# revitpythonwrapper Documentation

***Release 1.7.4***

**Gui Talarico**

**May 29, 2018**

# Contents

Version: 1.7.4

modindex | genindex

CHAPTER 1

# A Python Wrapper For the Revit API

*Python Revit Api code that looks like Python*

Revit Python Wrapper was created to help Python programmers write Revit API code.

Wrapper classes make the interaction with API objects less repetitive, and more consistent with Python's conventions.

**Caution:**

API breaking changes are expected on 2.0 release (Q4 2017)

Questions? Post them over in the project's Github Page or hit me up on twitter.

## 1.1 Release Notes

Release Notes On Github Repository

## 1.2 Contribute

https://www.github.com/gtalarico/revitpythonwrapper

## 1.3 License

MIT License

# CHAPTER 2

# Using RPW

There are several ways to use RevitPythonWrapper:

- pyRevit
- RevitPythonShell
- Dynamo

For more details on how to use pyRevit in these platforms, see the *Installation* page.

## 2.1 Benefits

- Normalizes Document and Application handlers for Revit + Dynamo
- Increase code re-use across platforms (ie. *rpw and rpw.revit*)
- Implements patterns to reduce repetitive tasks (ie. `rpw.db.Transaction`, `rpw.db.Collector`)
- Handles some data-type casting for speed and flexibility (ie. `rpw.db.Parameter`)
- Normalizes API calls for different Revit Versions
- Rpw Initializes all common variables such as document handling variables (`doc` and `uidoc`) so you can reuse code across platforms with no change to your import code. See *rpw and rpw.revit*.
- Preloads `clr`, and the required Revit assemblies such as `RevitAPI.dll` and `RevitAPIUI.dll` as well as .NET types such as `List` as `Enum`. See `rpw.utils.dotnet`
- Adds IronPython Standard Library to your `sys.path` (useful for Dynamo scripts).
- Easy to use WPF *Forms* and `TaskDialog` wrapper makes it easy to request additional user input with little effort.

## 2.2 Compatibility

RevitPythonWrapper has been tested on the following platforms:

- RevitPythonShell + Revit: 2015, 2016, 2017
- pyRevit 4.4+ on 2015, 2016, 2017, 2017.1
- Dynamo: 1.2, 1.3

CHAPTER 3

# Before You start

To make it easier to users, Rpw attempts to maintain close fidelity to names and terms use by the Revit Api. So if you know the Revit API, it should feel familiar. Alternative names are only used where Revit Api names are inconvenient or inadequate. For example, the rpw *Transaction* wrapper is also called `Transaction`, however, the FilteredElementCollector wrapper, is called `Collector`.

To minimize namespace collisions, the patterns below are highly recommended:

1. Avoid `from Something import *` . This is generally not a good idea anyway.
2. Use rpw imports instead of *import clr* and *from Autodesk.Revit …* See *rpw and rpw.revit* for more details. `rpw.utils.dotnet` has .NET classes such as List and Enum ready to go.
3. Keep rpw namespaces isolated from Revit Namespaces. Rpw's wrappers are lowercase the lowercase counterpart of their Revit equivalents, such as db, and ui. Revit Namespaces are DB, UI (‘‘from Autodesk.Revit import DB‘ and ‘‘from Autodesk.Revit import UI‘)

```
>>> from rpw import revit, db, ui, DB, UI
>>> # For rpw wrappers, especially those in rpw.db, keep them inside db:
>>> doors = db.Collector(of_category='Doors')
>>> with db.Transaction('Delete'):
...     [revit.doc.Delete(id) for id in doors.element_ids]
>>> # Keep Revit namespaces them under DB:
>>> invalid_id = DB.ElementId(-1)
```

# CHAPTER 4

# Contents

genindex | modindex

## 4.1 Installation

### 4.1.1 pyRevit

RevitPythonWrapper now ships with pyRevit. Just import rpw:

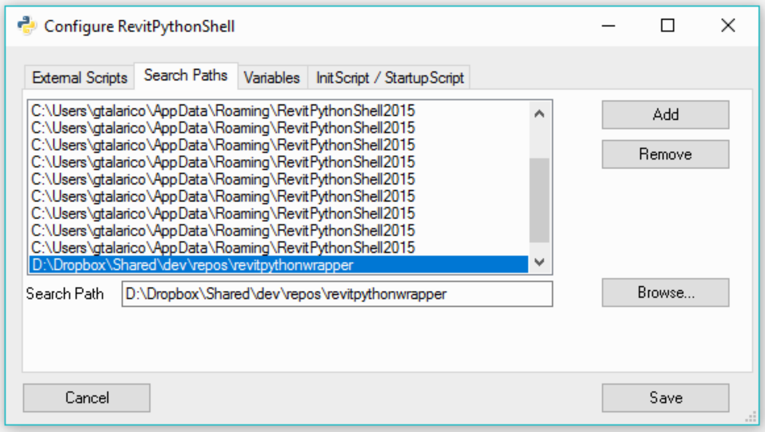
```
>>> from rpw import revit, db, ui
```

### 4.1.2 RevitPythonShell

Revit Python Wrapper works well with the RevitPythonShell

1. Clone the RevitPythonWrapper Repository
2. Add the the repository directory to RevitPythonShell's search path

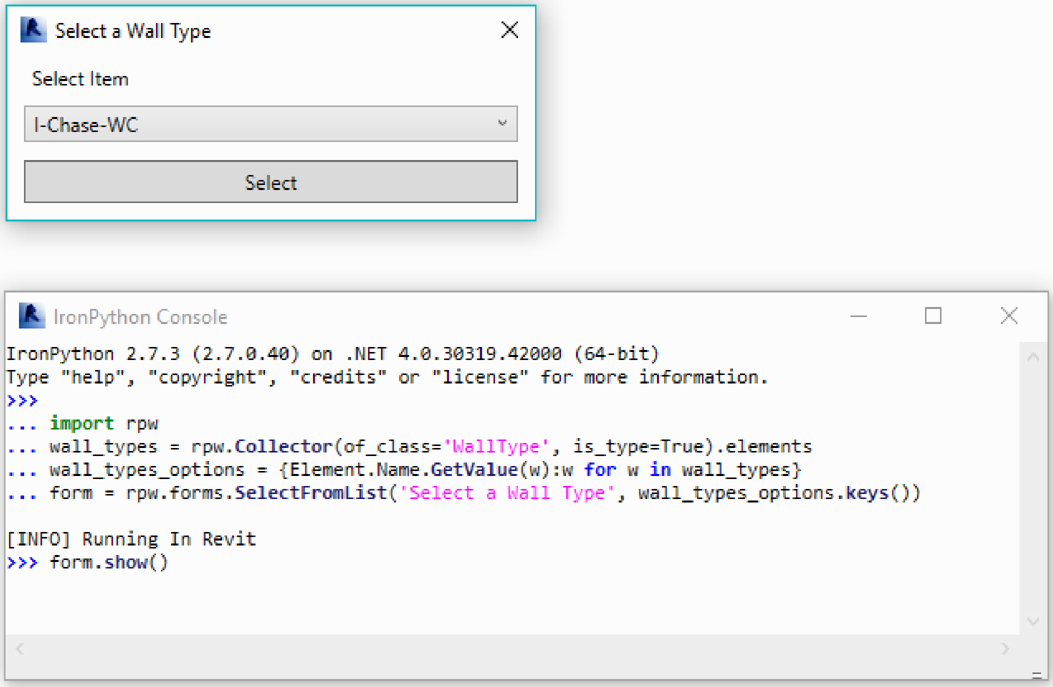
**Tip:** Add rpw to RevitPythonShell's Search Path to have it pre-loaded every you start it.



Configure RevitPythonShell

```
C:\Users\gtalarico\AppData\Roaming\RevitPythonShell2015
C:\Users\gtalarico\AppData\Roaming\RevitPythonShell2015
C:\Users\gtalarico\AppData\Roaming\RevitPythonShell2015
C:\Users\gtalarico\AppData\Roaming\RevitPythonShell2015
C:\Users\gtalarico\AppData\Roaming\RevitPythonShell2015
C:\Users\gtalarico\AppData\Roaming\RevitPythonShell2015
C:\Users\gtalarico\AppData\Roaming\RevitPythonShell2015
C:\Users\gtalarico\AppData\Roaming\RevitPythonShell2015
C:\Users\gtalarico\AppData\Roaming\RevitPythonShell2015
D:\Dropbox\Shared\dev\repos\revitpythonwrapper
```

Search Path D:\Dropbox\Shared\dev\repos\revitpythonwrapper



```
IronPython 2.7.3 (2.7.0.40) on .NET 4.0.30319.42000 (64-bit)
Type "help", "copyright", "credits" or "license" for more information.
>>>
... import rpw
... wall_types = rpw.Collector(of_class='WallType', is_type=True).elements
... wall_types_options = {Element.Name.GetValue(w):w for w in wall_types}
... form = rpw.forms.SelectFromList('Select a Wall Type', wall_types_options.keys())

[INFO] Running In Revit
>>> form.show()
```

### 4.1.3 Dynamo

RevitPythonWrapper is distributed through Dynamo's Package Manager ships with everything you need, but you can also download it your self and import it as you would with any other Python modules.

The package includes a Node that helps you find the location of the RPW library (see image below). Once you have the location, just add it to your `sys.path`, and you should be able to import the library. You can always manually add the library path; the node is only included for convenience.

For more details and question about this please see this post.

**Note:** Be sure the checkout the `RPW_GetStarted.dyn` file that is installed with the Package for practical examples. The file will typically be saved in: `.../Dynamo/1.X/packages/RevitPythonWrapper/extra/RPW_GetStarted.dyn`

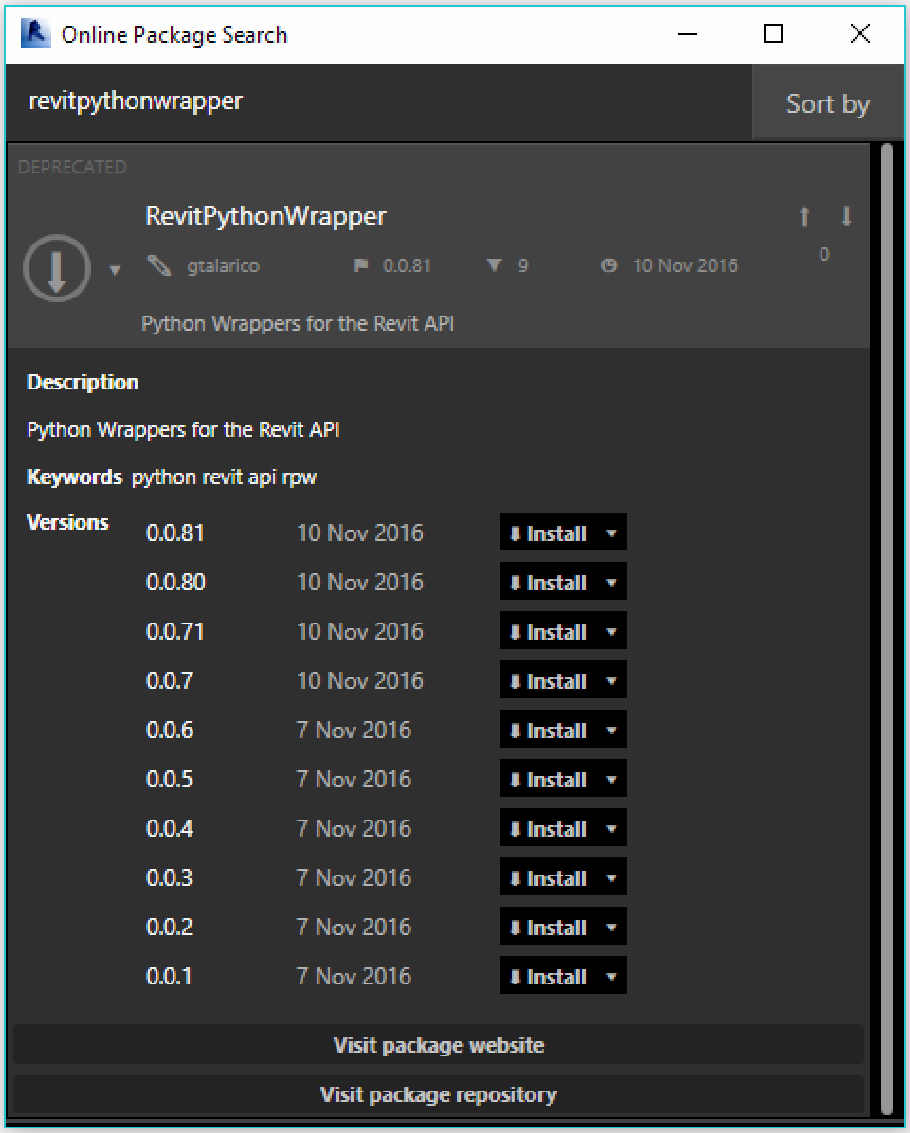
Python Code Sample

```
>>> # USe RPW_GetFilePath to find Rpw's path and plugged it into IN[0]
>>> # or append path manually
>>> import sys
>>> rpw_path = IN[0]  # IN[0] is a path to rpw.zip or the parent folder where rpw is␣
↪located
>>> sys.path.append(rpw_path)
>>> from rpw import db, ui
```

(continues on next page)

(continued from previous page)

```
>>> walls = db.Collector(of_class='Wall').elements
>>> ui.forms.Alert('Found {} walls'.format(len(walls)))
```



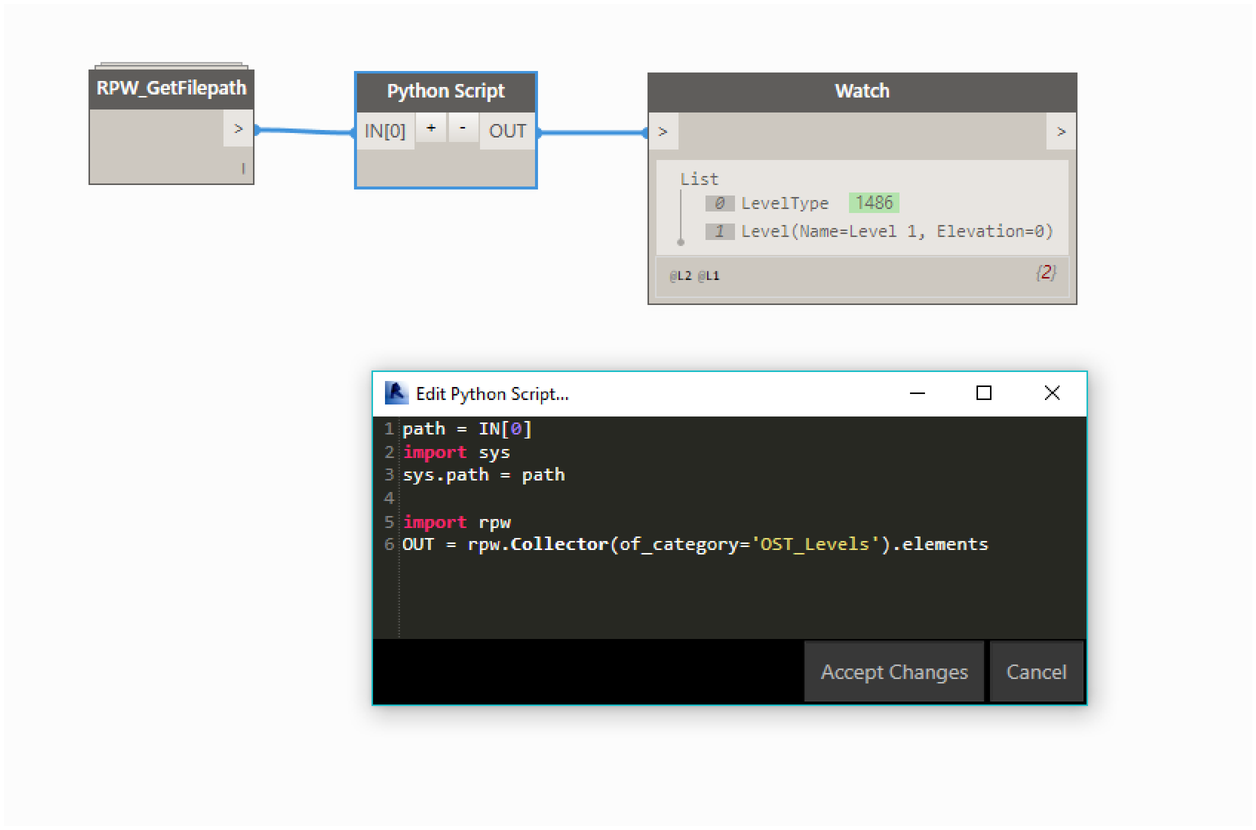
**Description**

Python Wrappers for the Revit API

**Keywords** python revit api rpw

**Versions**

| | | |
|---|---|---|
| 0.0.81 | 10 Nov 2016 | Install |
| 0.0.80 | 10 Nov 2016 | Install |
| 0.0.71 | 10 Nov 2016 | Install |
| 0.0.7 | 10 Nov 2016 | Install |
| 0.0.6 | 7 Nov 2016 | Install |
| 0.0.5 | 7 Nov 2016 | Install |
| 0.0.4 | 7 Nov 2016 | Install |
| 0.0.3 | 7 Nov 2016 | Install |
| 0.0.2 | 7 Nov 2016 | Install |
| 0.0.1 | 7 Nov 2016 | Install |



```
List
  0 LevelType 1486
  1 Level(Name=Level 1, Elevation=0)
@L2 @L1                              {2}
```

```
path = IN[0]
import sys
sys.path = path

import rpw
OUT = rpw.Collector(of_category='OST_Levels').elements
```

**Note:** Images below might be outdated

## 4.2 rpw and rpw.revit

### 4.2.1 Revit Python Wrapper

The main rpw namespace and rpw.revit provide you with most of the imports will need.

```
>>> from rpw import revit, db, ui
>>> db.Element(SomeElement)
>>> ui.Selection()
>>> revit.doc
>>> revit.uidoc.ActiveView
```

Revit Namespaces are also available:

```
>>> from rpw import DB, UI
>>> DB.ElementId(00000)
>>> UI.TaskDialog
```

In summary, if you use rpw, this could potentially be the only import line you would need:

```
>>> from rpw import revit, db, ui, DB, UI
```

`rpw.`**`UI Revit.UI Namespace`**

`rpw.DB` **Revit.DB Namespace**

### 4.2.2 Revit Wrapper

**class** `rpw.__revit.`**Revit**

Bases: `rpw.base.BaseObject`

Revit Application Wrapper

**Note:** The module path for the Revit Wrapper and its namespaces is `rpw.__revit.Revit`. However, the `Revit()` is always instantiated on the initialization of rpw, and is stored along with the `DB` and `UI` namespaces in the root of rpw module.

In other words, to use this wrapper all you need is to import `from rpw import revit`

```
>>> from rpw import revit
>>> revit.doc
<Autodesk.Revit.DB.Document>
>>> revit.username
gtalarico
>>> revit.host
'Dynamo'
```

**`__init__`**()

x.__init__(…) initializes x; see help(type(x)) for signature

**`active_view`**

*Returns* – uidoc.ActiveView

**`app`**

*Returns* – uidoc.Application

**`doc`**

*Returns* – uiapp.ActiveUIDocument.Document

**`docs`**

*Returns* – uidoc.Application.Documents

**`host`**

Host is set based on how revit handle was found.

**Returns** Revit Application Host [‘RPS’, ‘Dynamo’]

**Return type** Host (str)

**`open`**(*path*)

Opens New Document

**`process`**

*Returns* – Process.GetCurrentProcess()

**`process_id`**

*Returns* – Process.GetCurrentProcess()

**`process_name`**

*Returns* – Process.GetCurrentProcess()

**`uidoc`**

*Returns* – uiapp.ActiveUIDocument

**username**
*Returns* – uidoc.Application.Username

**version**
*Returns* – uidoc.Application.Username

---

**Hint:** Besides creating these global variables, the module's global variable initializer also adds the path to the Ironpython Library to your sys.path, so you can import standard python libraries right away, and skip the typical:

```
>>> import sys
>>> sys.path.append(r'C:\Program Files (x86)\IronPython 2.7\Lib')
```

---

### 4.2.3 Typical Methods

When RPW is not used, import code ends up being different for each platform:

```
>>> # RevitPythonShell / pyRevit
>>> import clr
>>> clr.AddReference('RevitAPI')
>>> clr.AddReference('RevitAPIUI')
>>>
>>> from Autodesk.Revit.DB import *
>>> from Autodesk.Revit.UI import *
>>>
>>> doc = __revit__.ActiveUIDocument.Document
>>> uidoc = __revit__.ActiveUIDocument
```

```
>>> # Dynamo
>>> import clr
>>> clr.AddReference('RevitAPI')
>>> clr.AddReference('RevitAPIUI')
>>> from Autodesk.Revit.DB import *
>>> from Autodesk.Revit.UI import *
>>>
>>> clr.AddReference("RevitServices")
>>> import RevitServices
>>> from RevitServices.Persistence import DocumentManager
>>> from RevitServices.Transactions import TransactionManager
>>>
>>> doc = DocumentManager.Instance.CurrentDBDocument
>>> uiapp = DocumentManager.Instance.CurrentUIApplication
>>> app = uiapp.Application
>>> uidoc = DocumentManager.Instance.CurrentUIApplication.ActiveUIDocument
```

---

### 4.2.4 Implementation

```
import rpw
from rpw.utils.dotnet import clr, Process
from rpw.utils.logger import logger
from rpw.base import BaseObject
```

(continues on next page)

(continued from previous page)

```
class Revit(BaseObject):
    """
    Revit Application Wrapper

    Note:
        The module path for the Revit Wrapper and its namespaces is ``rpw.__revit.
↪Revit``.
        However, the ``Revit()`` is always instantiated on the initialization of rpw,
        and is stored along with the ``DB`` and ``UI`` namespaces in the
        root of rpw module.

        In other words, to use this wrapper all you need is to import
        ``from rpw import revit``

    >>> from rpw import revit
    >>> revit.doc
    <Autodesk.Revit.DB.Document>
    >>> revit.username
    gtalarico
    >>> revit.host
    'Dynamo'

    """

    class HOSTS():
        RPS = 'RPS'
        DYNAMO = 'Dynamo'

    def __init__(self):
        try:
            self.uiapp = __revit__
            self._host = Revit.HOSTS.RPS
        except NameError:
            try:
                # Try Getting handler from Dynamo RevitServices
                self.uiapp = self.find_dynamo_uiapp()
                self._host = Revit.HOSTS.DYNAMO
            except Exception as errmsg:
                logger.warning('Revit Application handle could not be found')

        try:
            # Add DB UI Import to globals so it can be imported by rpw
            clr.AddReference('RevitAPI')
            clr.AddReference('RevitAPIUI')
            from Autodesk.Revit import DB, UI
            globals().update({'DB': DB, 'UI': UI})
        except Exception:
            # Replace Globals with Mock Objects for Sphinx and ipy direct exec.
            logger.warning('RevitAPI References could not be added')
            from rpw.utils.sphinx_compat import MockObject
            globals().update({'DB': MockObject(fullname='Autodesk.Revit.DB'),
                              'UI': MockObject(fullname='Autodesk.Revit.DB')})
            self.uiapp = MockObject(fullname='Autodesk.Revit.UI.UIApplication')
            self._host = None
```

(continues on next page)

(continued from previous page)

```
    def find_dynamo_uiapp(self):
        clr.AddReference("RevitServices")
        import RevitServices
        from RevitServices.Persistence import DocumentManager

        import sys
        sys.path.append(r'C:\Program Files (x86)\IronPython 2.7\Lib')
        return DocumentManager.Instance.CurrentUIApplication

    @property
    def host(self):
        """ Host is set based on how revit handle was found.

        Returns:
            Host (str): Revit Application Host ['RPS', 'Dynamo']
        """
        return self._host

    def open(self, path):
        """ Opens New Document """

    @property
    def doc(self):
        """ Returns: uiapp.ActiveUIDocument.Document """
        return getattr(self.uiapp.ActiveUIDocument, 'Document', None)

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

    @property
    def active_view(self):
        """ Returns: uidoc.ActiveView """
        return rpw.db.Element(self.uidoc.ActiveView)

    @active_view.setter
    def active_view(self, view_reference):
        self.uidoc.ActiveView = view_reference

    @property
    def app(self):
        """ Returns: uidoc.Application """
        return self.uiapp.Application

    @property
    def docs(self):
        """ Returns: uidoc.Application.Documents """
        return [doc for doc in self.app.Documents]

    @property
    def username(self):
        """ Returns: uidoc.Application.Username """
        return self.uiapp.Application.Username

    @property
    def version(self):
```

(continues on next page)

(continued from previous page)

```
        """ Returns: uidoc.Application.Username """
        return RevitVersion(self.uiapp)

    @property
    def process(self):
        """ Returns: Process.GetCurrentProcess() """
        return Process.GetCurrentProcess()

    @property
    def process_id(self):
        """ Returns: Process.GetCurrentProcess() """
        return self.process.Id

    @property
    def process_name(self):
        """ Returns: Process.GetCurrentProcess() """
        return self.process.ProcessName

    def __repr__(self):
        return '<{version} [{process}:{pid}]>'.format(version=self.version,
                                                       process=self.process_name,
                                                       pid=self.process_id)

    # Check what this is
    # @property
    # def process(self):
    #     clr.AddReferenceByPartialName('System.Windows.Forms')
    #     # noinspection PyUnresolvedReferences
    #     from System.Windows.Forms import Screen
    #     return Screen.FromHandle(Process.GetCurrentProcess().MainWindowHandle)


class RevitVersion():
    def __init__(self, uiapp):
        self.uiapp = uiapp

    @property
    def year(self):
        return self.uiapp.Application.VersionNumber

    @property
    def name(self):
        return self.uiapp.Application.VersionName

    @property
    def build(self):
        return self.uiapp.Application.VersionBuild

    def __lt__(self, other):
        """ Handle Version Comparison Logic"""
        raise NotImplemented

    def __gt__(self, other):
        """ Handle Version Comparison Logic"""
        raise NotImplemented

    def __repr__(self):
        return '<Version: {year}: {build}>'.format(year=self.name,
```

(continues on next page)

(continued from previous page)

```
                                                              build=self.build)

    def __str__(self):
        return '{name}:{build}'.format(name=self.name, build=self.build)


revit = Revit()
```

## 4.3 rpw.db

Autodesk.Revit.DB Wrappers

### 4.3.1 Element

#### Element Wrapper

Element Model Wrappers provide a consitent interface for acccessing parameters and properties of commonly used elements.

**class** rpw.db.**Element**(*element*, *doc=None*)

Bases: *rpw.base.BaseObjectWrapper*, *rpw.utils.mixins.CategoryMixin*

Inheriting from element extends wrapped elements with a new *parameters* attribute, well as the *unwrap()* method inherited from the *BaseObjectWrapper* class.

It can be created by instantiating `rpw.db.Element` , or one of the helper static methods shown below.

