. This is the module that the output window interacts with to create the charts.

The charts module provides the chart object and handles the creation of datasets. The first thing it does when drawing the graph is to create a html `<canvas>` element and assign a unique id to it:

```
<canvas id="chart123456"></canvas>
```

Then it parses the input data and creates a JSON representation of the data. The JSON string (*json_data*) will be inserted into a template javascript. This javascript creates a Chart object from the `Chart.js` library:

```
var ctx = document.getElementById('{}').getContext('2d');
var chart = new Chart(ctx, json_data);
```

and finally, the pyRevit chart object, injects this dynamically created javascript into the <*head*> of the output window `WebBrowser` component:

```
output.inject_script(js_code)
```

CHAPTER 3

Keyboard Shortcuts

3.1 Shift-Click: Alternate/Config Script

Each pyRevit command bundle can contain two scripts:

- `*script.py` is the main script.
- `*config.py` is the Alternale/Config script.

SHIFT-clicking on a ui button will run the alternate/config script. This alternate script is generally used to configure the main tool. Try Shift clicking on the Match tool in pyRevit > Modify panel and see the configuration window. Then try Shift clicking on the Settings tool in pyRevit panel slide-out and see what it does.

If you don't define the configuration script, you can check the value of `__shiftclick__` in your scripts to change script behaviour. This is the method that the Settings command is using to open the config file location in explorer:

```
if __shiftclick__:
    do_task_A()
else:
    do_task_B()
```

3.2 Ctrl-Click: Debug Mode

CTRL-clicking on a ui button will run the script in DEBUG mode and will allow the script to print all debug messages. You can check the value of `__forceddebugmode__` variable to see if the script is running in Debug mode to change script behaviour if neccessary.

```
if __forceddebugmode__:
    do_task_A()
else:
    do_task_B()
```

3.3 Alt-Click: Show Script file in Explorer

ALT-clicking on a ui button will show the associated script file in windows explorer.

3.4 Ctrl-Shift-Alt-Click: Reload Engine

If you're using pyRevit Rocket mode, this keyboard combination will force pyRevit to discard the cached engine for this command and use a new fresh engine. If you are developing scripts for pyRevit and using external modules, you'll need to use this keyboard combination after changes to the imported module source codes. Since the modules are already imported in the cached engine, you'd need a new fresh engine to reload the modules.

3.5 Shift-Win-Click: pyRevit Button Context Menu

Shows the context menu for the pyRevit command. See image below:

CHAPTER 4

Extensions and Commmands

4.1 Why do I need an Extension

pyRevit's extensions system has evolved again to be more flexible and easier to work with. We'll dive right into how you can add your own extension, but first let's answer one important question:

Q: Why would I need to create a separate extension? Why Can't I just add my scripts to the current pyRevit tools?

A: Because pyRevit is a git repository and the real benefit of that is that you can keep it always updated without the need to unsinstall and install the newer versions. To keep this system running without issues, I highly recommend not to mess with the pyRevit git repository folders and contents and pyRevit makes it really easy to add your own extensions. You can even add tools to the standard pyRevit tab in your own extensions. I'll show you how.

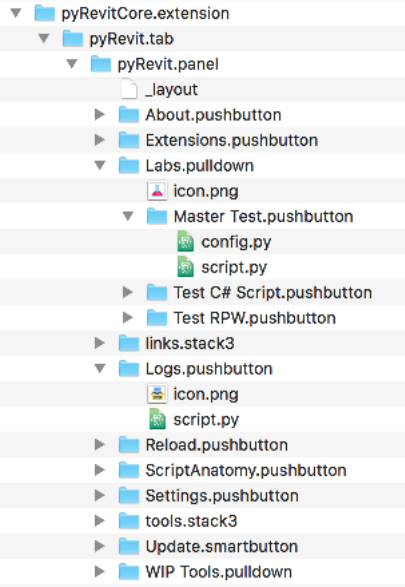
Besides, by creating a separate extension, you'll have all your precious scripts and tools in a safe place and away from the changes being made to the core pyRevit. They can even live somewhere on your company shared drives and be shared between your teams.

4.2 Extensions

Each extension is a group of tools, organized in bundles to be easily accessible through the user interface.

Extensions are organized in a folder bundle with `.extension` postfix.

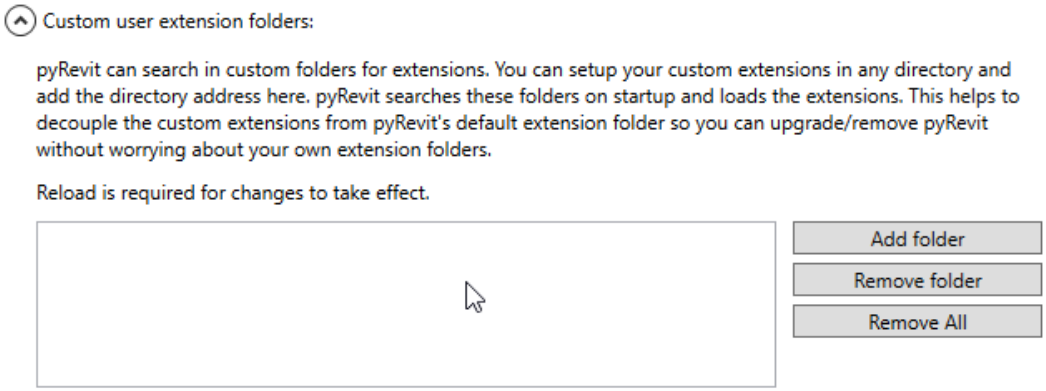
Like this one:



There are two steps that you need to follow to create your own extensions:

- Setup external extension folder:

  First, is to create a separate folder for all your custom extensions and tell pyRevit to load your extensions from this folder. This is done in the Settings window, under the Custom Extension folders section. This way your precious extensions will stay out of the pyRevit installation and are safe.



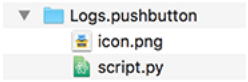
- Create custom extension bundle:

  Next, create your `<your extension name>.extension` folder under your custom extensions folder. Read the sections below on how to create bundles for your commands and the user interface.

4.3 Command Bundles

A bundle is a folder named in the format `bundle_name.bundle_type`.

Like these:



The most basic bundle is a command bundle. There are more than one type of command bundles but a `.pushbutton` bundle explained here covers 90% of the use cases.

4.3.1 Pushbutton Bundle

Each command bundle needs to include a script either in python or C#:

script.py:

> The first script file under the bundle that ends with `script.py` will be used as the script for this command bundle.
>
> Examples: `BuildWall_script.py` `Analyse-script.py`

config.py:

> This for python commands configuration. If this script is provided then Shift-Clicking on the button will run this command instead. Also a black dot will be added to the button name in the user interface to show that this command has a custom configuration tool. See *Shift-Click: Alternate/Config Script*

script.cs:

> This for C# commands and works similarly to python scripts. This C# script will be compiled in runtime.

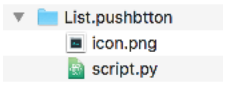
icon.png:

> Command bundles can include an icon for their user interface.

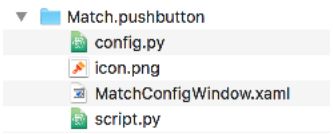
lib/:

> Bundles can define a python library (a sub-folder named *lib* inside the bundle will do). This library will be accessible to the python script in this bundle. This organizes all the python modules that are necessary for this python script to work into one folder.

This is how a command bundle looks like:



And this is a more advanced command bundle with a configuration script and configuration window definition file:



4.4 Group Bundles

Now that we have explained the command bundles, we need a way to organize these commands into a user-friendly interface. Let's introduce **Group Bundles**

A group bundle is a bundle that can contain command bundles and other group bundles. They come in all different shapes and sizes but they have a few features in common:

- They can contain command bundles and other group bundles. (But I've already said that)

- **icon.png**: Bundle can include an icon for their user interface.
- **lib/**: The can define a python library (a sub-folder named `lib` inside the bundle will do). This library will be accessible to all the commands in this bundle and other child group bundles. This folder can contain all the python modules that are being shared between the child commands.
- **_layout**: This is a text file inside the bundle that defines the order in which the bundle contents should be created in the user interface. The contents of this file should be the names of the component in the order that they should be created in the user interface.

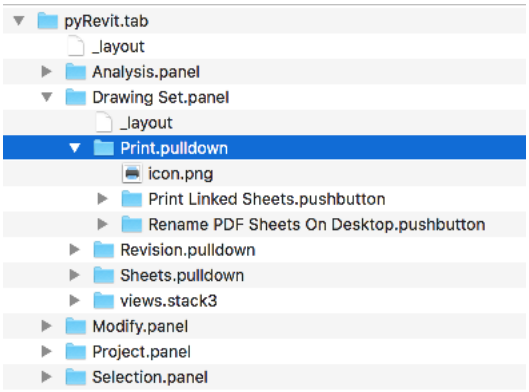
Here is `_layout` file example. This is a layout file for a Group Bundle that has a series of push buttons and other group bundles under itself:

```
PushButton A
PushButton B
PullDown A
---
PullDown B
Stack3 A
>>>
PushButton C
PullDown C
```

Oh, and also:

- `---` This line will add a separator to the interface (You can use more than 3 - characters. For example `----------` still works as a separator)
- `>>>` Any bundle after this line will be created inside a slide-out. This works for panel bundles only. (You can use more than 3 > characters. For example `>>>>>>>>>` still works as a slide-out)

And this is how a typical Group Bundle looks like:



Now let's talk about the different Group Bundles:

4.4.1 Tab Bundle

This bundle creates a Tab in the Ribbon with the bundle name.

Example	**Can Contain**
`pyRevit.tab`	Only `.panel` Group Bundles.

4.4.2 Panel Bundle

This bundle creates a Panel in a Ribbon Tab with the bundle name.

Example	**Can Contain**
`pyRevit.panel`	Any other bundle type

4.4.3 PullDown Bundle

This bundle creates a Pulldown Button in a Ribbon Panel or a Stack, with the bundle name and icon.

Example	**Can Contain**
`pyRevit.pulldown`	Only command bundles

4.4.4 SplitButton Bundle

This bundle creates a Split Button button in a Ribbon Panel or a Stack, with the bundle name and icon.

Example	**Can Contain**
`pyRevit.splitbutton`	Only command bundles

4.4.5 SplitPushButton Bundle

This bundle creates a Split Push Button button (The sticky split button) in a Ribbon Panel or a Stack, with the bundle name and icon.

Example	**Can Contain**
`pyRevit.splitpushbutton`	Only command bundles

4.4.6 Stack Bundle: Two Buttons

This bundle creates a stack of 2 buttons in a panel.

Example	**Can Contain**
`pyRevit.stack2`	Can contain: *.pulldown .splitbutton .splitpushbutton*

4.4.7 Stack Bundle: Three Buttons

Just like the *.stack2* bundle but with 3 buttons instead.

4.5 Advanced Bundles

There are a few more advanced bundle types in pyRevit as well. Here is some quick intro on these bundles.

4.5.1 Smart Button Bundle

Example
pyRevit.smartbutton

Smart buttons are python scripts that are written like modules. They should define `__selfinit__` function as shown below. This function gets executed at startup time to give a chance to the button to initialize itself (e.g set its icon based on its state).

The `__selfinit__` must return `True` if the initialization is successful and `False` if it is not. pyRevit will not create the button if the initialization returns `False` and is unsuccessful.

```
def __selfinit__(script_cmp, ui_button_cmp, __rvt__):
    """
    Args:
        script_cmp: script component that contains info on this script
        ui_button_cmp: this is the UI button component
        __rvt__: Revit UIApplication

    Returns:
                    bool: Return True if successful, False if not
    """

    run_self_initialization()
```

4.5.2 No Button Bundle

Example
pyRevit.nobutton

No-Button bundles are just like Pushbutton bundles except that they will never show up inside Revit UI and thus don't need any icons. The only method to run these commands is through pyRevit Search tool. These commands are meant for more advanced commands that not every user needs.

4.5.3 Panel Button Bundle

Example
pyRevit.panelbutton

Panle Button bundles are just like Pushbutton bundles except that they will be set as the panel configuration button (small arrow at the corner of UI Panels). These bundles do not need to have an icon as the standard Small arrow icon is used for panel configuration buttons by default. These commands work just like any pyRevit command but their primary purpose should be to configure set of related tools in a panel.

4.5.4 Link Button Bundle

Example
pyRevit.linkbutton

Link buttons can call a function from another Addin. To make a link button define the parameters below in the bundles `script.py`:

Note: For this button to work properly, the target addin must be already loaded when this button is being created, otherwise Revit can not tie the UI button to an assembly that is not loaded.

```
__assembly__ = 'Addin assembly name'
__commandclass__ = 'Class name for the command'
```

For example to call the Interactive Python Shell from RevitPythonShell addin:

```
__assembly__ = 'RevitPythonShell'
__commandclass__ = 'IronPythonConsoleCommand'
```

4.6 Other Extension Types

4.6.1 Library Extensions

Library extensions are created to share IronPython modules between all extensions. They're in essence IronPython module packages. Some users might decide to develop an IronPython library (e.g. RevitPythonWrapper Library) that other users can use in their tools and benefit from.

Library extensions are identified by `.lib` postfix. The library extension folder address will be added to the `sys.path` of all the other extensions by the loader.

CHAPTER 5

pyRevit Configuration

work in progress

CHAPTER 6

Usage Logger

work in progress

CHAPTER 7

pyRevit Installer

work in progress

pyRevit Core

- *Load Sequence, Step 1: Revit Addon*
- *Load Sequence, Step 2: IronPython Module*

CHAPTER 8

Load Sequence, Step 1: Revit Addon

8.1 The Complex Relationship of a C# Addin and a Python Script

Let's talk basics:

- Revit Addons are written in C# and are windows `.dll` files.
- pyRevit is written as an IronPython module. (actually a bit more complex than that)
- Revit doesn't have an option to run external python scripts.

Thus, we need a way to teach Revit how to run a python script when it's starting up.

The solution was to create a custom C# addin to create a python engine and run a script. We'll call this addin `pyRevitLoader.dll`. I wanted to keep this addin as simple as possible since it's the only statically-compiled piece of code in this project. The rest of the task of loading pyRevit were assigned to a loader python script that is being run by the loader addin.

So:

- `pyRevitLoader.dll` is a simple C# addin for Revit that runs python scripts
- `pyRevitLoader.dll` loads `pyRevitLoader.py` at startup.
- `pyRevitLoader.py` sets up the environment and loads pyRevit.

It's that simple really. See the sources below.

From here on, the documentation page for the *pyrevit.loader* module will take you through all the steps of parsing extensions, making dll assemblies and creating the user interface for the parsed extensions.

8.2 pyRevit loader script

Here is the full source of `pyRevitLoader.py`. The docstring explains how it works.

```
# -*- coding: utf-8 -*-
"""
This is the starting point for pyRevit. At Revit loads the PyRevitLoader.dll
 addon at startup. This dll then creates an ironpython engine and runs
 pyRevitLoader.py (this script). It's the job of this script to setup the
 environment for the pyrevit module (pyrevitlib\pyrevit) and load a new pyRevit
 session. This script needs to add the directory path of the pyrevit lib folder
 so the pyrevit module can be imported and used.
"""

import sys
import os.path as op

# add the library location to the system search paths
sys.path.append(op.dirname(op.dirname(op.dirname(op.dirname(__file__)))))

# now pyrevit can be imported
from pyrevit.loader import sessionmgr

# ask sessionmgr to start a new session
sessionmgr.load_session()
```

8.3 pyRevitLoader Addin Source

The source code for pyRevitLoader addin is under: `pyrevitlib/pyrevit/addin/<loader version>/Source`

CHAPTER 9

Load Sequence, Step 2: IronPython Module

work in progress

Modules

- *pyrevit*
- *pyrevit.api*
- *pyrevit.compat*
- *pyrevit.forms*
- *pyrevit.framework*
- *pyrevit.script*
- *pyrevit.userconfig*
- *pyrevit.coreutils*
- *pyrevit.output*

CHAPTER 10

pyrevit

10.1 Usage

```
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
```

10.2 Documentation

class pyrevit.**PyRevitException**
Base class for all pyRevit Exceptions.

Parameters args and message are derived from Exception class.

class pyrevit.**PyRevitIOError**
Generic IO error in pyRevit.

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

name
str – Postable command name

key
str – Postable command key string

id
int – Postable command id

rvtobj
RevitCommandId – Postable command Id Object

class pyrevit.**_HostApplication**(*host_uiapp*)
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(__revit__)
>>> hostapp.is_newer_than(2017)
```

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

app
Return Application provided to the running command.

available_servers
Return list of available Revit server names.

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

doc
Return active Document.

docs
Return list of open Document objects.

get_postable_commands()
Return list of postable commands.

Returns list of *_HostAppPostableCommand*

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*)
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.

proc
System.Diagnostics.Process – Return current process object.

proc_id
int – Return current process id.

proc_name
str – Return current process name.

proc_path
str – Return file path for the current process main module.

proc_screen
`intptr` – Return handle to screen hosting current process.

proc_screen_scalefactor
float – Return scaling for screen hosting current process.

proc_screen_workarea
`System.Drawing.Rectangle` – Return screen working area.

uiapp
Return UIApplication provided to the running command.

uidoc
Return active UIDocument.

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

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

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

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

command_alt_path
str – Return current command alternate script path.

command_bundle
str – Return current command bundle name.

command_data
`ExternalCommandData` – Return current command data.

command_extension
str – Return current command extension name.

command_mode
bool – Check if pyrevit is running in pyrevit command context.

command_name
str – Return current command name.

command_path
str – Return current command path.

command_uniqueid
str – Return current command unique id.

doc_mode
bool – Check if pyrevit is running by doc generator.

engine_mgr
`PyRevitBaseClasses.EngineManager` – Return engine manager.

engine_ver
str – Return PyRevitLoader.ScriptExecutor hardcoded version.

executed_from_ui
bool – Check if command was executed from ui.

first_load
bool – Check whether pyrevit is not running in pyrevit command.

forced_debug_mode
bool – Check if command is in debug mode.

pyrevit_command
`PyRevitBaseClasses.PyRevitCommandRuntime` – Return command.

result_dict
`Dictionary<String, String>` – Return results dict for logging.

window_handle
`PyRevitBaseClasses.ScriptOutput` – Return output window.

10.3 Implementation

```
"""pyRevit root level config for all pyrevit sub-modules."""

import clr
import sys
import os
import os.path as op
from collections import namedtuple
import traceback


try:
    clr.AddReference('PyRevitLoader')
except Exception as e:
    # probably older IronPython engine not being able to
    # resolve to an already loaded assembly.
    # PyRevitLoader is executing this script so it should be referabe.
    pass

try:
    import PyRevitLoader
except ImportError:
    # this means that pyRevit is _not_ being loaded from a pyRevit engine
    # e.g. when importing from RevitPythonShell
    PyRevitLoader = None


PYREVIT_ADDON_NAME = 'pyRevit'
VERSION_MAJOR = 4
VERSION_MINOR = 5
BUILD_METADATA = ''

# ------------------------------------------------------------------------------
# config environment paths
# ------------------------------------------------------------------------------
# main pyrevit repo folder
try:
    # 3 steps back for <home>/Lib/pyrevit
```

(continues on next page)

(continued from previous page)

```
    HOME_DIR = op.dirname(op.dirname(op.dirname(__file__)))
except NameError:
    raise Exception('Critical Error. Can not find home directory.')