Most importantly, all other *Element-related* classes inhert from this class so it can provide parameter access.

```
>>> from rpw import db
>>> element = db.Element(SomeElement)
>>> element = db.Element.from_id(ElementId)
>>> element = db.Element.from_int(Integer)
```

```
>>> wall = db.Element(RevitWallElement)
>>> wall.Id
>>> wall.parameters['Height'].value
10.0
```

The `Element` Constructor can be used witout specifying the exact class. On instantiation, it will attempt to map the type provided, if a match is not found, an Element will be used. If the element does not inherit from DB.Element, and exception is raised.

```
>>> wall_instance = db.Element(SomeWallInstance)
>>> type(wall_instance)
'rpw.db.WallInstance'
>>> wall_symbol = db.Element(SomeWallSymbol)
>>> type(wall_symbol)
'rpw.db.WallSymbol'
```

**parameters**

*ParameterSet* – Access *ParameterSet* class.

parameters.**builtins**
*ParameterSet* – BuitIn *ParameterSet* object

**unwrap**()
Wrapped Revit Reference

**__init__**(*element*, *doc=None*)
Main Element Instantiation

```
>>> from rpw import db
>>> wall = db.Element(SomeElementId)
<rpw: WallInstance % DB.Wall >
>>> wall.parameters['Height']
10.0
>>> wall.parameters.builtins['WALL_LOCATION_LINE']
1
```

**Parameters** **element** (*Element Reference*) – Can be `DB.Element`, `DB.ElementId`, or `int`.

**Returns** Instance of Wrapped Element.

**Return type** *Element*

**static __new__**(*cls*, *element*, ***kwargs*)
Factory Constructor will chose the best Class for the Element. This function iterates through all classes in the rpw.db module, and will find one that wraps the corresponding class. If and exact match is not found *Element* is used

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *WallType* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**static from_id**(*element_id*, *doc=None*)
Instantiate Element from an ElementId

**Parameters**

- **id** (`ElementId`) – ElementId of Element to wrap
- **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` instance

**Return type** (`Element`)

**static from_int**(*id_int*, *doc=None*)
Instantiate Element from an Integer representing and Id

**Parameters**

- **id** (`int`) – ElementId of Element to wrap
- **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` instance

**Return type** (`Element`)

**static from_list**(*element_references*, *doc=None*)
Instantiate Elements from a list of DB.Element instances

**Parameters elements** (`[DB.Element, DB.ElementId]`) – List of element references

**Returns** List of `rpw.db.Element` instances

**Return type** (`list`)

**name**
Name Property

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

## Parameters

Parameter Wrapper

```
>>> wrapped_element.parameters['Length']
10.0
>>> wrapped_element.parameters['Length'] = 5
5.0
```

## ParameterSet

**Note:** These are used internally by all Classes that inherit from `rpw.db.element`, but can be used on their own.

**class** rpw.db.**ParameterSet**(*element*)
Bases: *rpw.base.BaseObjectWrapper*

Allows you to treat an element's parameters as a dictionary.

This is used internally by Element Wrapper. An instance of this class is returned on the `parameters` attribute of wrapped elements.

```
>>> element.parameters.all()
>>> element.parameters['Comments'].value
>>> element.parameters['Comments'].type
```

ParameterSet can also be used for setting values: >>> element.parameters['Comments'].value = 'Something'

```
>>> parameters = ParameterSet(Element)
```

**_revit_object**
*DB.Element*

**__getitem__**(*param_name*)
Get's parameter by name.

**Returns** The first parameter found with a matching name (wrapper),

**Return type** *Parameter*

**Raises** `RpwParameterNotFound`

**__init__**(*element*)

**Parameters** **element** (*DB.Element*) – Element to create ParameterSet

**__setitem__**(*param_name*, *value*)
Sets value to element's parameter. This is a shorcut to using *parameters['Name'].value = value*

```
>>> element.parameters['Height'] = value
```

**all**
*Returns* – Flat list of wrapped parameter elements

**to_dict**()
WIP: Returns a Serializable Dictionary

## Parameter

**class** rpw.db.**Parameter**(*parameter*)
Bases: *rpw.base.BaseObjectWrapper*

Primarily for internal use by *rpw.db.Element*, but can be used on it's own.

```
>>> parameter = Parameter(DB.Parameter)
>>> parameter.type
<type: str>
>>> parameter.value
'Some String'
>>> parameter.name
'Parameter Name'
>>> parameter.builtin
Revit.DB.BuiltInParameter.SOME_BUILT_IN_NAME
```

**_revit_object**
*DB.Parameter*

**Note:** Parameter Wrapper handles the following types:

- Autodesk.Revit.DB.StorageType.String
- Autodesk.Revit.DB.StorageType.Double
- Autodesk.Revit.DB.StorageType.ElementId
- Autodesk.Revit.DB.StorageType.Integer

- Autodesk.Revit.DB.StorageType.None

---

**`__init__`**(*parameter*)
Parameter Wrapper Constructor

**Parameters** **`DB.Parameter`** – Parameter to be wrapped

**Returns** Wrapped Parameter Class

**Return type** *Parameter*

**`builtin`**
Returns the BuiltInParameter name of Parameter. Same as DB.Parameter.Definition.BuiltIn

**Usage:**

```
>>> element.parameters['Comments'].builtin_name
Revit.DB.BuiltInParameter.ALL_MODEL_INSTANCE_COMMENTS
```

**Returns** BuiltInParameter Enumeration Member

**Return type** Revit.DB.BuiltInParameter

**`builtin_id`**
ElementId of BuiltIn

**Usage:**

```
>>> wall.parameters['Unconnected Height'].builtin_id
Revit.DB.BuiltInParameter.WALL_USER_HEIGHT_PARAM
```

**`name`**
Returns Parameter name

```
>>> element.parameters['Comments'].name
>>> 'Comments'
```

**`to_dict`**()
Returns Parameter as a dictionary. Included properties are:

- name: Parameter.Definition.Name
- type: Parameter.StorageType.Name
- value: Uses best parameter method based on StorageType
- value_string: Parameter.AsValueString

**`type`**
Returns the Python Type of the Parameter

```
>>> element.parameters['Height'].type
<type: float>
```

**Returns** Python Built in type

**Return type** (`type`)

**value**

*Gets Parameter Value* – >>> desk.parameters['Height'].value >>> 3.0

Sets Parameter Value (must be in Transaction Context):

```
>>> desk.parameters['Height'].value = 3
```

**Returns** parameter value in python type

**Return type** (`type`)

**Note:** *Parameter* value setter automatically handles a few type castings:

- Storage is `str` and value is `None`; value is converted to `blank_string`
- Storage is `str` and value is `any`; value is converted to `string`
- Storage is `ElementId` and value is `None`; value is converted to `ElemendId.InvalidElementId`
- Storage is `int` and value is `float`; value is converted to `int`
- Storage is `float` and value is `int`; value is converted to `float`

**value_string**

Ensure Human Readable String Value

## Implementation

```
"""
Parameter Wrapper

>>> wrapped_element.parameters['Length']
10.0
>>> wrapped_element.parameters['Length'] = 5
5.0

"""  #
from rpw import revit, DB
from rpw.db.builtins import BipEnum
from rpw.base import BaseObjectWrapper
from rpw.exceptions import RpwException, RpwWrongStorageType
from rpw.exceptions import RpwParameterNotFound, RpwTypeError
from rpw.utils.logger import logger


class ParameterSet(BaseObjectWrapper):
    """
    Allows you to treat an element's parameters as a dictionary.

    This is used internally by Element Wrapper.
    An instance of this class is returned on the ``parameters``
    attribute of wrapped elements.
```

(continues on next page)

(continued from previous page)

```
    >>> element.parameters.all()
    >>> element.parameters['Comments'].value
    >>> element.parameters['Comments'].type

    ParameterSet can also be used for setting values:
    >>> element.parameters['Comments'].value = 'Something'

    >>> parameters = ParameterSet(Element)

    Attributes:
        _revit_object (DB.Element) = Revit Reference

    """

    _revit_object_class = DB.Element

    def __init__(self, element):
        """
        Args:
            element(DB.Element): Element to create ParameterSet
        """
        super(ParameterSet, self).__init__(element)
        self.builtins = _BuiltInParameterSet(self._revit_object)

    def get_value(self, param_name, default_value=None):
        try:
            return self.__getitem__(param_name).value
        except RpwParameterNotFound:
            return default_value

    def __getitem__(self, param_name):
        """ Get's parameter by name.

        Returns:
            :any:`Parameter`: The first parameter found with a matching name␣
↪(wrapper),

        Raises:
            :class:`RpwParameterNotFound`

        """
        # TODO: Any advantage of using ParameterMap Instead
        parameter = self._revit_object.LookupParameter(param_name)
        # return _Parameter(parameter) if parameter else None
        if not parameter:
            raise RpwParameterNotFound(self._revit_object, param_name)
        return Parameter(parameter)

    def __setitem__(self, param_name, value):
        """ Sets value to element's parameter.
        This is a shorcut to using `parameters['Name'].value = value`

        >>> element.parameters['Height'] = value
        """
        parameter = self.__getitem__(param_name)
        parameter.value = value
```

(continues on next page)

(continued from previous page)

```
    @property
    def all(self):
        """ Returns: Flat list of wrapped parameter elements
        """
        return [Parameter(parameter) for parameter in self._revit_object.Parameters]

    def to_dict(self):
        """ WIP: Returns a Serializable Dictionary """
        return [p.to_dict() for p in self.all]

    def __len__(self):
        return len(self.all)

    def __repr__(self):
        """ Adds data to Base __repr__ to add Parameter List Name """
        return super(ParameterSet, self).__repr__(data={'count': len(self)})


class _BuiltInParameterSet(BaseObjectWrapper):
    """ Built In Parameter Manager

    Usage:

        location_line = element.parameters.builtins['WALL_LOCATION_LINE']

    Note:
        Item Getter can take the BuilInParameter name string, or the Enumeration.
        >>> element.parameters.builtins['WALL_LOCATION_LINE']

        or

        >>>element.parameters.builtins[Revit.DB.BuiltInParameter.WALL_LOCATION_LINE]

    Attributes:
        _revit_object (DB.Element) = Revit Reference

    """

    _revit_object_class = DB.Element

    def __getitem__(self, builtin_enum):
        """ Retrieves Built In Parameter. """
        if isinstance(builtin_enum, str):
            builtin_enum = BipEnum.get(builtin_enum)
        parameter = self._revit_object.get_Parameter(builtin_enum)
        if not parameter:
            raise RpwParameterNotFound(self._revit_object, builtin_enum)
        return Parameter(parameter)

    def __setitem__(self, name, param_value):
        """ Sets value for an element's built in parameter. """
        builtin_parameter = self.__getitem__(name)
        builtin_parameter.value = param_value

    def __repr__(self):
        """ Adds data to Base __repr__ to add Parameter List Name """
        return super(_BuiltInParameterSet, self).__repr__()
```

(continues on next page)

(continued from previous page)

```
class Parameter(BaseObjectWrapper):
    """
    Primarily for internal use by :any:`rpw.db.Element`, but can be used on it's own.

    >>> parameter = Parameter(DB.Parameter)
    >>> parameter.type
    <type: str>
    >>> parameter.value
    'Some String'
    >>> parameter.name
    'Parameter Name'
    >>> parameter.builtin
    Revit.DB.BuiltInParameter.SOME_BUILT_IN_NAME

    Attributes:
        _revit_object (DB.Parameter) = Revit Reference

    Note:

        Parameter Wrapper handles the following types:

        * Autodesk.Revit.DB.StorageType.String
        * Autodesk.Revit.DB.StorageType.Double
        * Autodesk.Revit.DB.StorageType.ElementId
        * Autodesk.Revit.DB.StorageType.Integer
        * Autodesk.Revit.DB.StorageType.None


    """

    _revit_object_class = DB.Parameter
    STORAGE_TYPES = {
                    'String': str,
                    'Double': float,
                    'Integer': int,
                    'ElementId': DB.ElementId,
                    'None': None,
                     }

    def __init__(self, parameter):
        """ Parameter Wrapper Constructor

        Args:
            DB.Parameter: Parameter to be wrapped

        Returns:
            Parameter: Wrapped Parameter Class
        """
        if not isinstance(parameter, DB.Parameter):
            raise RpwTypeError(DB.Parameter, type(parameter))
        super(Parameter, self).__init__(parameter)

    @property
    def type(self):
        """ Returns the Python Type of the Parameter
```

(continues on next page)

(continued from previous page)

```
        >>> element.parameters['Height'].type
        <type: float>

        Returns:
            (``type``): Python Built in type

        """
        storage_type_name = self._revit_object.StorageType.ToString()
        python_type = Parameter.STORAGE_TYPES[storage_type_name]

        return python_type

    @property
    def parameter_type(self):
        parameter_type = self._revit_object.Definition.ParameterType
        return parameter_type

    @property
    def id(self):
        return self._revit_object.Id

    @property
    def value(self):
        """
        Gets Parameter Value:

        >>> desk.parameters['Height'].value
        >>> 3.0

        Sets Parameter Value (must be in Transaction Context):

        >>> desk.parameters['Height'].value = 3

        Returns:
            (``type``): parameter value in python type


        Note:

            `Parameter` value setter automatically handles a few type castings:

            * Storage is ``str`` and value is ``None``; value is converted to ``blank_
↪string``
            * Storage is ``str`` and value is ``any``; value is converted to␣
↪``string``
            * Storage is ``ElementId`` and value is ``None``; value is
              converted to ``ElemendId.InvalidElementId``
            * Storage is ``int`` and value is ``float``; value is converted to ``int``
            * Storage is ``float`` and value is ``int``; value is converted to␣
↪``float``

        """
        if self.type is str:
            return self._revit_object.AsString()
        if self.type is float:
            return self._revit_object.AsDouble()
```

(continues on next page)

(continued from previous page)

```
        if self.type is DB.ElementId:
            return self._revit_object.AsElementId()
        if self.type is int:
            return self._revit_object.AsInteger()

        raise RpwException('could not get storage type: {}'.format(self.type))

    @value.setter
    def value(self, value):
        if self._revit_object.IsReadOnly:
            definition_name = self._revit_object.Definition.Name
            raise RpwException('Parameter is Read Only: {}'.format(definition_name))

        # Check if value provided matches storage type
        if not isinstance(value, self.type):
            # If not, try to handle
            if self.type is str and value is None:
                value = ''
            if self.type is str and value is not None:
                value = str(value)
            elif self.type is DB.ElementId and value is None:
                value = DB.ElementId.InvalidElementId
            elif isinstance(value, int) and self.type is float:
                value = float(value)
            elif isinstance(value, float) and self.type is int:
                value = int(value)
            elif isinstance(value, bool) and self.type is int:
                value = int(value)
            else:
                raise RpwWrongStorageType(self.type, value)

        param = self._revit_object.Set(value)
        return param

    @property
    def value_string(self):
        """ Ensure Human Readable String Value """
        return self._revit_object.AsValueString() or \
                self._revit_object.AsString()

    def to_dict(self):
        """
        Returns Parameter as a dictionary. Included properties are:

            * name: Parameter.Definition.Name
            * type: Parameter.StorageType.Name
            * value: Uses best parameter method based on StorageType
            * value_string: Parameter.AsValueString
        """
        value = self.value if not isinstance(self.value, DB.ElementId) \
                           else self.value.IntegerValue
        return {
                'name': self.name,
                'type': self.type.__name__,
                'value': value,
                'value_string': self.value_string
                }
```

(continues on next page)

(continued from previous page)

```
    def __bool__(self):
        return bool(self.value)

    def __eq__(self, other):
        """ Equal Parameter Value Comparison """
        return self.value == other

    def __ne__(self, other):
        """ Not Equal Parameter Value Comparison """
        return self.value != other

    def __gt__(self, other):
        """ Greater than Parameter Value Comparison """
        return self.value > other

    def __ge__(self, other):
        """ Greater or Equal Parameter Value Comparison """
        return self.value >= other

    def __lt__(self, other):
        """ Less than Parameter Value Comparison """
        return self.value < other

    def __le__(self, other):
        """ Less or Equal Parameter Value Comparison """
        return self.value <= other

    @property
    def name(self):
        """
        Returns Parameter name

        >>> element.parameters['Comments'].name
        >>> 'Comments'
        """
        return self._revit_object.Definition.Name

    @property
    def builtin(self):
        """ Returns the BuiltInParameter name of Parameter.
        Same as DB.Parameter.Definition.BuiltIn

        Usage:
            >>> element.parameters['Comments'].builtin_name
            Revit.DB.BuiltInParameter.ALL_MODEL_INSTANCE_COMMENTS

        Returns:
            Revit.DB.BuiltInParameter: BuiltInParameter Enumeration Member
        """
        return self._revit_object.Definition.BuiltInParameter

    @property
    def builtin_id(self):
        """ ElementId of BuiltIn

        Usage:
```

(continues on next page)

(continued from previous page)

```
            >>> wall.parameters['Unconnected Height'].builtin_id
            Revit.DB.BuiltInParameter.WALL_USER_HEIGHT_PARAM
        """
        return DB.ElementId(self.builtin)

    def __repr__(self):
        """ Adds data to Base __repr__ to add selection count"""
        return super(Parameter, self).__repr__(data={
                                            'name': self.name,
                                            'value': self.value,
                                            'type': self.type.__name__
                                            })
```

## Element-based Wrappers

### Assembly

### AssemblyInstance

**class** rpw.db.**AssemblyInstance**(*element*, *doc=None*)

Bases: rpw.db.element.Element, *rpw.utils.mixins.CategoryMixin*

*DB.AssemblyInstance* Wrapper

**Attribute:** _revit_object (DB.AssemblyInstance): Wrapped DB.AssemblyInstance

**category**

Wrapped DB.Category

**classmethod collect**(***kwargs*)

Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *WallType* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()

Deletes Element from Model

**get_category**(*wrapped=True*)

Wrapped DB.Category

**get_elements**(*wrapped=True*)

Get other elements inside parent assembly.

**Returns** other elements inside the assembly

**name**
Name Property

**symbol**
Alias to AssemblyInstance.type

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

## AssemblyType

**class** rpw.db.**AssemblyType**(*element*, *doc=None*)
Bases: `rpw.db.family.FamilySymbol`, *`rpw.utils.mixins.CategoryMixin`*

*DB.AssemblyType* Wrapper Inherits from *`Element`*

**Attribute:** _revit_object (DB.AssemblyType): Wrapped `DB.AssemblyType`

**category**
Wrapped `DB.Category`

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *`WallType`* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**family**
*Returns* – *`Family`* – Wrapped `DB.Family` of the symbol

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**get_family**(*wrapped=True*)

**Returns** Wrapped `DB.Family` of the symbol

**Return type** *`Family`*

**get_instances**(*wrapped=True*)

**Returns**

**List of model instances of** the symbol (unwrapped)

**Return type** [`DB.FamilyInstance`]

**get_siblings**(*wrapped=True*)

**Returns**

**List of symbol Types** of the same Family (unwrapped)

**Return type** [`DB.FamilySymbol`]

**name**

Name Property

**siblings**

Returns all assembly types

**type**

Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()

Returns the Original Wrapped Element

## Implementation

```
""" Assembly Wrappers """

from rpw import revit, DB
from rpw.db.element import Element
from rpw.db.family import FamilyInstance, FamilySymbol

from rpw.utils.coerce import to_elements
from rpw.utils.mixins import CategoryMixin


# TODO: Tests
# TODO: Inherit from FamilyInstance Instead

class AssemblyInstance(Element, CategoryMixin):
    """
    `DB.AssemblyInstance` Wrapper

    Attribute:
        _revit_object (DB.AssemblyInstance): Wrapped ``DB.AssemblyInstance``
    """

    _revit_object_class = DB.AssemblyInstance
```

(continues on next page)

(continued from previous page)

```
    _collector_params = {'of_class': _revit_object_class, 'is_type': False}

    @property
    def symbol(self):
        """ Alias to AssemblyInstance.type """
        return self.type

    def get_elements(self, wrapped=True):
        """
        Get other elements inside parent assembly.

        Returns:
            other elements inside the assembly

        """
        member_ids = self._revit_object.GetMemberIds()
        elements = to_elements(member_ids, doc=self._revit_object.Document)
        return [Element(e) if wrapped else e for e in elements]

    def __repr__(self):
        return Element.__repr__(self, data={'name': self.Name})


class AssemblyType(FamilySymbol, CategoryMixin):
    """
    `DB.AssemblyType` Wrapper
    Inherits from :any:`Element`

    Attribute:
        _revit_object (DB.AssemblyType): Wrapped ``DB.AssemblyType``
    """

    _revit_object_class = DB.AssemblyType
    _collector_params = {'of_class': _revit_object_class, 'is_type': True}

    @property
    def siblings(self):
        """ Returns all assembly types """
        return [Element.from_id(t) for t in self._revit_object.GetSimilarTypes()]

    @property
    def instances(self):
        raise NotImplemented
        """ Returns all Instances of the assembly type """
        # bip = BipEnum.get_id('AREA_SCHEME_ID')
        # param_filter = rpw.db.Collector.ParameterFilter(bip, equals=self._revit_
↪object.Id)
        # collector = rpw.db.Collector(parameter_filter=param_filter,
        #                              **Area._collector_params)
        # return collector.wrapped_elements


    def __repr__(self):
        return Element.__repr__(self, data={'name': self.name})
```

### Family

#### Instance

**class** rpw.db.**FamilyInstance**(*element*, *doc=None*)

Bases: rpw.db.element.Element, *rpw.utils.mixins.CategoryMixin*

*DB.FamilyInstance* Wrapper

```
>>> instance = rpw.db.Element(SomeFamilyInstance)
<rpw:FamilyInstance % DB.FamilyInstance | name:72" x 36">
>>> instance.get_symbol().name
'72" x 36"'
>>> instance.get_family()
<RPW_Family:desk>
>>> instance.get_siblings()
[<rpw:FamilyInstance % DB.FamilyInstance | name:72" x 36">, ... ]
```

**Attribute:** _revit_object (DB.FamilyInstance): Wrapped DB.FamilyInstance

**__init__**(*element*, *doc=None*)

Main Element Instantiation

```
>>> from rpw import db
>>> wall = db.Element(SomeElementId)
<rpw: WallInstance % DB.Wall >
>>> wall.parameters['Height']
10.0
>>> wall.parameters.builtins['WALL_LOCATION_LINE']
1
```

**Parameters element** (*Element Reference*) – Can be DB.Element, DB.ElementId, or int.

**Returns** Instance of Wrapped Element.

**Return type** *Element*

**category**

Wrapped DB.Category

**classmethod collect**(***kwargs*)

Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *WallType* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**`delete`**`()`
Deletes Element from Model

**`family`**
Wrapped `DB.Family` of the `DB.FamilyInstance`

**`get_assembly`**
*Returns –*

**(bool, DB.Element) `None` if element not in Assembly, else** returns Element

**`get_category`**`(`*wrapped=True*`)`
Wrapped `DB.Category`

**`get_family`**`(`*wrapped=True*`)`
Wrapped `DB.Family` of the `DB.FamilyInstance`

**`get_siblings`**`(`*wrapped=True*`)`
Other `DB.FamilyInstance` of the same `DB.FamilySymbol`

**`get_symbol`**`(`*wrapped=True*`)`
`DB.FamilySymbol` of the `DB.FamilyInstance`

**`in_assembly`**
**Returns* – (bool)* – True if element is inside an AssemblyInstance

**`name`**
Name Property

**`siblings`**
Other `DB.FamilyInstance` of the same `DB.FamilySymbol`

**`symbol`**
Wrapped `DB.FamilySymbol` of the `DB.FamilyInstance`

**`type`**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **`doc`** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**`unwrap`**`()`
Returns the Original Wrapped Element

## Symbol

**`class`** `rpw.db.`**`FamilySymbol`**`(`*element*, *doc=None*`)`
Bases: `rpw.db.element.Element`, *`rpw.utils.mixins.CategoryMixin`*

*DB.FamilySymbol* Wrapper

```
>>> symbol = rpw.db.Element(SomeSymbol)
<rpw:FamilySymbol % DB.FamilySymbol | name:72" x 36">
>>> instance.get_symbol().name
'72" x 36"'
>>> instance.family
<rpw:Family % DB.Family | name:desk>
```

(continues on next page)

(continued from previous page)

```
>>> instance.siblings
<rpw:Family % DB.Family | name:desk>, ... ]
```

**Attribute:** _revit_object (DB.FamilySymbol): Wrapped `DB.FamilySymbol`

**`__init__`**(*element*, *doc=None*)
Main Element Instantiation

```
>>> from rpw import db
>>> wall = db.Element(SomeElementId)
<rpw: WallInstance % DB.Wall >
>>> wall.parameters['Height']
10.0
>>> wall.parameters.builtins['WALL_LOCATION_LINE']
1
```

**Parameters** **`element`** (*Element Reference*) – Can be `DB.Element`, `DB.ElementId`, or `int`.

**Returns** Instance of Wrapped Element.

**Return type** *`Element`*

**`category`**
Wrapped `DB.Category`

**`classmethod collect`**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *`WallType`* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**`delete`**()
Deletes Element from Model

**`family`**
*Returns* – *`Family`* – Wrapped `DB.Family` of the symbol

**`get_category`**(*wrapped=True*)
Wrapped `DB.Category`

**`get_family`**(*wrapped=True*)

**Returns** Wrapped `DB.Family` of the symbol

**Return type** *`Family`*

**`get_instances`**(*wrapped=True*)

**Returns**

**List of model instances of** the symbol (unwrapped)

**Return type** `[DB.FamilyInstance]`

**get_siblings**(*wrapped=True*)

**Returns**

**List of symbol Types** of the same Family (unwrapped)

**Return type** `[DB.FamilySymbol]`

**instances**

*Returns* –

**[DB.FamilyInstance]: List of model instances** of the symbol (unwrapped)

**name**

Name Property

**type**

Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()

Returns the Original Wrapped Element

## Family

**class** rpw.db.**Family**(*element, doc=None*)

Bases: `rpw.db.element.Element`, *`rpw.utils.mixins.CategoryMixin`*

*DB.Family* Wrapper

**Attribute:** _revit_object (DB.Family): Wrapped `DB.Family`

**__init__**(*element, doc=None*)

Main Element Instantiation

```
>>> from rpw import db
>>> wall = db.Element(SomeElementId)
<rpw: WallInstance % DB.Wall >
>>> wall.parameters['Height']
10.0
>>> wall.parameters.builtins['WALL_LOCATION_LINE']
1
```

**Parameters** **element** (*Element Reference*) – Can be `DB.Element`, `DB.ElementId`, or `int`.

**Returns** Instance of Wrapped Element.

**Return type** *`Element`*

**category**
Wrapped `DB.Category`

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *`WallType`* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**get_instances**(*wrapped=True*)
Returns: [`DB.FamilyInstance`]: List of model instances in this family (unwrapped)

**get_siblings**(*wrapped=True*)
Returns: [`DB.Family`]: List of Family elements in the same category (unwrapped)

**get_symbols**(*wrapped=True*)
Returns: [`DB.FamilySymbol`]: List of Symbol Types in the family (unwrapped)

**name**
Name Property

**siblings**
*Returns* – [`DB.Family`] – List of Family elements in the same category (unwrapped)

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

## Implementation

```python
import rpw
from rpw import revit, DB
from rpw.db.element import Element
from rpw.base import BaseObjectWrapper
from rpw.exceptions import RpwException
from rpw.utils.logger import logger, deprecate_warning
from rpw.utils.mixins import CategoryMixin
from rpw.db.builtins import BicEnum
from rpw.db.category import Category


class FamilyInstance(Element, CategoryMixin):
    """
    `DB.FamilyInstance` Wrapper

    >>> instance = rpw.db.Element(SomeFamilyInstance)
    <rpw:FamilyInstance % DB.FamilyInstance | name:72" x 36">
    >>> instance.get_symbol().name
    '72" x 36"'
    >>> instance.get_family()
    <RPW_Family:desk>
    >>> instance.get_siblings()
    [<rpw:FamilyInstance % DB.FamilyInstance | name:72" x 36">, ... ]

    Attribute:
        _revit_object (DB.FamilyInstance): Wrapped ``DB.FamilyInstance``
    """

    _revit_object_class = DB.FamilyInstance
    _collector_params = {'of_class': _revit_object_class, 'is_not_type': True}

    def get_symbol(self, wrapped=True):
        """ ``DB.FamilySymbol`` of the ``DB.FamilyInstance`` """
        symbol = self._revit_object.Symbol
        return FamilySymbol(symbol) if wrapped else symbol

    @property
    def symbol(self):
        """ Wrapped ``DB.FamilySymbol`` of the ``DB.FamilyInstance`` """
        deprecate_warning('FamilyInstance.symbol',
                          'FamilyInstance.get_symbol(wrapped=True)')
        return self.get_symbol(wrapped=True)

    def get_family(self, wrapped=True):
        """ Wrapped ``DB.Family`` of the ``DB.FamilyInstance`` """
        symbol = self.get_symbol()
        return symbol.get_family(wrapped=wrapped)

    @property
    def family(self):
        """ Wrapped ``DB.Family`` of the ``DB.FamilyInstance`` """
        deprecate_warning('FamilyInstance.family',
                          'FamilyInstance.get_family(wrapped=True)')
        return self.get_family(wrapped=True)

    def get_siblings(self, wrapped=True):
```

(continues on next page)