# default extensions directory
EXTENSIONS_DEFAULT_DIR = op.join(HOME_DIR, 'extensions')

# main pyrevit lib folders
MAIN_LIB_DIR = op.join(HOME_DIR, 'pyrevitlib')
MISC_LIB_DIR = op.join(HOME_DIR, 'site-packages')

# path to pyrevit module
PYREVIT_MODULE_DIR = op.join(MAIN_LIB_DIR, 'pyrevit')

# loader directory
LOADER_DIR = op.join(PYREVIT_MODULE_DIR, 'loader')

# addin directory
ADDIN_DIR = op.join(LOADER_DIR, 'addin')

# if loader module is available means pyRevit is being executed by Revit.
if PyRevitLoader:
    PYREVITLOADER_DIR = \
        op.join(ADDIN_DIR, PyRevitLoader.ScriptExecutor.EngineVersion)
    ADDIN_RESOURCE_DIR = op.join(PYREVITLOADER_DIR,
                                 'Source', 'pyRevitLoader', 'Resources')
# otherwise it might be under test, or documentation processing.
# so let's keep the symbols but set to None (fake the symbols)
else:
    PYREVITLOADER_DIR = ADDIN_RESOURCE_DIR = None

# add the framework dll path to the search paths
sys.path.append(ADDIN_DIR)
sys.path.append(PYREVITLOADER_DIR)


# pylama:ignore=E402
# now we can start importing stuff
from pyrevit.compat import safe_strtype
from pyrevit.framework import Process
from pyrevit.framework import Windows
from pyrevit.framework import Forms
from pyrevit.api import DB, UI  # noqa pylama ignore DB not being used here

# ------------------------------------------------------------------------------
# Base Exceptions
# ------------------------------------------------------------------------------
TRACEBACK_TITLE = 'Traceback:'


# General Exceptions
class PyRevitException(Exception):
    """Base class for all pyRevit Exceptions.

    Parameters args and message are derived from Exception class.
    """
```

(continues on next page)

(continued from previous page)

```
    def __str__(self):
        """Process stack trace and prepare report for output window."""
        sys.exc_type, sys.exc_value, sys.exc_traceback = sys.exc_info()
        try:
            tb_report = traceback.format_tb(sys.exc_traceback)[0]
            if self.args:
                message = self.args[0]
                return '{}\n\n{}\n{}'.format(message,
                                             TRACEBACK_TITLE,
                                             tb_report)
            else:
                return '{}\n{}'.format(TRACEBACK_TITLE, tb_report)
        except Exception:
            return Exception.__str__(self)


class PyRevitIOError(PyRevitException):
    """Generic IO error in pyRevit."""

    pass


# ------------------------------------------------------------------------------
# Wrapper for __revit__ builtin parameter set in scope by C# Script Executor
# ------------------------------------------------------------------------------
# namedtuple for passing information about a PostableCommand
_HostAppPostableCommand = namedtuple('_HostAppPostableCommand',
                                     ['name', 'key', 'id', 'rvtobj'])
"""Private namedtuple for passing information about a PostableCommand

Attributes:
    name (str): Postable command name
    key (str): Postable command key string
    id (int): Postable command id
    rvtobj (``RevitCommandId``): Postable command Id Object
"""


class _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.

    Args:
        host_uiapp (``UIApplication``): Instance of running host.

    Example:
        >>> hostapp = _HostApplication(__revit__)
        >>> hostapp.is_newer_than(2017)
    """

    def __init__(self, host_uiapp):
        self._uiapp = host_uiapp
        self._postable_cmds = []
```

(continues on next page)

(continued from previous page)

```
    @property
    def uiapp(self):
        """Return UIApplication provided to the running command."""
        return self._uiapp

    @property
    def app(self):
        """Return Application provided to the running command."""
        return self.uiapp.Application

    @property
    def uidoc(self):
        """Return active UIDocument."""
        return getattr(self.uiapp, 'ActiveUIDocument', None)

    @property
    def doc(self):
        """Return active Document."""
        return getattr(self.uidoc, 'Document', None)

    @property
    def activeview(self):
        """Return view that is active (UIDocument.ActiveView)."""
        return getattr(self.uidoc, 'ActiveView', None)

    @property
    def docs(self):
        """Return :obj:`list` of open :obj:`Document` objects."""
        return getattr(self.app, 'Documents', None)

    @property
    def available_servers(self):
        """Return :obj:`list` of available Revit server names."""
        return list(self.app.GetRevitServerNetworkHosts())

    @property
    def version(self):
        """str: Return version number (e.g. '2018')."""
        return self.app.VersionNumber

    @property
    def version_name(self):
        """str: Return version name (e.g. 'Autodesk Revit 2018')."""
        return self.app.VersionName

    @property
    def build(self):
        """str: Return build number (e.g. '20170927_1515(x64)')."""
        return self.app.VersionBuild

    @property
    def username(self):
        """str: Return the username from Revit API (Application.Username)."""
        uname = self.app.Username
        uname = uname.split('@')[0]  # if username is email
        # removing dots since username will be used in file naming
        uname = uname.replace('.', '')
```

(continues on next page)

(continued from previous page)

```
        return uname

    @property
    def proc(self):
        """System.Diagnostics.Process: Return current process object."""
        return Process.GetCurrentProcess()

    @property
    def proc_id(self):
        """int: Return current process id."""
        return Process.GetCurrentProcess().Id

    @property
    def proc_name(self):
        """str: Return current process name."""
        return Process.GetCurrentProcess().ProcessName

    @property
    def proc_path(self):
        """str: Return file path for the current process main module."""
        return Process.GetCurrentProcess().MainModule.FileName

    @property
    def proc_screen(self):
        """``intptr``: Return handle to screen hosting current process."""
        return Forms.Screen.FromHandle(
            Process.GetCurrentProcess().MainWindowHandle)

    @property
    def proc_screen_workarea(self):
        """``System.Drawing.Rectangle``: Return screen working area."""
        screen = HOST_APP.proc_screen
        if screen:
            return screen.WorkingArea

    @property
    def proc_screen_scalefactor(self):
        """float: Return scaling for screen hosting current process."""
        screen = HOST_APP.proc_screen
        if screen:
            actual_wdith = Windows.SystemParameters.PrimaryScreenWidth
            scaled_width = screen.PrimaryScreen.WorkingArea.Width
            return abs(scaled_width / actual_wdith)

    def is_newer_than(self, version):
        """bool: Return True if host app is newer than provided version.

        Args:
            version (str or int): version to check against.
        """
        return int(self.version) > int(version)

    def is_older_than(self, version):
        """bool: Return True if host app is older than provided version.

        Args:
            version (str or int): version to check against.
```

(continues on next page)

(continued from previous page)

```
        """
        return int(self.version) < int(version)

    def is_exactly(self, version):
        """bool: Return True if host app is equal to provided version.

        Args:
            version (str or int): version to check against.
        """
        return int(self.version) == int(version)

    def get_postable_commands(self):
        """Return list of postable commands.

        Returns:
            :obj:`list` of :obj:`_HostAppPostableCommand`
        """
        # if list of postable commands is _not_ already created
        # make the list and store in instance parameter
        if not self._postable_cmds:
            for pc in UI.PostableCommand.GetValues(UI.PostableCommand):
                try:
                    rcid = UI.RevitCommandId.LookupPostableCommandId(pc)
                    self._postable_cmds.append(
                        # wrap postable command info in custom namedtuple
                        _HostAppPostableCommand(name=safe_strtype(pc),
                                                key=rcid.Name,
                                                id=rcid.Id,
                                                rvtobj=rcid)
                        )
                except Exception:
                    # if any error occured when querying postable command
                    # or its info, pass silently
                    pass

        return self._postable_cmds


try:
    # Create an intance of host application wrapper
    # making sure __revit__ is available
    HOST_APP = _HostApplication(__revit__)  # noqa
except Exception:
    raise Exception('Critical Error: Host software is not supported. '
                    '(__revit__ handle is not available)')


# ------------------------------------------------------------------------------
# Wrapper to access builtin parameters set in scope by C# Script Executor
# ------------------------------------------------------------------------------
class _ExecutorParams(object):
    """Private Wrapper that provides runtime environment info."""

    @property   # read-only
    def engine_mgr(self):
        """``PyRevitBaseClasses.EngineManager``: Return engine manager."""
        try:
```

(continues on next page)

(continued from previous page)

```
            return __ipyenginemanager__
        except NameError:
            raise AttributeError()

    @property   # read-only
    def engine_ver(self):
        """str: Return PyRevitLoader.ScriptExecutor hardcoded version."""
        if PyRevitLoader:
            return PyRevitLoader.ScriptExecutor.EngineVersion

    @property  # read-only
    def first_load(self):
        """bool: Check whether pyrevit is not running in pyrevit command."""
        # if no output window is set by the executor, it means that pyRevit
        # is loading at Revit startup (not reloading)
        return True if EXEC_PARAMS.window_handle is None else False

    @property   # read-only
    def pyrevit_command(self):
        """``PyRevitBaseClasses.PyRevitCommandRuntime``: Return command."""
        try:
            return __externalcommand__
        except NameError:
            return None

    @property   # read-only
    def forced_debug_mode(self):
        """bool: Check if command is in debug mode."""
        if self.pyrevit_command:
            return self.pyrevit_command.DebugMode
        else:
            return False

    @property   # read-only
    def executed_from_ui(self):
        """bool: Check if command was executed from ui."""
        if self.pyrevit_command:
            return self.pyrevit_command.ExecutedFromUI
        else:
            return False

    @property   # read
    def window_handle(self):
        """``PyRevitBaseClasses.ScriptOutput``: Return output window."""
        if self.pyrevit_command:
            return self.pyrevit_command.OutputWindow

    @property   # read-only
    def command_path(self):
        """str: Return current command path."""
        if '__commandpath__' in __builtins__ \
                and __builtins__['__commandpath__']:
            return __builtins__['__commandpath__']
        elif self.pyrevit_command:
            return op.dirname(self.pyrevit_command.ScriptSourceFile)

    @property   # read-only
```

(continues on next page)

(continued from previous page)

```
    def command_alt_path(self):
        """str: Return current command alternate script path."""
        if '__alternatecommandpath__' in __builtins__ \
                and __builtins__['__alternatecommandpath__']:
            return __builtins__['__alternatecommandpath__']
        elif self.pyrevit_command:
            return op.dirname(self.pyrevit_command.AlternateScriptSourceFile)

    @property   # read-only
    def command_name(self):
        """str: Return current command name."""
        if '__commandname__' in __builtins__ \
                and __builtins__['__commandname__']:
            return __builtins__['__commandname__']
        elif self.pyrevit_command:
            return self.pyrevit_command.CommandName

    @property   # read-only
    def command_bundle(self):
        """str: Return current command bundle name."""
        if '__commandbundle__' in __builtins__ \
                and __builtins__['__commandbundle__']:
            return __builtins__['__commandbundle__']
        elif self.pyrevit_command:
            return self.pyrevit_command.CommandBundle

    @property   # read-only
    def command_extension(self):
        """str: Return current command extension name."""
        if '__commandextension__' in __builtins__ \
                and __builtins__['__commandextension__']:
            return __builtins__['__commandextension__']
        elif self.pyrevit_command:
            return self.pyrevit_command.CommandExtension

    @property   # read-only
    def command_uniqueid(self):
        """str: Return current command unique id."""
        if '__commanduniqueid__' in __builtins__ \
                and __builtins__['__commanduniqueid__']:
            return __builtins__['__commanduniqueid__']
        elif self.pyrevit_command:
            return self.pyrevit_command.CommandUniqueId

    @property
    def command_data(self):
        """``ExternalCommandData``: Return current command data."""
        if self.pyrevit_command:
            return self.pyrevit_command.CommandData

    @property
    def doc_mode(self):
        """bool: Check if pyrevit is running by doc generator."""
        try:
            return __sphinx__
        except NameError:
            return False
```

(continues on next page)

(continued from previous page)

```
    @property
    def command_mode(self):
        """bool: Check if pyrevit is running in pyrevit command context."""
        return self.pyrevit_command is not None

    @property
    def result_dict(self):
        """``Dictionary<String, String>``: Return results dict for logging."""
        if self.pyrevit_command:
            return self.pyrevit_command.GetResultsDictionary()


# create an instance of _ExecutorParams wrapping current runtime.
EXEC_PARAMS = _ExecutorParams()


# ------------------------------------------------------------------------------
# config user environment paths
# ------------------------------------------------------------------------------
# user env paths
USER_ROAMING_DIR = os.getenv('appdata')
USER_SYS_TEMP = os.getenv('temp')
USER_DESKTOP = op.expandvars('%userprofile%\\desktop')

# create paths for pyrevit files
if EXEC_PARAMS.doc_mode:
    PYREVIT_APP_DIR = PYREVIT_VERSION_APP_DIR = ' '
else:
    # pyrevit file directory
    PYREVIT_APP_DIR = op.join(USER_ROAMING_DIR, PYREVIT_ADDON_NAME)
    PYREVIT_VERSION_APP_DIR = op.join(PYREVIT_APP_DIR, HOST_APP.version)

    # add runtime paths to sys.paths
    # this will allow importing any dynamically compiled DLLs that
    # would be placed under this paths.
    for pyrvt_app_dir in [PYREVIT_APP_DIR, PYREVIT_VERSION_APP_DIR]:
        if not op.isdir(pyrvt_app_dir):
            try:
                os.mkdir(pyrvt_app_dir)
                sys.path.append(pyrvt_app_dir)
            except Exception as err:
                raise PyRevitException('Can not access pyRevit '
                                       'folder at: {} | {}'
                                       .format(pyrvt_app_dir, err))
        else:
            sys.path.append(pyrvt_app_dir)


# ------------------------------------------------------------------------------
# standard prefixes for naming pyrevit files (config, appdata and temp files)
# ------------------------------------------------------------------------------
if EXEC_PARAMS.doc_mode:
    PYREVIT_FILE_PREFIX_UNIVERSAL = PYREVIT_FILE_PREFIX = \
        PYREVIT_FILE_PREFIX_STAMPED = None
    PYREVIT_FILE_PREFIX_UNIVERSAL_USER = PYREVIT_FILE_PREFIX_USER = \
        PYREVIT_FILE_PREFIX_STAMPED_USER = None
```

(continues on next page)

(continued from previous page)

```
else:
    # e.g. pyRevit_
    PYREVIT_FILE_PREFIX_UNIVERSAL = '{}'.format(PYREVIT_ADDON_NAME)

    # e.g. pyRevit_2018_
    PYREVIT_FILE_PREFIX = '{}_{}'.format(PYREVIT_ADDON_NAME,
                                         HOST_APP.version)

    # e.g. pyRevit_2018_14422_
    PYREVIT_FILE_PREFIX_STAMPED = '{}_{}_{}'.format(PYREVIT_ADDON_NAME,
                                                    HOST_APP.version,
                                                    HOST_APP.proc_id)

    # e.g. pyRevit_eirannejad_
    PYREVIT_FILE_PREFIX_UNIVERSAL_USER = '{}_{}'.format(PYREVIT_ADDON_NAME,
                                                        HOST_APP.username)

    # e.g. pyRevit_2018_eirannejad_
    PYREVIT_FILE_PREFIX_USER = '{}_{}_{}'.format(PYREVIT_ADDON_NAME,
                                                 HOST_APP.version,
                                                 HOST_APP.username)

    # e.g. pyRevit_2018_eirannejad_14422_
    PYREVIT_FILE_PREFIX_STAMPED_USER = '{}_{}_{}_{}'.format(PYREVIT_ADDON_NAME,
                                                            HOST_APP.version,
                                                            HOST_APP.username,
                                                            HOST_APP.proc_id)
```

CHAPTER 11

pyrevit.api

11.1 Usage

```
from pyrevit.api import AdWindows
from pyrevit.api import NSJson
```

11.2 Documentation

Provide access to Revit API.

11.3 Implementation

```
"""Provide access to Revit API."""

from pyrevit.framework import clr

clr.AddReference('RevitAPI')
clr.AddReference('RevitAPIUI')
clr.AddReference('AdWindows')
clr.AddReference('UIFramework')
clr.AddReference('UIFrameworkServices')
clr.AddReference('Newtonsoft.Json')

# pylama:ignore=E402,W0611
# pylama ignore imports not on top and not used
import Autodesk.Internal as AdInternal
import Autodesk.Private as AdPrivate
import Autodesk.Windows as AdWindows
```

(continues on next page)

(continued from previous page)

```
import UIFramework
import UIFrameworkServices

import Autodesk.Revit.Attributes as Attributes

import Autodesk.Revit.DB as DB
import Autodesk.Revit.UI as UI

# try loading some utility modules shipped with revit
try:
    import Newtonsoft.Json as NSJson
except Exception as loaderr:
    pass
```

CHAPTER 12

pyrevit.compat

12.1 Usage

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

12.2 Documentation

python engine compatibility module.

12.3 Implementation

```
"""python engine compatibility module."""

import sys

PY2 = sys.version_info[0] == 2
PY3 = sys.version_info[0] == 3
IRONPY273 = sys.version_info[:3] == (2, 7, 3)
IRONPY277 = sys.version_info[:3] == (2, 7, 7)

safe_strtype = str

if PY2:
    safe_strtype = unicode  # noqa
```

CHAPTER 13

pyrevit.forms

13.1 Usage

```
from pyrevit.form import WPFWindow
```

13.2 Documentation

Reusable WPF forms for pyRevit.

class pyrevit.forms.**BaseCheckBoxItem**(*orig_item*)
Base class for checkbox option wrapping another object.

name
Name property.

unwrap()
Unwrap and return wrapped object.

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

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
...     )
>>> rops
'option2'
>>> rswitches
{'switch1': False, 'switch2': True}
```

_setup (***kwargs*)
Private method to be overriden by subclasses for window setup.