(continued from previous page)

```
        """ Other ``DB.FamilyInstance`` of the same ``DB.FamilySymbol`` """
        symbol = self.get_symbol()
        return symbol.get_instances(wrapped=wrapped)

    @property
    def siblings(self):
        """ Other ``DB.FamilyInstance`` of the same ``DB.FamilySymbol`` """
        deprecate_warning('FamilyInstance.siblings',
                          'FamilyInstance.get_siblings(wrapped=True)')
        return self.get_siblings(wrapped=True)

    @property
    def in_assembly(self):
        """
        Returns:
            (bool): True if element is inside an AssemblyInstance
        """
        if self._revit_object.AssemblyInstanceId.IntegerValue == -1:
            return False
        else:
            return True

    @property
    def get_assembly(self, wrapped=True):
        """
        Returns:
            (bool, DB.Element) ``None`` if element not in Assembly, else
                returns Element
        """
        if self.in_assembly:
            assembly_id = self._revit_object.AssemblyInstanceId
            assembly = self.doc.GetElement()
            return  Element(assembly) if wrapped else assembly
        else:
            return None

    def __repr__(self):
        symbol_name = self.get_symbol(wrapped=True).name
        return super(FamilyInstance, self).__repr__(data={'symbol': symbol_name})


class FamilySymbol(Element, CategoryMixin):
    """
    `DB.FamilySymbol` Wrapper

    >>> symbol = rpw.db.Element(SomeSymbol)
    <rpw:FamilySymbol % DB.FamilySymbol | name:72" x 36">
    >>> instance.get_symbol().name
    '72" x 36"'
    >>> instance.family
    <rpw:Family % DB.Family | name:desk>
    >>> instance.siblings
    <rpw:Family % DB.Family | name:desk>, ... ]

    Attribute:
        _revit_object (DB.FamilySymbol): Wrapped ``DB.FamilySymbol``
    """
```

(continues on next page)

(continued from previous page)

```
    _revit_object_class = DB.FamilySymbol
    _collector_params = {'of_class': _revit_object_class, 'is_type': True}

    def get_family(self, wrapped=True):
        """
        Returns:
            :any:`Family`: Wrapped ``DB.Family`` of the symbol

        """
        family = self._revit_object.Family
        return Family(family) if wrapped else family

    @property
    def family(self):
        """Returns:
            :any:`Family`: Wrapped ``DB.Family`` of the symbol """
        deprecate_warning('FamilySymbol.family',
                          'FamilySymbol.get_family(wrapped=True)')
        return self.get_family(wrapped=True)

    def get_instances(self, wrapped=True):
        """
        Returns:
            [``DB.FamilyInstance``]: List of model instances of
                the symbol (unwrapped)
        """
        collector = rpw.db.Collector(symbol=self._revit_object.Id, is_not_type=True)
        return collector.get_elements(wrapped)

    @property
    def instances(self):
        """
        Returns:
            [``DB.FamilyInstance``]: List of model instances
                of the symbol (unwrapped)
        """
        deprecate_warning('FamilySymbol.instances',
                          'FamilySymbol.get_instances(wrapped=True)')
        return self.get_instances(wrapped=True)

    def get_siblings(self, wrapped=True):
        """
        Returns:
            [``DB.FamilySymbol``]: List of symbol Types
                of the same Family (unwrapped)
        """
        symbols_ids = self._revit_object.GetSimilarTypes()
        return [self.doc.GetElement(i) for i in symbols_ids]
        # Same as: return self.family.symbols

    @property
    def siblings(self):
        deprecate_warning('FamilySymbol.siblings',
                          'FamilySymbol.get_siblings(wrapped=True)')
        return self.get_siblings(wrapped=True)

```

(continues on next page)

(continued from previous page)

```
    def __repr__(self):
        return super(FamilySymbol, self).__repr__(data={'name': self.name})


class Family(Element, CategoryMixin):
    """
    `DB.Family` Wrapper

    Attribute:
        _revit_object (DB.Family): Wrapped ``DB.Family``
    """

    _revit_object_class = DB.Family
    _collector_params = {'of_class': _revit_object_class}

    def get_instances(self, wrapped=True):
        """Returns:
            [``DB.FamilyInstance``]: List of model instances in this family␣
↪(unwrapped)
        """
        # There has to be a better way
        instances = []
        for symbol in self.get_symbols(wrapped=True):
            symbol_instances = symbol.get_instances(wrapped=wrapped)
            instances.append(symbol_instances)
        return instances

    @property
    def instances(self):
        deprecate_warning('Family.instances',
                          'Family.get_instances(wrapped=True)')
        return self.get_instances(wrapped=True)

    def get_symbols(self, wrapped=True):
        """Returns:
            [``DB.FamilySymbol``]: List of Symbol Types in the family (unwrapped)
        """
        symbols_ids = self._revit_object.GetFamilySymbolIds()
        elements = [self.doc.GetElement(i) for i in symbols_ids]
        return [Element(e) for e in elements] if wrapped else elements

    @property
    def symbols(self):
        deprecate_warning('Family.symbols',
                          'Family.get_symbols(wrapped=True)')
        return self.get_symbols(wrapped=True)

    def get_siblings(self, wrapped=True):
        """Returns:
            [``DB.Family``]: List of Family elements in the same category (unwrapped)
        """
        return self.category.get_families(wrapped=wrapped)

    @property
    def siblings(self):
        """Returns:
            [``DB.Family``]: List of Family elements in the same category (unwrapped)
```

(continues on next page)

(continued from previous page)

```
        """
        return self.get_siblings(wrapped=True)

    @property
    def _category(self):
        """Returns:
            :any:`Category`: Wrapped ``DB.Category`` of the Family """
        return self._revit_object.FamilyCategory

    def __repr__(self):
        return super(Family, self).__repr__({'name': self.name})
```

## Pattern

Pattern Wrappers

**class** rpw.db.**LinePatternElement**(*element*, *doc=None*)

Bases: `rpw.db.element.Element`, *rpw.utils.mixins.ByNameCollectMixin*

*DB.LinePatternElement* Wrapper

Solid, Dash, etc

**Attribute:** _revit_object (DB.LinePatternElement): Wrapped `DB.LinePatternElement`

**classmethod by_name**(*name*)

Mixin to provide instantiating by a name for classes that are collectible. This is a mixin so specifi usage will vary for each for. This method will call the *rpw.db.Element.collect* method of the class, and return the first element with a matching `.name` property.

```
>>> LinePatternElement.by_name('Dash')
<rpw:LinePatternElement name:Dash>
```

```
>>> FillPatternElement.by_name('Solid')
<rpw:FillPatternElement name:Solid>
```

**classmethod by_name_or_element_ref**(*reference*)

Mixin for collectible elements. This is to help cast elements from name, elemente, or element_id

**category**

Wrapped `DB.Category`

**classmethod collect**(***kwargs*)

Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *WallType* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
```

(continues on next page)

(continued from previous page)

```
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**name**
Name Property

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

**class** rpw.db.**FillPatternElement**(*element*, *doc=None*)
Bases: `rpw.db.pattern.LinePatternElement`

*DB.FillPatternElement* Wrapper

Solid, Horizontal, Vertical, Diagonal Down, etc

**Attribute:** _revit_object (DB.FillPatternElement): Wrapped `DB.FillPatternElement`

**classmethod by_name**(*name*)
Mixin to provide instantiating by a name for classes that are collectible. This is a mixin so specifi usage will vary for each for. This method will call the *`rpw.db.Element.collect`* method of the class, and return the first element with a matching `.name` property.

```
>>> LinePatternElement.by_name('Dash')
<rpw:LinePatternElement name:Dash>
```

```
>>> FillPatternElement.by_name('Solid')
<rpw:FillPatternElement name:Solid>
```

**classmethod by_name_or_element_ref**(*reference*)
Mixin for collectible elements. This is to help cast elements from name, elemente, or element_id

**category**
Wrapped `DB.Category`

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *`WallType`* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
```

(continues on next page)

(continued from previous page)

```
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**name**
Name Property

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

## Implementation

```
""" Pattern Wrappers """

from rpw import DB
from rpw.db.element import Element
from rpw.utils.mixins import ByNameCollectMixin


class LinePatternElement(Element, ByNameCollectMixin):
    """
    `DB.LinePatternElement` Wrapper

    Solid, Dash, etc

    Attribute:
        _revit_object (DB.LinePatternElement): Wrapped ``DB.LinePatternElement``

    """

    _revit_object_class = DB.LinePatternElement
    _collector_params = {'of_class': _revit_object_class, 'is_type': False}

    def __repr__(self):
        return Element.__repr__(self, data={'name': self.name})
```

(continues on next page)

(continued from previous page)

```
class FillPatternElement(LinePatternElement):
    """
    `DB.FillPatternElement` Wrapper

    Solid, Horizontal, Vertical, Diagonal Down, etc

    Attribute:
        _revit_object (DB.FillPatternElement): Wrapped ``DB.FillPatternElement``
    """

    _revit_object_class = DB.FillPatternElement
    _collector_params = {'of_class': _revit_object_class, 'is_type': False}
```

## Spatial Elements

### Room Wrapper

**class** rpw.db.**Room**(*element*, *doc=None*)

Bases: `rpw.db.element.Element`

*DB.Architecture.Room* Wrapper Inherits from *Element*

```
>>> from rpw import db
>>> room = db.Room(SomeRoom)
<rpw:Room % DB.Architecture.Room | name:Office number:122>
>>> room.name
'Office'
>>> room.number
'122'
>>> room.is_placed
True
>>> room.is_bounded
True
```

**Attribute:** _revit_object (DB.Architecture.Room): Wrapped `DB.Architecture.Room`

**__init__**(*element*, *doc=None*)

Main Element Instantiation

```
>>> from rpw import db
>>> wall = db.Element(SomeElementId)
<rpw: WallInstance % DB.Wall >
>>> wall.parameters['Height']
10.0
>>> wall.parameters.builtins['WALL_LOCATION_LINE']
1
```

**Parameters element** (*Element Reference*) – Can be `DB.Element`, `DB.ElementId`, or `int`.

**Returns** Instance of Wrapped Element.

**Return type** *Element*

**_category**
Default Category Access Parameter. Overwrite on wrapper as needed. See Family Wrapper for an example.

**category**
Wrapped `DB.Category`

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *WallType* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**is_bounded**
`bool` for whether Room is Bounded. Uses result of `Room.Area` attribute to define if room is Bounded.

**is_placed**
`bool` for whether Room is Placed. Uses result of `Room.Location` attribute to define if room is Placed.

**name**
*Room Name as parameter Value* – `ROOM_NAME` built-in parameter

**number**
*Room Number as parameter Value* – `ROOM_NUMBER` built-in parameter

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

### Area Wrapper

**class** `rpw.db.`**`Area`**(*element*, *doc=None*)

Bases: `rpw.db.spatial_element.Room`

*DB.Area* Wrapper Inherits from *Room*

```
>>> from rpw import db
>>> area = db.Area(SomeArea)
<rpw:Area % DB.Area | name:USF area: 100.0>
>>> area.name
'Rentable'
>>> area.is_placed
True
>>> area.is_bounded
True
```

**Attribute:** _revit_object (DB.Area): Wrapped `DB.Area`

**`__init__`**(*element*, *doc=None*)

Main Element Instantiation

```
>>> from rpw import db
>>> wall = db.Element(SomeElementId)
<rpw: WallInstance % DB.Wall >
>>> wall.parameters['Height']
10.0
>>> wall.parameters.builtins['WALL_LOCATION_LINE']
1
```

**Parameters** **`element`** (*Element Reference*) – Can be `DB.Element`, `DB.ElementId`, or `int`.

**Returns** Instance of Wrapped Element.

**Return type** *Element*

**`_category`**

Default Category Access Parameter. Overwrite on wrapper as needed. See Family Wrapper for an example.

**`area`**

*Area* – .Area attribute

**`category`**

Wrapped `DB.Category`

**classmethod** **`collect`**(***kwargs*)

Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *WallType* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
```

(continues on next page)

(continued from previous page)

```
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**is_bounded**
`bool` for whether Room is Bounded. Uses result of `Room.Area` attribute to define if room is Bounded.

**is_placed**
`bool` for whether Room is Placed. Uses result of `Room.Location` attribute to define if room is Placed.

**name**
*Area Scheme Name* – Area attribute parameter

**number**
*Room Number as parameter Value* – `ROOM_NUMBER` built-in parameter

**scheme**
*Area Scheme* – Wrapped Area Scheme

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

## Area Scheme Wrapper

**class** rpw.db.**AreaScheme**(*element*, *doc=None*)
Bases: `rpw.db.element.Element`

*DB.AreaScheme* Wrapper Inherits from *`Element`*

```
>>> scheme = wrapped_area.scheme
<rwp:AreaScheme % DB.AreaScheme | name:USF>
>>> scheme.areas
[ < Autodesk.Revit.DB.Area>, ...]
>>> scheme.name
'USF'
```

**Attribute:** _revit_object (DB.AreaScheme): Wrapped `DB.AreaScheme`

**areas**
Returns all Area Instances of this Area Scheme

**name**
*Area Scheme Name* – Area attribute parameter

### Implementation

```
import rpw
from rpw import revit, DB
from rpw.db import Element
from rpw.utils.logger import logger
from rpw.db.builtins import BipEnum


class Room(Element):
    """
    `DB.Architecture.Room` Wrapper
    Inherits from :any:`Element`

    >>> from rpw import db
    >>> room = db.Room(SomeRoom)
    <rpw:Room % DB.Architecture.Room | name:Office number:122>
    >>> room.name
    'Office'
    >>> room.number
    '122'
    >>> room.is_placed
    True
    >>> room.is_bounded
    True

    Attribute:
        _revit_object (DB.Architecture.Room): Wrapped ``DB.Architecture.Room``
    """

    _revit_object_class = DB.Architecture.Room
    _revit_object_category = DB.BuiltInCategory.OST_Rooms
    _collector_params = {'of_category': _revit_object_category,
                         'is_not_type': True}

    @property
    def name(self):
        """ Room Name as parameter Value: ``ROOM_NAME`` built-in parameter"""
        # Note: For an unknown reason, roominstance.Name does not work on IPY
        return self.parameters.builtins['ROOM_NAME'].value

    @name.setter
    def name(self, value):
        self.parameters.builtins['ROOM_NAME'].value = value

    @property
    def number(self):
        """ Room Number as parameter Value: ``ROOM_NUMBER`` built-in parameter"""
```

(continues on next page)

(continued from previous page)

```
        return self.parameters.builtins['ROOM_NUMBER'].value

    @number.setter
    def number(self, value):
        self.parameters.builtins['ROOM_NUMBER'].value = value

    # @property
    # def from_room(self, value):
        # TODO: from_room

    @property
    def is_placed(self):
        """ ``bool`` for whether Room is Placed.
        Uses result of ``Room.Location`` attribute to define if room is Placed.
        """
        return bool(self._revit_object.Location)

    @property
    def is_bounded(self):
        """ ``bool`` for whether Room is Bounded.
        Uses result of ``Room.Area`` attribute to define if room is Bounded.
        """
        return self._revit_object.Area > 0

    def __repr__(self):
        return super(Room, self).__repr__(data={'name': self.name,
                                               'number': self.number})


class Area(Room):
    """
    `DB.Area` Wrapper
    Inherits from :any:`Room`

    >>> from rpw import db
    >>> area = db.Area(SomeArea)
    <rpw:Area % DB.Area | name:USF area: 100.0>
    >>> area.name
    'Rentable'
    >>> area.is_placed
    True
    >>> area.is_bounded
    True

    Attribute:
        _revit_object (DB.Area): Wrapped ``DB.Area``
    """

    _revit_object_class = DB.Area
    _revit_object_category = DB.BuiltInCategory.OST_Areas
    _collector_params = {'of_category': _revit_object_category,
                         'is_not_type': True}

    @property
    def name(self):
        """ Area Scheme Name: Area attribute parameter"""
        return self.scheme.name
```

(continues on next page)

(continued from previous page)

```
    @property
    def scheme(self):
        """ Area Scheme: Wrapped Area Scheme"""
        return AreaScheme(self._revit_object.AreaScheme)

    @property
    def area(self):
        """ Area: .Area attribute"""
        return self._revit_object.Area

    def __repr__(self):
        return super(Element, self).__repr__(data={'name': self.name,
                                                   'area': self.area})


class AreaScheme(Element):
    """
    `DB.AreaScheme` Wrapper
    Inherits from :any:`Element`

    >>> scheme = wrapped_area.scheme
    <rwp:AreaScheme % DB.AreaScheme | name:USF>
    >>> scheme.areas
    [ < Autodesk.Revit.DB.Area>, ...]
    >>> scheme.name
    'USF'

    Attribute:
        _revit_object (DB.AreaScheme): Wrapped ``DB.AreaScheme``
    """

    _revit_object_class = DB.AreaScheme
    _collector_params = {'of_class': _revit_object_class}

    @property
    def name(self):
        """ Area Scheme Name: Area attribute parameter"""
        return self._revit_object.Name

    @property
    def areas(self):
        """ Returns all Area Instances of this Area Scheme """
        bip = BipEnum.get_id('AREA_SCHEME_ID')
        param_filter = rpw.db.Collector.ParameterFilter(bip, equals=self._revit_
↪object.Id)
        collector = rpw.db.Collector(parameter_filter=param_filter,
                                     **Area._collector_params)
        return collector.wrapped_elements

    def __repr__(self):
        return super(AreaScheme, self).__repr__(data={'name': self.name})
```

### View

View Wrappers

## View Classes

### View

**class** rpw.db.**View**(*element*, *doc=None*)

Bases: `rpw.db.element.Element`

This is the main View View Wrapper - wraps `DB.View`. All other View classes inherit from this class in the same way All API View classes inheir from `DB.View`.

This class is also used for view types that do not have more specific class, such as `DB.Legend`, `DB.ProjectBrowser`, `DB.SystemBrowser`.

As with other wrappers, you can use the Element() factory class to use the best wrapper available:

```
>>> from rpw import db
>>> wrapped_view = db.Element(some_view_plan)
<rpw:ViewPlan>
>>> wrapped_view = db.Element(some_legend)
<rpw:View>
>>> wrapped_view = db.Element(some_schedule)
<rpw:ViewSchedule>
```

```
>>> wrapped_view = db.Element(some_view_plan)
>>> wrapped_view.view_type
<rpw:ViewType | view_type: FloorPlan>
>>> wrapped_view.view_family_type
<rpw:ViewFamilyType % ..DB.ViewFamilyType | view_family:FloorPlan name:Floor Plan␣
↪id:1312>
>>> wrapped_view.view_family
<rpw:ViewFamily | family: FloorPlan>
>>> wrapped_view.siblings
[<rpw:ViewFamilyType % ..DB.ViewFamilyType> ... ]
```

View wrappers classes are collectible:

```
>>> rpw.db.ViewPlan.collect()
<rpw:Collector % ..DB.FilteredElementCollector | count:5>
>>> rpw.db.View3D.collect(where=lambda x: x.Name='MyView')
<rpw:Collector % ..DB.FilteredElementCollector | count:1>
```

**__init__**(*element*, *doc=None*)

Main Element Instantiation

```
>>> from rpw import db
>>> wall = db.Element(SomeElementId)
<rpw: WallInstance % DB.Wall >
>>> wall.parameters['Height']
10.0
>>> wall.parameters.builtins['WALL_LOCATION_LINE']
1
```

**Parameters** **element** (*Element Reference*) – Can be `DB.Element`, `DB.ElementId`, or `int`.

**Returns** Instance of Wrapped Element.

**Return type** *Element*

**category**
Wrapped `DB.Category`

**classmethod collect** (***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *WallType* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete** ()
Deletes Element from Model

**get_category** (*wrapped=True*)
Wrapped `DB.Category`

**name**
Name Property

**override**
Access to overrides.

For more information see *OverrideGraphicSettings*

```
>>> from rpw import db
>>> wrapped_view = db.Element(some_view)
>>> wrapped_view.override.projection_line(element, color=[0,255,0])
>>> wrapped_view.override.cut_line(category, weight=5)
```

**siblings**
Collect all views of the same `ViewType`

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap** ()
Returns the Original Wrapped Element

**view_family**
ViewFamily attribute

**view_family_type**
ViewFamilyType attribute

**view_type**
ViewType attribute

## ViewPlan

**class** rpw.db.**ViewPlan**(*element*, *doc=None*)
Bases: `rpw.db.view.View`

ViewPlan Wrapper.

`ViewType` is ViewType.FloorPlan or ViewType.CeilingPlan

**category**
Wrapped `DB.Category`

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *`WallType`* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**name**
Name Property

**override**
Access to overrides.

For more information see *`OverrideGraphicSettings`*

```
>>> from rpw import db
>>> wrapped_view = db.Element(some_view)
>>> wrapped_view.override.projection_line(element, color=[0,255,0])
>>> wrapped_view.override.cut_line(category, weight=5)
```

**siblings**
Collect all views of the same `ViewType`

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

**view_family**
ViewFamily attribute

**view_family_type**
ViewFamilyType attribute

**view_type**
ViewType attribute

## ViewSheet

**class** rpw.db.**ViewSheet**(*element*, *doc=None*)
Bases: `rpw.db.view.View`

ViewSheet Wrapper. `ViewType` is ViewType.DrawingSheet

**category**
Wrapped `DB.Category`

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, `WallType` uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**name**
Name Property

**override**
Access to overrides.

For more information see `OverrideGraphicSettings`

```
>>> from rpw import db
>>> wrapped_view = db.Element(some_view)
>>> wrapped_view.override.projection_line(element, color=[0,255,0])
>>> wrapped_view.override.cut_line(category, weight=5)
```

**siblings**
Collect all views of the same `ViewType`

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

**view_family**
ViewFamily attribute

**view_family_type**
ViewFamilyType attribute

**view_type**
ViewType attribute

## ViewSchedule

**class** rpw.db.**ViewSchedule**(*element*, *doc=None*)
Bases: `rpw.db.view.View`

ViewSchedule Wrapper. `ViewType` is ViewType.Schedule

**category**
Wrapped `DB.Category`

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *`WallType`* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**name**
Name Property

**override**
Access to overrides.

For more information see *OverrideGraphicSettings*

```
>>> from rpw import db
>>> wrapped_view = db.Element(some_view)
>>> wrapped_view.override.projection_line(element, color=[0,255,0])
>>> wrapped_view.override.cut_line(category, weight=5)
```

**siblings**
Collect all views of the same `ViewType`

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

**view_family**
ViewFamily attribute

**view_family_type**
ViewFamilyType attribute

**view_type**
ViewType attribute

## ViewSection

**class** rpw.db.**ViewSection**(*element*, *doc=None*)
Bases: `rpw.db.view.View`

DB.ViewSection Wrapper. `ViewType` is ViewType.DrawingSheet

**category**
Wrapped `DB.Category`

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *WallType* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**name**
Name Property

**override**
Access to overrides.

For more information see *OverrideGraphicSettings*

```
>>> from rpw import db
>>> wrapped_view = db.Element(some_view)
>>> wrapped_view.override.projection_line(element, color=[0,255,0])
>>> wrapped_view.override.cut_line(category, weight=5)
```

**siblings**
Collect all views of the same `ViewType`

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

**view_family**
ViewFamily attribute

**view_family_type**
ViewFamilyType attribute

**view_type**
ViewType attribute

## View3D

**class** rpw.db.**View3D**(*element*, *doc=None*)
Bases: `rpw.db.view.View`

DB.View3D Wrapper. `ViewType` is ViewType.ThreeD

**category**
Wrapped `DB.Category`

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *WallType* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**name**
Name Property

**override**
Access to overrides.

For more information see *OverrideGraphicSettings*

```
>>> from rpw import db
>>> wrapped_view = db.Element(some_view)
>>> wrapped_view.override.projection_line(element, color=[0,255,0])
>>> wrapped_view.override.cut_line(category, weight=5)
```

**siblings**
Collect all views of the same `ViewType`

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

**view_family**
ViewFamily attribute

**view_family_type**
ViewFamilyType attribute

**view_type**
ViewType attribute

### ViewFamilyType

**class** rpw.db.**ViewFamilyType**(*element*, *doc=None*)
Bases: `rpw.db.element.Element`

View Family Type Wrapper

**category**
Wrapped `DB.Category`

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *`WallType`* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Wrapped `DB.Category`

**name**
Name Property

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

**view_family**
Returns ViewFamily Enumerator

**views**
Collect All Views of the same ViewFamilyType

## View Enumeration

### ViewFamily

**class** rpw.db.**ViewFamily**(*revit_object*, *enforce_type=True*)
Bases: *`rpw.base.BaseObjectWrapper`*

ViewFamily Enumerator Wrapper. An enumerated type that corresponds to the type of a Revit view. http://www.revitapidocs.com/2015/916ed7b6-0a2e-c607-5d35-9ff9303b1f46.htm

This is returned on view.ViewFamily AreaPlan, CeilingPlan, CostReport Detail, Drafting, Elevation FloorPlan, GraphicalColumnSchedule, ImageView, Legend LoadsReport, PanelSchedule, PressureLossReport Schedule, Section, Sheet, StructuralPlan ThreeDimensional, Walkthrough

**name**
ToString() of View Family Enumerator

**unwrap**()
Returns the Original Wrapped Element

**views**
Collect All Views of the same ViewFamily

### ViewType

**class** rpw.db.**ViewType**(*revit_object*, *enforce_type=True*)
Bases: *rpw.base.BaseObjectWrapper*

ViewType Wrapper. An enumerated type listing available view types. http://www.revitapidocs.com/2015/bf04dabc-05a3-baf0-3564-f96c0bde3400.htm

**Can be on of the following types:** AreaPlan ,CeilingPlan, ColumnSchedule, CostReport, Detail, DraftingView, DrawingSheet, Elevation, EngineeringPlan, FloorPlan, Internal, Legend, LoadsReport, PanelSchedule, PresureLossReport, ProjectBrowser, Rendering, Report, Schedule, Section, SystemBrowser, ThreeD, Undefined, Walkthrough

**name**
ToString() of View Family Enumerator

**unwrap**()
Returns the Original Wrapped Element

**views**
Collect All Views of the same ViewType

### Override Wrappers

### OverrideGraphicSettings

**class** rpw.db.view.**OverrideGraphicSettings**(*wrapped_view*)
Bases: *rpw.base.BaseObjectWrapper*

Internal Wrapper for OverrideGraphicSettings - view.override

```
>>> from rpw import db
>>> wrapped_view = db.Element(some_view)
>>> wrapped_view.override.projection_line(target, color=(255,0,0))
>>> wrapped_view.override.projection_fill(target, color=(0,0,255),␣
↪pattern=pattern_id)
>>> wrapped_view.override.cut_line(target, color=(0,0,255), weight=2)
>>> wrapped_view.override.cut_fill(target, visible=False)
>>> wrapped_view.override.transparency(target, 50)
>>> wrapped_view.override.halftone(target, True)
>>> wrapped_view.override.detail_level(target, 'Coarse')
```

**Note:** Target can be any of the following:

- Element
- ElementId
- BuiltInCategory Enum
- BuiltInCategory Fuzzy Name (See `fuzzy_get()`)
- Category_id
- An iterable containing any of the above types

---

**`cut_fill`**(*target*, *color=None*, *pattern=None*, *visible=None*)
Sets CutFill overrides

**Parameters**

- **`target`** (`Element`, `ElementId`, `Category`) – Target Element(s) or Category(ies) to apply override. Can be list.
- **`color`** (`tuple`, `list`) – RGB Colors [ex. (255, 255, 0)]
- **`pattern`** (`DB.ElementId`) – ElementId of Pattern
- **`visible`** (`bool`) – Cut Fill Visibility

**`cut_line`**(*target*, *color=None*, *pattern=None*, *weight=None*)
Sets CutLine Overrides

**Parameters**

- **`target`** (`Element`, `ElementId`, `Category`) – Target Element(s) or Category(ies) to apply override. Can be list.
- **`color`** (`tuple`, `list`) – RGB Colors [ex. (255, 255, 0)]
- **`pattern`** (`DB.ElementId`) – ElementId of Pattern
- **`weight`** (`int`,``None``) – Line weight must be a positive integer less than 17 or None(sets invalidPenNumber)

**`detail_level`**(*target*, *detail_level*)
Sets DetailLevel Override. DetailLevel can be Enumeration memeber of DB.ViewDetailLevel or its name as a string. The Options are:

- Coarse
- Medium
- Fine

**Parameters**

- **`target`** (`Element`, `ElementId`, `Category`) – Target Element(s) or Category(ies) to apply override. Can be list.
- **`detail_level`** (`DB.ViewDetailLevel`, `str`) – Detail Level Enumerator or name

**`halftone`**(*target*, *halftone*)
Sets Halftone Override

**Parameters**

- **`target`** (`Element`, `ElementId`, `Category`) – Target Element(s) or Category(ies) to apply override. Can be list.