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.**DestDocOption** (*doc*)

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
```

_setup(***kwargs*)
Private method to be overriden by subclasses for prompt setup.

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

wait_async (*func*, *args=()*)
Call a method asynchronosely and show progress.

class pyrevit.forms.**RevisionOption** (*revision_element*)

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, switches = forms.SearchPrompt.show(
...     search_db=['target1', 'target2', 'target3', 'target4'],
...     switches=['/switch1', '/switch2'],
...     search_tip='pyRevit Search'
...     )
... matched_str
'target1'
... switches
{'/switch1': True, '/switch2': False}
```

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

find_switch_match ()
Find matching switches 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_matches
List of matches for the given search term.

search_term
Current cleaned up search term.

search_term_noswitch
Current cleaned up search term without the listed switches.

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(*input_term=None*)
Update search prompt results based on current input text.

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

Check box items passed in context to this standard form, must implement `name` and `state` parameter and `__nonzero__` method for truth value testing.

Parameters

- **context** (*list[object]*) – list of items to be selected from
- **title** (*str*) – window title
- **width** (*int*) – window width
- **height** (*int*) – window height
- **button_name** (*str*) – name of select button

Example

```
>>> from pyrevit import forms
>>> class MyOption(object):
...     def __init__(self, name, state=False):
...         self.state = state
...         self.name = name
...
...     def __nonzero__(self):
...         return self.state
...
...     def __str__(self):
...         return self.name
>>> ops = [MyOption('op1'), MyOption('op2', True), MyOption('op3')]
>>> res = forms.SelectFromCheckBoxes.show(ops,
...                                       button_name='Select Item')
>>> [bool(x) for x in res]  # or [x.state for x in res]
[True, False, True]
```

This module also provides a wrapper base class *BaseCheckBoxItem* 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.BaseCheckBoxItem)
...     @property
...     def name(self):
...         return '{} - {}{}'.format(self.item.SheetNumber,
...                                   self.item.SheetNumber)
```

(continues on next page)

(continued from previous page)

```
>>> ops = [MyOption('op1'), MyOption('op2', True), MyOption('op3')]
>>> res = forms.SelectFromCheckBoxes.show(ops,
...                                       button_name='Select Item')
>>> [bool(x) for x in res]  # or [x.state for x in res]
[True, False, True]
```

_setup(***kwargs*)
Private method to be overriden by subclasses for window setup.

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

Parameters

- **context** (*list[str]*) – list of items to be selected from
- **title** (*str*) – window title
- **width** (*int*) – window width
- **height** (*int*) – window height
- **button_name** (*str*) – name of select button
- **multiselect** (*bool*) – allow multi-selection

Example

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

_setup(***kwargs*)
Private method to be overriden by subclasses for window setup.

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

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

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

class pyrevit.forms.**SheetOption**(*sheet_element*)

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()*

_setup(***kwargs*)
Private method to be overriden by subclasses for prompt 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()*

_setup(***kwargs*)
Private method to be overriden by subclasses for window setup.

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

classmethod show(*context*, *title='User Input'*, *width=500*, *height=400*, ***kwargs*)
Show user input window.

Parameters

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

class pyrevit.forms.**ViewOption**(*view_element*)

class pyrevit.forms.**WPFWindow**(*xaml_source*, *literal_string=False*)
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

Example

```
>>> from pyrevit import forms
>>> layout = '<Window ShowInTaskbar="False" ResizeMode="NoResize" ' \
>>>          'WindowStartupLocation="CenterScreen" ' \
>>>          'HorizontalContentAlignment="Center">' \
>>>          '</Window>'
>>> w = forms.WPFWindow(layout, literal_string=True)
>>> w.show()
```

static hide_element(**wpf_elements*)
Collapse elements.

Parameters ***wpf_elements** (*str*) – element names to be collaped

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

Parameters

- **element_name** (*str*) – xaml image element name
- **image_file** (*str*) – image file path

show(*modal=False*)
Show window.

show_dialog()
Show modal window.

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

Parameters ***wpf_elements** (*str*) – element names to be set to visible.

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

Parameters ***wpf_elements** (*str*) – element names 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
```

_setup(***kwargs*)
Private method to be overriden by subclasses for prompt setup.

13.3 Implementation

```
"""Reusable WPF forms for pyRevit."""

import sys
import os
import os.path as op
import string
from collections import OrderedDict
import threading
from functools import wraps

from pyrevit import HOST_APP, EXEC_PARAMS
from pyrevit.compat import safe_strtype
from pyrevit import coreutils
from pyrevit.coreutils.logger import get_logger
from pyrevit import framework
from pyrevit.framework import System
from pyrevit.framework import Threading
from pyrevit.framework import Interop
from pyrevit.framework import wpf, Forms, Controls, Media
from pyrevit.api import AdWindows
from pyrevit import revit, UI, DB


logger = get_logger(__name__)


DEFAULT_CMDSWITCHWND_WIDTH = 600
DEFAULT_SEARCHWND_WIDTH = 600
DEFAULT_SEARCHWND_HEIGHT = 100
DEFAULT_INPUTWINDOW_WIDTH = 500
DEFAULT_INPUTWINDOW_HEIGHT = 400


class WPFWindow(framework.Windows.Window):
    r"""WPF Window base class for all pyRevit forms.

    Args:
        xaml_source (str): xaml source filepath or xaml content
        literal_string (bool): xaml_source contains xaml content, not filepath

    Example:
        >>> from pyrevit import forms
        >>> layout = '<Window ShowInTaskbar="False" ResizeMode="NoResize" ' \
        >>>          'WindowStartupLocation="CenterScreen" ' \
        >>>          'HorizontalContentAlignment="Center">' \
        >>>          '</Window>'
        >>> w = forms.WPFWindow(layout, literal_string=True)
        >>> w.show()
    """

    def __init__(self, xaml_source, literal_string=False):
        """Initialize WPF window and resources."""
        # self.Parent = self
        wih = Interop.WindowInteropHelper(self)
        wih.Owner = AdWindows.ComponentManager.ApplicationWindow
```

(continues on next page)

(continued from previous page)

```
        if not literal_string:
            if not op.exists(xaml_source):
                wpf.LoadComponent(self,
                                  os.path.join(EXEC_PARAMS.command_path,
                                               xaml_source)
                                  )
            else:
                wpf.LoadComponent(self, xaml_source)
        else:
            wpf.LoadComponent(self, framework.StringReader(xaml_source))

        #2c3e50 #noqa
        self.Resources['pyRevitDarkColor'] = \
            Media.Color.FromArgb(0xFF, 0x2c, 0x3e, 0x50)

        #23303d #noqa
        self.Resources['pyRevitDarkerDarkColor'] = \
            Media.Color.FromArgb(0xFF, 0x23, 0x30, 0x3d)

        #ffffff #noqa
        self.Resources['pyRevitButtonColor'] = \
            Media.Color.FromArgb(0xFF, 0xff, 0xff, 0xff)

        #f39c12 #noqa
        self.Resources['pyRevitAccentColor'] = \
            Media.Color.FromArgb(0xFF, 0xf3, 0x9c, 0x12)

        self.Resources['pyRevitDarkBrush'] = \
            Media.SolidColorBrush(self.Resources['pyRevitDarkColor'])
        self.Resources['pyRevitAccentBrush'] = \
            Media.SolidColorBrush(self.Resources['pyRevitAccentColor'])

        self.Resources['pyRevitDarkerDarkBrush'] = \
            Media.SolidColorBrush(self.Resources['pyRevitDarkerDarkColor'])

        self.Resources['pyRevitButtonForgroundBrush'] = \
            Media.SolidColorBrush(self.Resources['pyRevitButtonColor'])

    def show(self, modal=False):
        """Show window."""
        if modal:
            return self.ShowDialog()
        self.Show()

    def show_dialog(self):
        """Show modal window."""
        return self.ShowDialog()

    def set_image_source(self, element_name, image_file):
        """Set source file for image element.

        Args:
            element_name (str): xaml image element name
            image_file (str): image file path
        """
        wpfel = getattr(self, element_name)
```

(continues on next page)

(continued from previous page)

```
        if not op.exists(image_file):
            # noinspection PyUnresolvedReferences
            wpfel.Source = \
                framework.Imaging.BitmapImage(
                    framework.Uri(os.path.join(EXEC_PARAMS.command_path,
                                               image_file))
                    )
        else:
            wpfel.Source = \
                framework.Imaging.BitmapImage(framework.Uri(image_file))

    @staticmethod
    def hide_element(*wpf_elements):
        """Collapse elements.

        Args:
            *wpf_elements (str): element names to be collaped
        """
        for wpfel in wpf_elements:
            wpfel.Visibility = framework.Windows.Visibility.Collapsed

    @staticmethod
    def show_element(*wpf_elements):
        """Show collapsed elements.

        Args:
            *wpf_elements (str): element names to be set to visible.
        """
        for wpfel in wpf_elements:
            wpfel.Visibility = framework.Windows.Visibility.Visible

    @staticmethod
    def toggle_element(*wpf_elements):
        """Toggle visibility of elements.

        Args:
            *wpf_elements (str): element names to be toggled.
        """
        for wpfel in wpf_elements:
            if wpfel.Visibility == framework.Windows.Visibility.Visible:
                self.hide_element(wpfel)
            elif wpfel.Visibility == framework.Windows.Visibility.Collapsed:
                self.show_element(wpfel)


class TemplateUserInputWindow(WPFWindow):
    """Base class for pyRevit user input standard forms.

    Args:
        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 :func:`_setup`
    """

    xaml_source = 'BaseWindow.xaml'
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, context, title, width, height, **kwargs):
        """Initialize user input window."""
        WPFWindow.__init__(self,
                           op.join(op.dirname(__file__), self.xaml_source))
        self.Title = title
        self.Width = width
        self.Height = height

        self._context = context
        self.response = None
        self.PreviewKeyDown += self.handle_input_key

        self._setup(**kwargs)

    def _setup(self, **kwargs):
        """Private method to be overriden by subclasses for window setup."""
        pass

    def handle_input_key(self, sender, args):
        """Handle keyboard input."""
        if args.Key == framework.Windows.Input.Key.Escape:
            self.Close()

    @classmethod
    def show(cls, context,
             title='User Input',
             width=DEFAULT_INPUTWINDOW_WIDTH,
             height=DEFAULT_INPUTWINDOW_HEIGHT, **kwargs):
        """Show user input window.

        Args:
            context (any): window context element(s)
            title (type): window title
            width (type): window width
            height (type): window height
            **kwargs (type): other arguments to be passed to window
        """
        dlg = cls(context, title, width, height, **kwargs)
        dlg.ShowDialog()
        return dlg.response


class SelectFromList(TemplateUserInputWindow):
    """Standard form to select from a list of items.

    Args:
        context (list[str]): list of items to be selected from
        title (str): window title
        width (int): window width
        height (int): window height
        button_name (str): name of select button
        multiselect (bool): allow multi-selection

    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']
    """

    xaml_source = 'SelectFromList.xaml'

    def _setup(self, **kwargs):
        self.hide_element(self.clrsearch_b)
        self.clear_search(None, None)
        self.search_tb.Focus()

        if 'multiselect' in kwargs and not kwargs['multiselect']:
            self.list_lb.SelectionMode = Controls.SelectionMode.Single
        else:
            self.list_lb.SelectionMode = Controls.SelectionMode.Extended

        button_name = kwargs.get('button_name', None)
        if button_name:
            self.select_b.Content = button_name

        self._list_options()

    def _list_options(self, option_filter=None):
        if option_filter:
            option_filter = option_filter.lower()
            self.list_lb.ItemsSource = \
                [safe_strtype(option) for option in self._context
                 if option_filter in safe_strtype(option).lower()]
        else:
            self.list_lb.ItemsSource = \
                [safe_strtype(option) for option in self._context]

    def _get_options(self):
        return [option for option in self._context
                if safe_strtype(option) in self.list_lb.SelectedItems]

    def button_select(self, sender, args):
        """Handle select button click."""
        self.response = self._get_options()
        self.Close()

    def search_txt_changed(self, sender, args):
        """Handle text change in search box."""
        if self.search_tb.Text == '':
            self.hide_element(self.clrsearch_b)
        else:
            self.show_element(self.clrsearch_b)

        self._list_options(option_filter=self.search_tb.Text)

    def clear_search(self, sender, args):
        """Clear search box."""
        self.search_tb.Text = ' '
        self.search_tb.Clear()
        self.search_tb.Focus()
```

(continues on next page)

(continued from previous page)

```
class BaseCheckBoxItem(object):
    """Base class for checkbox option wrapping another object."""

    def __init__(self, orig_item):
        """Initialize the checkbox option and wrap given obj.

        Args:
            orig_item (any): object to wrap (must have name property
                             or be convertable to string with str()
        """
        self.item = orig_item
        self.state = False

    def __nonzero__(self):
        return self.state

    def __str__(self):
        return self.name or str(self.item)

    @property
    def name(self):
        """Name property."""
        return getattr(self.item, 'name', '')

    def unwrap(self):
        """Unwrap and return wrapped object."""
        return self.item


class SelectFromCheckBoxes(TemplateUserInputWindow):
    """Standard form to select from a list of check boxes.

    Check box items passed in context to this standard form, must implement
    ``name`` and ``state`` parameter and ``__nonzero__`` method for truth
    value testing.

    Args:
        context (list[object]): list of items to be selected from
        title (str): window title
        width (int): window width
        height (int): window height
        button_name (str): name of select button

    Example:
        >>> from pyrevit import forms
        >>> class MyOption(object):
        ...     def __init__(self, name, state=False):
        ...         self.state = state
        ...         self.name = name
        ...
        ...     def __nonzero__(self):
        ...         return self.state
        ...
        ...     def __str__(self):
        ...         return self.name
        >>> ops = [MyOption('op1'), MyOption('op2', True), MyOption('op3')]
        >>> res = forms.SelectFromCheckBoxes.show(ops,
```

(continues on next page)

(continued from previous page)

```
        ...                                          button_name='Select Item')
        >>> [bool(x) for x in res]  # or [x.state for x in res]
        [True, False, True]

        This module also provides a wrapper base class :obj:`BaseCheckBoxItem`
        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.BaseCheckBoxItem)
        ...     @property
        ...     def name(self):
        ...         return '{} - {}{}'.format(self.item.SheetNumber,
        ...                                   self.item.SheetNumber)
        >>> ops = [MyOption('op1'), MyOption('op2', True), MyOption('op3')]
        >>> res = forms.SelectFromCheckBoxes.show(ops,
        ...                                          button_name='Select Item')
        >>> [bool(x) for x in res]  # or [x.state for x in res]
        [True, False, True]
    """

    xaml_source = 'SelectFromCheckboxes.xaml'

    def _setup(self, **kwargs):
        self.hide_element(self.clrsearch_b)
        self.search_tb.Focus()

        self.checked_only = kwargs.get('checked_only', False)
        button_name = kwargs.get('button_name', None)
        if button_name:
            self.select_b.Content = button_name

        self.list_lb.SelectionMode = Controls.SelectionMode.Extended

        self._verify_context()
        self._list_options()

    def _verify_context(self):
        new_context = []
        for item in self._context:
            if not hasattr(item, 'state'):
                new_context.append(BaseCheckBoxItem(item))
            else:
                new_context.append(item)

        self._context = new_context

    def _list_options(self, checkbox_filter=None):
        if checkbox_filter:
            self.checkall_b.Content = 'Check'
            self.uncheckall_b.Content = 'Uncheck'
            self.toggleall_b.Content = 'Toggle'
            checkbox_filter = checkbox_filter.lower()
            self.list_lb.ItemsSource = \
                [checkbox for checkbox in self._context
                 if checkbox_filter in checkbox.name.lower()]
```

(continues on next page)

(continued from previous page)

```
        else:
            self.checkall_b.Content = 'Check All'
            self.uncheckall_b.Content = 'Uncheck All'
            self.toggleall_b.Content = 'Toggle All'
            self.list_lb.ItemsSource = self._context

    def _set_states(self, state=True, flip=False, selected=False):
        all_items = self.list_lb.ItemsSource
        if selected:
            current_list = self.list_lb.SelectedItems
        else:
            current_list = self.list_lb.ItemsSource
        for checkbox in current_list:
            if flip:
                checkbox.state = not checkbox.state
            else:
                checkbox.state = state

        # push list view to redraw
        self.list_lb.ItemsSource = None
        self.list_lb.ItemsSource = all_items

    def toggle_all(self, sender, args):
        """Handle toggle all button to toggle state of all check boxes."""
        self._set_states(flip=True)

    def check_all(self, sender, args):
        """Handle check all button to mark all check boxes as checked."""
        self._set_states(state=True)

    def uncheck_all(self, sender, args):
        """Handle uncheck all button to mark all check boxes as un-checked."""
        self._set_states(state=False)

    def check_selected(self, sender, args):
        """Mark selected checkboxes as checked."""
        self._set_states(state=True, selected=True)

    def uncheck_selected(self, sender, args):
        """Mark selected checkboxes as unchecked."""
        self._set_states(state=False, selected=True)

    def button_select(self, sender, args):
        """Handle select button click."""
        if self.checked_only:
            self.response = [x.item for x in self._context if x.state]
        else:
            self.response = self._context
        self.Close()

    def search_txt_changed(self, sender, args):
        """Handle text change in search box."""
        if self.search_tb.Text == '':
            self.hide_element(self.clrsearch_b)
        else:
            self.show_element(self.clrsearch_b)
```

(continues on next page)

(continued from previous page)

```
        self._list_options(checkbox_filter=self.search_tb.Text)

    def clear_search(self, sender, args):
        """Clear search box."""
        self.search_tb.Text = ' '
        self.search_tb.Clear()
        self.search_tb.Focus()


class CommandSwitchWindow(TemplateUserInputWindow):
    """Standard form to select from a list of command options.

    Args:
        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

    Returns:
        str: name of selected option

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

    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
        ...     )
        >>> rops
        'option2'
        >>> rswitches
        {'switch1': False, 'switch2': True}
    """

    xaml_source = 'CommandSwitchWindow.xaml'

    def _setup(self, **kwargs):
        self.selected_switch = ''
        self.Width = DEFAULT_CMDSWITCHWND_WIDTH
```

(continues on next page)

(continued from previous page)

```
        self.Title = 'Command Options'

        message = kwargs.get('message', None)
        self._switches = kwargs.get('switches', [])

        configs = kwargs.get('config', None)

        self.message_label.Content = \
            message if message else 'Pick a command option:'

        # creates the switches first
        for switch in self._switches:
            my_togglebutton = framework.Controls.Primitives.ToggleButton()
            my_togglebutton.Content = switch
            if configs and option in configs:
                self._set_config(my_togglebutton, configs[switch])
            self.button_list.Children.Add(my_togglebutton)

        for option in self._context:
            my_button = framework.Controls.Button()
            my_button.Content = option
            my_button.Click += self.process_option
            if configs and option in configs:
                self._set_config(my_button, configs[option])
            self.button_list.Children.Add(my_button)

        self._setup_response()
        self.search_tb.Focus()
        self._filter_options()

    @staticmethod
    def _set_config(item, config_dict):
        bg = config_dict.get('background', None)
        if bg:
            bg = bg.replace('0x', '#')
            item.Background = Media.BrushConverter().ConvertFrom(bg)

    def _setup_response(self, response=None):
        if self._switches:
            switches = [x for x in self.button_list.Children
                        if hasattr(x, 'IsChecked')]
            self.response = response, {x.Content: x.IsChecked
                                       for x in switches}
        else:
            self.response = response

    def _filter_options(self, option_filter=None):
        if option_filter:
            self.search_tb.Tag = ''
            option_filter = option_filter.lower()
            for button in self.button_list.Children:
                if option_filter not in button.Content.lower():
                    button.Visibility = framework.Windows.Visibility.Collapsed
                else:
                    button.Visibility = framework.Windows.Visibility.Visible
        else:
            self.search_tb.Tag = \
```

(continues on next page)

(continued from previous page)

```
                'Type to Filter / Tab to Select / Enter or Click to Run'
            for button in self.button_list.Children:
                button.Visibility = framework.Windows.Visibility.Visible

    def _get_active_button(self):
        buttons = []
        for button in self.button_list.Children:
            if button.Visibility == framework.Windows.Visibility.Visible:
                buttons.append(button)
        if len(buttons) == 1:
            return buttons[0]
        else:
            for x in buttons:
                if x.IsFocused:
                    return x

    def handle_click(self, sender, args):
        """Handle mouse click."""
        self.Close()

    def handle_input_key(self, sender, args):
        """Handle keyboard inputs."""
        if args.Key == framework.Windows.Input.Key.Escape:
            if self.search_tb.Text:
                self.search_tb.Text = ''
            else:
                self.Close()
        elif args.Key == framework.Windows.Input.Key.Enter:
            self.process_option(self._get_active_button(), None)
        elif args.Key != framework.Windows.Input.Key.Tab \
                and args.Key != framework.Windows.Input.Key.Space\
                and args.Key != framework.Windows.Input.Key.LeftShift\
                and args.Key != framework.Windows.Input.Key.RightShift:
            self.search_tb.Focus()

    def search_txt_changed(self, sender, args):
        """Handle text change in search box."""
        self._filter_options(option_filter=self.search_tb.Text)

    def process_option(self, sender, args):
        """Handle click on command option button."""
        self.Close()
        if sender:
            self._setup_response(response=sender.Content)


class TemplatePromptBar(WPFWindow):
    """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.

    Args:
        height (int): window height
        **kwargs: other arguments to be passed to :func:`_setup`
    """
```

(continues on next page)

(continued from previous page)

```
    xaml_source = 'TemplatePromptBar.xaml'

    def __init__(self, height=32, **kwargs):
        """Initialize user prompt window."""
        WPFWindow.__init__(self,
                           op.join(op.dirname(__file__), self.xaml_source))

        self.user_height = height
        self.update_window()

        self._setup(**kwargs)

    def update_window(self):
        """Update the prompt bar to match Revit window."""
        screen_area = HOST_APP.proc_screen_workarea
        scale_factor = 1.0 / HOST_APP.proc_screen_scalefactor
        top = left = width = height = 0

        window_rect = revit.get_window_rectangle()

        # set width and height
        width = window_rect.Right - window_rect.Left
        height = self.user_height

        top = window_rect.Top
        # in maximized window, the top might be off the active screen
        # due to windows thicker window frames
        # lets cut the height and re-adjust the top
        top_diff = abs(screen_area.Top - top)
        if 10 > top_diff > 0 and top_diff < height:
            height -= top_diff
            top = screen_area.Top

        left = window_rect.Left
        # in maximized window, Left also might be off the active screen
        # due to windows thicker window frames
        # let's fix the width to accomodate the extra pixels as well
        left_diff = abs(screen_area.Left - left)
        if 10 > left_diff > 0 and left_diff < width:
            # deduct two times the left negative offset since this extra
            # offset happens on both left and right side
            width -= left_diff * 2
            left = screen_area.Left

        self.Top = top * scale_factor
        self.Left = left * scale_factor
        self.Width = width * scale_factor
        self.Height = height

    def _setup(self, **kwargs):
        """Private method to be overriden by subclasses for prompt setup."""
        pass

    def __enter__(self):
        self.Show()
        return self
```

(continues on next page)

(continued from previous page)

```
    def __exit__(self, exception, exception_value, traceback):
        self.Close()


class WarningBar(TemplatePromptBar):
    """Show warning bar at the top of Revit window.

    Args:
        title (string): warning bar text

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

    xaml_source = 'WarningBar.xaml'

    def _setup(self, **kwargs):
        self.message_tb.Text = kwargs.get('title', '')


class ProgressBar(TemplatePromptBar):
    """Show progress bar at the top of Revit window.

    Args:
        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:
```

(continues on next page)

(continued from previous page)

```
        >>> 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
    """

    xaml_source = 'ProgressBar.xaml'

    def _setup(self, **kwargs):
        self.max_value = 1
        self.new_value = 0
        self.step = kwargs.get('step', 0)

        self.cancelled = False
        has_cancel = kwargs.get('cancellable', False)
        if has_cancel:
            self.show_element(self.cancel_b)

        self.pbar.IsIndeterminate = kwargs.get('indeterminate', False)
        self._title = kwargs.get('title', '{value}/{max_value}')

    def _update_pbar(self):
        self.update_window()
        self.pbar.Maximum = self.max_value
        self.pbar.Value = self.new_value

        # updating title
        title_text = \
            string.Formatter().vformat(self._title,
                                       (),
                                       coreutils.SafeDict(
                                           {'value': self.new_value,
                                            'max_value': self.max_value}
                                           ))

        self.pbar_text.Text = title_text

    def _dispatch_updater(self):
        # ask WPF dispatcher for gui update
        self.pbar.Dispatcher.Invoke(System.Action(self._update_pbar),
                                    Threading.DispatcherPriority.Background)

    @staticmethod
    def _make_return_getter(f, ret):
        # WIP
        @wraps(f)
        def wrapped_f(*args, **kwargs):
            ret.append(f(*args, **kwargs))
        return wrapped_f
```

(continues on next page)

(continued from previous page)

```
    @property
    def title(self):
        """Progress bar title."""
        return self._title

    @title.setter
    def title(self, value):
        if type(value) == str:
            self._title = value

    @property
    def indeterminate(self):
        """Progress bar indeterminate state."""
        return self.pbar.IsIndeterminate

    @indeterminate.setter
    def indeterminate(self, value):
        self.pbar.IsIndeterminate = value

    def clicked_cancel(self, sender, args):
        """Handler for cancel button clicked event."""
        self.cancel_b.Content = 'Cancelling...'
        self.cancelled = True

    def wait_async(self, func, args=()):
        """Call a method asynchronosely and show progress."""
        returns = []
        self.indeterminate = True
        rgfunc = self._make_return_getter(func, returns)
        t = threading.Thread(target=rgfunc, args=args)
        t.start()
        while t.is_alive():
            self._dispatch_updater()

        return returns[0] if returns else None

    def reset(self):
        """Reset progress value to 0."""
        self.update_progress(0, 1)

    def update_progress(self, new_value, max_value=1):
        """Update progress bar state with given min, max values.

        Args:
            new_value (float): current progress value
            max_value (float): total progress value
        """
        self.max_value = max_value
        self.new_value = new_value
        if self.new_value == 0:
            self._dispatch_updater()
        elif self.step > 0:
            if self.new_value % self.step == 0:
                self._dispatch_updater()
        else:
            self._dispatch_updater()
```

(continues on next page)

(continued from previous page)

```
class SearchPrompt(WPFWindow):
    """Standard prompt for pyRevit search.

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

    Example:
        >>> from pyrevit import forms
        >>> # assume search input of '/switch1 target1'
        >>> matched_str, switches = forms.SearchPrompt.show(
        ...     search_db=['target1', 'target2', 'target3', 'target4'],
        ...     switches=['/switch1', '/switch2'],
        ...     search_tip='pyRevit Search'
        ...     )
        ... matched_str
        'target1'
        ... switches
        {'/switch1': True, '/switch2': False}
    """
    def __init__(self, search_db, width, height, **kwargs):
        """Initialize search prompt window."""
        WPFWindow.__init__(self,
                           op.join(op.dirname(__file__), 'SearchPrompt.xaml'))
        self.Width = width
        self.MinWidth = self.Width
        self.Height = height

        self.search_tip = kwargs.get('search_tip', '')

        self._search_db = sorted(search_db)
        self._switches = kwargs.get('switches', [])
        self._setup_response()

        self.search_tb.Focus()
        self.hide_element(self.tab_icon)
        self.hide_element(self.return_icon)
        self.search_tb.Text = ''
        self.set_search_results()

    def _setup_response(self, response=None):
        if self._switches:
            switch_dict = dict.fromkeys(self._switches)
            for mswitch in self.find_switch_match():
                switch_dict[mswitch] = True
            self.response = response, switch_dict
        else:
            self.response = response
```

(continues on next page)

(continued from previous page)

```
    @property
    def search_input(self):
        """Current search input."""
        return self.search_tb.Text

    @search_input.setter
    def search_input(self, value):
        self.search_tb.Text = value

    @property
    def search_term(self):
        """Current cleaned up search term."""
        return self.search_input.lower().strip()

    @property
    def search_term_noswitch(self):
        """Current cleaned up search term without the listed switches."""
        term = self.search_term
        for switch in self._switches:
            term = term.replace(switch.lower() + ' ', '')
        return term.strip()

    @property
    def search_matches(self):
        """List of matches for the given search term."""
        # remove duplicates while keeping order
        # results = list(set(self._search_results))
        return OrderedDict.fromkeys(self._search_results).keys()

    def update_results_display(self, input_term=None):
        """Update search prompt results based on current input text."""
        self.directmatch_tb.Text = ''
        self.wordsmatch_tb.Text = ''

        results = self.search_matches
        res_cout = len(results)

        logger.debug('unique results count: {}'.format(res_cout))
        logger.debug('unique results: {}'.format(results))

        if res_cout > 1:
            self.show_element(self.tab_icon)
            self.hide_element(self.return_icon)
        elif res_cout == 1:
            self.hide_element(self.tab_icon)
            self.show_element(self.return_icon)
        else:
            self.hide_element(self.tab_icon)
            self.hide_element(self.return_icon)

        if self._result_index >= res_cout:
            self._result_index = 0

        if self._result_index < 0:
            self._result_index = res_cout - 1

        if not input_term:
```

(continues on next page)

(continued from previous page)

```
            input_term = self.search_term_noswitch

        if not self.search_input:
            self.directmatch_tb.Text = self.search_tip
            return

        if results:
            cur_res = results[self._result_index]
            logger.debug('current result: {}'.format(cur_res))
            if cur_res.lower().startswith(input_term):
                logger.debug('directmatch_tb.Text: {}'.format(cur_res))
                self.directmatch_tb.Text = \
                    self.search_input + cur_res[len(input_term):]
            else:
                logger.debug('wordsmatch_tb.Text: {}'.format(cur_res))
                self.wordsmatch_tb.Text = '- {}'.format(cur_res)

            self._setup_response(cur_res)
            return True

        self._setup_response()
        return False

    def set_search_results(self, *args):
        """Set search results for returning."""
        self._result_index = 0
        self._search_results = []

        for resultset in args:
            logger.debug('result set: {}'.format(resultset))
            self._search_results.extend(sorted(resultset))

        logger.debug('results: {}'.format(self._search_results))

    def find_switch_match(self):
        """Find matching switches in search term."""
        results = []
        cur_txt = self.search_term
        for switch in self._switches:
            if switch.lower() in cur_txt:
                results.append(switch)
        return results

    def find_direct_match(self, input_text):
        """Find direct text matches in search term."""
        results = []
        if input_text:
            for cmd_name in self._search_db:
                if cmd_name.lower().startswith(input_text):
                    results.append(cmd_name)

        return results

    def find_word_match(self, input_text):
        """Find direct word matches in search term."""
        results = []
        if input_text:
```

(continues on next page)

(continued from previous page)

```
            cur_words = input_text.split(' ')
            for cmd_name in self._search_db:
                if all([x in cmd_name.lower() for x in cur_words]):
                    results.append(cmd_name)

        return results

    def search_txt_changed(self, sender, args):
        """Handle text changed event."""
        input_term = self.search_term_noswitch
        dmresults = self.find_direct_match(input_term)
        wordresults = self.find_word_match(input_term)
        self.set_search_results(dmresults, wordresults)
        self.update_results_display(input_term)

    def handle_kb_key(self, sender, args):
        """Handle keyboard input event."""
        if args.Key == framework.Windows.Input.Key.Escape:
            self._setup_response()
            self.Close()
        elif args.Key == framework.Windows.Input.Key.Enter:
            self.Close()
        elif args.Key == framework.Windows.Input.Key.Tab:
            self._result_index += 1
            self.update_results_display()
        elif args.Key == framework.Windows.Input.Key.Down:
            self._result_index += 1
            self.update_results_display()
        elif args.Key == framework.Windows.Input.Key.Up:
            self._result_index -= 1
            self.update_results_display()

    @classmethod
    def show(cls, search_db,
             width=DEFAULT_SEARCHWND_WIDTH,
             height=DEFAULT_SEARCHWND_HEIGHT, **kwargs):
        """Show search prompt."""
        dlg = cls(search_db, width, height, **kwargs)
        dlg.ShowDialog()
        return dlg.response


class RevisionOption(BaseCheckBoxItem):
    def __init__(self, revision_element):
        super(RevisionOption, self).__init__(revision_element)

    @property
    def name(self):
        revnum = self.item.SequenceNumber
        if hasattr(self.item, 'RevisionNumber'):
            revnum = self.item.RevisionNumber
        return '{}-{}-{}'.format(revnum,
                                 self.item.Description,
                                 self.item.RevisionDate)


class SheetOption(BaseCheckBoxItem):
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, sheet_element):
        super(SheetOption, self).__init__(sheet_element)

    @property
    def name(self):
        return '{} - {}{}' \
            .format(self.item.SheetNumber,
                    self.item.Name,
                    ' (placeholder)' if self.item.IsPlaceholder else '')

    @property
    def number(self):
        return self.item.SheetNumber


class ViewOption(BaseCheckBoxItem):
    def __init__(self, view_element):
        super(ViewOption, self).__init__(view_element)

    @property
    def name(self):
        return '{} ({})'.format(self.item.ViewName, self.item.ViewType)


class DestDocOption(BaseCheckBoxItem):
    def __init__(self, doc):
        super(DestDocOption, self).__init__(doc)

    @property
    def name(self):
        return getattr(self.item, 'Title', '')


def select_revisions(title='Select Revision',
                     button_name='Select',
                     width=DEFAULT_INPUTWINDOW_WIDTH, multiselect=True,
                     filterfunc=None, doc=None):
    revisions = sorted(revit.query.get_revisions(doc=doc),
                       key=lambda x: x.SequenceNumber)

    if filterfunc:
        revisions = filter(filterfunc, revisions)
    revision_options = [RevisionOption(x) for x in revisions]

    # ask user for revisions
    return_options = \
        SelectFromList.show(
            revision_options,
            title=title,
            button_name=button_name,
            width=width,
            multiselect=multiselect
            )

    if return_options:
        if not multiselect and len(return_options) == 1:
            return return_options[0].unwrap()
```

(continues on next page)

(continued from previous page)

```
        else:
            return [x.unwrap() for x in return_options]


def select_sheets(title='Select Sheets', button_name='Select',
                  width=DEFAULT_INPUTWINDOW_WIDTH, multiple=True,
                  filterfunc=None, doc=None):
    doc = doc or HOST_APP.doc
    all_sheets = DB.FilteredElementCollector(doc) \
                   .OfClass(DB.ViewSheet) \
                   .WhereElementIsNotElementType() \
                   .ToElements()
    if filterfunc:
        all_sheets = filter(filterfunc, all_sheets)

    # ask user for multiple sheets
    if multiple:
        return_options = \
            SelectFromCheckBoxes.show(
                sorted([SheetOption(x) for x in all_sheets],
                       key=lambda x: x.number),
                title=title,
                button_name=button_name,
                width=width)
        if return_options:
            return [x.unwrap() for x in return_options if x.state]
    else:
        return_option = \
            SelectFromList.show(
                sorted([SheetOption(x) for x in all_sheets],
                       key=lambda x: x.number),
                title=title,
                button_name=button_name,
                width=width,
                multiselect=False)
        if return_option:
            return return_option[0].unwrap()


def select_views(title='Select Views', button_name='Select',
                 width=DEFAULT_INPUTWINDOW_WIDTH, multiple=True,
                 filterfunc=None, doc=None):
    all_graphviews = revit.query.get_all_views(doc=doc)

    if filterfunc:
        all_graphviews = filter(filterfunc, all_graphviews)

    # ask user for multiple sheets
    if multiple:
        return_options = \
            SelectFromCheckBoxes.show(
                sorted([ViewOption(x) for x in all_graphviews],
                       key=lambda x: x.name),
                title=title,
                button_name=button_name,
                width=width)
        if return_options:
```

(continues on next page)