- **halftone** (bool) – Halftone

**match_element** (*target*, *element_to_match*)
Matches the settings of another element

**Parameters**

- **target** (Element, ElementId, Category) – Target Element(s) or Category(ies) to apply override. Can be list.
- **element_to_match** (Element, ElementId) – Element to match

**projection_fill** (*target*, *color=None*, *pattern=None*, *visible=None*)
Sets ProjectionFill overrides

**Parameters**

- **target** (Element, ElementId, Category) – Target Element(s) or Category(ies) to apply override. Can be list.
- **color** (tuple, list) – RGB Colors [ex. (255, 255, 0)]
- **pattern** (DB.ElementId) – ElementId of Pattern
- **visible** (bool) – Cut Fill Visibility

**projection_line** (*target*, *color=None*, *pattern=None*, *weight=None*)
Sets ProjectionLine overrides

**Parameters**

- **target** (Element, ElementId, Category) – Target Element(s) or Category(ies) to apply override. Can be list.
- **color** (tuple, list) – RGB Colors [ex. (255, 255, 0)]
- **pattern** (DB.ElementId) – ElementId of Pattern
- **weight** (int,‘‘None‘‘) – Line weight must be a positive integer less than 17 or None(sets invalidPenNumber)

**transparency** (*target*, *transparency*)
Sets SurfaceTransparency override

**Parameters**

- **target** (Element, ElementId, Category) – Target Element(s) or Category(ies) to apply override. Can be list.
- **transparency** (int) – Value of the transparency of the projection surface (0 = opaque, 100 = fully transparent)

**unwrap** ()
Returns the Original Wrapped Element

## Implementation

```
"""
View Wrappers

"""  #
```

(continues on next page)

(continued from previous page)

```
import rpw
from rpw import revit, DB
from rpw.db.element import Element
from rpw.db.pattern import LinePatternElement, FillPatternElement
from rpw.db.collector import Collector
from rpw.base import BaseObjectWrapper
from rpw.utils.coerce import to_element_ids, to_element_id, to_element
from rpw.utils.coerce import to_category_id, to_iterable
from rpw.exceptions import RpwTypeError, RpwCoerceError
from rpw.utils.logger import logger


class View(Element):
    """
    This is the main View View Wrapper - wraps ``DB.View``.
    All other View classes inherit from this class in the same way All
    API View classes inheir from ``DB.View``.

    This class is also used for view types that do not have more specific
    class, such as ``DB.Legend``, ``DB.ProjectBrowser``, ``DB.SystemBrowser``.

    As with other wrappers, you can use the Element() factory class to
    use the best wrapper available:

    >>> from rpw import db
    >>> wrapped_view = db.Element(some_view_plan)
    <rpw:ViewPlan>
    >>> wrapped_view = db.Element(some_legend)
    <rpw:View>
    >>> wrapped_view = db.Element(some_schedule)
    <rpw:ViewSchedule>


    >>> wrapped_view = db.Element(some_view_plan)
    >>> wrapped_view.view_type
    <rpw:ViewType | view_type: FloorPlan>
    >>> wrapped_view.view_family_type
    <rpw:ViewFamilyType % ..DB.ViewFamilyType | view_family:FloorPlan name:Floor Plan␣
↪id:1312>
    >>> wrapped_view.view_family
    <rpw:ViewFamily | family: FloorPlan>
    >>> wrapped_view.siblings
    [<rpw:ViewFamilyType % ..DB.ViewFamilyType> ... ]

    View wrappers classes are collectible:

    >>> rpw.db.ViewPlan.collect()
    <rpw:Collector % ..DB.FilteredElementCollector | count:5>
    >>> rpw.db.View3D.collect(where=lambda x: x.Name='MyView')
    <rpw:Collector % ..DB.FilteredElementCollector | count:1>

    """

    _revit_object_category = DB.BuiltInCategory.OST_Views
    _revit_object_class = DB.View
    _collector_params = {'of_class': _revit_object_class, 'is_type': False}
```

(continues on next page)

(continued from previous page)

```
    @property
    def view_type(self):
        """ ViewType attribute """
        return ViewType(self._revit_object.ViewType)

    @property
    def view_family_type(self):
        """ ViewFamilyType attribute """
        # NOTE: This can return Empty, as Some Views like SystemBrowser have no Type
        view_type_id = self._revit_object.GetTypeId()
        view_type = self.doc.GetElement(view_type_id)
        if view_type:
            return ViewFamilyType(self.doc.GetElement(view_type_id))

    @property
    def view_family(self):
        """ ViewFamily attribute """
        # Some Views don't have a ViewFamilyType
        return getattr(self.view_family_type, 'view_family', None)

    @property
    def siblings(self):
        """ Collect all views of the same ``ViewType`` """
        return self.view_type.views

    @property
    def override(self):
        """ Access to overrides.

        For more information see :any:`OverrideGraphicSettings`

        >>> from rpw import db
        >>> wrapped_view = db.Element(some_view)
        >>> wrapped_view.override.projection_line(element, color=[0,255,0])
        >>> wrapped_view.override.cut_line(category, weight=5)

        """
        return OverrideGraphicSettings(self)

    def change_type(self, type_reference):
        raise NotImplemented
        # self._revit_object.ChangeTypeId(type_reference)

    def __repr__(self):
        return super(View, self).__repr__(data={
                                'view_name': self.name,
                                'view_family_type': getattr(self.view_family_type,
↪'name', None),
                                'view_type': self.view_type.name,
                                'view_family': getattr(self.view_family, 'name', None)
                                                })


class ViewPlan(View):
    """
    ViewPlan Wrapper.
```

(continues on next page)

(continued from previous page)

```
    ``ViewType`` is ViewType.FloorPlan or  ViewType.CeilingPlan

    """
    _revit_object_class = DB.ViewPlan
    _collector_params = {'of_class': _revit_object_class, 'is_type': False}

    @property
    def level(self):
        return self._revit_object.GenLevel


class ViewSheet(View):
    """ ViewSheet Wrapper. ``ViewType`` is ViewType.DrawingSheet """
    _revit_object_class = DB.ViewSheet
    _collector_params = {'of_class': _revit_object_class, 'is_type': False}


class ViewSchedule(View):
    """ ViewSchedule Wrapper. ``ViewType`` is ViewType.Schedule """
    _revit_object_class = DB.ViewSchedule
    _collector_params = {'of_class': _revit_object_class, 'is_type': False}


class ViewSection(View):
    """ DB.ViewSection Wrapper. ``ViewType`` is ViewType.DrawingSheet """
    _revit_object_class = DB.ViewSection
    _collector_params = {'of_class': _revit_object_class, 'is_type': False}


class View3D(View):
    """ DB.View3D Wrapper. ``ViewType`` is ViewType.ThreeD """
    _revit_object_class = DB.View3D
    _collector_params = {'of_class': _revit_object_class, 'is_type': False}


class ViewFamilyType(Element):
    """ View Family Type Wrapper """
    _revit_object_class = DB.ViewFamilyType
    _collector_params = {'of_class': _revit_object_class, 'is_type': True}

    @property
    def view_family(self):
        """ Returns ViewFamily Enumerator """
        # Autodesk.Revit.DB.ViewFamily.FloorPlan
        return ViewFamily(self._revit_object.ViewFamily)

    @property
    def views(self):
        """ Collect All Views of the same ViewFamilyType """
        views = Collector(of_class='View').get_elements(wrapped=True)
        return [view for view in views if
                getattr(view.view_family_type, '_revit_object', None) == self.
↪unwrap()]

    def __repr__(self):
        return super(ViewFamilyType, self).__repr__(data={'name': self.name,
```

(continues on next page)

(continued from previous page)

```
                                                                'view_family': self.view_
↪family.name,
                                                                })


class ViewFamily(BaseObjectWrapper):
    """ ViewFamily Enumerator Wrapper.
    An enumerated type that corresponds to the type of a Revit view.
    http://www.revitapidocs.com/2015/916ed7b6-0a2e-c607-5d35-9ff9303b1f46.htm

    This is returned on view.ViewFamily
    AreaPlan, CeilingPlan, CostReport
    Detail, Drafting, Elevation
    FloorPlan, GraphicalColumnSchedule, ImageView, Legend
    LoadsReport, PanelSchedule, PressureLossReport
    Schedule, Section, Sheet, StructuralPlan
    ThreeDimensional, Walkthrough
    """
    _revit_object_class = DB.ViewFamily

    @property
    def name(self):
        """ ToString() of View Family Enumerator """
        return self._revit_object.ToString()

    @property
    def views(self):
        """ Collect All Views of the same ViewFamily """
        views = Collector(of_class='View').get_elements(wrapped=True)
        return [view for view in views if
                getattr(view.view_family, '_revit_object', None) == self.unwrap()]

    def __repr__(self):
        return super(ViewFamily, self).__repr__(data={'family': self.name})


class ViewType(BaseObjectWrapper):
    """ ViewType Wrapper.
    An enumerated type listing available view types.
    http://www.revitapidocs.com/2015/bf04dabc-05a3-baf0-3564-f96c0bde3400.htm

    Can be on of the following types:
        AreaPlan ,CeilingPlan, ColumnSchedule, CostReport,
        Detail, DraftingView, DrawingSheet, Elevation, EngineeringPlan,
        FloorPlan, Internal, Legend,
        LoadsReport, PanelSchedule, PresureLossReport,
        ProjectBrowser, Rendering, Report,
        Schedule, Section, SystemBrowser,
        ThreeD, Undefined, Walkthrough
    """
    _revit_object_class = DB.ViewType

    @property
    def name(self):
        """ ToString() of View Family Enumerator """
        return self._revit_object.ToString()
```

(continues on next page)

(continued from previous page)

```
    @property
    def views(self):
        """ Collect All Views of the same ViewType """
        views = Collector(of_class='View').get_elements(wrapped=True)
        return [view for view in views if view.view_type.unwrap() == self.unwrap()]

    def __repr__(self):
        return super(ViewType, self).__repr__(data={'view_type': self.name})


class ViewPlanType(BaseObjectWrapper):
    """
    Enumerator
        ViewPlanType.FloorPlan, ViewPlanType.CeilingPlan
    No Wrapper Need. Enum is only used as arg for when creating ViewPlan
    """


class OverrideGraphicSettings(BaseObjectWrapper):

    """ Internal Wrapper for OverrideGraphicSettings - view.override



    >>> from rpw import db
    >>> wrapped_view = db.Element(some_view)
    >>> wrapped_view.override.projection_line(target, color=(255,0,0))
    >>> wrapped_view.override.projection_fill(target, color=(0,0,255),␣
↪pattern=pattern_id)
    >>> wrapped_view.override.cut_line(target, color=(0,0,255), weight=2)
    >>> wrapped_view.override.cut_fill(target, visible=False)
    >>> wrapped_view.override.transparency(target, 50)
    >>> wrapped_view.override.halftone(target, True)
    >>> wrapped_view.override.detail_level(target, 'Coarse')

    Note:
        Target can be any of the following:

        * Element
        * ElementId
        * BuiltInCategory Enum
        * BuiltInCategory Fuzzy Name (See :func:`fuzzy_get`)
        * Category_id
        * An iterable containing any of the above types

    """

    # TODO: Pattern: Add pattern_id from name. None sets InvalidElementId
    # TODO: Weight: None to set InvalidPenNumber
    # TODO: Color: Add color from name util
    # ISSUE: Cannot set LinePatterns to Solid because it's not collectible:
    # https://forums.autodesk.com/t5/revit-api-forum/solid-linepattern/td-p/4651067

    _revit_object_class = DB.OverrideGraphicSettings

    def __init__(self, wrapped_view):
        super(OverrideGraphicSettings, self).__init__(DB.OverrideGraphicSettings())
```

(continues on next page)

(continued from previous page)

```
        self.view = wrapped_view.unwrap()

    def _set_overrides(self, target):
        targets = to_iterable(target)
        for target in targets:
            try:
                category_id = to_category_id(target)
                self._set_category_overrides(category_id)
            except (RpwTypeError, RpwCoerceError) as errmsg:
                logger.debug('Not Category, Trying Element Override')
                element_id = to_element_id(target)
                self._set_element_overrides(element_id)

    def _set_element_overrides(self, element_id):
        self.view.SetElementOverrides(element_id, self._revit_object)

    def _set_category_overrides(self, category_id):
        self.view.SetCategoryOverrides(category_id, self._revit_object)

    def match_element(self, target, element_to_match):
        """
        Matches the settings of another element

        Args:
            target (``Element``, ``ElementId``, ``Category``): Target
                Element(s) or Category(ies) to apply override. Can be list.
            element_to_match (``Element``, ``ElementId``): Element to match

        """
        element_to_match = to_element_id(element_to_match)

        self._revit_object = self.view.GetElementOverrides(element_to_match)
        self._set_overrides(target)

    def projection_line(self, target, color=None, pattern=None, weight=None):
        """
        Sets ProjectionLine overrides

        Args:
            target (``Element``, ``ElementId``, ``Category``): Target
                Element(s) or Category(ies) to apply override. Can be list.
            color (``tuple``, ``list``): RGB Colors [ex. (255, 255, 0)]
            pattern (``DB.ElementId``): ElementId of Pattern
            weight (``int``,``None``): Line weight must be a positive integer
                less than 17 or None(sets invalidPenNumber)

        """
        if color:
            Color = DB.Color(*color)
            self._revit_object.SetProjectionLineColor(Color)
        if pattern:
            line_pattern = LinePatternElement.by_name_or_element_ref(pattern)
            self._revit_object.SetProjectionLinePatternId(line_pattern.Id)
        if weight:
            self._revit_object.SetProjectionLineWeight(weight)

        self._set_overrides(target)
```

(continues on next page)

(continued from previous page)

```
    def cut_line(self, target, color=None, pattern=None, weight=None):
        """
        Sets CutLine Overrides

        Args:
            target (``Element``, ``ElementId``, ``Category``): Target
                Element(s) or Category(ies) to apply override. Can be list.
            color (``tuple``, ``list``): RGB Colors [ex. (255, 255, 0)]
            pattern (``DB.ElementId``): ElementId of Pattern
            weight (``int``,``None``): Line weight must be a positive integer
                less than 17 or None(sets invalidPenNumber)
        """
        if color:
            Color = DB.Color(*color)
            self._revit_object.SetCutLineColor(Color)
        if pattern:
            line_pattern = LinePatternElement.by_name_or_element_ref(pattern)
            self._revit_object.SetCutLinePatternId(line_pattern.Id)
        if weight:
            self._revit_object.SetCutLineWeight(weight)

        self._set_overrides(target)

    def projection_fill(self, target, color=None, pattern=None, visible=None):
        """
        Sets ProjectionFill overrides

        Args:
            target (``Element``, ``ElementId``, ``Category``): Target
                Element(s) or Category(ies) to apply override. Can be list.
            color (``tuple``, ``list``): RGB Colors [ex. (255, 255, 0)]
            pattern (``DB.ElementId``): ElementId of Pattern
            visible (``bool``): Cut Fill Visibility
        """
        if color:
            Color = DB.Color(*color)
            self._revit_object.SetProjectionFillColor(Color)
        if pattern:
            fill_pattern = FillPatternElement.by_name_or_element_ref(pattern)
            self._revit_object.SetProjectionFillPatternId(fill_pattern.Id)
        if visible is not None:
            self._revit_object.SetProjectionFillPatternVisible(visible)

        self._set_overrides(target)

    def cut_fill(self, target, color=None, pattern=None, visible=None):
        """
        Sets CutFill overrides

        Args:
            target (``Element``, ``ElementId``, ``Category``): Target
                Element(s) or Category(ies) to apply override. Can be list.
            color (``tuple``, ``list``): RGB Colors [ex. (255, 255, 0)]
            pattern (``DB.ElementId``): ElementId of Pattern
            visible (``bool``): Cut Fill Visibility
        """
```

(continues on next page)

(continued from previous page)

```
        if color:
            Color = DB.Color(*color)
            self._revit_object.SetCutFillColor(Color)
        if pattern:
            fill_pattern = FillPatternElement.by_name_or_element_ref(pattern)
            self._revit_object.SetCutFillPatternId(fill_pattern.Id)
        if visible is not None:
            self._revit_object.SetCutFillPatternVisible(visible)

        self._set_overrides(target)

    def transparency(self, target, transparency):
        """
        Sets SurfaceTransparency override

        Args:
            target (``Element``, ``ElementId``, ``Category``): Target
                Element(s) or Category(ies) to apply override. Can be list.
            transparency (``int``): Value of the transparency of the projection␣
↪surface
                                        (0 = opaque, 100 = fully transparent)
        """
        self._revit_object.SetSurfaceTransparency(transparency)
        self._set_overrides(target)

    def halftone(self, target, halftone):
        """
        Sets Halftone Override

        Args:
            target (``Element``, ``ElementId``, ``Category``): Target
                Element(s) or Category(ies) to apply override. Can be list.
            halftone (``bool``): Halftone
        """
        self._revit_object.SetHalftone(halftone)
        self._set_overrides(target)

    def detail_level(self, target, detail_level):
        """
        Sets DetailLevel Override. DetailLevel can be Enumeration memeber of
        DB.ViewDetailLevel or its name as a string. The Options are:

            * Coarse
            * Medium
            * Fine

        Args:
            target (``Element``, ``ElementId``, ``Category``): Target
                Element(s) or Category(ies) to apply override. Can be list.
            detail_level (``DB.ViewDetailLevel``, ``str``): Detail Level
                Enumerator or name
        """

        if isinstance(detail_level, str):
            detail_level = getattr(DB.ViewDetailLevel, detail_level)
```

(continues on next page)

(continued from previous page)

```
        self._revit_object.SetDetailLevel(detail_level)
        self._set_overrides(target)
```

### Walls

#### Wall Wrappers

**Note:** These classes inherit from the classes listed above, but make some adjustments to compensate for dissimilarities in in Wall Families.

When retrieving the FamilySymbol from an instance, and the Family from a Symbol, one might uses `instance.Symbol` and `symbol.Family`.

Unfortunately, this would not be the case with Wall Elements. A Wall Instance is actually a `DB.Wall`; the *Family Type* of a wall is not a `DB.FamilySymbol` type, but a `DB.WallType`; and instead of `.Family`, walls use `.Kind`.

These wrappers create a more consistent navigation by allowing to retrieve the "symbol" and "family" of a wall using: *wall.symbol*, and *wall.family*

```
>>> wall = rpw.db.Wall(SomeWallInstance)
>>> wall.symbol
<rpw: WallType % DB.WallType | type:Wall 1>
>>> wall.family
<rpw: WallKind % DB.WallKind | type:Basic 1>
```

**class** `rpw.db.`**`Wall`**(*element*, *doc=None*)

Bases: `rpw.db.family.FamilyInstance`

Inherits base `FamilyInstance` and overrides symbol attribute to get *Symbol* equivalent of Wall - WallType *(GetTypeId)*

**`__init__`**(*element*, *doc=None*)

Main Element Instantiation

```
>>> from rpw import db
>>> wall = db.Element(SomeElementId)
<rpw: WallInstance % DB.Wall >
>>> wall.parameters['Height']
10.0
>>> wall.parameters.builtins['WALL_LOCATION_LINE']
1
```

**Parameters** **`element`** (*Element Reference*) – Can be `DB.Element`, `DB.ElementId`, or `int`.

**Returns** Instance of Wrapped Element.

**Return type** *`Element`*

**`category`**

Wrapped `DB.Category` of the `DB.Wall`

**`change_type`**(*wall_type_reference*)

Change Wall Type

**Parameters** **wall_type_reference** (`ElementId`, `WallType`, `str`) – Wall Type Reference

**classmethod collect** (***kwargs*)

Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, `WallType` uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()

Deletes Element from Model

**get_assembly**

*Returns* –

**(bool, DB.Element) None if element not in Assembly, else** returns Element

**get_category**(*wrapped=True*)

Get Wall Category

**get_family**(*wrapped=True*)

Get WallKind Alias

**get_siblings**(*wrapped=True*)

Other `DB.FamilyInstance` of the same `DB.FamilySymbol`

**get_symbol**(*wrapped=True*)

Get Wall Type Alias

**get_wall_type**(*wrapped=True*)

Get Wall Type

**in_assembly**

**Returns* – (bool)* – True if element is inside an AssemblyInstance

**name**

Name Property

**siblings**

Other `DB.FamilyInstance` of the same `DB.FamilySymbol`

**type**

Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

**class** rpw.db.**WallType**(*element*, *doc=None*)
Bases: rpw.db.family.FamilySymbol, *rpw.utils.mixins.ByNameCollectMixin*

**Inherits from *FamilySymbol* and overrides:**

- *wall_kind()* to get the *Family* equivalent of Wall *(.Kind)*
- Uses a different method to get instances.

**__init__**(*element*, *doc=None*)
Main Element Instantiation

```
>>> from rpw import db
>>> wall = db.Element(SomeElementId)
<rpw: WallInstance % DB.Wall >
>>> wall.parameters['Height']
10.0
>>> wall.parameters.builtins['WALL_LOCATION_LINE']
1
```

**Parameters element** (*Element Reference*) – Can be DB.Element, DB.ElementId, or int.

**Returns** Instance of Wrapped Element.

**Return type** *Element*

**classmethod by_name**(*name*)
Mixin to provide instantiating by a name for classes that are collectible. This is a mixin so specifi usage will vary for each for. This method will call the *rpw.db.Element.collect* method of the class, and return the first element with a matching .name property.

```
>>> LinePatternElement.by_name('Dash')
<rpw:LinePatternElement name:Dash>
```

```
>>> FillPatternElement.by_name('Solid')
<rpw:FillPatternElement name:Solid>
```

**classmethod by_name_or_element_ref**(*reference*)
Mixin for collectible elements. This is to help cast elements from name, elemente, or element_id

**category**
Wrapped DB.Category of the DB.Wall

**classmethod collect**(***kwargs*)
Collect all elements of the wrapper using the default collector. This method is defined on the main Element wrapper, but the collector parameters are defined in each wrapper. For example, *WallType* uses the *_collector_params*: {'of_class': DB.WallType, 'is_type': True}

These default collector parameters can be overriden by passing keyword args to the collectors call.

```
>>> from rpw import db
>>> wall_types_collector = db.WallType.collect()
<rpw:Collector % FilteredElementCollector [count:4]>
>>> wall_types_collector.get_elements()  # All Wall Types
[<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
```

(continues on next page)

(continued from previous page)

```
>>> wall_types_collector.get_elements()
[<rpw:Area % DB.Area | Rentable:30.2>]
>>> rooms = db.WallInstance.collect(level="Level 1")
[<rpw:WallInstance % DB.Wall symbol:Basic Wall>]
```

**delete**()
Deletes Element from Model

**get_category**(*wrapped=True*)
Get Wall Category

**get_family**(*wrapped=True*)
Returns: *Family*: Wrapped `DB.Family` of the symbol

**get_instances**(*wrapped=True*)
Returns all Instances of this Wall Types

**get_siblings**(*wrapped=True*)
Returns: [`DB.FamilySymbol`]: List of symbol Types

of the same Family (unwrapped)

**get_wall_kind**(*wrapped=True*)
Returns `DB.Family` of the Symbol

**instances**
Returns all Instances of this Wall Types

**name**
Name Property

**type**
Get's Element Type using the default GetTypeId() Method. For some Elements, this is the same as `element.Symbol` or `wall.WallType`

**Parameters** **doc** (`DB.Document`, optional) – Document of Element [default: revit.doc]

**Returns** Wrapped `rpw.db.Element` element type

**Return type** (`Element`)

**unwrap**()
Returns the Original Wrapped Element

**wall_kind**
Returns `DB.Family` of the Symbol

**class** rpw.db.**WallKind**(*revit_object*, *enforce_type=True*)
Bases: *rpw.base.BaseObjectWrapper*

Equivalent of `Family` but is Enumerator for Walls.

Can be Basic, Stacked, Curtain, Unknown

**__init__**(*revit_object*, *enforce_type=True*)
Child classes can use self._revit_object to refer back to Revit Element

> **Warning:** Any Wrapper that inherits and overrides __init__ class MUST ensure `_revit_object` is created by calling super().__init__ before setting any self attributes. Not doing so will cause recursion errors and Revit will crash. BaseObjectWrapper should define a class variable _revit_object_class to define the object class being wrapped.

**`get_instances`**(*wrapped=True*)
Returns all Wall instances of this given Wall Kind

**`get_symbols`**(*wrapped=True*)
Get Wall Types Alias

**`get_wall_types`**(*wrapped=True*)
Get Wall Types Alias

**`instances`**
Returns all Wall instances of this given Wall Kind

**`name`**
*Retuns Pretty Name as shown on UI* – Basic > Basic Wall

**`unwrap`**()
Returns the Original Wrapped Element

**class** `rpw.db.`**`WallCategory`**(*revit_object*, *enforce_type=True*)
Bases: `rpw.db.category.Category`

`DB.Category` Wall Category Wrapper

**Attribute:** _revit_object (DB.Family): Wrapped `DB.Category`

**`__init__`**(*revit_object*, *enforce_type=True*)
Child classes can use self._revit_object to refer back to Revit Element

> **Warning:** Any Wrapper that inherits and overrides __init__ class MUST ensure `_revit_object` is created by calling super().__init__ before setting any self attributes. Not doing so will cause recursion errors and Revit will crash. BaseObjectWrapper should define a class variable _revit_object_class to define the object class being wrapped.