(continued from previous page)

```
            return [x.unwrap() for x in return_options if x.state]
    else:
        return_option = \
            SelectFromList.show(
                sorted([ViewOption(x) for x in all_graphviews],
                       key=lambda x: x.name),
                title=title,
                button_name=button_name,
                width=width,
                multiselect=False)
        if return_option:
            return return_option[0].unwrap()


def select_dest_docs():
    # find open documents other than the active doc
    open_docs = [d for d in revit.docs if not d.IsLinked]
    open_docs.remove(revit.doc)

    if len(open_docs) < 1:
        alert('Only one active document is found. '
              'At least two documents must be open. '
              'Operation cancelled.')
        return

    return_options = \
        SelectFromCheckBoxes.show([DestDocOption(x) for x in open_docs],
                                  title='Select Destination Documents',
                                  button_name='OK')
    if return_options:
        return [x.unwrap() for x in return_options if x]


def alert(msg, title='pyRevit', ok=True,
          cancel=False, yes=False, no=False, retry=False, exit=False):
    buttons = UI.TaskDialogCommonButtons.None   # noqa
    if ok:
        buttons |= UI.TaskDialogCommonButtons.Ok
    if cancel:
        buttons |= UI.TaskDialogCommonButtons.Cancel
    if yes:
        buttons |= UI.TaskDialogCommonButtons.Yes
    if no:
        buttons |= UI.TaskDialogCommonButtons.No
    if retry:
        buttons |= UI.TaskDialogCommonButtons.Retry

    res = UI.TaskDialog.Show(title, msg, buttons)

    if not exit:
        if res == UI.TaskDialogResult.Ok \
                or res == UI.TaskDialogResult.Yes \
                or res == UI.TaskDialogResult.Retry:
            return True
        else:
            return False
```

(continues on next page)

(continued from previous page)

```
    sys.exit()


def pick_folder():
    fb_dlg = Forms.FolderBrowserDialog()
    if fb_dlg.ShowDialog() == Forms.DialogResult.OK:
        return fb_dlg.SelectedPath


def pick_file(file_ext='', files_filter='', init_dir='',
              restore_dir=True, multi_file=False, unc_paths=False):
    of_dlg = Forms.OpenFileDialog()
    if files_filter:
        of_dlg.Filter = files_filter
    else:
        of_dlg.Filter = '|*.{}'.format(file_ext)
    of_dlg.RestoreDirectory = restore_dir
    of_dlg.Multiselect = multi_file
    if init_dir:
        of_dlg.InitialDirectory = init_dir
    if of_dlg.ShowDialog() == Forms.DialogResult.OK:
        if unc_paths:
            return coreutils.dletter_to_unc(of_dlg.FileName)
        return of_dlg.FileName


def save_file(file_ext='', files_filter='', init_dir='', default_name='',
              restore_dir=True, unc_paths=False):
    sf_dlg = Forms.SaveFileDialog()
    if files_filter:
        sf_dlg.Filter = files_filter
    else:
        sf_dlg.Filter = '|*.{}'.format(file_ext)
    sf_dlg.RestoreDirectory = restore_dir
    if init_dir:
        sf_dlg.InitialDirectory = init_dir

    # setting default filename
    sf_dlg.FileName = default_name

    if sf_dlg.ShowDialog() == Forms.DialogResult.OK:
        if unc_paths:
            return coreutils.dletter_to_unc(sf_dlg.FileName)
        return sf_dlg.FileName


def check_workshared(doc):
    if not doc.IsWorkshared:
        alert('Model is not workshared.')
        return False
    return True
```

CHAPTER 14

pyrevit.framework

14.1 Usage

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

14.2 Documentation

Provide access to DotNet Framework.

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

Parameters **fw_object** – Dotnet Framework Object Instance

14.3 Implementation

```
"""Provide access to DotNet Framework."""

import clr

import System


clr.AddReference("System.Core")
clr.AddReference('System.Management')
clr.AddReferenceByPartialName('System.Windows.Forms')
clr.AddReferenceByPartialName('System.Drawing')
clr.AddReference('PresentationCore')
clr.AddReference('PresentationFramework')
```

(continues on next page)

(continued from previous page)

```
clr.AddReference('System.Xml.Linq')
clr.AddReferenceByPartialName('WindowsBase')

# add linq extensions?
clr.ImportExtensions(System.Linq)

# pylama:ignore=E402,W0611
# pylama ignore imports not on top and not used
from System import AppDomain, Version
from System import Type
from System import Uri, Guid
from System import EventHandler
from System import Array, IntPtr
from System.Collections import IEnumerator, IEnumerable
from System.Collections.Generic import List, Dictionary
from System import DateTime, DateTimeOffset

from System import Diagnostics
from System.Diagnostics import Process
from System.Diagnostics import Stopwatch

from System import Reflection
from System.Reflection import Assembly, AssemblyName
from System.Reflection import TypeAttributes, MethodAttributes
from System.Reflection import CallingConventions
from System.Reflection import BindingFlags
from System.Reflection.Emit import AssemblyBuilderAccess
from System.Reflection.Emit import CustomAttributeBuilder, OpCodes

from System import IO
from System.IO import IOException, DriveInfo, Path, StringReader

from System import Net
from System.Net import WebClient, WebRequest, WebProxy

from System import Drawing
from System import Windows
from System.Windows import Forms
from System.Windows.Forms import Clipboard
from System.Windows import Controls
from System.Windows import Media
from System.Windows import Threading
from System.Windows import Interop
from System.Windows.Media import Imaging, SolidColorBrush, Color

from System import Math

from System.CodeDom import Compiler
from Microsoft.CSharp import CSharpCodeProvider

from System.Management import ManagementObjectSearcher

from System.Runtime.Serialization import FormatterServices

clr.AddReference('IronPython.Wpf')
import wpf
```

(continues on next page)

(continued from previous page)

```
def get_type(fw_object):
    """Return CLR type of an object.

    Args:
        fw_object: Dotnet Framework Object Instance
    """
    return clr.GetClrType(fw_object)
```

CHAPTER 15

pyrevit.script

15.1 Usage

See *pyrevit.script Module*

```
from pyrevit import script

script.clipboard_copy('some text')
data = script.journal_read('data-key')
script.exit()
```

15.2 Documentation

Provide basic utilities for pyRevit scripts.

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

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

pyrevit.script.**get_alt_script_path**()
Return alternate script path of the current pyRevit command.

Returns alternate 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_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**()
Create and return config section parser object for current script.

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.usagelog.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_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.**open_url**(*url*)
Open url in a new tab in default webbrowser.

pyrevit.script.**reset_config**()
Reset pyRevit config.

Script should call this to reset any saved configuration by removing section related to current script

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.**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'

15.3 Implementation

```
"""Provide basic utilities for pyRevit scripts."""

import sys
import os
import os.path as op

from pyrevit import EXEC_PARAMS
from pyrevit.coreutils import logger
from pyrevit.coreutils import appdata
from pyrevit.coreutils import envvars
from pyrevit import framework
from pyrevit import revit
from pyrevit import output

# suppress any warning generated by native or third-party modules
import warnings
warnings.filterwarnings("ignore")


mlogger = logger.get_logger(__name__)


def get_info():
    """Return info on current pyRevit command.

    Returns:
        :obj:`pyrevit.extensions.genericcomps.GenericUICommand`:
            Command info object
    """
    from pyrevit.extensions.extensionmgr import get_command_from_path
    return get_command_from_path(EXEC_PARAMS.command_path)
```

(continues on next page)

(continued from previous page)

```
def get_script_path():
    """Return script path of the current pyRevit command.

    Returns:
        str: script path
    """
    return EXEC_PARAMS.command_path


def get_alt_script_path():
    """Return alternate script path of the current pyRevit command.

    Returns:
        str: alternate script path
    """
    return EXEC_PARAMS.command_alt_path


def get_bundle_name():
    """Return bundle name of the current pyRevit command.

    Returns:
        str: bundle name (e.g. MyButton.pushbutton)
    """
    return EXEC_PARAMS.command_bundle


def get_extension_name():
    """Return extension name of the current pyRevit command.

    Returns:
        str: extension name (e.g. MyExtension.extension)
    """
    return EXEC_PARAMS.command_extension


def get_unique_id():
    """Return unique id of the current pyRevit command.

    Returns:
        str: command unique id
    """
    return EXEC_PARAMS.command_uniqueid


def get_results():
    """Return command results dictionary for logging.

    Returns:
        :obj:`pyrevit.usagelog.record.CommandCustomResults`:
            Command results dict
    """
    from pyrevit.usagelog.record import CommandCustomResults
    return CommandCustomResults()
```

(continues on next page)

(continued from previous page)

```
def get_pyrevit_version():
    """Return pyRevit version.

    Returns:
        :obj:`pyrevit.versionmgr.PyRevitVersion`: pyRevit version provider
    """
    from pyrevit.versionmgr import PYREVIT_VERSION
    return PYREVIT_VERSION


def get_logger():
    """Create and return logger named for current script.

    Returns:
        :obj:`pyrevit.coreutils.logger.LoggerWrapper`: Logger object
    """
    return logger.get_logger(EXEC_PARAMS.command_name)


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

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


def get_config():
    """Create and return config section parser object for current script.

    Returns:
        :obj:`pyrevit.coreutils.configparser.PyRevitConfigSectionParser`:
            Config section parser object
    """
    from pyrevit.userconfig import user_config
    script_cfg_postfix = 'config'

    try:
        return user_config.get_section(EXEC_PARAMS.command_name +
                                       script_cfg_postfix)
    except Exception:
        return user_config.add_section(EXEC_PARAMS.command_name +
                                       script_cfg_postfix)


def 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.
    """
    from pyrevit.userconfig import user_config
    user_config.save_changes()


def reset_config():
```

(continues on next page)

(continued from previous page)

```
    """Reset pyRevit config.

    Script should call this to reset any saved configuration by removing section
↪related to current script
    """
    from pyrevit.userconfig import user_config
    script_cfg_postfix = 'config'

    user_config.remove_section(EXEC_PARAMS.command_name +
                               script_cfg_postfix)
    user_config.save_changes()


def 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.

    Args:
        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:
        str: full file path
    """
    if add_cmd_name:
        script_file_id = '{}_{}'.format(EXEC_PARAMS.command_name, file_id)
    else:
        script_file_id = file_id

    return appdata.get_universal_data_file(script_file_id, file_ext)


def 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.
```

(continues on next page)

(continued from previous page)

```
    Script should manage cleaning up these files.

    Args:
        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:
        str: full file path
    """
    if add_cmd_name:
        script_file_id = '{}_{}'.format(EXEC_PARAMS.command_name, file_id)
    else:
        script_file_id = file_id

    return appdata.get_data_file(script_file_id, file_ext)


def 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.

    Args:
        file_id (str): unique id for the filename
        add_cmd_name (bool, optional): add command name to file name

    Returns:
        str: full file path
    """
    if add_cmd_name:
        script_file_id = '{}_{}'.format(EXEC_PARAMS.command_name, file_id)
    else:
        script_file_id = file_id

    return appdata.get_instance_data_file(script_file_id)


def 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)
```

(continues on next page)

(continued from previous page)

```
        '.../pyRevit_2018_Command Name_mydata_Project1.data'

    Document data files are not cleaned up at pyRevit startup.
    Script should manage cleaning up these files.

    Args:
        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:
        str: full file path
    """
    proj_info = revit.get_project_info()

    if add_cmd_name:
        script_file_id = '{}_{}_{}'.format(EXEC_PARAMS.command_name,
                                           file_id,
                                           proj_info.filename
                                           or proj_info.name)
    else:
        script_file_id = '{}_{}'.format(file_id,
                                        proj_info.filename
                                        or proj_info.name)

    return appdata.get_data_file(script_file_id, file_ext)


def get_bundle_file(file_name):
    """Return full path to file under current script bundle.

    Args:
        file_name (str): bundle file name

    Returns:
        str: full bundle file path
    """
    return op.join(EXEC_PARAMS.command_path, file_name)


def journal_write(data_key, msg):
    """Write key and value to active Revit journal for current command.

    Args:
        data_key (str): data key
        msg (str): data value string
    """
    # Get the StringStringMap class which can write data into.
    # noinspection PyUnresolvedReferences
    data_map = EXEC_PARAMS.command_data.JournalData
    data_map.Clear()

    # Begin to add the support data
    data_map.Add(data_key, msg)


def journal_read(data_key):
```

(continues on next page)

(continued from previous page)

```
    """Read value for provided key from active Revit journal.

    Args:
        data_key (str): data key

    Returns:
        str: data value string
    """
    # Get the StringStringMap class which can write data into.
    # noinspection PyUnresolvedReferences
    data_map = EXEC_PARAMS.command_data.JournalData

    # Begin to get the support data
    return data_map[data_key]


def get_button():
    """Find and return current script ui button.

    Returns:
        :obj:`pyrevit.coreutils.ribbon._PyRevitRibbonButton`: ui button object
    """
    from pyrevit.coreutils.ribbon import get_current_ui
    pyrvt_tabs = get_current_ui().get_pyrevit_tabs()
    for tab in pyrvt_tabs:
        button = tab.find_child(EXEC_PARAMS.command_name)
        if button:
            return button
    return None


def 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.

    Args:
        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'
    """
    # find the ui button
    uibutton = get_button()
    if not uibutton:
        mlogger.debug('Can not find ui button.')
        return

    # get icon for on state
    if not on_icon_path:
        on_icon_path = get_bundle_file('on.png')
        if not on_icon_path:
            mlogger.debug('Script does not have icon for on state.')
            return
```

(continues on next page)

(continued from previous page)

```
    # get icon for off state
    if not off_icon_path:
        off_icon_path = get_bundle_file('off.png')
        if not off_icon_path:
            mlogger.debug('Script does not have icon for on state.')
            return

    icon_path = on_icon_path if new_state else off_icon_path
    mlogger.debug('Setting icon state to: {} ({})'
                  .format(new_state, icon_path))
    uibutton.set_icon(icon_path)


def exit():
    """Stop the script execution and exit."""
    sys.exit()


def show_file_in_explorer(file_path):
    """Show file in Windows Explorer."""
    import subprocess
    subprocess.Popen(r'explorer /select,"{}"'
                     .format(os.path.normpath(file_path)))


def open_url(url):
    """Open url in a new tab in default webbrowser."""
    import webbrowser
    return webbrowser.open_new_tab(url)


def clipboard_copy(string_to_copy):
    """Copy string to Windows Clipboard."""
    framework.Clipboard.SetText(string_to_copy)


def 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.

    Args:
        index_file (str, optional): full path of html file.
    """
    outputwindow = get_output()
    if not op.isfile(index_file):
        index_file = get_bundle_file(index_file)
    outputwindow.open_page(index_file)


def 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
```

(continues on next page)

(continued from previous page)

```
    environment variables could be sees in pyRevit settings window.

    Args:
        envvar (str): name of environment variable

    Returns:
        any: type of object stored in environment variable

    Example:
        >>> script.get_envvar('ToolActiveState')
        True
    """
    return envvars.get_pyrevit_env_var(envvar)


def 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.

    Args:
        envvar (str): name of environment variable
        value (any): value of environment variable

    Example:
        >>> script.set_envvar('ToolActiveState', False)
        >>> script.get_envvar('ToolActiveState')
        False
    """
    return envvars.set_pyrevit_env_var(envvar, value)
```

CHAPTER 16

pyrevit.userconfig

16.1 Usage

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.

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():
    """Create and return config section parser object for current script.

    Returns:
        :obj:`pyrevit.coreutils.configparser.PyRevitConfigSectionParser`:
            Config section parser object
    """
    from pyrevit.userconfig import user_config
    script_cfg_postfix = 'config'

    try:
        return user_config.get_section(EXEC_PARAMS.command_name +
                                       script_cfg_postfix)
    except Exception:
        return user_config.add_section(EXEC_PARAMS.command_name +
                                       script_cfg_postfix)
```

Example:

```
>>> from pyrevit import script
>>> cfg = script.get_config()
>>> cfg.property = value
>>> cfg.get('property', default_value)
>>> script.save_config()
```

16.2 Documentation

Handle reading and parsing, writin and saving of all user configurations.

All other modules use this module to query user config.

class pyrevit.userconfig.**PyRevitConfig**(*cfg_file_path=None*)

Provide read/write access to pyRevit configuration.

Parameters **cfg_file_path** (*str*) – full path to config file to be used.

Example

```
>>> cfg = PyRevitConfig(cfg_file_path)
>>> cfg.add_section('sectionname')
>>> cfg.sectionname.property = value
>>> cfg.sectionname.get('property', default_value)
>>> cfg.save_changes()
```

get_config_version()

Return version of config file used for change detection.

get_ext_root_dirs()

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.

16.3 Implementation

```
"""Handle reading and parsing, writin and saving of all user configurations.

All other modules use this module to query user config.
"""

import os
import os.path as op
import shutil

from pyrevit import EXEC_PARAMS, EXTENSIONS_DEFAULT_DIR
from pyrevit.framework import IOException
```

(continues on next page)

(continued from previous page)

```
from pyrevit.coreutils import touch
import pyrevit.coreutils.appdata as appdata
from pyrevit.coreutils.configparser import PyRevitConfigParser
from pyrevit.coreutils.logger import get_logger, set_file_logging
from pyrevit.versionmgr.upgrade import upgrade_user_config


logger = get_logger(__name__)


INIT_SETTINGS_SECTION = 'core'


# location for default pyRevit config files
if not EXEC_PARAMS.doc_mode:
    ADMIN_CONFIG_DIR = op.join(os.getenv('programdata'), 'pyRevit')

    # setup config file name and path
    CONFIG_FILE_PATH = appdata.get_universal_data_file(file_id='config',
                                                       file_ext='ini')
    logger.debug('User config file: {}'.format(CONFIG_FILE_PATH))
else:
    ADMIN_CONFIG_DIR = CONFIG_FILE_PATH = None


# ==============================================================================
# fix obsolete config file naming
# config file (and all appdata files) used to include username in the filename
# this fixes the existing config file with obsolete naming, to new format
# pylama:ignore=E402
from pyrevit import PYREVIT_APP_DIR, PYREVIT_FILE_PREFIX_UNIVERSAL_USER

OBSOLETE_CONFIG_FILENAME = '{}_{}'.format(PYREVIT_FILE_PREFIX_UNIVERSAL_USER,
                                          'config.ini')
OBSOLETE_CONFIG_FILEPATH = op.join(PYREVIT_APP_DIR, OBSOLETE_CONFIG_FILENAME)

if op.exists(OBSOLETE_CONFIG_FILEPATH):
    try:
        os.rename(OBSOLETE_CONFIG_FILEPATH, CONFIG_FILE_PATH)
    except Exception as rename_err:
        logger.error('Failed to update the config file name to new format. '
                     'A new configuration file has been created for you '
                     'under \n{}'
                     '\nYour previous pyRevit configuration file still '
                     'existing under the same folder. Please close Revit, '
                     'open both configuration files and copy and paste '
                     'settings from the old config file to new config file. '
                     'Then you can remove the old config file as pyRevit '
                     'will not be using that anymore. | {}'
                     .format(CONFIG_FILE_PATH, rename_err))
# end fix obsolete config file naming
# ==============================================================================


class PyRevitConfig(PyRevitConfigParser):
    """Provide read/write access to pyRevit configuration.
```

(continues on next page)

(continued from previous page)

```
    Args:
        cfg_file_path (str): full path to config file to be used.

    Example:
        >>> cfg = PyRevitConfig(cfg_file_path)
        >>> cfg.add_section('sectionname')
        >>> cfg.sectionname.property = value
        >>> cfg.sectionname.get('property', default_value)
        >>> cfg.save_changes()
    """

    def __init__(self, cfg_file_path=None):
        """Load settings from provided config file and setup parser."""
        self.config_file = cfg_file_path

        # try opening and reading config file in order.
        PyRevitConfigParser.__init__(self, cfg_file_path=cfg_file_path)

        # set log mode on the logger module based on
        # user settings (overriding the defaults)
        self._update_env()

    def _update_env(self):
        # update the debug level based on user config
        logger.reset_level()

        try:
            # first check to see if command is not in forced debug mode
            if not EXEC_PARAMS.forced_debug_mode:
                if self.core.debug:
                    logger.set_debug_mode()
                    logger.debug('Debug mode is enabled in user settings.')
                elif self.core.verbose:
                    logger.set_verbose_mode()

            set_file_logging(self.core.filelogging)
        except Exception as env_update_err:
            logger.debug('Error updating env variable per user config. | {}'
                         .format(env_update_err))

    def get_config_version(self):
        """Return version of config file used for change detection."""
        return self.get_config_file_hash()

    def get_ext_root_dirs(self):
        """Return a list of external extension directories set by the user.

        Returns:
            :obj:`list`: list of strings. External user extension directories.
        """
        dir_list = list()
        dir_list.append(EXTENSIONS_DEFAULT_DIR)
        try:
            dir_list.extend([p for p in self.core.userextensions])
        except Exception as read_err:
            logger.error('Error reading list of user extension folders. | {}'
```

(continues on next page)

(continued from previous page)

```
                        .format(read_err))

        return dir_list

    def save_changes(self):
        """Save user config into associated config file."""
        try:
            PyRevitConfigParser.save(self, self.config_file)
        except Exception as save_err:
            logger.error('Can not save user config to: {} | {}'
                         .format(self.config_file, save_err))

        # adjust environment per user configurations
        self._update_env()


def _set_hardcoded_config_values(parser):
    """Set default config values for user configuration.

    Args:
        parser (:obj:`pyrevit.userconfig.PyRevitConfig`):
            parser to accept the default values
    """
    # hard-coded values
    parser.add_section('core')
    parser.core.checkupdates = False
    parser.core.verbose = True
    parser.core.debug = False
    parser.core.filelogging = True
    parser.core.startuplogtimeout = 10
    parser.core.userextensions = []
    parser.core.compilecsharp = True
    parser.core.compilevb = True
    parser.core.loadbeta = False
    parser.core.rocketmode = False


def _get_default_config_parser(config_file_path):
    """Create a user settings file.

    Args:
        config_file_path (str): config file full name and path

    Returns:
        :obj:`pyrevit.userconfig.PyRevitConfig`: pyRevit config file handler
    """
    logger.debug('Creating default config file at: {} '
                 .format(CONFIG_FILE_PATH))
    touch(config_file_path)

    try:
        parser = PyRevitConfig(cfg_file_path=config_file_path)
    except Exception as read_err:
        # can not create default user config file under appdata folder
        logger.debug('Can not create config file under: {} | {}'
                     .format(config_file_path, read_err))
        parser = PyRevitConfig()
```

(continues on next page)

(continued from previous page)

```
    # set hard-coded values
    _set_hardcoded_config_values(parser)

    # save config into config file
    parser.save_changes()
    logger.debug('Default config saved to: {}'
                 .format(config_file_path))

    return parser


def _setup_admin_config():
    """Setup the default config file with hardcoded values."""
    if not op.exists(CONFIG_FILE_PATH) \
            and op.isdir(ADMIN_CONFIG_DIR):
        for entry in os.listdir(ADMIN_CONFIG_DIR):
            if entry.endswith('.ini'):
                sourcecfg = op.join(ADMIN_CONFIG_DIR, entry)
                try:
                    shutil.copyfile(sourcecfg, CONFIG_FILE_PATH)
                    logger.debug('Configured from admin file: {}'
                                 .format(sourcecfg))
                except Exception as copy_err:
                    logger.debug('Error copying admin config file: {}'
                                 .format(sourcecfg))
                return True


if not EXEC_PARAMS.doc_mode:
    # check to see if there is any config file provided by admin
    # if yes, copy that and use as default
    _setup_admin_config()

    # read user config, or setup default config file if not available
    # this pushes reading settings at first import of this module.
    try:
        user_config = PyRevitConfig(cfg_file_path=CONFIG_FILE_PATH)
        upgrade_user_config(user_config)
    except Exception as cfg_err:
        logger.debug('Can not read existing confing file at: {} | {}'
                     .format(CONFIG_FILE_PATH, cfg_err))
        user_config = _get_default_config_parser(CONFIG_FILE_PATH)
else:
    user_config = None
```

CHAPTER 17

pyrevit.coreutils

Misc Helper functions for pyRevit.

17.1 pyrevit.coreutils

17.1.1 Usage

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

17.1.2 Documentation

Misc Helper functions for pyRevit.

class pyrevit.coreutils.**FileWatcher**(*filepath*)

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**
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*)
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_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**
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.**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 True if internet connection is present.

Return type bool

pyrevit.coreutils.**cleanup_filename**(*file_name*)

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')
"Myfile3.txt"
```

pyrevit.coreutils.**cleanup_string**(*input_str*)

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.**create_ext_command_attrs**()

Create dotnet attributes for Revit extenrnal commads.

This method is used in creating custom dotnet types for pyRevit commands and compiling them into a DLL assembly. Current implementation sets `RegenerationOption.Manual` and `TransactionMode.Manual`

Returns list of `CustomAttributeBuilder` for `RegenerationOption` and `TransactionMode` attributes.

Return type list

pyrevit.coreutils.**create_type**(*modulebuilder*, *type_class*, *class_name*, *custom_attr_list*, **args*)

Create a dotnet type for a pyRevit command.

See `baseclasses.cs` code for the template pyRevit command dotnet type and its constructor default arguments that must be provided here.

Parameters

- **modulebuilder** (`ModuleBuilder`) – dotnet module builder
- **type_class** (*type*) – source dotnet type for the command
- **class_name** (*str*) – name for the new type
- **custom_attr_list** (`list`) – list of dotnet attributes for the type
- ***args** – list of arguments to be used with type constructor

Returns returns created dotnet type

Return type type

Example

```
>>> asm_builder = AppDomain.CurrentDomain.DefineDynamicAssembly(
... win_asm_name, AssemblyBuilderAccess.RunAndSave, filepath
... )
>>> module_builder = asm_builder.DefineDynamicModule(
... ext_asm_file_name, ext_asm_full_file_name
... )
>>> create_type(
... module_builder,
... PyRevitCommand,
... "PyRevitSomeCommandUniqueName",
... coreutils.create_ext_command_attrs(),
... [scriptpath, atlscriptpath, searchpath, helpurl, name,
... bundle, extension, uniquename, False, False])
<type PyRevitSomeCommandUniqueName>
```

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*)
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.**exract_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.**find_loaded_asm**(*asm_info*, *by_partial_name=False*, *by_location=False*)
Find loaded assembly based on name, partial name, or location.

Parameters

- **asm_info** (*str*) – name or location of the assembly
- **by_partial_name** (*bool*) – returns all assemblies that has the asm_info
- **by_location** (*bool*) – returns all assemblies matching location

Returns List of all loaded assemblies matching the provided info If only one assembly has been found, it returns the assembly. None will be returned if assembly is not loaded.

Return type list

pyrevit.coreutils.**find_type_by_name**(*assembly*, *type_name*)
Find type by name in assembly.

Parameters

- **assembly** (Assembly) – assembly to find the type in
- **type_name** (*str*) – type name

Returns returns the type if found.

Raises PyRevitException if type not found.

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

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

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_file_name**(*file_path*)
Return file basename of the given file.

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

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

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.**increment_str**(*input_str*, *step*)
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.**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_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.**load_asm**(*asm_name*)
Load assembly by name into current domain.

Parameters **asm_name** (*str*) – assembly name

Returns returns the loaded assembly, None if not loaded.

pyrevit.coreutils.**load_asm_file**(*asm_file*)
Load assembly by file into current domain.

Parameters **asm_file** (*str*) – assembly file path

Returns returns the loaded assembly, None if not loaded.

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.**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 ditinquishes 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.**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 ditinquishes 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=''*)
Run shell process silently.

Parameters

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

Exmaple:

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

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.

17.1.3 Implementation

```
"""Misc Helper functions for pyRevit."""

import os
import os.path as op
import re
import ast
import hashlib
import time
import datetime
import shutil
import random
import stat
from collections import defaultdict

from pyrevit import HOST_APP, PyRevitException
from pyrevit.compat import safe_strtype
from pyrevit import framework
from pyrevit import api


# pylama:ignore=D105
DEFAULT_SEPARATOR = ';'


class Timer:
    """Timer class using python native time module.

    Example:
        >>> timer = Timer()
        >>> timer.get_time()
        12
    """
```

(continues on next page)

(continued from previous page)

```
    def __init__(self):
        """Initialize and Start Timer."""
        self.start = time.time()

    def restart(self):
        """Restart Timer."""
        self.start = time.time()

    def get_time(self):
        """Get Elapsed Time."""
        return time.time() - self.start


class ScriptFileParser:
    """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', [])
        []
    """

    def __init__(self, file_address):
        """Initialize and read provided python script.

        Args:
            file_address (str): python script file path
        """
        self.file_addr = file_address
        with open(file_address, 'r') as f:
            self.ast_tree = ast.parse(f.read())

    def get_docstring(self):
        """Get global docstring."""
        doc_str = ast.get_docstring(self.ast_tree)
        if doc_str:
            return doc_str.decode('utf-8')
        return None

    def extract_param(self, param_name, default_value=None):
        """Find variable and extract its value.

        Args:
            param_name (str): variable name
            default_value (any):
                default value to be returned if variable does not exist

        Returns:
            any: value of the variable or :obj:`None`
        """
        try:
```

(continues on next page)

(continued from previous page)

```
            for child in ast.iter_child_nodes(self.ast_tree):
                if hasattr(child, 'targets'):
                    for target in child.targets:
                        if hasattr(target, 'id') and target.id == param_name:
                            param_value = ast.literal_eval(child.value)
                            if isinstance(param_value, str):
                                param_value = param_value.decode('utf-8')
                            return param_value
        except Exception as err:
            raise PyRevitException('Error parsing parameter: {} '
                                   'in script file for : {} | {}'
                                   .format(param_name, self.file_addr, err))

        return default_value


class FileWatcher(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
    """

    def __init__(self, filepath):
        """Initialize and read timestamp of provided file.

        Args:
            filepath (str): file path
        """
        self._cached_stamp = 0
        self._filepath = filepath
        self.update_tstamp()

    def update_tstamp(self):
        """Update the cached timestamp for later comparison."""
        self._cached_stamp = os.stat(self._filepath).st_mtime

    @property
    def has_changed(self):
        """Compare current file timestamp to the cached timestamp."""
        return os.stat(self._filepath).st_mtime != self._cached_stamp


class SafeDict(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',
```

(continues on next page)

(continued from previous page)

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

    def __missing__(self, key):
        return '{' + key + '}'


def get_all_subclasses(parent_classes):
    """Return all subclasses of a python class.

    Args:
        parent_classes (list): list of python classes

    Returns:
        list: list of python subclasses
    """
    sub_classes = []
    # if super-class, get a list of sub-classes.
    # Otherwise use component_class to create objects.
    for parent_class in parent_classes:
        try:
            derived_classes = parent_class.__subclasses__()
            if len(derived_classes) == 0:
                sub_classes.append(parent_class)
            else:
                sub_classes.extend(derived_classes)
        except AttributeError:
            sub_classes.append(parent_class)
    return sub_classes


def get_sub_folders(search_folder):
    """Get a list of all subfolders directly inside provided folder.

    Args:
        search_folder (str): folder path

    Returns:
        list: list of subfolder names
    """
    sub_folders = []
    for f in os.listdir(search_folder):
        if op.isdir(op.join(search_folder, f)):
            sub_folders.append(f)
    return sub_folders


def verify_directory(folder):
    """Check if the folder exists and if not create the folder.

    Args:
        folder (str): path of folder to verify

    Returns:
        str: path of verified folder, equals to provided folder
```

(continues on next page)

(continued from previous page)

```
    Raises:
        OSError on folder creation error.
    """
    if not op.exists(folder):
        try:
            os.makedirs(folder)
        except OSError as err:
            raise err
    return folder


def join_strings(str_list, separator=DEFAULT_SEPARATOR):
    """Join strings using provided separator.

    Args:
        str_list (list): list of string values
        separator (str): single separator character,
            defaults to DEFAULT_SEPARATOR

    Returns:
        str: joined string
    """
    if str_list:
        return separator.join(str_list)
    return ''


# character replacement list for cleaning up file names
SPECIAL_CHARS = {' ': '',
                 '~': '',
                 '!': 'EXCLAM',
                 '@': 'AT',
                 '#': 'SHARP',
                 '$': 'DOLLAR',
                 '%': 'PERCENT',
                 '^': '',
                 '&': 'AND',
                 '*': 'STAR',
                 '+': 'PLUS',
                 ';': '', ':': '', ',': '', '\"': '',
                 '{': '', '}': '', '[': '', ']': '', '\(': '', '\)': '',
                 '-': 'MINUS',
                 '=': 'EQUALS',
                 '<': '', '>': '',
                 '?': 'QMARK',
                 '.': 'DOT',
                 '_': 'UNDERS',
                 '|': 'VERT',
                 '\/': '', '\\': ''}


def cleanup_string(input_str):
    """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
```

(continues on next page)

(continued from previous page)

```
    unique names.

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

    Args:
        input_str (str): input string to be cleaned

    Example:
        >>> src_str = 'TEST@Some*<value>'
        >>> cleanup_string(src_str)
        "TESTATSomeSTARvalue"
    """
    # remove spaces and special characters from strings
    for char, repl in SPECIAL_CHARS.items():
        input_str = input_str.replace(char, repl)

    return input_str


def get_revit_instance_count():
    """Return number of open host app instances.

    Returns:
        int: number of open host app instances.
    """
    return len(list(framework.Process.GetProcessesByName(HOST_APP.proc_name)))


def run_process(proc, cwd=''):
    """Run shell process silently.

    Args:
        proc (str): process executive name
        cwd (str): current working directory

    Exmaple:
        >>> run_process('notepad.exe', 'c:/')
    """
    import subprocess
    return subprocess.Popen(proc,
                            stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                            cwd=cwd, shell=True)


def 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.

    Args:
        variable_name (str): variable name to look up in caller local scope
    """
    import inspect

    frame = inspect.stack()[1][0]
```

(continues on next page)

(continued from previous page)

```
    while variable_name not in frame.f_locals:
        frame = frame.f_back
        if frame is None:
            return None
    return frame.f_locals[variable_name]


def inspect_calling_scope_global_var(variable_name):
    """Trace back the stack to find the variable in the caller global stack.

    Args:
        variable_name (str): variable name to look up in caller global scope
    """
    import inspect

    frame = inspect.stack()[1][0]
    while variable_name not in frame.f_globals:
        frame = frame.f_back
        if frame is None:
            return None
    return frame.f_locals[variable_name]


def find_loaded_asm(asm_info, by_partial_name=False, by_location=False):
    """Find loaded assembly based on name, partial name, or location.

    Args:
        asm_info (str): name or location of the assembly
        by_partial_name (bool): returns all assemblies that has the asm_info
        by_location (bool): returns all assemblies matching location

    Returns:
        list: List of all loaded assemblies matching the provided info
        If only one assembly has been found, it returns the assembly.
        :obj:`None` will be returned if assembly is not loaded.
    """
    loaded_asm_list = []
    for loaded_assembly in framework.AppDomain.CurrentDomain.GetAssemblies():
        if by_partial_name:
            if asm_info.lower() in \
                    safe_strtype(loaded_assembly.GetName().Name).lower():
                loaded_asm_list.append(loaded_assembly)
        elif by_location:
            try:
                if op.normpath(loaded_assembly.Location) == \
                        op.normpath(asm_info):
                    loaded_asm_list.append(loaded_assembly)
            except Exception:
                continue
        elif asm_info.lower() == \
                safe_strtype(loaded_assembly.GetName().Name).lower():
            loaded_asm_list.append(loaded_assembly)

    return loaded_asm_list


def load_asm(asm_name):
```

(continues on next page)

(continued from previous page)

```
    """Load assembly by name into current domain.

    Args:
        asm_name (str): assembly name

    Returns:
        returns the loaded assembly, None if not loaded.
    """
    return framework.AppDomain.CurrentDomain.Load(asm_name)


def load_asm_file(asm_file):
    """Load assembly by file into current domain.

    Args:
        asm_file (str): assembly file path

    Returns:
        returns the loaded assembly, None if not loaded.
    """
    try:
        return framework.Assembly.LoadFrom(asm_file)
    except Exception:
        return None


def find_type_by_name(assembly, type_name):
    """Find type by name in assembly.

    Args:
        assembly (:obj:`Assembly`): assembly to find the type in
        type_name (str): type name

    Returns:
        returns the type if found.

    Raises:
        :obj:`PyRevitException` if type not found.
    """
    base_class = assembly.GetType(type_name)
    if base_class is not None:
        return base_class
    else:
        raise PyRevitException('Can not find base class type: {}'
                               .format(type_name))


def make_canonical_name(*args):
    """Join arguments with dot creating a unique id.