**`builtin`**
Returns BuiltInCategory of the Category

**`families`**
Returns `DB.WallKind` elements in the category

**`get_families`**(*wrapped=True*)
Returns `DB.WallKind` elements in the category

**`get_instances`**(*wrapped=True*)

**Returns** List of Symbol Instances in the Category.

**Return type** (`DB.FamilyInstance`)

**`get_symbols`**(*wrapped=True*)

**Returns** List of Symbol Types in the Category

**Return type** Symbols (`DB.FamilySymbol`)

**`name`**
Returns name of the Category

**`unwrap`**()
Returns the Original Wrapped Element

### Implementation

```
import rpw
from rpw import revit, DB
from rpw.db import Element
from rpw.db import FamilyInstance, FamilySymbol, Family, Category
from rpw.base import BaseObjectWrapper
from rpw.utils.logger import logger, deprecate_warning
from rpw.utils.coerce import to_element_id
from rpw.db.builtins import BipEnum
from rpw.exceptions import RpwTypeError, RpwCoerceError
from rpw.utils.mixins import ByNameCollectMixin


class Wall(FamilyInstance):
    """
    Inherits base ``FamilyInstance`` and overrides symbol attribute to
    get `Symbol` equivalent of Wall - WallType `(GetTypeId)`
    """

    _revit_object_category = DB.BuiltInCategory.OST_Walls
    _revit_object_class = DB.Wall
    _collector_params = {'of_class': _revit_object_class, 'is_type': False}

    def change_type(self, wall_type_reference):
        """
        Change Wall Type

        Args:
            wall_type_reference (``ElementId``, ``WallType``, ``str``): Wall Type␣
↪Reference
        """
        wall_type = WallType.by_name_or_element_ref(wall_type_reference)
        wall_type_id = to_element_id(wall_type)
        self._revit_object.ChangeTypeId(wall_type_id)

    def get_symbol(self, wrapped=True):
        """ Get Wall Type Alias """
        return self.get_wall_type(wrapped)

    @property
    def symbol(self):
        deprecate_warning('Wall.symbol', 'Wall.get_symbol()')
        return self.get_symbol(wrapped=True)

    def get_wall_type(self, wrapped=True):
        """ Get Wall Type """
        wall_type_id = self._revit_object.GetTypeId()
        wall_type = self.doc.GetElement(wall_type_id)
        return WallType(wall_type) if wrapped else wall_type

    @property
    def wall_type(self):
        deprecate_warning('Wall.wall_type', 'Wall.get_wall_type()')
        return self.get_wall_type(wrapped=True)
```

(continues on next page)

(continued from previous page)

```
    def get_wall_kind(self, wrapped=True):
        wall_type = self.get_wall_type(wrapped=True)
        return wall_type.get_wall_kind(wrapped=wrapped)

    @property
    def wall_kind(self):
        deprecate_warning('Wall.wall_kind', 'Wall.get_wall_kind()')
        return self.get_wall_kind(wrapped=True)

    def get_family(self, wrapped=True):
        """ Get WallKind Alias """
        return self.get_wall_kind(wrapped=wrapped)

    @property
    def family(self):
        deprecate_warning('Wall.family', 'Wall.get_family()')
        return self.get_family(wrapped=True)

    def get_category(self, wrapped=True):
        """ Get Wall Category """
        return WallCategory(self._revit_object.Category)

    @property
    def category(self):
        """ Wrapped ``DB.Category`` of the ``DB.Wall`` """
        deprecate_warning('Wall.category', 'Wall.get_category()')
        return self.get_category(wrapped=True)


class WallType(FamilySymbol, ByNameCollectMixin):
    """
    Inherits from :any:`FamilySymbol` and overrides:
        * :func:`wall_kind` to get the `Family` equivalent of Wall `(.Kind)`
        * Uses a different method to get instances.
    """

    _revit_object_class = DB.WallType
    _collector_params = {'of_class': _revit_object_class, 'is_type': True}

    def get_family(self, wrapped=True):
        return self.get_wall_kind(wrapped=wrapped)

    @property
    def family(self):
        deprecate_warning('WallType.family', 'WallType.get_family()')
        return self.get_wall_kind(wrapped=True)

    def get_wall_kind(self, wrapped=True):
        """ Returns ``DB.Family`` of the Symbol """
        kind = self._revit_object.Kind
        return WallKind(kind) if wrapped else kind

    @property
    def wall_kind(self):
        """ Returns ``DB.Family`` of the Symbol """
        deprecate_warning('WallType.wall_kind', 'WallType.get_wall_kind()')
        return self.get_wall_kind(wrapped=True)
```

(continues on next page)

(continued from previous page)

```
    def get_instances(self, wrapped=True):
        """ Returns all Instances of this Wall Types """
        bip = BipEnum.get_id('SYMBOL_NAME_PARAM')
        param_filter = rpw.db.ParameterFilter(bip, equals=self.name)
        return rpw.db.Collector(parameter_filter=param_filter,
                                **Wall._collector_params).wrapped_elements

    @property
    def instances(self):
        """ Returns all Instances of this Wall Types """
        deprecate_warning('WallType.instances', 'WallType.get_instances()')
        return self.get_instances(wrapped=True)

    def get_siblings(self, wrapped=True):
        wall_kind = self.get_wall_kind(wrapped=True)
        return wall_kind.get_wall_types(wrapped=wrapped)

    @property
    def siblings(self):
        deprecate_warning('WallType.siblings', 'WallType.get_siblings()')
        return self.get_siblings(wrapped=True)

    def get_category(self, wrapped=True):
        """ Get Wall Category """
        return WallCategory(self._revit_object.Category)

    @property
    def category(self):
        """ Wrapped ``DB.Category`` of the ``DB.Wall`` """
        deprecate_warning('Wall.category', 'Wall.get_category()')
        return self.get_category(wrapped=True)


# class WallKind(Family):
class WallKind(BaseObjectWrapper):
    """
    Equivalent of ``Family`` but is Enumerator for Walls.

    Can be Basic, Stacked, Curtain, Unknown
    """

    _revit_object_class = DB.WallKind

    @property
    def name(self):
        """ Retuns Pretty Name as shown on UI: Basic > Basic Wall """
        # Same method as Family Works, but requires Code duplication
        # Since this should not inherit from Family.
        # Solution copy code or Mixin. Or return Enum Name:  'Basic'
        # This works but requires Lookup.
        # wall_type = self.get_wall_types()[0]
        # return wall_type.parameters.builtins['SYMBOL_FAMILY_NAME_PARAM'].value
        # return '{} Wall'.format(self._revit_object.ToString())
        return self._revit_object.ToString()

    def get_symbols(self, wrapped=True):
```

(continues on next page)

(continued from previous page)

```
        """ Get Wall Types Alias """
        return self.get_wall_types(wrapped=wrapped)

    @property
    def symbols(self):
        deprecate_warning('WallKind.symbols', 'WallKind.get_symbols()')
        return self.get_symbols(wrapped=True)

    def get_wall_types(self, wrapped=True):
        """ Get Wall Types Alias """
        type_collector = rpw.db.WallType.collect()
        wall_types = type_collector.get_elements(wrapped=wrapped)
        return [wall_type for wall_type in wall_types
                if wall_type.Kind == self._revit_object]

    @property
    def wall_types(self):
        deprecate_warning('WallKind.wall_types', 'WallKind.get_wall_types()')
        return self.get_wall_types(wrapped=True)

    def get_instances(self, wrapped=True):
        """ Returns all Wall instances of this given Wall Kind"""
        instances = []
        for wall_type in self.get_wall_types(wrapped=True):
            instances.extend(wall_type.get_instances(wrapped=wrapped))
        return instances

    @property
    def instances(self):
        """ Returns all Wall instances of this given Wall Kind"""
        deprecate_warning('WallKind.instances', 'WallKind.get_instances()')
        return self.get_instances(wrapped=True)

    def get_category(self, wrapped=True):
        cat = DB.Category.GetCategory(revit.doc, DB.BuiltInCategory.OST_Walls)
        return WallCategory(cat) if wrapped else cat

    @property
    def category(self):
        deprecate_warning('WallKind.category', 'WallKind.get_category()')
        return self.get_category(wrapped=True)

    def __repr__(self):
        return super(WallKind, self).__repr__({'name': self.name})

class WallCategory(Category):
    """
    ``DB.Category`` Wall Category Wrapper

    Attribute:
        _revit_object (DB.Family): Wrapped ``DB.Category``
    """

    _revit_object_class = DB.Category

    def get_families(self, wrapped=True):
        """ Returns ``DB.WallKind`` elements in the category """
```

(continues on next page)

(continued from previous page)

```
        wall_kinds = []
        for member in dir(DB.WallKind):
            if type(getattr(DB.WallKind, member)) is DB.WallKind:
                wall_kind = getattr(DB.WallKind, member)
                wall_kind = WallKind(wall_kind) if wrapped else wall_kind
                wall_kinds.append(wall_kind)
        return wall_kinds

    @property
    def families(self):
        """ Returns ``DB.WallKind`` elements in the category """
        deprecate_warning('WallCategory.families',
                          'WallCategory.get_families()')
        return self.get_families(wrapped=True)
```

## Implementation

```
import rpw
from rpw import revit, DB
from rpw.db.parameter import Parameter, ParameterSet
from rpw.base import BaseObjectWrapper
from rpw.exceptions import RpwException, RpwWrongStorageType
from rpw.exceptions import RpwParameterNotFound, RpwTypeError
from rpw.utils.logger import logger, deprecate_warning
from rpw.utils.mixins import CategoryMixin
from rpw.db.builtins import BicEnum, BipEnum
from rpw.utils.coerce import to_element_ids


class Element(BaseObjectWrapper, CategoryMixin):
    """
    Inheriting from element extends wrapped elements with a new :class:`parameters`
    attribute, well as the :func:`unwrap` method inherited from the␣
↪:any:`BaseObjectWrapper` class.

    It can be created by instantiating ``rpw.db.Element`` , or one of the helper
    static methods shown below.

    Most importantly, all other `Element-related` classes inhert from this class
    so it can provide parameter access.

    >>> from rpw import db
    >>> element = db.Element(SomeElement)
    >>> element = db.Element.from_id(ElementId)
    >>> element = db.Element.from_int(Integer)

    >>> wall = db.Element(RevitWallElement)
    >>> wall.Id
    >>> wall.parameters['Height'].value
    10.0

    The ``Element`` Constructor can be used witout specifying the
    exact class. On instantiation, it will attempt to map the type provided,
    if a match is not found, an Element will be used.
```

(continues on next page)

(continued from previous page)

```
    If the element does not inherit from DB.Element, and exception is raised.

    >>> wall_instance = db.Element(SomeWallInstance)
    >>> type(wall_instance)
    'rpw.db.WallInstance'
    >>> wall_symbol = db.Element(SomeWallSymbol)
    >>> type(wall_symbol)
    'rpw.db.WallSymbol'

    Attributes:

        parameters (:any:`ParameterSet`): Access :any:`ParameterSet` class.
        parameters.builtins (:any:`ParameterSet`): BuitIn :any:`ParameterSet` object

    Methods:
        unwrap(): Wrapped Revit Reference

    """

    _revit_object_class = DB.Element

    def __new__(cls, element, **kwargs):
        """
        Factory Constructor will chose the best Class for the Element.
        This function iterates through all classes in the rpw.db module,
        and will find one that wraps the corresponding class. If and exact
        match is not found :any:`Element` is used
        """
        defined_wrapper_classes = rpw.db.__all__

        _revit_object_class = cls._revit_object_class

        if element is None:
            raise RpwTypeError('Element or Element Child', 'None')

        # TODO: Handle double wrapping
        if hasattr(element, 'unwrap'):
            raise RpwTypeError('revit element', 'wrapped element: {}'.format(element))

        # Ensure Wrapped Element is instance of Class Wrapper or decendent
        if not isinstance(element, _revit_object_class):
            raise RpwTypeError(_revit_object_class.__name__,
                               element.__class__.__name__)

        # Ensure Wrapped Element is instance of Class Wrapper or decendent
        if not isinstance(element, _revit_object_class):
            raise RpwTypeError(_revit_object_class, element.__class__)

        # If explicit constructor was called, use that and skip discovery
        if type(element) is _revit_object_class:
            return super(Element, cls).__new__(cls, element, **kwargs)

        for wrapper_class in defined_wrapper_classes:
            class_name = wrapper_class.__name__
            if type(element) is getattr(wrapper_class, '_revit_object_class', None):
                # Found Mathing Class, Use Wrapper
                # print('Found Mathing Class, Use Wrapper: {}'.format(class_name))
```

(continues on next page)

(continued from previous page)

```
                return super(Element, cls).__new__(wrapper_class, element, **kwargs)
        else:
            # Could Not find a Matching Class, Use Element if related
            return super(Element, cls).__new__(cls, element, **kwargs)

        # No early return. Should not reach this point
        element_class_name = element.__class__.__name__
        raise RpwException('Factory does not support type: {}'.format(element_class_
↪name))

    def __init__(self, element, doc=None):
        """
        Main Element Instantiation

        >>> from rpw import db
        >>> wall = db.Element(SomeElementId)
        <rpw: WallInstance % DB.Wall >
        >>> wall.parameters['Height']
        10.0
        >>> wall.parameters.builtins['WALL_LOCATION_LINE']
        1

        Args:
            element (`Element Reference`): Can be ``DB.Element``, ``DB.ElementId``,␣
↪or ``int``.

        Returns:
            :class:`Element`: Instance of Wrapped Element.

        """
        # rpw.ui.forms.Console(context=locals())
        super(Element, self).__init__(element)
        self.doc = element.Document if doc is None else revit.doc
        if isinstance(element, DB.Element):
            # WallKind Inherits from Family/Element, but is not Element,
            # so ParameterSet fails. Parameters are only added if Element
            # inherits from element
            # NOTE: This is no longer the case. Verify if it can be removed
            self.parameters = ParameterSet(element)

    @property
    def type(self):
        """
        Get's Element Type using the default GetTypeId() Method.
        For some Elements, this is the same as ``element.Symbol`` or ``wall.WallType``

        Args:
            doc (``DB.Document``, optional): Document of Element [default: revit.doc]

        Returns:
            (``Element``): Wrapped ``rpw.db.Element`` element type

        """
        element_type_id = self._revit_object.GetTypeId()
        element_type = self._revit_object.Document.GetElement(element_type_id)
        return Element(element_type)
```

(continues on next page)

(continued from previous page)

```
    @property
    def name(self):
        """ Name Property """
        return DB.Element.Name.__get__(self.unwrap())

    @name.setter
    def name(self, value):
        """ Name Property Setter """
        return DB.Element.Name.__set__(self.unwrap(), value)

    @classmethod
    def collect(cls, **kwargs):
        """
        Collect all elements of the wrapper using the default collector.
        This method is defined on the main Element wrapper, but the
        collector parameters are defined in each wrapper. For example,
        :any:`WallType` uses the `_collector_params`:
        {'of_class': DB.WallType, 'is_type': True}

        These default collector parameters can be overriden by passing keyword
        args to the collectors call.

        >>> from rpw import db
        >>> wall_types_collector = db.WallType.collect()
        <rpw:Collector % FilteredElementCollector [count:4]>
        >>> wall_types_collector.get_elements()  # All Wall Types
        [<rpw:WallType [name:Wall 1] [id:1557]>, ... ]
        >>> wall_types_collector.get_elements()
        [<rpw:Area % DB.Area | Rentable:30.2>]
        >>> rooms = db.WallInstance.collect(level="Level 1")
        [<rpw:WallInstance % DB.Wall symbol:Basic Wall>]

        """
        _collector_params = getattr(cls, '_collector_params', None)

        if _collector_params:
            kwargs.update(_collector_params)
            return rpw.db.Collector(**kwargs)
        else:
            raise RpwException('Wrapper cannot collect by class: {}'.format(cls.__
↪name__))

    @staticmethod
    def from_int(id_int, doc=None):
        """
        Instantiate Element from an Integer representing and Id

        Args:
            id (``int``): ElementId of Element to wrap
            doc (``DB.Document``, optional): Document of Element [default: revit.doc]

        Returns:
            (``Element``): Wrapped ``rpw.db.Element`` instance
        """
        doc = revit.doc if doc is None else doc
        element_id = DB.ElementId(id_int)
        return Element.from_id(element_id, doc=doc)
```

(continues on next page)

(continued from previous page)

```
    @staticmethod
    def from_id(element_id, doc=None):
        """
        Instantiate Element from an ElementId

        Args:
            id (``ElementId``): ElementId of Element to wrap
            doc (``DB.Document``, optional): Document of Element [default: revit.doc]

        Returns:
            (``Element``): Wrapped ``rpw.db.Element`` instance

        """
        doc = doc or revit.doc
        element = doc.GetElement(element_id)
        return Element(element)

    @staticmethod
    def from_list(element_references, doc=None):
        """
        Instantiate Elements from a list of DB.Element instances

        Args:
            elements (``[DB.Element, DB.ElementId]``): List of element references

        Returns:
            (``list``): List of ``rpw.db.Element`` instances

        """
        doc = doc or revit.doc
        try:
            return [Element(e) for e in element_references]
        except RpwTypeError:
            pass
        try:
            element_ids = to_element_ids(element_references)
            return [Element.from_id(id_, doc=doc) for id_ in element_ids]
        except RpwTypeError:
            raise


    @staticmethod
    def Factory(element):
        deprecate_warning('Element.Factory()', replaced_by='Element()')
        return Element(element)

    def delete(self):
        """ Deletes Element from Model """
        self.doc.Delete(self._revit_object.Id)

    def __repr__(self, data=None):
        if data is None:
            data = {}
        element_id = getattr(self._revit_object, 'Id', None)
        if element_id:
            data.update({'id': element_id})
```

(continues on next page)

(continued from previous page)

```
        return super(Element, self).__repr__(data=data)
```

### 4.3.2 Category

#### Category

**class** rpw.db.**Category**(*revit_object*, *enforce_type=True*)

Bases: *rpw.base.BaseObjectWrapper*

*DB.Category* Wrapper

**Attribute:** _revit_object (DB.Family): Wrapped `DB.Category`

**builtin**

Returns BuiltInCategory of the Category

**get_families**(*wrapped=True*, *doc=None*)

**Returns** List of Family elements in this same category

**Return type** Families (`DB.Family`)

**get_instances**(*wrapped=True*)

**Returns** List of Symbol Instances in the Category.

**Return type** (`DB.FamilyInstance`)

**get_symbols**(*wrapped=True*)

**Returns** List of Symbol Types in the Category

**Return type** Symbols (`DB.FamilySymbol`)

**name**

Returns name of the Category

#### Implementation

```
import rpw
from rpw import revit, DB
from rpw.db.element import Element
from rpw.base import BaseObjectWrapper
from rpw.utils.logger import logger, deprecate_warning
from rpw.db.builtins import BicEnum


class Category(BaseObjectWrapper):
    """
    `DB.Category` Wrapper

    Attribute:
        _revit_object (DB.Family): Wrapped ``DB.Category``
    """
```

(continues on next page)

(continued from previous page)

```
    _revit_object_class = DB.Category

    @property
    def name(self):
        """ Returns name of the Category """
        return self._revit_object.Name

    def get_families(self, wrapped=True, doc=None):
        """
        Returns:
            Families (``DB.Family``): List of Family elements in this
            same category

        """
        # There has to be a better way, but perhaps not: https://goo.gl/MqdzWg
        unique_family_ids = set()
        for symbol in self.get_symbols(wrapped=True):
            unique_family_ids.add(symbol.family.Id)
        doc = doc or revit.doc
        elements = [doc.GetElement(family_id) for family_id in unique_family_ids]
        return [Element(e) for e in elements] if wrapped else elements

    @property
    def families(self):
        deprecate_warning('Category.families',
                          'Category.get_families(wrapped=True')
        return self.get_families(wrapped=True)

    def get_symbols(self, wrapped=True):
        """
        Returns:
            Symbols (``DB.FamilySymbol``): List of Symbol Types in the Category
        """
        collector = rpw.db.Collector(of_category=self.builtin, is_type=True)
        return collector.get_elements(wrapped)

    @property
    def symbols(self):
        deprecate_warning('Category.symbols',
                          'Category.get_symbols(wrapped=True')
        return self.get_symbols(wrapped=True)

    def get_instances(self, wrapped=True):
        """
        Returns:
            (``DB.FamilyInstance``): List of Symbol Instances in the Category.
        """
        collector = rpw.db.Collector(of_category=self.builtin, is_not_type=True)
        return collector.get_elements(wrapped)

    @property
    def instances(self):
        deprecate_warning('Category.instances',
                          'Category.get_instances(wrapped=True')
        return self.get_instances(wrapped=True)
```

(continues on next page)

(continued from previous page)

```
    @property
    def builtin(self):
        """ Returns BuiltInCategory of the Category """
        return BicEnum.from_category_id(self._revit_object.Id)

    @property
    def _builtin_enum(self):
        deprecate_warning('Category._builtin_enum()', 'Category.builtin')
        return self.builtin

    def __repr__(self):
        return super(Category, self).__repr__({'name': self.name})
```

### 4.3.3 Curve

Curve Wrappers

#### Curve

**class** `rpw.db.`**`Curve`**(*revit_object*, *enforce_type=True*)

Bases: *`rpw.base.BaseObjectWrapper`*

DB.Curve Wrapper

```
>>> curve = Curve.new(ExistingCurveObject)
>>> curve.create_detail()
```

**`create_detail`**(*view=None*, *doc=Document*)

**Parameters**

- **view** (`DB.View`) – Optional View. Default: `uidoc.ActiveView`
- **doc** (`DB.Document`) – Optional Document. Default: `doc`

**`unwrap`**()

Returns the Original Wrapped Element

#### Line

**class** `rpw.db.`**`Line`**(*revit_object*, *enforce_type=True*)

Bases: `rpw.db.curve.Curve`

DB.Line Wrapper

```
>>> line = Line.new([-10,0], [10,0])
>>> # or
>>> line = Line.new(ExistingLineObject)
>>> line.create_detail()
```

**`create_detail`**(*view=None*, *doc=Document*)

**Parameters**

- **view** (`DB.View`) – Optional View. Default: `uidoc.ActiveView`

- **doc** (DB.Document) – Optional Document. Default: doc

**end_point**
End Point of line

**end_points**
End Points of line

**mid_point**
Mid Point of line

**classmethod new** (*pt1*, *pt2*)

**Parameters**

- **point1** (point) – Point like object. See *XYZ*
- **point2** (point) – Point like object. See *XYZ*

**start_point**
Start Point of line

**unwrap**()
Returns the Original Wrapped Element

## Ellipse

**class** rpw.db.**Ellipse** (*revit_object*, *enforce_type=True*)
Bases: rpw.db.curve.Curve

```
>>> ellipse = Ellipse.new([-10,0], [10,0])
>>> # or
>>> ellipse = Ellipse.new(ExistingEllipseObject)
>>> ellipse.create_detail()
```

**create_detail** (*view=None*, *doc=Document*)

**Parameters**

- **view** (DB.View) – Optional View. Default: uidoc.ActiveView
- **doc** (DB.Document) – Optional Document. Default: doc

**classmethod new** (*center*, *x_radius*, *y_radius*, *x_axis=None*, *y_axis=None*, *start_param=0.0*, *end_param=6.283185307179586*)

**Parameters**

- **center** (point) – Center of Ellipse
- **x_radius** (float) – X Radius
- **y_radius** (float) – Y Radius
- **x_axis** (point) – X Axis
- **y_axis** (point) – Y Axis
- **start_param** (float) – Start Parameter
- **end_param** (float) – End Parameter

**unwrap**()
Returns the Original Wrapped Element

## Circle

**class** rpw.db.**Circle**(*revit_object*, *enforce_type=True*)

Bases: `rpw.db.curve.Ellipse`

```
>>> circle = Circle.new([-10,0], 2)
>>> # or
>>> circle = Circle.new(ExistingCircleObject)
>>> circle.create_detail()
```

**create_detail**(*view=None*, *doc=Document*)

**Parameters**

- **view** (`DB.View`) – Optional View. Default: `uidoc.ActiveView`
- **doc** (`DB.Document`) – Optional Document. Default: `doc`

**classmethod new**(*center*, *radius*, *x_axis=None*, *y_axis=None*, *start_param=0.0*, *end_param=6.283185307179586*)

**Parameters**

- **center** (`point`) – Center of Ellipse
- **x_radius** (`float`) – X Radius
- **x_axis** (`point`) – X Axis
- **y_axis** (`point`) – Y Axis
- **start_param** (`float`) – Start Parameter
- **end_param** (`float`) – End Parameter

**unwrap**()

Returns the Original Wrapped Element

## Implementation

```
""" Curve Wrappers """

from math import pi as PI

from rpw import revit, DB
from rpw.base import BaseObjectWrapper
from rpw.db.element import Element
from rpw.db.xyz import XYZ
from rpw.utils.mixins import ByNameCollectMixin


class Curve(BaseObjectWrapper):
    """
    DB.Curve Wrapper

    >>> curve = Curve.new(ExistingCurveObject)
    >>> curve.create_detail()

    """
```

(continues on next page)

(continued from previous page)

```
    _revit_object_class = DB.Curve

    def create_detail(self, view=None, doc=revit.doc):
        """
        Args:
            view (``DB.View``): Optional View. Default: ``uidoc.ActiveView``
            doc (``DB.Document``): Optional Document. Default: ``doc``
        """
        # TODO: Accept Detail Type (GraphicStyle)
        view = view or revit.active_view.unwrap()
        return doc.Create.NewDetailCurve(view, self._revit_object)

    def create_model(self, view=None, doc=revit.doc):
        # http://www.revitapidocs.com/2017.1/b880c4d7-9841-e44e-2a1c-36fefe274e2e.htm
        raise NotImplemented


class Line(Curve):

    """
    DB.Line Wrapper

    >>> line = Line.new([-10,0], [10,0])
    >>> # or
    >>> line = Line.new(ExistingLineObject)
    >>> line.create_detail()

    """
    _revit_object_class = DB.Line

    @classmethod
    def new(cls, pt1, pt2):
        """
        Args:
            point1 (``point``): Point like object. See :any:`XYZ`
            point2 (``point``): Point like object. See :any:`XYZ`
        """
        pt1 = XYZ(pt1)
        pt2 = XYZ(pt2)
        line = DB.Line.CreateBound(pt1.unwrap(), pt2.unwrap())
        return cls(line)

    @property
    def start_point(self):
        """ Start Point of line """
        return XYZ(self._revit_object.GetEndPoint(0))

    @property
    def end_point(self):
        """ End Point of line """
        return XYZ(self._revit_object.GetEndPoint(1))

    @property
    def mid_point(self):
        """ Mid Point of line """
        return XYZ(self._revit_object.GetEndPoint(0.5))
```

(continues on next page)

(continued from previous page)

```
    @property
    def end_points(self):
        """ End Points of line """
        return (XYZ(self.start_point), XYZ(self.end_point))


class Ellipse(Curve):
    """

    >>> ellipse = Ellipse.new([-10,0], [10,0])
    >>> # or
    >>> ellipse = Ellipse.new(ExistingEllipseObject)
    >>> ellipse.create_detail()

    """
    _revit_object_class = DB.Ellipse

    @classmethod
    def new(cls, center, x_radius, y_radius, x_axis=None, y_axis=None,
            start_param=0.0, end_param=2*PI):
        """
        Args:
            center (``point``): Center of Ellipse
            x_radius (``float``): X Radius
            y_radius (``float``): Y Radius
            x_axis (``point``): X Axis
            y_axis (``point``): Y Axis
            start_param (``float``): Start Parameter
            end_param (``float``): End Parameter
        """
        center = XYZ(center).unwrap()
        x_axis = DB.XYZ(1,0,0) if x_axis is None else XYZ(x_axis).unwrap().Normalize()
        y_axis = DB.XYZ(0,1,0) if y_axis is None else XYZ(y_axis).unwrap().Normalize()

        start_param = start_param or 0.0
        end_param = start_param or PI*2

        ellipse = DB.Ellipse.Create(center, x_radius, y_radius, x_axis, y_axis, start_
↪param, end_param)
        return cls(ellipse)

class Circle(Ellipse):
    """

    >>> circle = Circle.new([-10,0], 2)
    >>> # or
    >>> circle = Circle.new(ExistingCircleObject)
    >>> circle.create_detail()

    """

    @classmethod
    def new(cls, center,
            radius,
            x_axis=None, y_axis=None,
            start_param=0.0, end_param=2*PI):
```

(continues on next page)

(continued from previous page)

```
        """
        Args:
            center (``point``): Center of Ellipse
            x_radius (``float``): X Radius
            x_axis (``point``): X Axis
            y_axis (``point``): Y Axis
            start_param (``float``): Start Parameter
            end_param (``float``): End Parameter
        """
        center = XYZ(center).unwrap()
        x_axis = DB.XYZ(1,0,0) if x_axis is None else XYZ(x_axis).unwrap().Normalize()
        y_axis = DB.XYZ(0,1,0) if y_axis is None else XYZ(y_axis).unwrap().Normalize()

        start_param = start_param or 0.0
        end_param = start_param or PI*2

        circle = DB.Ellipse.Create(center, radius, radius, x_axis, y_axis, start_
↪param, end_param)
        return cls(circle)


class Arc(Curve):
    """

    >>> arc = Arc.new([0,0], [0,0], [0,0])
    >>> # or
    >>> arc = Arc.new(ExistingArcObject)
    >>> arc.create_detail()

    """

    @classmethod
    def new(cls, *args):
            # http://www.revitapidocs.com/2017.1/19c3ba08-5443-c9d4-3a3f-0e78901fe6d4.
↪htm
            # XYZ, XYZ, XYZ
            # Plane, Double, Double, Double (Plane, Radius, startAngle, endAngle)
            # XYZ, Double, Double, Double ,XYZ, XYZ (Center, radius, vectors, angles)
        """
        Args:
            start_pt (``point``): Start Point
            mid_pt (``point``): End Point
            end_pt (``point``): Mid Point
        """
        if len(args) == 3:
            start_pt, end_pt, mid_pt = [XYZ(pt).unwrap() for pt in args]
            arc = DB.Arc.Create(start_pt, end_pt, mid_pt)
        else:
            raise NotImplemented('only arc by 3 pts available')
        return cls(arc)
```

### 4.3.4 Geometry

#### XYZ Wrapper

**class** rpw.db.**XYZ**(**point_reference*)

Bases: *rpw.base.BaseObjectWrapper*

*DB.XYZ* Wrapper

XYZ light wrapper with a few helpful methods:

```
>>> from rpw.db import db
>>> pt = db.XYZ(some_point)
>>> pt.as_tuple
(0,0,0)
>>> pt.x = 10
<rpw:XYZ % DB.XYZ: 0,0,10>
>>> pt.at_z(5)
<rpw:XYZ % DB.XYZ: 0,0,5>
>>> pt.as_dict()
{'x': 0, 'y':0, 'z':5}
```