    Args:
        *args: Variable length argument list of type :obj:`str`

    Returns:
        str: dot separated unique name

    Example:
```

(continues on next page)

(continued from previous page)

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


def get_file_name(file_path):
    """Return file basename of the given file.

    Args:
        file_path (str): file path
    """
    return op.splitext(op.basename(file_path))[0]


def get_str_hash(source_str):
    """Calculate hash value of given string.

    Current implementation uses :func:`hashlib.md5` hash function.

    Args:
        source_str (str): source str

    Returns:
        str: hash value as string
    """
    return hashlib.md5(source_str.encode('utf-8', 'ignore')).hexdigest()


def calculate_dir_hash(dir_path, dir_filter, file_filter):
    r"""Create a unique hash to represent state of directory.

    Args:
        dir_path (str): target directory
        dir_filter (str): exclude directories matching this regex
        file_filter (str): exclude files matching this regex

    Returns:
        str: hash value as string

    Example:
        >>> calculate_dir_hash(source_path, '\.extension', '\.json')
        "1a885a0cae99f53d6088b9f7cee3bf4d"
    """
    mtime_sum = 0
    for root, dirs, files in os.walk(dir_path):
        if re.search(dir_filter, op.basename(root), flags=re.IGNORECASE):
            mtime_sum += op.getmtime(root)
            for filename in files:
                if re.search(file_filter, filename, flags=re.IGNORECASE):
                    modtime = op.getmtime(op.join(root, filename))
                    mtime_sum += modtime
    return get_str_hash(safe_strtype(mtime_sum))


def prepare_html_str(input_string):
    """Reformat html string and prepare for pyRevit output window.
```

(continues on next page)

(continued from previous page)

```
    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
    ditinquishes between <> printed from python and <> that define html.

    Args:
        input_string (str): input html string

    Example:
        >>> prepare_html_str('<p>Some text</p>')
        "&clt;p&cgt;Some text&clt;/p&cgt;"
    """
    return input_string.replace('<', '&clt;').replace('>', '&cgt;')


def 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
    ditinquishes between <> printed from python and <> that define html.

    Args:
        input_html (str): input codified html string

    Example:
        >>> prepare_html_str('&clt;p&cgt;Some text&clt;/p&cgt;')
        "<p>Some text</p>"
    """
    return input_html.replace('&clt;', '<').replace('&cgt;', '>')


# def check_internet_connection():
#     client = framework.WebClient()
#     try:
#         client.OpenRead("http://www.google.com")
#         return True
#     except:
#         return False
#


# def check_internet_connection():
    # import urllib2
    #
    # def internet_on():
    #     try:
    #         urllib2.urlopen('http://216.58.192.142', timeout=1)
    #         return True
    #     except urllib2.URLError as err:
    #         return False
```

(continues on next page)

(continued from previous page)

```
def check_internet_connection(timeout=1000):
    """Check if internet connection is available.

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

    Args:
        timeout (int): timeout in milliseconds

    Returns:
        bool: True if internet connection is present.
    """
    def can_access(url_to_open):
        try:
            client = framework.WebRequest.Create(url_to_open)
            client.Method = "HEAD"
            client.Timeout = timeout
            client.Proxy = framework.WebProxy.GetDefaultProxy()
            response = client.GetResponse()
            response.GetResponseStream()
            return True
        except Exception:
                return False

    for url in ["http://google.com/",
                "http://github.com/",
                "http://bitbucket.com/"]:
        if can_access(url):
            return url

    return False


def touch(fname, times=None):
    """Update the timestamp on the given file.

    Args:
        fname (str): target file path
        times (int): number of times to touch the file
    """
    with open(fname, 'a'):
        os.utime(fname, times)


def read_source_file(source_file_path):
    """Read text file and return contents.

    Args:
        source_file_path (str): target file path

    Returns:
        str: file contents

    Raises:
        :obj:`PyRevitException` on read error
    """
```

(continues on next page)

(continued from previous page)

```
    try:
        with open(source_file_path, 'r') as code_file:
            return code_file.read()
    except Exception as read_err:
        raise PyRevitException('Error reading source file: {} | {}'
                               .format(source_file_path, read_err))


def create_ext_command_attrs():
    """Create dotnet attributes for Revit extenrnal commads.

    This method is used in creating custom dotnet types for pyRevit commands
    and compiling them into a DLL assembly. Current implementation sets
    ``RegenerationOption.Manual`` and ``TransactionMode.Manual``

    Returns:
        list: list of :obj:`CustomAttributeBuilder` for
        :obj:`RegenerationOption` and :obj:`TransactionMode` attributes.
    """
    regen_const_info = \
        framework.clr.GetClrType(api.Attributes.RegenerationAttribute) \
        .GetConstructor(
               framework.Array[framework.Type](
                   (api.Attributes.RegenerationOption,)
                   )
               )

    regen_attr_builder = \
        framework.CustomAttributeBuilder(
            regen_const_info,
            framework.Array[object](
                (api.Attributes.RegenerationOption.Manual,)
                )
            )

    # add TransactionAttribute to framework.Type
    trans_constructor_info = \
        framework.clr.GetClrType(api.Attributes.TransactionAttribute) \
        .GetConstructor(
               framework.Array[framework.Type](
                   (api.Attributes.TransactionMode,)
                   )
               )

    trans_attrib_builder = \
        framework.CustomAttributeBuilder(
            trans_constructor_info,
            framework.Array[object](
                (api.Attributes.TransactionMode.Manual,)
                )
            )

    return [regen_attr_builder, trans_attrib_builder]


def create_type(modulebuilder,
                type_class, class_name, custom_attr_list, *args):
```

(continues on next page)

(continued from previous page)

```
    """Create a dotnet type for a pyRevit command.

    See ``baseclasses.cs`` code for the template pyRevit command dotnet type
    and its constructor default arguments that must be provided here.

    Args:
        modulebuilder (:obj:`ModuleBuilder`): dotnet module builder
        type_class (type): source dotnet type for the command
        class_name (str): name for the new type
        custom_attr_list (:obj:`list`): list of dotnet attributes for the type
        *args: list of arguments to be used with type constructor

    Returns:
        type: returns created dotnet type

    Example:
        >>> asm_builder = AppDomain.CurrentDomain.DefineDynamicAssembly(
        ... win_asm_name, AssemblyBuilderAccess.RunAndSave, filepath
        ... )
        >>> module_builder = asm_builder.DefineDynamicModule(
        ... ext_asm_file_name, ext_asm_full_file_name
        ... )
        >>> create_type(
        ... module_builder,
        ... PyRevitCommand,
        ... "PyRevitSomeCommandUniqueName",
        ... coreutils.create_ext_command_attrs(),
        ... [scriptpath, atlscriptpath, searchpath, helpurl, name,
        ... bundle, extension, uniquename, False, False])
        <type PyRevitSomeCommandUniqueName>
    """
    # create type builder
    type_builder = \
        modulebuilder.DefineType(
            class_name,
            framework.TypeAttributes.Class | framework.TypeAttributes.Public,
            type_class
            )

    for custom_attr in custom_attr_list:
        type_builder.SetCustomAttribute(custom_attr)

    # prepare a list of input param types to find the matching constructor
    type_list = []
    param_list = []
    for param in args:
        if type(param) == str \
                or type(param) == int:
            type_list.append(type(param))
            param_list.append(param)

    # call base constructor
    ci = type_class.GetConstructor(framework.Array[framework.Type](type_list))
    # create class constructor builder
    const_builder = \
        type_builder.DefineConstructor(framework.MethodAttributes.Public,
                                       framework.CallingConventions.Standard,
```

(continues on next page)

(continued from previous page)

```
                                        framework.Array[framework.Type](()))
    # add constructor parameters to stack
    gen = const_builder.GetILGenerator()
    gen.Emit(framework.OpCodes.Ldarg_0)  # Load "this" onto eval stack

    # add constructor input params to the stack
    for param_type, param in zip(type_list, param_list):
        if param_type == str:
            gen.Emit(framework.OpCodes.Ldstr, param)
        elif param_type == int:
            gen.Emit(framework.OpCodes.Ldc_I4, param)

    # call base constructor (consumes "this" and the created stack)
    gen.Emit(framework.OpCodes.Call, ci)
    # Fill some space - this is how it is generated for equivalent C# code
    gen.Emit(framework.OpCodes.Nop)
    gen.Emit(framework.OpCodes.Nop)
    gen.Emit(framework.OpCodes.Nop)
    gen.Emit(framework.OpCodes.Ret)
    type_builder.CreateType()


def open_folder_in_explorer(folder_path):
    """Open given folder in Windows Explorer.

    Args:
        folder_path (str): directory path
    """
    import subprocess
    subprocess.Popen(r'explorer /open,"{}"'
                     .format(os.path.normpath(folder_path)))


def fully_remove_dir(dir_path):
    """Remove directory recursively.

    Args:
        dir_path (str): directory path
    """
    def del_rw(action, name, exc):
        os.chmod(name, stat.S_IWRITE)
        os.remove(name)

    shutil.rmtree(dir_path, onerror=del_rw)


def cleanup_filename(file_name):
    """Cleanup file name from special characters.

    Args:
        file_name (str): file name

    Returns:
        str: cleaned up file name

    Example:
        >>> cleanup_filename('Myfile-(3).txt')
```

(continues on next page)

(continued from previous page)

```
        "Myfile3.txt"
    """
    return re.sub('[^\w_.)( -]', '', file_name)


def _inc_or_dec_string(str_id, shift):
    """Increment or decrement identifier.

    Args:
        str_id (str): identifier e.g. A310a
        shift (int): number of steps to change the identifier

    Returns:
        str: modified identifier

    Example:
        >>> _inc_or_dec_string('A319z')
        'A320a'
    """
    next_str = ""
    index = len(str_id) - 1
    carry = shift

    while index >= 0:
        if str_id[index].isalpha():
            if str_id[index].islower():
                reset_a = 'a'
                reset_z = 'z'
            else:
                reset_a = 'A'
                reset_z = 'Z'

            curr_digit = (ord(str_id[index]) + carry)
            if curr_digit < ord(reset_a):
                curr_digit = ord(reset_z) - ((ord(reset_a) - curr_digit) - 1)
                carry = shift
            elif curr_digit > ord(reset_z):
                curr_digit = ord(reset_a) + ((curr_digit - ord(reset_z)) - 1)
                carry = shift
            else:
                carry = 0

            curr_digit = chr(curr_digit)
            next_str += curr_digit

        elif str_id[index].isdigit():

            curr_digit = int(str_id[index]) + carry
            if curr_digit > 9:
                curr_digit = 0 + ((curr_digit - 9)-1)
                carry = shift
            elif curr_digit < 0:
                curr_digit = 9 - ((0 - curr_digit)-1)
                carry = shift
            else:
                carry = 0
            next_str += safe_strtype(curr_digit)
```

(continues on next page)

(continued from previous page)

```
        else:
            next_str += str_id[index]

        index -= 1

    return next_str[::-1]


def increment_str(input_str, step):
    """Incremenet identifier.

    Args:
        input_str (str): identifier e.g. A310a
        step (int): number of steps to change the identifier

    Returns:
        str: modified identifier

    Example:
        >>> increment_str('A319z')
        'A320a'
    """
    return _inc_or_dec_string(input_str, abs(step))


def decrement_str(input_str, step):
    """Decrement identifier.

    Args:
        input_str (str): identifier e.g. A310a
        step (int): number of steps to change the identifier

    Returns:
        str: modified identifier

    Example:
        >>> decrement_str('A310a')
        'A309z'
    """
    return _inc_or_dec_string(input_str, -abs(step))


def filter_null_items(src_list):
    """Remove None items in the given list.

    Args:
        src_list (:obj:`list`): list of any items

    Returns:
        :obj:`list`: cleaned list
    """
    return list(filter(bool, src_list))


def reverse_dict(input_dict):
    """Reverse the key, value pairs.
```

(continues on next page)

(continued from previous page)

```
    Args:
        input_dict (:obj:`dict`): source ordered dict

    Returns:
        :obj:`defaultdict`: reversed dictionary

    Example:
        >>> reverse_dict({1: 2, 3: 4})
        defaultdict(<type 'list'>, {2: [1], 4: [3]})
    """
    output_dict = defaultdict(list)
    for key, value in input_dict.items():
        output_dict[value].append(key)
    return output_dict


def timestamp():
    """Return timestamp for current time.

    Returns:
        str: timestamp in string format

    Example:
        >>> timestamp()
        '01003075032506808'
    """
    return datetime.datetime.now().strftime("%m%j%H%M%S%f")


def current_time():
    """Return formatted current time.

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

    Returns:
        str: formatted current time.

    Example:
        >>> current_time()
        '07:50:53'
    """
    return datetime.datetime.now().strftime("%H:%M:%S")


def current_date():
    """Return formatted current date.

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

    Returns:
        str: formatted current date.

    Example:
        >>> current_date()
        '2018-01-03'
    """
```

(continues on next page)

(continued from previous page)

```
    return datetime.datetime.now().strftime("%Y-%m-%d")


def is_blank(input_string):
    """Check if input string is blank (multiple white spaces is blank).

    Args:
        input_string (str): input string

    Returns:
        bool: True if string is blank

    Example:
        >>> is_blank('   ')
        True
    """
    if input_string and input_string.strip():
        return False
    return True


def is_url_valid(url_string):
    """Check if given URL is in valid format.

    Args:
        url_string (str): URL string

    Returns:
        bool: True if URL is in valid format

    Example:
        >>> is_url_valid('https://www.google.com')
        True
    """
    regex = re.compile(
            r'^(?:http|ftp)s?://'                   # http:// or https://
            r'(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\.)+'
            r'(?:[A-Z]{2,6}\.?|[A-Z0-9-]{2,}\.?)|'  # domain...
            r'localhost|'                           # localhost...
            r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})'  # ...or ip
            r'(?::\d+)?'                            # optional port
            r'(?:/?|[/?]\S+)$', re.IGNORECASE)

    return regex.match(url_string)


def 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.

    Args:
        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)
```

(continues on next page)

(continued from previous page)

```
    Returns:
        str: Reformatted string

    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)'
    """
    # find the tags
    tag_extractor = re.compile('{(.*?)}')
    tags = tag_extractor.findall(orig_format)

    # replace the tags with regex patterns
    # to create a regex pattern that finds values
    tag_replacer = re.compile('{.*?}')
    value_extractor_pattern = tag_replacer.sub('(.+)', orig_format)
    # find all values
    value_extractor = re.compile(value_extractor_pattern)
    values = value_extractor.findall(orig_str)
    if len(values) > 0:
        values = values[0]

    # create a dictionary of tags and values
    reformat_dict = {}
    for k, v in zip(tags, values):
        reformat_dict[k] = v

    # use dictionary to reformat the string into new
    return new_format.format(**reformat_dict)


def get_mapped_drives_dict():
    """Return a dictionary of currently mapped network drives."""
    searcher = framework.ManagementObjectSearcher(
        "root\\CIMV2",
        "SELECT * FROM Win32_MappedLogicalDisk"
        )

    return {x['DeviceID']: x['ProviderName'] for x in searcher.Get()}


def dletter_to_unc(dletter_path):
    """Convert drive letter path into UNC path of that drive.

    Args:
        dletter_path (str): drive letter path

    Returns:
        str: UNC path

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

(continues on next page)

(continued from previous page)

```
    dletter = dletter_path[:2]
    for mapped_drive, server_path in drives.items():
        if dletter.lower() == mapped_drive.lower():
            return dletter_path.replace(dletter, server_path)


def unc_to_dletter(unc_path):
    """Convert UNC path into drive letter path.

    Args:
        unc_path (str): UNC path

    Returns:
        str: drive letter path

    Example:
        >>> # assuming J: is mapped to //filestore/server/jdrive
        >>> unc_to_dletter('//filestore/server/jdrive/somefile.txt')
        'J:/somefile.txt'
    """
    drives = get_mapped_drives_dict()
    for mapped_drive, server_path in drives.items():
        if server_path in unc_path:
            return unc_path.replace(server_path, mapped_drive)


def random_color():
    """Return a random color channel value (between 0 and 255)."""
    return random.randint(0, 255)


def random_alpha():
    """Return a random alpha value (between 0 and 1.00)."""
    return round(random.random(), 2)


def random_hex_color():
    """Return a random color in hex format.

    Example:
        >>> random_hex_color()
        '#FF0000'
    """
    return '#%02X%02X%02X' % (random_color(),
                              random_color(),
                              random_color())


def random_rgb_color():
    """Return a random color in rgb format.

    Example:
        >>> random_rgb_color()
        'rgb(255, 0, 0)'
    """
    return 'rgb(%d, %d, %d)' % (random_color(),
                                random_color(),
```

(continues on next page)

(continued from previous page)

```
                                random_color())


def random_rgba_color():
    """Return a random color in rgba format.

    Example:
        >>> random_rgba_color()
        'rgba(255, 0, 0, 0.5)'
    """
    return 'rgba(%d, %d, %d, %.2f)' % (random_color(),
                                       random_color(),
                                       random_color(),
                                       random_alpha())


def exract_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

    Args:
        formatted_str (str): string specifying range

    Returns:
        list: list of names in the specified range

    Example:
        >>> exract_range('A103:A106')
        ['A103', 'A104', 'A105', 'A106']
        >>> exract_range('S203-S206')
        ['S203', 'S204', 'S205', 'S206']
        >>> exract_range('M00A,M00B')
        ['M00A', 'M00B']
    """
    for rchar, rchartype in {':': 'range', '-': 'range',
                             ',': 'list', ';': 'list'}.items():
        if rchar in formatted_str:
            if rchartype == 'range' \
                    and formatted_str.count(rchar) == 1:
                items = []
                start, end = formatted_str.split(rchar)
                assert len(start) == len(end), \
                    'Range start and end must have same length'
                items.append(start)
                item = increment_str(start, 1)
                safe_counter = 0
                while item != end:
                    items.append(item)
                    item = increment_str(item, 1)
                    safe_counter += 1
                    assert safe_counter < max_range, 'Max range reached.'
```

(continues on next page)

(continued from previous page)

```
                items.append(end)
                return items
            elif rchartype == 'list':
                return [x.strip() for x in formatted_str.split(rchar)]
    return [formatted_str]
```

17.2 pyrevit.coreutils.pyutils

17.2.1 Usage

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

17.2.2 Documentation

Helper functions for python.

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.**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
```

17.2.3 Implementation

```
"""Helper functions for python."""

import re


def pairwise(iterable, step=2):
    """Iterate through items in pairs.

    Args:
        iterable (iterable): any iterable object
        step (int): number of steps to move when making pairs

    Returns:
        iterable: list of pairs

    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)]
    """
    if step == 1:
        from itertools import tee, izip
        a, b = tee(iterable)
        next(b, None)
        return izip(a, b)
    elif step == 2:
        a = iter(iterable)
        return zip(a, a)


def safe_cast(val, to_type, default=None):
    """Convert value to type gracefully.
```

(continues on next page)

(continued from previous page)

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

    Args:
        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
    """
    try:
        return to_type(val)
    except (ValueError, TypeError):
        return default


def isnumber(token):
    """Verify if given string token is int or float.

    Args:
        token (str): string value

    Returns:
        bool: True of token is int or float

    Example:
        >>> isnumber('12.3')
        True
    """
    return re.match("^[0-9.]+?$", token) is not None
```

17.3 pyrevit.coreutils.mathnet

17.3.1 Usage

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

17.3.2 Documentation

MathNet importer module.

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

17.3.3 Implementation

```
"""MathNet importer module.

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

(continues on next page)

(continued from previous page)

```
"""

from pyrevit import EXEC_PARAMS
from pyrevit.framework import clr
from pyrevit.coreutils.logger import get_logger
from pyrevit.loader.addin import get_addin_dll_file


logger = get_logger(__name__)


MATHNET_LIB = 'MathNet.Numerics'

if not EXEC_PARAMS.doc_mode:
    mathnet_dll = get_addin_dll_file(MATHNET_LIB)
    logger.debug('Loading dll: {}'.format(mathnet_dll))
    try:
        clr.AddReferenceToFileAndPath(mathnet_dll)
        import MathNet
    except Exception as load_err:
        logger.error('Can not load {} module. | {}'
                     .format(MATHNET_LIB, load_err))
```

CHAPTER 18

pyrevit.output

Provides access and control over the pyRevit output window.

18.1 pyrevit.output

18.1.1 Usage

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()
```

18.1.2 Documentation

Provide access to output window and its functionality.

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; }')
```

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.

static emojize(*md_str*)

Replace emoji codes with emoji images and print.

Parameters **md_str** (*str*) – string containing emoji code

Example

```
>>> output = pyrevit.output.get_output()
>>> output.emojize('Process completed. :thumbs_up:')
```

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_progress()

Hide output window progress bar.

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

Parameters

- **script_code** (*str*) – javascript code
- **attribs** (dict) – dictionary of attribute names and value

Example

```
>>> output = pyrevit.output.get_output()
>>> output.inject_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()
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.

make_bar_chart()
PyRevitOutputChart: Return bar chart object.

make_bubble_chart()
PyRevitOutputChart: Return bubble chart object.

make_chart()
PyRevitOutputChart: Return chart object.

make_doughnut_chart()
PyRevitOutputChart: Return dougnut chart object.

make_line_chart()
PyRevitOutputChart: Return line chart object.

make_pie_chart()
PyRevitOutputChart: Return pie chart object.

make_polar_chart()
PyRevitOutputChart: Return polar chart object.

make_radar_chart()
PyRevitOutputChart: Return radar chart object.

make_stacked_chart()
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
str – 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.

output_uniqueid
str – 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.

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>')
```

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=[]*, *formats=[]*, *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_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_title(*new_title*)
Set window title to the new title.

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

show()
Show the window.

unhide_progress()
Unhide output window progress bar.

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
PyRevitBaseClasses.ScriptOutput – Return output window object.

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

18.1.3 Implementation

```
"""Provide access to output window and its functionality."""

from __future__ import print_function
import os.path as op
import itertools

from pyrevit import EXEC_PARAMS
from pyrevit.compat import safe_strtype
from pyrevit import framework
from pyrevit import coreutils
from pyrevit.coreutils import logger
from pyrevit.coreutils import markdown, charts
from pyrevit.coreutils import emoji
from pyrevit.coreutils import envvars
from pyrevit.coreutils.loadertypes import EnvDictionaryKeys
from pyrevit.coreutils.loadertypes import ScriptOutputManager
from pyrevit.output import linkmaker
from pyrevit.userconfig import user_config


mlogger = logger.get_logger(__name__)


DEFAULT_STYLESHEET_NAME = 'outputstyles.css'


def set_stylesheet(stylesheet):
    """Set active css stylesheet used by output window.

    Args:
        stylesheet (str): full path to stylesheet file
    """
    envvars.set_pyrevit_env_var(EnvDictionaryKeys.outputStyleSheet,
                                stylesheet)


def get_stylesheet():
    """Return active css stylesheet used by output window."""
    return envvars.get_pyrevit_env_var(EnvDictionaryKeys.outputStyleSheet)


def get_default_stylesheet():
    """Return default css stylesheet used by output window."""
    return op.join(op.dirname(__file__), DEFAULT_STYLESHEET_NAME)


def reset_stylesheet():
    """Reset active stylesheet to default."""
    envvars.set_pyrevit_env_var(EnvDictionaryKeys.outputStyleSheet,
                                get_default_stylesheet())
```

(continues on next page)

(continued from previous page)

```
# setup output window stylesheet
if not EXEC_PARAMS.doc_mode:
    active_stylesheet = \
        user_config.core.get_option('outputstylesheet',
                                    default_value=get_default_stylesheet())
    set_stylesheet(active_stylesheet)


class PyRevitOutputWindow(object):
    """Wrapper to interact with the output window."""

    @property
    def window(self):
        """``PyRevitBaseClasses.ScriptOutput``: Return output window object."""
        return EXEC_PARAMS.window_handle

    @property
    def renderer(self):
        """Return html renderer inside output window.

        Returns:
            ``System.Windows.Forms.WebBrowser`` (In current implementation)
        """
        if self.window:
            return self.window.renderer

    @property
    def output_id(self):
        """str: 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.
        """
        if self.window:
            return self.window.OutputId

    @property
    def output_uniqueid(self):
        """str: 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.
        """
        if self.window:
            return self.window.OutputUniqueId

    @property
    def debug_mode(self):
        """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.
```

(continues on next page)

(continued from previous page)

```
        """
        return EXEC_PARAMS.pyrevit_command.OutputStream.PrintDebugInfo

    @debug_mode.setter
    def debug_mode(self, value):
        EXEC_PARAMS.pyrevit_command.OutputStream.PrintDebugInfo = value

    def _get_head_element(self):
        return self.renderer.Document.GetElementsByTagName('head')[0]

    def self_destruct(self, seconds):
        """Set self-destruct (close window) timer.

        Args:
            seconds (int): number of seconds after which window is closed.
        """
        if self.window:
            self.window.SelfDestructTimer(seconds)

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

        Args:
            element_tag (str): html tag of the element e.g. 'div'
            element_contents (str): html code of the element contents
            attribs (:obj:`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})
        """
        html_element = self.renderer.Document.CreateElement(element_tag)
        if element_contents:
            html_element.InnerHtml = element_contents

        if attribs:
            for attribute, value in attribs.items():
                html_element.SetAttribute(attribute, value)

        # inject the script into head
        head_el = self._get_head_element()
        head_el.AppendChild(html_element)

    def inject_script(self, script_code, attribs=None):
        """Inject script tag into current html head of the output window.

        Args:
            script_code (str): javascript code
            attribs (:obj:`dict`): dictionary of attribute names and value

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

(continues on next page)

(continued from previous page)

```
        self.inject_to_head('script', script_code, attribs=attribs)

    def add_style(self, style_code, attribs=None):
        """Inject style tag into current html head of the output window.

        Args:
            style_code (str): css styling code
            attribs (:obj:`dict`): dictionary of attribute names and value

        Example:
            >>> output = pyrevit.output.get_output()
            >>> output.add_style('body { color: blue; }')
        """
        self.inject_to_head('style', style_code, attribs=attribs)

    def get_head_html(self):
        """str: Return inner code of html head element."""
        return self._get_head_element().InnerHtml

    def set_title(self, new_title):
        """Set window title to the new title."""
        if self.window:
            self.window.Title = new_title

    def set_width(self, width):
        """Set window width to the new width."""
        if self.window:
            self.window.Width = width

    def set_height(self, height):
        """Set window height to the new height."""
        if self.window:
            self.window.Height = height

    def set_font(self, font_family, font_size):
        """Set window font family to the new font family and size.

        Args:
            font_family (str): font family name e.g. 'Courier New'
            font_size (int): font size e.g. 16
        """
        # noinspection PyUnresolvedReferences
        self.renderer.Font = \
            framework.Drawing.Font(font_family,
                                   font_size,
                                   framework.Drawing.FontStyle.Regular,
                                   framework.Drawing.GraphicsUnit.Point)

    def resize(self, width, height):
        """Resize window to the new width and height."""
        self.set_width(width)
        self.set_height(height)

    def get_title(self):
        """str: Return current window title."""
        if self.window:
            return self.window.Text
```

(continues on next page)

(continued from previous page)

```
    def get_width(self):
        """int: Return current window width."""
        if self.window:
            return self.window.Width

    def get_height(self):
        """int: Return current window height."""
        if self.window:
            return self.window.Height

    def close(self):
        """Close the window."""
        if self.window:
            self.window.Close()

    def close_others(self, all_open_outputs=False):
        """Close all other windows that belong to the current command.

        Args:
            all_open_outputs (bool): Close all any other windows if True
        """
        if all_open_outputs:
            ScriptOutputManager.CloseActiveOutputWindows(self.window)
        else:
            ScriptOutputManager.CloseActiveOutputWindows(self.window,
                                                         self.output_id)

    def hide(self):
        """Hide the window."""
        if self.window:
            self.window.Hide()

    def show(self):
        """Show the window."""
        if self.window:
            self.window.Show()

    def lock_size(self):
        """Lock window size."""
        if self.window:
            self.window.LockSize()

    def save_contents(self, dest_file):
        """Save html code of the window.

        Args:
            dest_file (str): full path of the destination html file
        """
        if self.renderer:
            html = \
                self.renderer.Document.Body.OuterHtml.encode('ascii', 'ignore')
            doc_txt = self.renderer.DocumentText
            full_html = doc_txt.lower().replace('<body></body>', html)
            with open(dest_file, 'w') as output_file:
                output_file.write(full_html)
```

(continues on next page)

(continued from previous page)

```
    def open_url(self, dest_url):
        """Open url page in output window.

        Args:
            dest_url (str): web url of the target page
        """
        if self.renderer:
            self.renderer.Navigate(dest_url, False)

    def open_page(self, dest_file):
        """Open html page in output window.

        Args:
            dest_file (str): full path of the target html file
        """
        self.show()
        self.open_url('file:///' + dest_file)

    def update_progress(self, cur_value, max_value):
        """Activate and update the output window progress bar.

        Args:
            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)
        """
        if self.window:
            self.window.UpdateProgressBar(cur_value, max_value)

    def reset_progress(self):
        """Reset output window progress bar to zero."""
        if self.window:
            self.window.UpdateProgressBar(0, 1)

    def hide_progress(self):
        """Hide output window progress bar."""
        if self.window:
            self.window.SetProgressBarVisibility(False)

    def unhide_progress(self):
        """Unhide output window progress bar."""
        if self.window:
            self.window.SetProgressBarVisibility(True)

    @staticmethod
    def emojize(md_str):
        """Replace emoji codes with emoji images and print.

        Args:
            md_str (str): string containing emoji code

        Example:
            >>> output = pyrevit.output.get_output()
```

(continues on next page)

(continued from previous page)

```
            >>> output.emojize('Process completed. :thumbs_up:')
        """
        print(emoji.emojize(md_str), end="")

    @staticmethod
    def 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(coreutils.prepare_html_str(emoji.emojize(html_str)),
              end="")

    @staticmethod
    def print_code(code_str):
        """Print code to the output window with special formatting.

        Example:
            >>> output = pyrevit.output.get_output()
            >>> output.print_code('value = 12')
        """
        code_div = '<div class="code">{}</div>'
        print(coreutils.prepare_html_str(
                code_div.format(
                    code_str.replace('    ', ' '*4))), end="")

    @staticmethod
    def print_md(md_str):
        """Process markdown code and print to output window.

        Example:
            >>> output = pyrevit.output.get_output()
            >>> output.print_md('### Title')
        """
        tables_ext = 'pyrevit.coreutils.markdown.extensions.tables'
        markdown_html = markdown.markdown(md_str, extensions=[tables_ext])
        markdown_html = markdown_html.replace('\n', '').replace('\r', '')
        html_code = emoji.emojize(coreutils.prepare_html_str(markdown_html))
        print(html_code, end="")

    def print_table(self, table_data, columns=[], formats=[],
                    title='', last_line_style=''):
        """Print provided data in a table in output window.

        Args:
            table_data (:obj:`list` of iterables): 2D array of data
            title (str): table title
            columns (:obj:`list` str): list of column names
            formats (:obj:`list` str): column data formats
            last_line_style (str): css style of last row

        Example:
            >>> data = [
            ... ['row1', 'data', 'data', 80 ],
            ... ['row2', 'data', 'data', 45 ],
```

(continues on next page)

(continued from previous page)

```
            ... ]
            >>> output.print_table(
            ... table_data=data,
            ... title="Example Table",
            ... columns=["Row Name", "Column 1", "Column 2", "Percentage"],
            ... formats=['', '', '', '{}%'],
            ... last_line_style='color:red;'
            ... )
        """
        if last_line_style:
            self.add_style('tr:last-child {{ {style} }}'
                           .format(style=last_line_style))

        zipper = itertools.izip_longest
        adjust_base_col = '|'
        adjust_extra_col = ':---|'
        base_col = '|'
        extra_col = '{data}|'

        # find max column count
        max_col = max([len(x) for x in table_data])

        header = ''
        if columns:
            header = base_col
            for idx, col_name in zipper(range(max_col), columns, fillvalue=''):
                header += extra_col.format(data=col_name)

            header += '\n'

        justifier = adjust_base_col
        for idx in range(max_col):
            justifier += adjust_extra_col

        justifier += '\n'

        rows = ''
        for entry in table_data:
            row = base_col
            for idx, attrib, attr_format \
                    in zipper(range(max_col), entry, formats, fillvalue=''):
                if attr_format:
                    value = attr_format.format(attrib)
                else:
                    value = attrib
                row += extra_col.format(data=value)
            rows += row + '\n'

        table = header + justifier + rows
        self.print_md('### {title}'.format(title=title))
        self.print_md(table)

    def insert_divider(self):
        """Add horizontal rule to the output window."""
        self.print_md('-----')

    def next_page(self):
```

(continues on next page)

(continued from previous page)

```
        """Add hidden next page tag to the output window.

        This is helpful to silently separate the output to multiple pages
        for better printing.
        """
        self.print_html('<div class="nextpage"></div><div> </div>')

    @staticmethod
    def linkify(element_ids, title=None):
        """Create clickable link for the provided element ids.

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

        Args:
            element_ids (`ElementId`) or
            element_ids (:obj:`list` of `ElementId`): 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)))
        """
        return coreutils.prepare_html_str(
            linkmaker.make_link(element_ids, contents=title)
            )

    def make_chart(self):
        """:obj:`PyRevitOutputChart`: Return chart object."""
        return charts.PyRevitOutputChart(self)

    def make_line_chart(self):
        """:obj:`PyRevitOutputChart`: Return line chart object."""
        return charts.PyRevitOutputChart(self, chart_type=charts.LINE_CHART)

    def make_stacked_chart(self):
        """:obj:`PyRevitOutputChart`: Return stacked chart object."""
        chart = charts.PyRevitOutputChart(self, chart_type=charts.LINE_CHART)
        chart.options.scales = {'yAxes': [{'stacked': True, }]}
        return chart

    def make_bar_chart(self):
        """:obj:`PyRevitOutputChart`: Return bar chart object."""
        return charts.PyRevitOutputChart(self, chart_type=charts.BAR_CHART)

    def make_radar_chart(self):
        """:obj:`PyRevitOutputChart`: Return radar chart object."""
        return charts.PyRevitOutputChart(self, chart_type=charts.RADAR_CHART)

    def make_polar_chart(self):
        """:obj:`PyRevitOutputChart`: Return polar chart object."""
        return charts.PyRevitOutputChart(self, chart_type=charts.POLAR_CHART)

    def make_pie_chart(self):
        """:obj:`PyRevitOutputChart`: Return pie chart object."""
        return charts.PyRevitOutputChart(self, chart_type=charts.PIE_CHART)
```

(continues on next page)

(continued from previous page)

```
    def make_doughnut_chart(self):
        """:obj:`PyRevitOutputChart`: Return dougnut chart object."""
        return charts.PyRevitOutputChart(self,
                                         chart_type=charts.DOUGHNUT_CHART)

    def make_bubble_chart(self):
        """:obj:`PyRevitOutputChart`: Return bubble chart object."""
        return charts.PyRevitOutputChart(self, chart_type=charts.BUBBLE_CHART)


def get_output():
    """:obj:`pyrevit.output.PyRevitOutputWindow` : Return output window."""
    return PyRevitOutputWindow()
```

18.2 pyrevit.output.linkmaker

18.2.1 Documentation

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)))
```

18.2.2 Implementation

```
"""Handle creation of output window helper links."""

import json

from pyrevit import HOST_APP
from pyrevit.compat import safe_strtype
from pyrevit import DB
from pyrevit.coreutils.logger import get_logger


logger = get_logger(__name__)


PROTOCOL_NAME = 'revit://outputhelpers?'
```

(continues on next page)

(continued from previous page)

```
DEFAULT_LINK = '<a title="Click to select or show element" ' \
                  'class="elementlink" {}>{}</a>'


def 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 :func:`linkify`
    method.

    Example:
        >>> output = pyrevit.output.get_output()
        >>> for idx, elid in enumerate(element_ids):
        >>>     print('{}: {}'.format(idx+1, output.linkify(elid)))
    """
    elementquery = []
    if isinstance(element_ids, list):
        strids = [safe_strtype(x.IntegerValue) for x in element_ids]
    elif isinstance(element_ids, DB.ElementId):
        strids = [safe_strtype(element_ids.IntegerValue)]

    for strid in strids:
        elementquery.append('element[]={}'.format(strid))

    reviturl = '&'.join(elementquery)
    linkname = ', '.join(strids)

    if len(reviturl) >= 2000:
        alertjs = 'alert(&quot;Url was too long and discarded!&quot;);'
        linkattrs = 'href="#" onClick="{}"'.format(alertjs)
    else:
        linkattrs = 'href="{}{}{}"'.format(PROTOCOL_NAME,
                                           '&command=select&',
                                           reviturl)

    return DEFAULT_LINK.format(linkattrs, contents or linkname)
```

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