**_revit_object**

*DB.XYZ* – Wrapped `DB.XYZ`

**__add__**(*point*)

Addition Method

**__eq__**(*other*)

Equality Method

**__init__**(**point_reference*)

XYZ Supports a wide variety of instantiation overloads:

```
>>> XYZ(0,0)
>>> XYZ(0,0,0)
>>> XYZ([0,0])
>>> XYZ([0,0,0])
>>> XYZ(DB.XYZ(0,0,0))
```

**Parameters** **point_reference** (`DB.XYZ`,‘‘iterable‘‘, `args`) – Point like data

**__mul__**(*value*)

Multiplication Method

**__sub__**(*point*)

Subtraction Method

**as_dict**

Dictionary representing the xyz coordinate of the Point

**Returns** dict with float of XYZ values

**Return type** (dict)

**as_tuple**

Tuple representing the xyz coordinate of the Point

**Returns** tuple float of XYZ values

**Return type** (tuple)

**at_z** (*z*, *wrapped=True*)
Returns a new point at the passed Z value

**Parameters** **z** (*float*) – Elevation of new Points

**Returns** New Points

**Return type** (*XYZ*)

**x**
X Value

**y**
Y Value

**z**
Z Value

## Implementation

```
from rpw import DB
from rpw.base import BaseObjectWrapper
from rpw.exceptions import RpwCoerceError
from rpw.db.transform import Transform
from collections import OrderedDict

class XYZ(BaseObjectWrapper):
    """
    `DB.XYZ` Wrapper

    XYZ light wrapper with a few helpful methods:

    >>> from rpw.db import db
    >>> pt = db.XYZ(some_point)
    >>> pt.as_tuple
    (0,0,0)
    >>> pt.x = 10
    <rpw:XYZ % DB.XYZ: 0,0,10>
    >>> pt.at_z(5)
    <rpw:XYZ % DB.XYZ: 0,0,5>
    >>> pt.as_dict()
    {'x': 0, 'y':0, 'z':5}

    Attributes:
        _revit_object (DB.XYZ): Wrapped ``DB.XYZ``
    """

    _revit_object_class = DB.XYZ

    def __init__(self, *point_reference):
        """
        XYZ Supports a wide variety of instantiation overloads:

        >>> XYZ(0,0)
        >>> XYZ(0,0,0)
        >>> XYZ([0,0])
        >>> XYZ([0,0,0])
```

(continues on next page)

(continued from previous page)

```
        >>> XYZ(DB.XYZ(0,0,0))

        Args:
            point_reference (``DB.XYZ``,``iterable``, ``args``): Point like data
        """
        # XYZ(0,0,0)
        if len(point_reference) == 3:
            xyz = DB.XYZ(*point_reference)
        # XYZ(0,0)
        elif len(point_reference) == 2:
            xyz = DB.XYZ(point_reference[0], point_reference[1], 0)
        # XYZ([0,0,0]) or # XYZ([0,0])
        elif len(point_reference) == 1 and isinstance(point_reference[0], (tuple,␣
↪list)):
            # Assumes one arg, tuple
            xyz = XYZ(*point_reference[0])
            xyz = DB.XYZ(*xyz.as_tuple)
        # XYZ(DB.XYZ(0,0,0))
        elif len(point_reference) == 1 and isinstance(point_reference[0], DB.XYZ):
            # Assumes one arg, DB.XYZ
            xyz = point_reference[0]
        elif len(point_reference) == 1 and isinstance(point_reference[0], XYZ):
            # Assumes one arg, DB.XYZ
            xyz = point_reference[0].unwrap()
        else:
            raise RpwCoerceError(point_reference, 'point-like object')
        super(XYZ, self).__init__(xyz)

    @property
    def x(self):
        """X Value"""
        return self._revit_object.X

    @property
    def y(self):
        """Y Value"""
        return self._revit_object.Y

    @property
    def z(self):
        """Z Value"""
        return self._revit_object.Z

    @x.setter
    def x(self, value):
        self._revit_object = DB.XYZ(value, self.y, self.z)

    @y.setter
    def y(self, value):
        self._revit_object = DB.XYZ(self.x, value, self.z)

    @z.setter
    def z(self, value):
        self._revit_object = DB.XYZ(self.x, self.y, value)

    def at_z(self, z, wrapped=True):
        """ Returns a new point at the passed Z value
```

(continues on next page)

(continued from previous page)

```
        Args:
            z(float): Elevation of new Points

        Returns:
            (:any:`XYZ`): New Points
        """
        return XYZ(self.x, self.y, z) if wrapped else DB.XYZ(self.x, self.y, z)

    @property
    def as_tuple(self):
        """
        Tuple representing the xyz coordinate of the Point

        Returns:
            (tuple): tuple float of XYZ values

        """
        return (self.x, self.y, self.z)

    @property
    def as_dict(self):
        """
        Dictionary representing the xyz coordinate of the Point

        Returns:
            (dict): dict with float of XYZ values

        """
        return OrderedDict([('x', self.x), ('y', self.y), ('z', self.z)])

    def rotate(self, rotation, axis=None, radians=False):
        rotated_xyz = Transform.rotate_vector(self.unwrap(),
                                              rotation,
                                              center=None,
                                              axis=axis,
                                              radians=radians)
        return rotated_xyz

    def __mul__(self, value):
        """ Multiplication Method """
        return XYZ(self.unwrap() * value)

    def __add__(self, point):
        """ Addition Method """
        return XYZ(self.unwrap() + XYZ(point).unwrap())

    def __sub__(self, point):
        """ Subtraction Method """
        return XYZ(self.unwrap() - XYZ(point).unwrap())

    def __eq__(self, other):
        """ Equality Method """
        return self._revit_object.IsAlmostEqualTo(XYZ(other).unwrap())

    def __repr__(self):
        return super(XYZ, self).__repr__(data=self.as_dict,
```

(continues on next page)

(continued from previous page)

```
                                   to_string='Autodesk.Revit.DB.XYZ')
```

### 4.3.5 Reference

Reference Wrappers

```
>>> pick = rpw.ui.Pick()
>>> references = pick.pick_element(multiple=True)
>>> references
[<rpw:Reference>, <rpw:Reference>]
>>> references[0].as_global_pt
<rpw:XYZ>
```

Linked Element

```
>>> reference = pick.pick_linked_element()
>>> element = reference.get_element()
```

**class** `rpw.db.`**`Reference`**(*reference*, *linked=False*)

Bases: `rpw.db.element.Element`

*DB.Reference* Wrapper Inherits from *`Element`*

```
>>>
```

**Attribute:** _revit_object (DB.Reference): Wrapped `DB.Reference` doc (Document): Element Document

**`__init__`**(*reference*, *linked=False*)

Main Element Instantiation

```
>>> from rpw import db
>>> wall = db.Element(SomeElementId)
<rpw: WallInstance % DB.Wall >
>>> wall.parameters['Height']
10.0
>>> wall.parameters.builtins['WALL_LOCATION_LINE']
1
```

**Parameters** **`element`** (*Element Reference*) – Can be `DB.Element`, `DB.ElementId`, or `int`.

**Returns** Instance of Wrapped Element.

**Return type** *`Element`*

**`as_global_pt`**

Returns `GlobalPoint` property of Reference

**`as_uv_pt`**

Returns `UVPoint` property of Reference - Face references only

**`get_element`**(*wrapped=True*)

Element of Reference

**`get_geometry`**()

GeometryObject from Reference

**id**
ElementId of Reference

## Implementation

```
"""
Reference Wrappers

>>> pick = rpw.ui.Pick()
>>> references = pick.pick_element(multiple=True)
>>> references
[<rpw:Reference>, <rpw:Reference>]
>>> references[0].as_global_pt
<rpw:XYZ>

Linked Element

>>> reference = pick.pick_linked_element()
>>> element = reference.get_element()

"""

import rpw
from rpw import revit, DB
from rpw.db.element import Element
from rpw.db.xyz import XYZ
from rpw.utils.logger import logger
# from rpw.db.builtins import BipEnum


class Reference(Element):
    """
    `DB.Reference` Wrapper
    Inherits from :any:`Element`

    >>>

    Attribute:
        _revit_object (DB.Reference): Wrapped ``DB.Reference``
        doc (Document): Element Document
    """

    _revit_object_class = DB.Reference

    def __init__(self, reference, linked=False):
        if not linked:
            doc = revit.doc
        else:
            link_instance = revit.doc.GetElement(reference.ElementId)
            doc = link_instance.GetLinkDocument()

        super(Reference, self).__init__(reference, doc=doc)
        self.doc = doc
        self.linked = linked
```

(continues on next page)

(continued from previous page)

```
    def __repr__(self):
        return super(Reference, self).__repr__(data={'id': self.id})

    @property
    def as_global_pt(self):
        """ Returns ``GlobalPoint`` property of Reference """
        pt = self._revit_object.GlobalPoint
        if pt:
            return XYZ(pt)

    @property
    def as_uv_pt(self):
        """ Returns ``UVPoint`` property of Reference - Face references only """
        pt = self._revit_object.UVPoint
        if pt:
            # TODO XYZ needs to handle XYZ
            return pt
            # return XYZ(pt)

    @property
    def id(self):
        """ ElementId of Reference """
        return self._revit_object.ElementId if not self.linked else self._revit_
↪object.LinkedElementId

    def get_element(self, wrapped=True):
        """ Element of Reference """
        element = self.doc.GetElement(self.id)
        return element if not wrapped else Element(element)

    def get_geometry(self):
        """ GeometryObject from Reference """
        ref = self._revit_object
        return self.doc.GetElement(ref).GetGeometryObjectFromReference(ref)
```

### 4.3.6 Transaction

Wrappers to make Revit Transactions work with Python Context Manager.

**class** rpw.db.**Transaction**(*name=None*, *doc=Document*)

Bases: *rpw.base.BaseObjectWrapper*

Simplifies transactions by applying `Transaction.Start()` and `Transaction.Commit()` before and after the context. Automatically rolls back if exception is raised.

```
>>> from rpw import db
>>> with db.Transaction('Move Wall'):
>>>     wall.DoSomething()
```

```
>>> with db.Transaction('Move Wall') as t:
>>>     wall.DoSomething()
>>>     assert t.HasStarted() is True
>>> assert t.HasEnded() is True
```

**Wrapped Element:** self._revit_object = *Revit.DB.Transaction*

**__init__**(*name=None*, *doc=Document*)
Child classes can use self._revit_object to refer back to Revit Element

> **Warning:** Any Wrapper that inherits and overrides __init__ class MUST ensure `_revit_object` is created by calling super().__init__ before setting any self attributes. Not doing so will cause recursion errors and Revit will crash. BaseObjectWrapper should define a class variable _revit_object_class to define the object class being wrapped.

**static ensure**(*name*)
Transaction Manager Decorator

Decorate any function with `@Transaction.ensure('Transaction Name')` and the funciton will run within a Transaction Context.

**Parameters name** (`str`) – Name of the Transaction

```
>>> from rpw import db
>>> @db.Transaction.ensure('Do Something')
>>> def set_some_parameter(wall, value):
>>>     wall.parameters['Comments'].value = value
>>>
>>> set_some_parameter(wall, value)
```

## Implementation

```
import traceback
from rpw import revit, DB
from rpw.base import BaseObjectWrapper
from rpw.exceptions import RpwException
from rpw.utils.logger import logger


class Transaction(BaseObjectWrapper):
    """
    Simplifies transactions by applying ``Transaction.Start()`` and
    ``Transaction.Commit()`` before and after the context.
    Automatically rolls back if exception is raised.

    >>> from rpw import db
    >>> with db.Transaction('Move Wall'):
    >>>     wall.DoSomething()

    >>> with db.Transaction('Move Wall') as t:
    >>>     wall.DoSomething()
    >>>     assert t.HasStarted() is True
    >>> assert t.HasEnded() is True

    Wrapped Element:
        self._revit_object = `Revit.DB.Transaction`

    """

    _revit_object_class = DB.Transaction
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, name=None, doc=revit.doc):
        if name is None:
            name = 'RPW Transaction'
        super(Transaction, self).__init__(DB.Transaction(doc, name))
        self.transaction = self._revit_object

    def __enter__(self):
        self.transaction.Start()
        return self

    def __exit__(self, exception, exception_msg, tb):
        if exception:
            self.transaction.RollBack()
            logger.error('Error in Transaction Context: has rolled back.')
            # traceback.print_tb(tb)
            # raise exception # Let exception through
        else:
            try:
                self.transaction.Commit()
            except Exception as exc:
                self.transaction.RollBack()
                logger.error('Error in Transaction Commit: has rolled back.')
                logger.error(exc)
                raise

    @staticmethod
    def ensure(name):
        """ Transaction Manager Decorator

        Decorate any function with ``@Transaction.ensure('Transaction Name')``
        and the funciton will run within a Transaction Context.

        Args:
            name (str): Name of the Transaction

        >>> from rpw import db
        >>> @db.Transaction.ensure('Do Something')
        >>> def set_some_parameter(wall, value):
        >>>     wall.parameters['Comments'].value = value
        >>>
        >>> set_some_parameter(wall, value)
        """
        from functools import wraps

        def wrap(f):
            @wraps(f)
            def wrapped_f(*args, **kwargs):
                with Transaction(name):
                    return_value = f(*args, **kwargs)
                return return_value
            return wrapped_f
        return wrap

    # TODO: Add  __repr__ with Transaction Status
    # TODO: Merge Transaction Status
    # TODO: add check for if transaction is in progress, especially for ensure
```

(continues on next page)

(continued from previous page)

```
    # TODO: add ensure to TransactionGroup


class TransactionGroup(BaseObjectWrapper):
    """
    Similar to Transaction, but for ``DB.Transaction Group``

    >>> from rpw import db
    >>> with db.TransacationGroup('Do Major Task'):
    >>>     with db.Transaction('Do Task'):
    >>>         # Do Stuff

    >>> from rpw import db
    >>> with db.TransacationGroup('Do Major Task', assimilate=False):
    >>>     with db.Transaction('Do Task'):
    >>>         # Do Stuff
    """

    _revit_object_class = DB.TransactionGroup

    def __init__(self, name=None, assimilate=True, doc=revit.doc):
        """
            Args:
                name (str): Name of the Transaction
                assimilate (bool): If assimilates is ``True``,
                    transaction history is `squashed`.
        """
        if name is None:
            name = 'RPW Transaction Group'
        super(TransactionGroup, self).__init__(DB.TransactionGroup(doc, name))
        self.transaction_group = self._revit_object
        self.assimilate = assimilate

    def __enter__(self):
        self.transaction_group.Start()
        return self.transaction_group

    def __exit__(self, exception, exception_msg, tb):
        if exception:
            self.transaction_group.RollBack()
            logger.error('Error in TransactionGroup Context: has rolled back.')
        else:
            try:
                if self.assimilate:
                    self.transaction_group.Assimilate()
                else:
                    self.transaction_group.Commit()
            except Exception as exc:
                self.transaction_group.RollBack()
                logger.error('Error in TransactionGroup Commit: \
                              has rolled back.')
                logger.error(exc)
                raise exc


class DynamoTransaction(object):
```

(continues on next page)

(continued from previous page)

```
    # TODO: Use Dynamo Transaction when HOST is 'Dynamo'

    def __init__(self, name):
        raise NotImplemented
    #     from rpw import TransactionManager
    #     self.transaction = TransactionManager.Instance
    #
    # def __enter__(self):
    #     self.transaction.EnsureInTransaction(doc)
    #
    # def __exit__(self, exception, exception_msg, traceback):
    #     if exception:
    #         pass # self.transaction.RollBack()
    #     else:
    #         try:
    #             self.transaction.TransactionTaskDone()
    #         except:
    #             try:
    #                 self.transaction.ForceCloseTransaction()
    #             except:
    #                 raise RpwException('Failed to complete transaction')
```

## 4.3.7 Collector

### Overview

Usage

```
>>> from rpw import db
>>> levels = db.Collector(of_category='Levels', is_type=True)
>>> walls = db.Collector(of_class='Wall', where=lambda x: x.parameters['Length'] > 5)
>>> desks = db.Collector(of_class='FamilyInstance', level='Level 1')
```

**Note:** As of June 2017, these are the filters that have been implemented:

ElementCategoryFilter = of_category

ElementClassFilter = of_class

ElementIsCurveDrivenFilter = is_curve_driven

ElementIsElementTypeFilter = is_type + is_not_type

ElementOwnerViewFilter = view

ElementLevelFilter = level + not_level

ElementOwnerViewFilter = owner_view + is_view_independent

FamilySymbolFilter = family

FamilyInstanceFilter = symbol

ElementParameterFilter = parameter_filter

Exclusion = exclude

UnionWith = or_collector

IntersectWith = and_collector

Custom = where

## FilteredElementCollector

**class** rpw.db.**Collector**(***filters*)

Bases: `rpw.base.BaseObjectWrapper`

Revit FilteredElement Collector Wrapper

**Usage:**

```
>>> collector = Collector(of_class='View')
>>> elements = collector.get_elements()
```

Multiple Filters:

```
>>> Collector(of_class='Wall', is_not_type=True)
>>> Collector(of_class='ViewSheet', is_not_type=True)
>>> Collector(of_category='OST_Rooms', level=some_level)
>>> Collector(symbol=SomeSymbol)
>>> Collector(owner_view=SomeView)
>>> Collector(owner_view=None)
>>> Collector(parameter_filter=parameter_filter)
```

Use Enumeration member or its name as a string:

```
>>> Collector(of_category='OST_Walls')
>>> Collector(of_category=DB.BuiltInCategory.OST_Walls)
>>> Collector(of_class=DB.ViewType)
>>> Collector(of_class='ViewType')
```

Search Document, View, or list of elements

```
>>> Collector(of_category='OST_Walls') # doc is default
>>> Collector(view=SomeView, of_category='OST_Walls') # Doc is default
>>> Collector(doc=SomeLinkedDoc, of_category='OST_Walls')
>>> Collector(elements=[Element1, Element2,...], of_category='OST_Walls')
>>> Collector(owner_view=SomeView)
>>> Collector(owner_view=None)
```

collector.**get_elements**
Returns list of all *collected* elements

collector.**get_first**
Returns first found element, or `None`

collector.**get_elements**
Returns list with all elements wrapped. Elements will be instantiated using `Element`

**Wrapped Element:** self._revit_object = `Revit.DB.FilteredElementCollector`

**__init__**(***filters*)

**Parameters** ****filters** (`keyword args`) – Scope and filters

**Returns** Collector Instance

**Return type** Collector (`Collector`)

**Scope Options:**

- `view` *(DB.View)*: View Scope (Optional)
- `element_ids` *([ElementId])*: List of Element Ids to limit Collector Scope
- `elements` *([Element])*: List of Elements to limit Collector Scope

**Warning:** Only one scope filter should be used per query. If more then one is used, only one will be applied, in this order `view` > `elements` > `element_ids`

**Filter Options:**

- is_type (`bool`): Same as `WhereElementIsElementType`
- is_not_type (`bool`): Same as `WhereElementIsNotElementType`
- of_class (`Type`): Same as `OfClass`. Type can be `DB.SomeType` or string: `DB.Wall` or `'Wall'`
- of_category (`BuiltInCategory`): Same as `OfCategory`. Can be `DB.BuiltInCategory.OST_Wall` or `'Wall'`
- owner_view (`DB.ElementId, View`): ``WhereElementIsViewIndependent(True)`
- is_view_independent (`bool`): `WhereElementIsViewIndependent(True)`
- family (`DB.ElementId`, `DB.Element`): Element or ElementId of Family
- symbol (`DB.ElementId`, `DB.Element`): Element or ElementId of Symbol
- level (`DB.Level`, `DB.ElementId`, `Level Name`): Level, ElementId of Level, or Level Name
- not_level (`DB.Level`, `DB.ElementId`, `Level Name`): Level, ElementId of Level, or Level Name
- parameter_filter (*`ParameterFilter`*): Applies `ElementParameterFilter`
- exclude (*element_references*): Element(s) or ElementId(s) to exlude from result
- and_collector (`collector`): Collector to intersect with. Elements must be present in both
- or_collector (`collector`): Collector to Union with. Elements must be present on of the two.
- where (*function*): function to test your elements against

**`_collect`** (*doc*, *collector*, *filters*)
Main Internal Recursive Collector Function.

**Parameters**

- **`doc`** (*UI.UIDocument*) – Document for the collector.
- **`collector`** (*FilteredElementCollector*) – FilteredElementCollector
- **`filters`** (*dict*) – Filters - {‘doc’: revit.doc, ‘of_class’: ‘Wall’}

**Returns** FilteredElementCollector

**Return type** collector (*FilteredElementCollector*)

**`elements`**
Returns list with all elements

**get_element_ids**()
Returns list with all elements instantiated using `Element`

**get_elements**(*wrapped=True*)
Returns list with all elements instantiated using `Element`

**get_first**(*wrapped=True*)
Returns first element or *None*

**Returns** First element or None

**Return type** Element (*DB.Element*, *None*)

**select**()
Selects Collector Elements on the UI

**wrapped_elements**
Returns list with all elements instantiated using `Element`

## ParameterFilter

**class** rpw.db.**ParameterFilter**(*parameter_reference*, ***conditions*)
Bases: `rpw.base.BaseObjectWrapper`

Parameter Filter Wrapper. Used to build a parameter filter to be used with the Collector.

**Usage:**

```
>>> param_id = DB.ElementId(DB.BuiltInParameter.TYPE_NAME)
>>> parameter_filter = ParameterFilter(param_id, equals='Wall 1')
>>> collector = Collector(parameter_filter=parameter_filter)
```

**Returns** A filter rule object, depending on arguments.

**Return type** FilterRule

**__init__**(*parameter_reference*, ***conditions*)
Creates Parameter Filter Rule

```
>>> param_rule = ParameterFilter(param_id, equals=2)
>>> param_rule = ParameterFilter(param_id, not_equals='a', case_
↪sensitive=True)
>>> param_rule = ParameterFilter(param_id, not_equals=3, reverse=True)
```

**Parameters**

- **param_id** (`DB.ElementID`) – ElementId of parameter
- ****conditions** – Filter Rule Conditions and options.
- **conditions** –

  `begins`, `not_begins`
  `contains`, `not_contains`
  `ends`, `not_ends`
  `equals`, `not_equals`
  `less`, `not_less`
  `less_equal`, `not_less_equal`
  `greater`, `not_greater`

`greater_equal`, `not_greater_equal`

- **options** –

  `case_sensitive`: Enforces case sensitive, String only

  `reverse`: Reverses result of Collector

**static from_element_and_parameter**(*element*, *param_name*, ***conditions*)
Alternative constructor to built Parameter Filter from Element + Parameter Name instead of parameter Id

```
>>> parameter_filter = ParameterFilter.from_element(element,param_name, less_
↪than=10)
>>> Collector(parameter_filter=parameter_filter)
```

## Implementation

```
"""
Usage

>>> from rpw import db
>>> levels = db.Collector(of_category='Levels', is_type=True)
>>> walls = db.Collector(of_class='Wall', where=lambda x: x.parameters['Length'] > 5)
>>> desks = db.Collector(of_class='FamilyInstance', level='Level 1')

Note:
    As of June 2017, these are the filters that have been implemented:

    | ``ElementCategoryFilter`` = ``of_category``
    | ``ElementClassFilter`` = ``of_class``
    | ``ElementIsCurveDrivenFilter`` = ``is_curve_driven``
    | ``ElementIsElementTypeFilter`` = ``is_type`` + ``is_not_type``
    | ``ElementOwnerViewFilter`` = ``view``
    | ``ElementLevelFilter`` = ``level`` + ``not_level``
    | ``ElementOwnerViewFilter`` = ``owner_view`` + ``is_view_independent``
    | ``FamilySymbolFilter`` = ``family``
    | ``FamilyInstanceFilter`` = ``symbol``
    | ``ElementParameterFilter`` = ``parameter_filter``
    | ``Exclusion`` = ``exclude``
    | ``UnionWith`` = ``or_collector``
    | ``IntersectWith`` = ``and_collector``
    | ``Custom`` = where

"""

from rpw import revit, DB
from rpw.utils.dotnet import List
from rpw.base import BaseObjectWrapper, BaseObject
from rpw.exceptions import RpwException, RpwTypeError, RpwCoerceError
from rpw.db.element import Element
from rpw.db.builtins import BicEnum, BipEnum
from rpw.ui.selection import Selection
from rpw.db.collection import ElementSet
from rpw.utils.coerce import to_element_id, to_element_ids
from rpw.utils.coerce import to_category, to_class
from rpw.utils.logger import logger
```

(continues on next page)

(continued from previous page)

```
from rpw.utils.logger import deprecate_warning

# More Info on Performance and ElementFilters:
# http://thebuildingcoder.typepad.com/blog/2015/12/quick-slow-and-linq-element-
↪filtering.html


class BaseFilter(BaseObject):
    """ Base Filter and Apply Logic """

    method = 'WherePasses'

    @classmethod
    def process_value(cls, value):
        """
        Filters must implement this method to process the input values and
        convert it into the proper filter or value.

        For example, if the user inputs `level=Level`,
        process value will create a ElementLevelFilter() with the id of Level.

        Additionally, this method can be used for more advanced input
        processing, for example, converting a 'LevelName' into a Level
        to allow for more flexible input options
        """
        raise NotImplemented

    @classmethod
    def apply(cls, doc, collector, value):
        """
        Filters can overide this method to define how the filter is applied
        The default behavious is to chain the ``method`` defined by the filter
        class (ie. WherePasses) to the collector, and feed it the input `value`
        """
        method_name = cls.method
        method = getattr(collector, method_name)

        # FamilyInstanceFilter is the only Filter that  requires Doc
        if cls is not FilterClasses.FamilyInstanceFilter:
            value = cls.process_value(value)
        else:
            value = cls.process_value(value, doc)

        return method(value)


class SuperQuickFilter(BaseFilter):
    """ Preferred Quick """
    priority_group = 0


class QuickFilter(BaseFilter):
    """ Typical Quick """
    priority_group = 1


class SlowFilter(BaseFilter):
```

(continues on next page)

(continued from previous page)

```
    """ Typical Slow """
    priority_group = 2


class SuperSlowFilter(BaseFilter):
    """ Leave it for Last. Must unpack results """
    priority_group = 3

class LogicalFilter(BaseFilter):
    """ Leave it after Last as it must be completed """
    priority_group = 4

class FilterClasses():
    """
    Groups FilterClasses to facilitate discovery.

    # TODO: Move Filter doc to Filter Classes

    Implementation Tracker:
    Quick
        X Revit.DB.ElementCategoryFilter = of_category
        X Revit.DB.ElementClassFilter = of_class
        X Revit.DB.ElementIsCurveDrivenFilter = is_curve_driven
        X Revit.DB.ElementIsElementTypeFilter = is_type / is_not_type
        X Revit.DB.ElementOwnerViewFilter = view
        X Revit.DB.FamilySymbolFilter = family
        X Revit.DB.ExclusionFilter = exclude
        X Revit.DB.IntersectWidth = and_collector
        X Revit.DB.UnionWidth = or_collector
        _ Revit.DB.BoundingBoxContainsPointFilter
        _ Revit.DB.BoundingBoxIntersectsFilter
        _ Revit.DB.BoundingBoxIsInsideFilter
        _ Revit.DB.ElementDesignOptionFilter
        _ Revit.DB.ElementMulticategoryFilter
        _ Revit.DB.ElementMulticlassFilter
        _ Revit.DB.ElementStructuralTypeFilter
        _ Revit.DB.ElementWorksetFilter
        _ Revit.DB.ExtensibleStorage ExtensibleStorageFilter
    Slow
        X Revit.DB.ElementLevelFilter
        X Revit.DB.FamilyInstanceFilter = symbol
        X Revit.DB.ElementParameterFilter
        _ Revit.DB.Architecture RoomFilter
        _ Revit.DB.Architecture RoomTagFilter
        _ Revit.DB.AreaFilter
        _ Revit.DB.AreaTagFilter
        _ Revit.DB.CurveElementFilter
        _ Revit.DB.ElementIntersectsFilter
        _ Revit.DB.ElementPhaseStatusFilter
        _ Revit.DB.Mechanical SpaceFilter
        _ Revit.DB.Mechanical SpaceTagFilter
        _ Revit.DB.PrimaryDesignOptionMemberFilter
        _ Revit.DB.Structure FamilyStructuralMaterialTypeFilter
        _ Revit.DB.Structure StructuralInstanceUsageFilter
        _ Revit.DB.Structure StructuralMaterialTypeFilter
        _ Revit.DB.Structure StructuralWallUsageFilter
        _ Autodesk.Revit.UI.Selection SelectableInViewFilter
```

(continues on next page)

(continued from previous page)

```
    Logical
        _ Revit.DB.LogicalAndFilter = and_filter
        _ Revit.DB.LogicalOrFilter = or_filter

    Others
        X Custom where - uses lambda

    """
    @classmethod
    def get_available_filters(cls):
        """ Discover all Defined Filter Classes """
        filters = []
        for filter_class_name in dir(FilterClasses):
            if filter_class_name.endswith('Filter'):
                filters.append(getattr(FilterClasses, filter_class_name))
        return filters

    @classmethod
    def get_sorted(cls):
        """ Returns Defined Filter Classes sorted by priority """
        return sorted(FilterClasses.get_available_filters(),
                      key=lambda f: f.priority_group)

    class ClassFilter(SuperQuickFilter):
        keyword = 'of_class'

        @classmethod
        def process_value(cls, class_reference):
            class_ = to_class(class_reference)
            return DB.ElementClassFilter(class_)

    class CategoryFilter(SuperQuickFilter):
        keyword = 'of_category'

        @classmethod
        def process_value(cls, category_reference):
            category = to_category(category_reference)
            return DB.ElementCategoryFilter(category)

    class IsTypeFilter(QuickFilter):
        keyword = 'is_type'

        @classmethod
        def process_value(cls, bool_value):
            return DB.ElementIsElementTypeFilter(not(bool_value))

    class IsNotTypeFilter(IsTypeFilter):
        keyword = 'is_not_type'

        @classmethod
        def process_value(cls, bool_value):
            return DB.ElementIsElementTypeFilter(bool_value)

    class FamilySymbolFilter(QuickFilter):
        keyword = 'family'
```

(continues on next page)

(continued from previous page)

```
        @classmethod
        def process_value(cls, family_reference):
            family_id = to_element_id(family_reference)
            return DB.FamilySymbolFilter(family_id)

    class ViewOwnerFilter(QuickFilter):
        keyword = 'owner_view'
        reverse = False

        @classmethod
        def process_value(cls, view_reference):
            if view_reference is not None:
                view_id = to_element_id(view_reference)
            else:
                view_id = DB.ElementId.InvalidElementId
            return DB.ElementOwnerViewFilter(view_id, cls.reverse)

    class ViewIndependentFilter(QuickFilter):
        keyword = 'is_view_independent'

        @classmethod
        def process_value(cls, bool_value):
            view_id = DB.ElementId.InvalidElementId
            return DB.ElementOwnerViewFilter(view_id, not(bool_value))

    class CurveDrivenFilter(QuickFilter):
        keyword = 'is_curve_driven'

        @classmethod
        def process_value(cls, bool_value):
            return DB.ElementIsCurveDrivenFilter(not(bool_value))

    class FamilyInstanceFilter(SlowFilter):
        keyword = 'symbol'

        @classmethod
        def process_value(cls, symbol_reference, doc):
            symbol_id = to_element_id(symbol_reference)
            return DB.FamilyInstanceFilter(doc, symbol_id)

    class LevelFilter(SlowFilter):
        keyword = 'level'
        reverse = False

        @classmethod
        def process_value(cls, level_reference):
            """ Process level= input to allow for level name """
            if isinstance(level_reference, str):
                level = Collector(of_class='Level', is_type=False,
                                  where=lambda x:
                                  x.Name == level_reference)
                try:
                    level_id = level[0].Id
                except IndexError:
                    RpwCoerceError(level_reference, DB.Level)
            else:
                level_id = to_element_id(level_reference)
```

(continues on next page)

(continued from previous page)

```
            return DB.ElementLevelFilter(level_id, cls.reverse)

    class NotLevelFilter(LevelFilter):
        keyword = 'not_level'
        reverse = True

    class ParameterFilter(SlowFilter):
        keyword = 'parameter_filter'

        @classmethod
        def process_value(cls, parameter_filter):
            if isinstance(parameter_filter, ParameterFilter):
                return parameter_filter.unwrap()
            else:
                raise Exception('Shouldnt get here')

    class WhereFilter(SuperSlowFilter):
        """
        Requires Unpacking of each Element. As per the API design,
        this filter must be combined.

        By default, function will test against wrapped elements for easier
        parameter access

        >>> Collector(of_class='FamilyInstance', where=lambda x: 'Desk' in x.name)
        >>> Collector(of_class='Wall', where=lambda x: 'Desk' in x.parameters['Length
↪'] > 5.0)
        """
        keyword = 'where'

        @classmethod
        def apply(cls, doc, collector, func):
            excluded_elements = set()
            for element in collector:
                wrapped_element = Element(element)
                if not func(wrapped_element):
                    excluded_elements.add(element.Id)
            excluded_elements = List[DB.ElementId](excluded_elements)
            if excluded_elements:
                return collector.Excluding(excluded_elements)
            else:
                return collector

    class ExclusionFilter(QuickFilter):
        keyword = 'exclude'

        @classmethod
        def process_value(cls, element_references):
            element_set = ElementSet(element_references)
            return DB.ExclusionFilter(element_set.as_element_id_list)

    class InteresectFilter(LogicalFilter):
        keyword = 'and_collector'

        @classmethod
        def process_value(cls, collector):
            if hasattr(collector, 'unwrap'):
```

(continues on next page)

(continued from previous page)

```
                collector = collector.unwrap()
            return collector

        @classmethod
        def apply(cls, doc, collector, value):
            new_collector = cls.process_value(value)
            return collector.IntersectWith(new_collector)

    class UnionFilter(InteresectFilter):
        keyword = 'or_collector'

        @classmethod
        def apply(cls, doc, collector, value):
            new_collector = cls.process_value(value)
            return collector.UnionWith(new_collector)


class Collector(BaseObjectWrapper):
    """
    Revit FilteredElement Collector Wrapper

    Usage:
        >>> collector = Collector(of_class='View')
        >>> elements = collector.get_elements()

        Multiple Filters:

        >>> Collector(of_class='Wall', is_not_type=True)
        >>> Collector(of_class='ViewSheet', is_not_type=True)
        >>> Collector(of_category='OST_Rooms', level=some_level)
        >>> Collector(symbol=SomeSymbol)
        >>> Collector(owner_view=SomeView)
        >>> Collector(owner_view=None)
        >>> Collector(parameter_filter=parameter_filter)

        Use Enumeration member or its name as a string:

        >>> Collector(of_category='OST_Walls')
        >>> Collector(of_category=DB.BuiltInCategory.OST_Walls)
        >>> Collector(of_class=DB.ViewType)
        >>> Collector(of_class='ViewType')

        Search Document, View, or list of elements

        >>> Collector(of_category='OST_Walls') # doc is default
        >>> Collector(view=SomeView, of_category='OST_Walls') # Doc is default
        >>> Collector(doc=SomeLinkedDoc, of_category='OST_Walls')
        >>> Collector(elements=[Element1, Element2,...], of_category='OST_Walls')
        >>> Collector(owner_view=SomeView)
        >>> Collector(owner_view=None)

    Attributes:
        collector.get_elements(): Returns list of all `collected` elements
        collector.get_first(): Returns first found element, or ``None``
        collector.get_elements(): Returns list with all elements wrapped.
                                        Elements will be instantiated using :any:`Element`
```

(continues on next page)

(continued from previous page)

```
    Wrapped Element:
        self._revit_object = ``Revit.DB.FilteredElementCollector``

    """

    _revit_object_class = DB.FilteredElementCollector

    def __init__(self, **filters):
        """
        Args:
            **filters (``keyword args``): Scope and filters

        Returns:
            Collector (:any:`Collector`): Collector Instance

        Scope Options:
            * ``view`` `(DB.View)`: View Scope (Optional)
            * ``element_ids`` `([ElementId])`: List of Element Ids to limit Collector␣
↪Scope
            * ``elements`` `([Element])`: List of Elements to limit Collector Scope

        Warning:
            Only one scope filter should be used per query. If more then one is used,
            only one will be applied, in this order ``view`` > ``elements`` >␣
↪``element_ids``

        Filter Options:
            * is_type (``bool``): Same as ``WhereElementIsElementType``
            * is_not_type (``bool``): Same as ``WhereElementIsNotElementType``
            * of_class (``Type``): Same as ``OfClass``. Type can be ``DB.SomeType``␣
↪or string: ``DB.Wall`` or ``'Wall'``
            * of_category (``BuiltInCategory``): Same as ``OfCategory``. Can be ``DB.
↪BuiltInCategory.OST_Wall`` or ``'Wall'``
            * owner_view (``DB.ElementId, View`):␣
↪``WhereElementIsViewIndependent(True)``
            * is_view_independent (``bool``): ``WhereElementIsViewIndependent(True)``
            * family (``DB.ElementId``, ``DB.Element``): Element or ElementId of␣
↪Family
            * symbol (``DB.ElementId``, ``DB.Element``): Element or ElementId of␣
↪Symbol
            * level (``DB.Level``, ``DB.ElementId``, ``Level Name``): Level,␣
↪ElementId of Level, or Level Name
            * not_level (``DB.Level``, ``DB.ElementId``, ``Level Name``): Level,␣
↪ElementId of Level, or Level Name
            * parameter_filter (:any:`ParameterFilter`): Applies␣
↪``ElementParameterFilter``
            * exclude (`element_references`): Element(s) or ElementId(s) to exlude␣
↪from result
            * and_collector (``collector``): Collector to intersect with. Elements␣
↪must be present in both
            * or_collector (``collector``): Collector to Union with. Elements must be␣
↪present on of the two.
            * where (`function`): function to test your elements against

        """
        # Define Filtered Element Collector Scope + Doc
        collector_doc = filters.pop('doc') if 'doc' in filters else revit.doc
```

(continues on next page)

(continued from previous page)

```
        if 'view' in filters:
            view = filters.pop('view')
            view_id = view if isinstance(view, DB.ElementId) else view.Id
            collector = DB.FilteredElementCollector(collector_doc, view_id)
        elif 'elements' in filters:
            elements = filters.pop('elements')
            element_ids = to_element_ids(elements)
            collector = DB.FilteredElementCollector(collector_doc, List[DB.
↪ElementId](element_ids))
        elif 'element_ids' in filters:
            element_ids = filters.pop('element_ids')
            collector = DB.FilteredElementCollector(collector_doc, List[DB.
↪ElementId](element_ids))
        else:
            collector = DB.FilteredElementCollector(collector_doc)

        super(Collector, self).__init__(collector)

        for key in filters.keys():
            if key not in [f.keyword for f in FilterClasses.get_sorted()]:
                raise RpwException('Filter not valid: {}'.format(key))

        self._collector = self._collect(collector_doc, collector, filters)

    def _collect(self, doc, collector, filters):
        """
        Main Internal Recursive Collector Function.

        Args:
            doc (`UI.UIDocument`): Document for the collector.
            collector (`FilteredElementCollector`): FilteredElementCollector
            filters (`dict`): Filters - {'doc': revit.doc, 'of_class': 'Wall'}

        Returns:
            collector (`FilteredElementCollector`): FilteredElementCollector
        """
        for filter_class in FilterClasses.get_sorted():
            if filter_class.keyword not in filters:
                continue
            filter_value = filters.pop(filter_class.keyword)
            logger.debug('Applying Filter: {}:{}'.format(filter_class, filter_value))
            new_collector = filter_class.apply(doc, collector, filter_value)
            return self._collect(doc, new_collector, filters)
        return collector

    def __iter__(self):
        """ Uses iterator to reduce unecessary memory usage """
        # TODO: Depracate or Make return Wrapped ?
        for element in self._collector:
            yield element

    def get_elements(self, wrapped=True):
        """
        Returns list with all elements instantiated using :any:`Element`
        """
        if wrapped:
```

(continues on next page)

(continued from previous page)

```
            return [Element(el) for el in self.__iter__()]
        else:
            return [element for element in self.__iter__()]

    @property
    def elements(self):
        """ Returns list with all elements """
        deprecate_warning('Collector.elements',
                          'Collector.get_elements(wrapped=True)')
        return self.get_elements(wrapped=False)

    @property
    def wrapped_elements(self):
        """ Returns list with all elements instantiated using :any:`Element`"""
        deprecate_warning('Collector.wrapped_elements',
                          'Collector.get_elements(wrapped=True)')
        return self.get_elements(wrapped=True)

    def select(self):
        """ Selects Collector Elements on the UI """
        Selection(self.element_ids)

    def get_first(self, wrapped=True):
        """
        Returns first element or `None`

        Returns:
            Element (`DB.Element`, `None`): First element or None
        """
        try:
            element = self[0]
            return Element(element) if wrapped else element
        except IndexError:
            return None


    # @property
    # def get_first(self):
    #     deprecate_warning('Collector.first', 'Collector.get_first()')
    #     return self.get_first(wrapped=False)

    def get_element_ids(self):
        """
        Returns list with all elements instantiated using :any:`Element`
        """
        return [element_id for element_id in self._collector.ToElementIds()]

    @property
    def element_ids(self):
        deprecate_warning('Collector.element_ids',
                          'Collector.get_element_ids()')
        return self.get_element_ids()

    def __getitem__(self, index):
        # TODO: Depracate or Make return Wrapped ?
        for n, element in enumerate(self.__iter__()):
            if n == index:
```

(continues on next page)

(continued from previous page)

```
                return element
        else:
            raise IndexError('Index {} not in collector {}'.format(index,
                                                                   self))

    def __bool__(self):
        """ Evaluates to `True` if Collector.elements is not empty [] """
        return bool(self.get_elements(wrapped=False))

    def __len__(self):
        """ Returns length of collector.get_elements() """
        try:
            return self._collector.GetElementCount()
        except AttributeError:
            return len(self.get_elements(wrapped=False))  # Revit 2015

    def __repr__(self):
        return super(Collector, self).__repr__(data={'count': len(self)})


class ParameterFilter(BaseObjectWrapper):
    """
    Parameter Filter Wrapper.
    Used to build a parameter filter to be used with the Collector.

    Usage:
        >>> param_id = DB.ElementId(DB.BuiltInParameter.TYPE_NAME)
        >>> parameter_filter = ParameterFilter(param_id, equals='Wall 1')
        >>> collector = Collector(parameter_filter=parameter_filter)

    Returns:
        FilterRule: A filter rule object, depending on arguments.
    """
    _revit_object_class = DB.ElementParameterFilter

    RULES = {
            'equals': 'CreateEqualsRule',
            'not_equals': 'CreateEqualsRule',
            'contains': 'CreateContainsRule',
            'not_contains': 'CreateContainsRule',
            'begins': 'CreateBeginsWithRule',
            'not_begins': 'CreateBeginsWithRule',
            'ends': 'CreateEndsWithRule',
            'not_ends': 'CreateEndsWithRule',
            'greater': 'CreateGreaterRule',
            'not_greater': 'CreateGreaterRule',
            'greater_equal': 'CreateGreaterOrEqualRule',
            'not_greater_equal': 'CreateGreaterOrEqualRule',
            'less': 'CreateLessRule',
            'not_less': 'CreateLessRule',
            'less_equal': 'CreateLessOrEqualRule',
            'not_less_equal': 'CreateLessOrEqualRule',
           }

    CASE_SENSITIVE = True                 # Override with case_sensitive=False
    FLOAT_PRECISION = 0.0013020833333333  # 1/64" in ft:(1/64" = 0.015625)/12
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, parameter_reference, **conditions):
        """
        Creates Parameter Filter Rule

        >>> param_rule = ParameterFilter(param_id, equals=2)
        >>> param_rule = ParameterFilter(param_id, not_equals='a', case_
↪sensitive=True)
        >>> param_rule = ParameterFilter(param_id, not_equals=3, reverse=True)

        Args:
            param_id(DB.ElementID): ElementId of parameter
            **conditions: Filter Rule Conditions and options.

            conditions:
                | ``begins``, ``not_begins``
                | ``contains``, ``not_contains``
                | ``ends``, ``not_ends``
                | ``equals``, ``not_equals``
                | ``less``, ``not_less``
                | ``less_equal``, ``not_less_equal``
                | ``greater``, ``not_greater``
                | ``greater_equal``, ``not_greater_equal``

            options:
                | ``case_sensitive``: Enforces case sensitive, String only
                | ``reverse``: Reverses result of Collector

        """
        parameter_id = self.coerce_param_reference(parameter_reference)
        reverse = conditions.get('reverse', False)
        case_sensitive = conditions.get('case_sensitive', ParameterFilter.CASE_
↪SENSITIVE)
        precision = conditions.get('precision', ParameterFilter.FLOAT_PRECISION)

        for condition in conditions.keys():
            if condition not in ParameterFilter.RULES:
                raise RpwException('Rule not valid: {}'.format(condition))

        rules = []
        for condition_name, condition_value in conditions.iteritems():

            # Returns on of the CreateRule factory method names above
            rule_factory_name = ParameterFilter.RULES.get(condition_name)
            filter_value_rule = getattr(DB.ParameterFilterRuleFactory,
                                        rule_factory_name)

            args = [condition_value]

            if isinstance(condition_value, str):
                args.append(case_sensitive)

            if isinstance(condition_value, float):
                args.append(precision)

            filter_rule = filter_value_rule(parameter_id, *args)
            if 'not_' in condition_name:
                filter_rule = DB.FilterInverseRule(filter_rule)
```

(continues on next page)

(continued from previous page)

```
            logger.debug('ParamFilter Conditions: {}'.format(conditions))
            logger.debug('Case sensitive: {}'.format(case_sensitive))
            logger.debug('Reverse: {}'.format(reverse))
            logger.debug('ARGS: {}'.format(args))
            logger.debug(filter_rule)
            logger.debug(str(dir(filter_rule)))

            rules.append(filter_rule)
        if not rules:
            raise RpwException('malformed filter rule: {}'.format(conditions))

        _revit_object = DB.ElementParameterFilter(List[DB.FilterRule](rules),
                                                  reverse)
        super(ParameterFilter, self).__init__(_revit_object)
        self.conditions = conditions

    def coerce_param_reference(self, parameter_reference):
        if isinstance(parameter_reference, str):
            param_id = BipEnum.get_id(parameter_reference)
        elif isinstance(parameter_reference, DB.ElementId):
            param_id = parameter_reference
        else:
            RpwCoerceError(parameter_reference, ElementId)
        return param_id

    @staticmethod
    def from_element_and_parameter(element, param_name, **conditions):
        """
        Alternative constructor to built Parameter Filter from Element +
        Parameter Name instead of parameter Id

        >>> parameter_filter = ParameterFilter.from_element(element,param_name, less_
↪than=10)
        >>> Collector(parameter_filter=parameter_filter)
        """
        parameter = element.LookupParameter(param_name)
        param_id = parameter.Id
        return ParameterFilter(param_id, **conditions)

    def __repr__(self):
        return super(ParameterFilter, self).__repr__(data=self.conditions)
```

## 4.3.8 Collections

API Related Sets and Collections

### ElementSet

**class** rpw.db.**ElementSet**(*elements_or_ids=None*, *doc=Document*)

Bases: `rpw.base.BaseObject`

Provides helpful methods for managing a set of unique of `DB.ElementId` Sets are unordered.

```
>>> element_set = ElementSet([element, element])
>>> element_set = ElementSet()
>>> element_set.add(SomeElement)
>>> SomeElement in element_set
True
>>> element_set.clear()
```

**Note:** Similar to DB.ElementSet, doesnt wrap since there is no advantage

**Parameters** **`DB.ElementID, optional)`** (*`(DB.Element,`*) – Elements or Element Ids.

**`__init__`**(*elements_or_ids=None*, *doc=Document*)
x.__init__(...) initializes x; see help(type(x)) for signature

**`add`**(*elements_or_ids*)
Adds elements or element_ids to set. Handles single or list

**Parameters** **`element_reference`** (*DB.Element*, DB.Element_ids) – Iterable Optional

**`as_element_list`**
*Returns* – IList<DB.Element>

**`clear`**()
Clears Set

**`get_element_ids`**(*as_list=True*)
ElementId of Elements in ElementSet

**Parameters** **`as_list`** (*`bool`*) – True if you want list as List[DB.ElementId], False for regular python list. Default is True

**Returns** List of ElementIds Objects

**Return type** ElementIds (List, List[DB.ElementId])

**`get_elements`**(*wrapped=True*, *as_list=False*)
Elements in ElementSet

**Parameters**

- **`wrapped`** (*`bool`*) – True for wrapped Elements. Default is True.
- **`as_list`** (*`bool`*) – True if you want list as List[DB.Element], False for regular python list. Default is False. If `as_list` is True, `wrapped` will be set to False.

**Returns** Elements stored in ElementSet

**Return type** Elements (`List`)

**`pop`**(*element_reference*, *wrapped=True*)
Removed from set using ElementIds

**Parameters** **`element_reference`** (*`DB.ElementId, DB.Element`*) –

**Returns** (DB.Element, db.Element)

**`select`**()
Selects Set in UI

### ElementCollection

**class** rpw.db.**ElementCollection**(*elements_or_ids=None*, *doc=Document*)

Bases: `rpw.base.BaseObject`

List Collection for managing a list of `DB.Element`.

```
>>> element_set = ElementCollection([element, element])
>>> element_set = ElementCollection()
>>> element_set.add(SomeElement)
>>> SomeElement in element_set
True
>>> element_set.clear()
```

**Parameters** **DB.ElementID, optional)** (*(DB.Element,)*) – Elements or Element Ids.

**__init__**(*elements_or_ids=None*, *doc=Document*)
x.__init__(…) initializes x; see help(type(x)) for signature

**append**(*elements_or_ids*)
Adds elements or element_ids to set. Handles single or list

**as_element_id_list**
*Returns* – IList<DB.Element>

**clear**()
Clears Set

**element_ids**
*Returns* – ElementIds (`List`) – List of ElementIds Objects

**get_element_ids**(*as_list=True*)
ElementId of Elements in ElementCollection

**Parameters** **as_list** (*bool*) – True if you want list as List[DB.ElementId], False for regular python list. Default is True

**Returns** List of ElementIds Objects

**Return type** ElementIds (List, List[DB.ElementId])

**get_elements**(*wrapped=True*, *as_list=False*)
List of Elements in Collection

**Parameters**

- **wrapped** (*bool*) – True for wrapped Elements. Default is True.
- **as_list** (*bool*) – True if you want list as List[DB.ElementId], False for regular python list. Default is True

**Returns** List of Elements Objects or List[DB.Element]

**Return type** Elements (`List`)

**get_first**(*wrapped=True*)
Get First Item in Collection

**Parameters** **wrapped** (*bool*) – True for wrapped. Default is True.

**Returns** First Element, or None if empty.

**Return type** (*db.Element*, DB.Element)

**pop** (*index=0*, *wrapped=True*)
Removed from set using ElementIds

**Parameters** **index** (int) – Index of Element [Default: 0]

**select** ()
Selects Set in UI

## XYZCollection

**class** rpw.db.**XyzCollection** (*points*)
Bases: rpw.base.BaseObject

Provides helpful methods for managing a collection(list) of *XYZ* instances.

```
>>> points = [p1,p2,p3,p4, ...]
>>> point_collection = XyzCollection(points)
```

point_collection.**average**

point_collection.**min**

point_collection.**max**

**__init__** (*points*)
x.__init__(...) initializes x; see help(type(x)) for signature

**average**

```
>>> points = [XYZ(0,0,0), XYZ(4,4,2)]
>>> points.average
(2,2,1)
```

**Returns** Average of point collection.

**Return type** XYZ (*DB.XYZ*)

**max**

```
>>> points = [(0,0,5), (2,2,2)]
>>> points.max
(2,2,5)
```

**Returns** Max of point collection.

**Return type** XYZ (*DB.XYZ*)

**min**

```
>>> points = [(0,0,5), (2,2,2)]
>>> points.min = (0,0,2)
```

**Returns** Min of point collection.

**Return type** XYZ (*DB.XYZ*)

**sorted_by** (*x_y_z*)
Sorts Point Collection by axis.

```
>>> pts = XyzCollection(XYZ(0,10,0), XYZ(0,0,0))
>>> pts.sorted_by('y')
[XYZ(0,0,0), XYZ(0,10,0)]
```

**Parameters** **axis** (*str*) – Axist to sort by.

## Implementation

```
""" API Related Sets and Collections """


from collections import OrderedDict

import rpw
from rpw import revit, DB
from rpw.db.xyz import XYZ
from rpw.db.element import Element
from rpw.base import BaseObject
from rpw.utils.coerce import to_elements, to_element_ids, to_element_id
from rpw.utils.dotnet import List
from rpw.utils.logger import deprecate_warning


class ElementSet(BaseObject):
    """
    Provides helpful methods for managing a set of unique of ``DB.ElementId``
    Sets are unordered.

    >>> element_set = ElementSet([element, element])
    >>> element_set = ElementSet()
    >>> element_set.add(SomeElement)
    >>> SomeElement in element_set
    True
    >>> element_set.clear()

    NOTE:
        Similar to DB.ElementSet, doesnt wrap since there is no advantage

    Args:
        (`DB.Element`, `DB.ElementID`, optional): Elements or Element Ids.

    """

    def __init__(self, elements_or_ids=None, doc=revit.doc):
        self.doc = doc
        self._element_id_set = []
        if elements_or_ids:
            self.add(elements_or_ids)

    def add(self, elements_or_ids):
```

(continues on next page)

(continued from previous page)

```
        """
        Adds elements or element_ids to set. Handles single or list

        Args:
            element_reference (`DB.Element`, DB.Element_ids): Iterable Optional

        """
        element_ids = to_element_ids(elements_or_ids)
        for id_ in element_ids:
            if id_ not in self._element_id_set:
                self._element_id_set.append(id_)

    def pop(self, element_reference, wrapped=True):
        """
        Removed from set using ElementIds

        Args:
            element_reference (DB.ElementId, DB.Element)

        Returns:
            (DB.Element, db.Element)

        """
        element_id = to_element_id(element_reference)
        element = self.__getitem__(element_id)
        self._element_id_set.remove(element_id)
        return element if wrapped else element.unwrap()

    def clear(self):
        """ Clears Set """
        self._element_id_set = []

    @property
    def _elements(self):
        return [self.doc.GetElement(e) for e in self._element_id_set]

    @property
    def _wrapped_elements(self):
        return Element.from_list(self._element_id_set)

    def get_elements(self, wrapped=True, as_list=False):
        """
        Elements in ElementSet

        Args:
            wrapped(bool): True for wrapped Elements. Default is True.
            as_list(bool): True if you want list as List[DB.Element], False
                for regular python list. Default is False.
                If ``as_list`` is True, ``wrapped`` will be set to False.

        Returns:
            Elements (``List``): Elements stored in ElementSet
        """
        if as_list or not wrapped:
            elements = self._elements
            return List[DB.Element](elements) if as_list else elements
        else:
```

(continues on next page)

(continued from previous page)

```
            return self._wrapped_elements

    @property
    def wrapped_elements(self):
        deprecate_warning('ElementSet.wrapped_elements',
                          'ElementSet.get_elements(wrapped=True)')
        return self.get_elements(wrapped=True)

    @property
    def elements(self):
        deprecate_warning('ElementSet.wrapped_elements',
                          'ElementSet.get_elements(wrapped=False)')
        return self.get_elements(wrapped=False)

    @property
    def as_element_list(self):
        """
        Returns:
            IList<DB.Element>
        """
        return self.get_element_ids(as_list=True)

    def get_element_ids(self, as_list=True):
        """
        ElementId of Elements in ElementSet

        Args:
            as_list(bool): True if you want list as List[DB.ElementId], False
                for regular python list. Default is True

        Returns:
            ElementIds (List, List[DB.ElementId]): List of ElementIds Objects

        """
        if as_list:
            return List[DB.ElementId](self._element_id_set)
        else:
            return list(self._element_id_set)

    @property
    def element_ids(self):
        deprecate_warning('ElementSet.element_ids',
                          'ElementSet.get_element_ids(as_list=False)')
        return self.get_element_ids(as_list=False)

    @property
    def as_element_id_list(self):
        deprecate_warning('ElementSet.as_element_id_list',
                          'ElementSet.get_element_ids(as_list=True)')
        return self.get_element_ids(as_list=True)

    def select(self):
        """ Selects Set in UI """
        return rpw.ui.Selection(self._element_id_set)

    def __len__(self):
        return len(self._element_id_set)
```

(continues on next page)

(continued from previous page)

```
    def __iter__(self):
        """ Iterator: Wrapped """
        for element in self._element_id_set:
            yield Element.from_id(element)

    def __getitem__(self, element_reference):
        """
        Get Element from set from an element ElementId

        Args:
            element_reference (DB.Element, DB.ElementID)

        Returns:
            (wrapped_element): Wrapped Element. Raises Key Error if not found.
        """
        eid_key = to_element_id(element_reference)
        for element in self.__iter__():
            if element.Id == eid_key:
                return element
        raise KeyError(eid_key)

    def __contains__(self, element_or_id):
        """
        Checks if selection contains the element Reference.

        Args:
            Reference: Element, ElementId, or Integer

        Returns:
            bool: ``True`` or ``False``
        """
        # TODO Write Tests
        element_id = to_element_id(element_or_id)
        return bool(element_id in self._element_id_set)

    def __bool__(self):
        return bool(self._element_id_set)

    def __repr__(self, data=None):
        return super(ElementSet, self).__repr__(data={'count': len(self)})


class ElementCollection(BaseObject):
    """
    List Collection for managing a list of ``DB.Element``.

    >>> element_set = ElementCollection([element, element])
    >>> element_set = ElementCollection()
    >>> element_set.add(SomeElement)
    >>> SomeElement in element_set
    True
    >>> element_set.clear()

    Args:
        (`DB.Element`, `DB.ElementID`, optional): Elements or Element Ids.
    """
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, elements_or_ids=None, doc=revit.doc):
        self.doc = doc
        self._elements = []
        if elements_or_ids:
            self.append(elements_or_ids)

    def append(self, elements_or_ids):
        """ Adds elements or element_ids to set. Handles single or list """
        elements = to_elements(elements_or_ids)
        for element in elements:
            self._elements.append(element)

    def clear(self):
        """ Clears Set """
        self._elements = []

    def pop(self, index=0, wrapped=True):
        """
        Removed from set using ElementIds

        Args:
            index (``int``): Index of Element [Default: 0]
        """
        element = self._elements.pop(index)
        return Element(element) if wrapped else element

    @property
    def _element_ids(self):
        return [e.Id for e in self._elements]

    @property
    def _wrapped_elements(self):
        return Element.from_list(self._elements)

    def get_elements(self, wrapped=True, as_list=False):
        """
        List of Elements in Collection

        Args:
            wrapped(bool): True for wrapped Elements. Default is True.
            as_list(bool): True if you want list as List[DB.ElementId], False
                for regular python list. Default is True

        Returns:
            Elements (``List``): List of Elements Objects or List[DB.Element]
        """
        elements = self._elements

        if as_list or not wrapped:
            elements = self._elements
            return List[DB.Element](elements) if as_list else elements
        else:
            return self._wrapped_elements

    @property
    def elements(self):
        deprecate_warning('ElementCollection.elements',
```

(continues on next page)

(continued from previous page)

```
                          'ElementCollection.get_elements()')
        return self.get_elements(wrapped=True)

    @property
    def as_element_list(self):
        return self.get_elements(as_list=True)

    def select(self):
        """ Selects Set in UI """
        return rpw.ui.Selection(self._elements)

    def get_element_ids(self, as_list=True):
        """
        ElementId of Elements in ElementCollection

        Args:
            as_list(bool): True if you want list as List[DB.ElementId], False
                for regular python list. Default is True

        Returns:
            ElementIds (List, List[DB.ElementId]): List of ElementIds Objects

        """
        if as_list:
            return List[DB.ElementId](self._element_ids)
        else:
            return self._element_ids

    @property
    def element_ids(self):
        """
        Returns:
            ElementIds (``List``): List of ElementIds Objects
        """
        deprecate_warning('ElementCollection.element_ids',
                          'ElementCollection.get_element_ids()')
        return self.get_element_ids(as_list=False)

    @property
    def as_element_id_list(self):
        """
        Returns:
            IList<DB.Element>
        """
        deprecate_warning('ElementCollection.as_element_id_list',
                          'ElementCollection.get_element_ids()')
        return self.get_element_ids(as_list=True)

    def get_first(self, wrapped=True):
        """ Get First Item in Collection

        Args:
            wrapped (bool): True for wrapped. Default is True.

        Returns:
            (db.Element, DB.Element): First Element, or None if empty.
        """
```

(continues on next page)

(continued from previous page)

```
        try:
            element = self._elements[0]
        except IndexError:
            return None
        else:
            return Element(element) if wrapped else element

    def __iter__(self):
        """ Iterator: Wrapped """
        for wrapped_element in self._wrapped_elements:
            yield wrapped_element

    def __len__(self):
        return len(self._elements)

    def __getitem__(self, index):
        """ Getter: Wrapped """
        for n, wrapped_element in enumerate(self.__iter__()):
            if n == index:
                return wrapped_element
        raise IndexError(index)

    def __contains__(self, element_or_id):
        """
        Checks if selection contains the element Reference.

        Args:
            Reference: Element, ElementId, or Integer

        Returns:
            bool: ``True`` or ``False``
        """
        element_id = to_element_id(element_or_id)
        return bool(element_id in self.get_element_ids(as_list=False))

    def __bool__(self):
        return bool(self._elements)

    def __repr__(self, data=None):
        return super(ElementCollection, self).__repr__(
                                            data={'count': len(self)})


class XyzCollection(BaseObject):
    """
    Provides helpful methods for managing a
    collection(list) of :any:`XYZ` instances.

    >>> points = [p1,p2,p3,p4, ...]
    >>> point_collection = XyzCollection(points)

    Attributes:
        point_collection.average
        point_collection.min
        point_collection.max
    """
    # TODO: Implement unwrapped return.
```

(continues on next page)

(continued from previous page)

```
    # TODO: Implement Collection methods (Add, pop, as list, etc)

    def __init__(self, points):
        self.points = points if points is not None else []

    def __iter__(self):
        for point in self.points:
            yield point

    @property
    def average(self):
        """
        >>> points = [XYZ(0,0,0), XYZ(4,4,2)]
        >>> points.average
        (2,2,1)

        Returns:
            XYZ (`DB.XYZ`): Average of point collection.

        """
        x_values = [point.X for point in self.points]
        y_values = [point.Y for point in self.points]
        z_values = [point.Z for point in self.points]
        x_avg = sum(x_values) / len(x_values)
        y_avg = sum(y_values) / len(y_values)
        z_avg = sum(z_values) / len(z_values)

        return XYZ(x_avg, y_avg, z_avg)

    @property
    def max(self):
        """
        >>> points = [(0,0,5), (2,2,2)]
        >>> points.max
        (2,2,5)

        Returns:
            XYZ (`DB.XYZ`): Max of point collection.

        """
        x_values = [point.X for point in self.points]
        y_values = [point.Y for point in self.points]
        z_values = [point.Z for point in self.points]
        x_max = max(x_values)
        y_max = max(y_values)
        z_max = max(z_values)
        return XYZ(x_max, y_max, z_max)

    @property
    def min(self):
        """
        >>> points = [(0,0,5), (2,2,2)]
        >>> points.min = (0,0,2)

        Returns:
            XYZ (`DB.XYZ`): Min of point collection.
```

(continues on next page)

(continued from previous page)

```
        """
        x_values = [point.X for point in self.points]
        y_values = [point.Y for point in self.points]
        z_values = [point.Z for point in self.points]
        x_min = min(x_values)
        y_min = min(y_values)
        z_min = min(z_values)
        return XYZ(x_min, y_min, z_min)

    def sorted_by(self, x_y_z):
        """ Sorts Point Collection by axis.

        >>> pts = XyzCollection(XYZ(0,10,0), XYZ(0,0,0))
        >>> pts.sorted_by('y')
        [XYZ(0,0,0), XYZ(0,10,0)]

        Args:
            axis (`str`): Axist to sort by.
        """
        sorted_points = self.points[:]
        sorted_points.sort(key=lambda p: getattr(p, x_y_z.upper()))
        return sorted_points

    def __len__(self):
        return len(self.points)

    def __repr__(self):
        return super(PointCollection, self).__repr__(data=len(self))
```

### 4.3.9 BuiltIns

```
>>> BipEnum.get('WALL_LOCATION_LINE')
Revit.DB.BuiltInParameter.WALL_LOCATION_LINE
>>> BipEnum.get_id('WALL_LOCATION_LINE')
Revit.DB.ElementId
```

**Note:** These classes were created to be used internally, but are documented here as some its functionalities could be helpful to others.

**class** rpw.db.builtins.**_BiCategory**

Bases: *rpw.base.BaseObjectWrapper*

Enumeration Wrapper

```
>>> BiCategory.get('OST_Rooms')
Revit.DB.BuiltInCategory.OST_Rooms
>>> BiCategory.get_id('OST_Rooms')
Revit.DB.ElementId
>>> BiCategory.from_category_id(furniture.Category.Id)
DB.BuiltInCategory.OST_Furniture
```

**`__init__`**()

Child classes can use self._revit_object to refer back to Revit Element

> **Warning:** Any Wrapper that inherits and overrides __init__ class MUST ensure `_revit_object` is created by calling super().__init__ before setting any self attributes. Not doing so will cause recursion errors and Revit will crash. BaseObjectWrapper should define a class variable _revit_object_class to define the object class being wrapped.

**`from_category_id`**(*category_id*)

Casts `DB.BuiltInCategory` Enumeration member from a Category ElementId

**Parameters** **`category_id`** (`DB.ElementId`) – ElementId reference of a category

**Returns** `DB.BuiltInCategory` member

**`fuzzy_get`**(*loose_category_name*)

Gets Built In Category by Fuzzy Name. Similar to get() but ignores case, and does not require **OST_** prefix.

```
>>> BiCategory.fuzzy_get('OST_Rooms')
< BuiltInCategory >
>>> BiCategory.fuzzy_get('Rooms')
< BuiltInCategory >
>>> BiCategory.fuzzy_get('rooms')
< BuiltInCategory >
```

**Parameters** **`str`** – Name of Category

**Returns** BuiltInCategory Enumeration Member

**Return type** `DB.BuiltInCategory`

**`get`**(*category_name*)

Gets Built In Category by Name

**Parameters** **`str`** – Name of Category

**Returns** BuiltInCategory Enumeration Member

**Return type** `DB.BuiltInCategory`

**`get_id`**(*category_name*)

Gets ElementId of Category by name

**Parameters** **`str`** – Name of Category

**Returns** BuiltInCategory Enumeration Member

**Return type** `DB.BuiltInCategory`

**class** `rpw.db.builtins.`**`_BiParameter`**

Bases: *`rpw.base.BaseObjectWrapper`*

BuiltInParameter Wrapper

```
>>> BiParameter.get('WALL_LOCATION_LINE')
Revit.DB.BuiltInParameter.WALL_LOCATION_LINE
>>> BiParameter.get_id('WALL_LOCATION_LINE')
Revit.DB.ElementId
```

**__getattr__**(*attr*)
Getter for original methods and properties or the element. This method is only called if the attribute name does not already exists.

**__init__**()
Child classes can use self._revit_object to refer back to Revit Element

> **Warning:** Any Wrapper that inherits and overrides __init__ class MUST ensure `_revit_object` is created by calling super().__init__ before setting any self attributes. Not doing so will cause recursion errors and Revit will crash. BaseObjectWrapper should define a class variable _revit_object_class to define the object class being wrapped.

**get**(*parameter_name*)
Gets Built In Parameter by Name

**Parameters** **str** – Name of Parameter

**Returns** BuiltInParameter Enumeration Member

**Return type** `DB.BuiltInParameter`

**get_id**(*parameter_name*)
Gets ElementId of Category by name

**Parameters** **parameter_name** (`str`) – Name of Built In Parameter

**Returns** BuiltInParameter Enumeration Member

**Return type** `DB.BuitInParameter`

## Implementation

```
import re
from rpw import revit, DB
from rpw.base import BaseObject, BaseObjectWrapper
from rpw.utils.dotnet import Enum
from rpw.exceptions import RpwCoerceError


class _BiParameter(BaseObjectWrapper):
    """
    BuiltInParameter Wrapper

    >>> BiParameter.get('WALL_LOCATION_LINE')
    Revit.DB.BuiltInParameter.WALL_LOCATION_LINE
    >>> BiParameter.get_id('WALL_LOCATION_LINE')
    Revit.DB.ElementId

    """

    _revit_object_class = DB.BuiltInParameter

    def __init__(self):
        super(_BiParameter, self).__init__(DB.BuiltInParameter,
```

(continues on next page)

(continued from previous page)

```
                                                    enforce_type=False)

    def __getattr__(self, attr):
        return self.get(attr)

    def get(self, parameter_name):
        """ Gets Built In Parameter by Name

        Args:
            ``str``: Name of Parameter

        Returns:
            ``DB.BuiltInParameter``: BuiltInParameter Enumeration Member

        """
        try:
            enum = getattr(DB.BuiltInParameter, parameter_name)
        except AttributeError:
            raise RpwCoerceError(parameter_name, DB.BuiltInParameter)
        return enum

    def get_id(self, parameter_name):
        """
        Gets ElementId of Category by name

        Args:
            parameter_name(``str``): Name of Built In Parameter

        Returns:
            ``DB.BuitInParameter``: BuiltInParameter Enumeration Member
        """
        enum = self.get(parameter_name)
        return DB.ElementId(enum)

    def __repr__(self):
        return super(_BiParameter, self).__repr__(to_string='Autodesk.Revit.DB.
↪BuiltInParameter')


class _BiCategory(BaseObjectWrapper):
    """
    Enumeration Wrapper

    >>> BiCategory.get('OST_Rooms')
    Revit.DB.BuiltInCategory.OST_Rooms
    >>> BiCategory.get_id('OST_Rooms')
    Revit.DB.ElementId
    >>> BiCategory.from_category_id(furniture.Category.Id)
    DB.BuiltInCategory.OST_Furniture
    """

    _revit_object_class = DB.BuiltInCategory

    def __init__(self):
        super(_BiCategory, self).__init__(DB.BuiltInCategory,
                                          enforce_type=False)
```

(continues on next page)

(continued from previous page)

```
    def get(self, category_name):
        """ Gets Built In Category by Name

        Args:
            ``str``: Name of Category

        Returns:
            ``DB.BuiltInCategory``: BuiltInCategory Enumeration Member
        """

        try:
            enum = getattr(DB.BuiltInCategory, category_name)
        except AttributeError:
            raise RpwCoerceError(category_name, DB.BuiltInCategory)
        return enum

    def fuzzy_get(self, loose_category_name):
        """ Gets Built In Category by Fuzzy Name.
        Similar to get() but ignores case, and does not require OST_ prefix.

        >>> BiCategory.fuzzy_get('OST_Rooms')
        < BuiltInCategory >
        >>> BiCategory.fuzzy_get('Rooms')
        < BuiltInCategory >
        >>> BiCategory.fuzzy_get('rooms')
        < BuiltInCategory >

        Args:
            ``str``: Name of Category

        Returns:
            ``DB.BuiltInCategory``: BuiltInCategory Enumeration Member
        """
        loose_category_name = loose_category_name.replace(' ', '').lower()
        loose_category_name = loose_category_name.replace('ost_', '')
        for category_name in dir(DB.BuiltInCategory):
            exp = '(OST_)({})$'.format(loose_category_name)
            if re.search(exp, category_name, re.IGNORECASE):
                return self.get(category_name)
        # If not Found Try regular method, handle error
        return self.get(loose_category_name)

    def get_id(self, category_name):
        """ Gets ElementId of Category by name

        Args:
            ``str``: Name of Category

        Returns:
            ``DB.BuiltInCategory``: BuiltInCategory Enumeration Member
        """
        enum = self.get(category_name)
        return DB.ElementId(enum)

    def from_category_id(self, category_id):
        """
        Casts ``DB.BuiltInCategory`` Enumeration member from a Category ElementId
```

(continues on next page)

(continued from previous page)

```
        Args:
            category_id (``DB.ElementId``): ElementId reference of a category

        Returns:
            ``DB.BuiltInCategory`` member
        """
        bic = Enum.ToObject(DB.BuiltInCategory, category_id.IntegerValue)
        if DB.ElementId(bic).IntegerValue < -1:
            return bic
        else:
            # If you pass a regular element to category_id, it converts it to BIC.
            # It should fail, because result is not a valid Category Enum
            raise RpwCoerceError('category_id: {}'.format(category_id),
                                 DB.BuiltInCategory)
        # Similar to: Category.GetCategory(doc, category.Id).Name

    def __repr__(self):
        return super(_BiCategory, self).__repr__(to_string='Autodesk.Revit.DB.
↪BuiltInCategory')


# Classes should already be instantiated
BiParameter = _BiParameter()
BiCategory = _BiCategory()
# TODO: Replace on Tests and Code!
BipEnum = BiParameter
BicEnum = BiCategory
```

## 4.4 rpw.ui

Autodesk.Revit.UI Wrappers

### 4.4.1 Forms

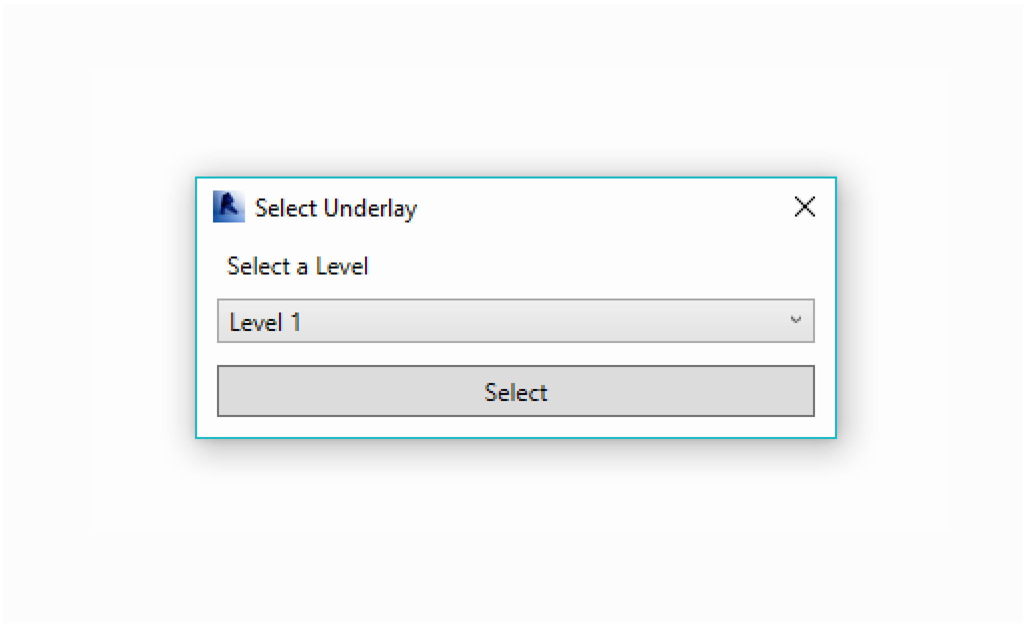
The forms module provide several pre-build forms as well as a framework from which you can build your own forms.

All classes documented in this section can be imported as such:

```
>>> from rpw.ui.forms import Console
```

## QuickForms

### SelectFromList



rpw.ui.forms.**SelectFromList**(*title*, *options*, *description=None*, *sort=True*, *exit_on_close=True*)

Simple FlexForm wrapped function with ComboBox and button

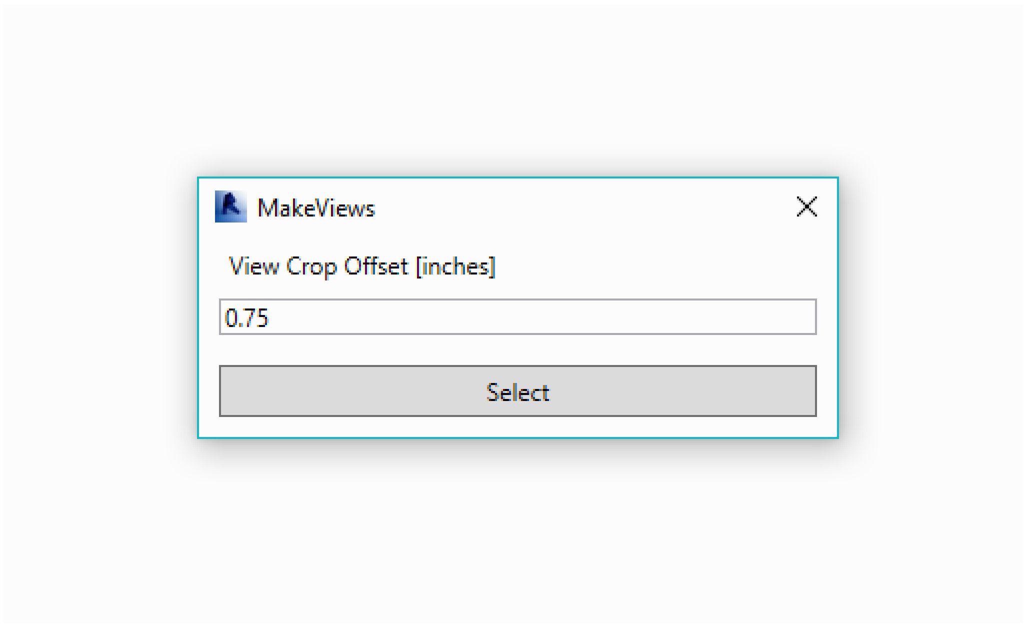
**Parameters**

- **title** (*str*) – Title of form
- **options** (*dict, list[str]*) – Dictionary (string keys) or List[strings]
- **description** (*str*) – Optional Description of input requested [default: None]
- **sort** (*bool*) – Optional sort flag - sorts keys [default: True]
- **exit_on_close** (*bool*) – Form will call sys.exit() if Closed on X. [default: True]

Usage:

```
>>> from rpw.ui.forms import SelectFromList
>>> value = SelectFromList('Test Window', ['1','2','3'])
>>> # Dropdown shows '1', '2' ,'3'. User clicks Select '1'
>>> print(value)
'1'
>>> # Dictionary
>>> value = SelectFromList('Test Window', {'Text':str, 'Number':int})
>>> # User clicks Text
>>> print(value)
str
```

### TextInput



`rpw.ui.forms.`**`TextInput`**(*title*, *default=None*, *description=None*, *sort=True*, *exit_on_close=True*)

Simple FlexForm wrapped function with TextBox and button

**Parameters**

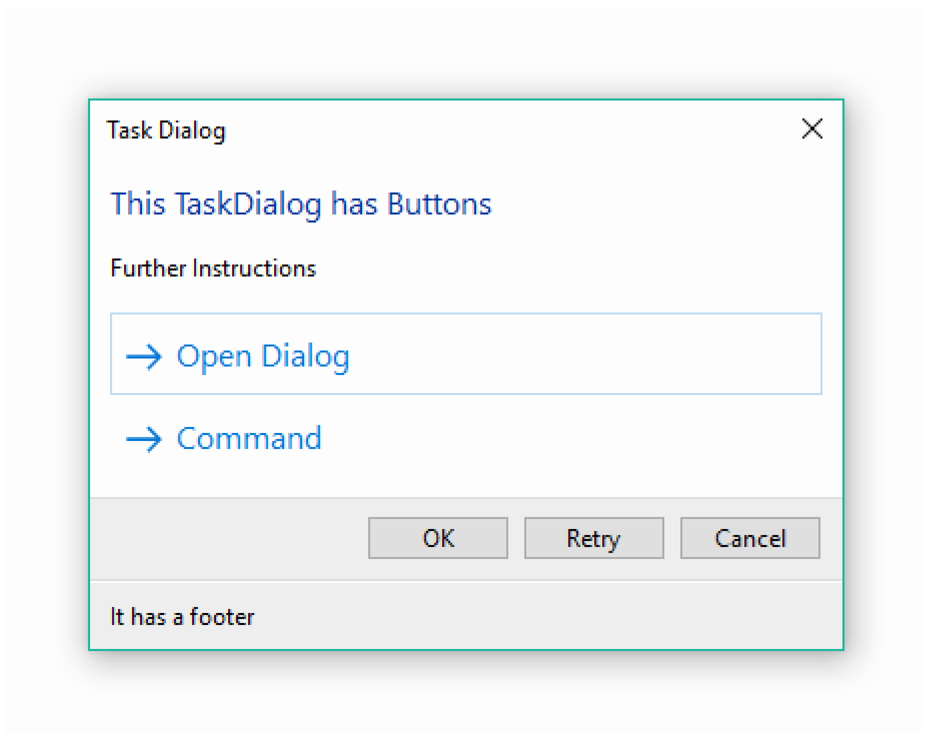
- **`title`** (*str*) – Title of form
- **`default`** (*str*) – Optional default value for text box [default: None]
- **`description`** (*str*) – Optional Description of input requested [default: None]
- **`exit_on_close`** (*bool*) – Form will call sys.exit() if Closed on X. [default: True]

Usage:

```
>>> from rpw.ui.forms import TextInput
>>> value = TextInput('Title', default="3")
>>> print(value)
3
```

### TaskDialogs

#### TaskDialog



**class** `rpw.ui.forms.`**`TaskDialog`**(*instruction*, *commands=None*, *buttons=None*, *title='Task Dialog'*, *content=''*, *title_prefix=False*, *show_close=False*, *footer=''*, *expanded_content=''*, *verification_text=''*)

Task Dialog Wrapper

```
>>> from rpw.ui.forms import CommandLink, TaskDialog
>>> commands= [CommandLink('Open Dialog', return_value='Open'),
>>> ...           CommandLink('Command', return_value=lambda: True)]
>>> ...
>>> dialog = TaskDialog('This TaskDialog has Buttons ',
>>> ...                  title_prefix=False,
>>> ...                  content="Further Instructions",
>>> ...                  commands=commands,
>>> ...                  buttons=['Cancel', 'OK', 'RETRY'],
>>> ...                  footer='It has a footer',
>>> ...                  # verification_text='Add Verification Checkbox',
>>> ...                  # expanded_content='Add Expanded Content',
>>> ...                  show_close=True)
>>> dialog.show()
'Open'
```

**Wrapped Element:** self._revit_object = *Revit.UI.TaskDialog*

**Parameters**

- **`content`** (*`str`*) – Main text of TaskDialog.

- **commands** (*list, optional*) – List of CommandLink Instances. Default is no buttons.
- **buttons** (*list, optional*) – List of TaskDialogCommonButtons names. Default is no buttons. 'Close' is shown if no commands are passed.
- **title** (*str, optional*) – Title of TaskDialog. Default is 'Task Dialog'.p
- **instruction** (*str, optional*) – Main Instruction.
- **footer** (*str, optional*) – Footer Text. Default is `blank`.
- **expanded_content** (*str, optional*) – Expandable Text. Default is `blank`.
- **verification_text** (*str, optional*) – Checkbox text. Default is `blank`.
- **title_prefix** (*bool, optional*) – Prefix Title with app name. Default is `False`
- **show_close** (*bool, optional*) – Show X to close. Default is False.

**__init__**(*instruction, commands=None, buttons=None, title='Task Dialog', content='', title_prefix=False, show_close=False, footer='', expanded_content='', verification_text=''*)
Child classes can use self._revit_object to refer back to Revit Element

**Warning:** Any Wrapper that inherits and overrides __init__ class MUST ensure `_revit_object` is created by calling super().__init__ before setting any self attributes. Not doing so will cause recursion errors and Revit will crash. BaseObjectWrapper should define a class variable _revit_object_class to define the object class being wrapped.

**show**(*exit=False*)
Show TaskDialog

**Parameters exit** (*bool, optional*) – Exit Script after Dialog. Useful for displaying Errors. Default is False.

**Returns** Returns is `False` if dialog is Cancelled (X or Cancel button). If CommandLink button is clicked, `CommandLink.return_value` is returned - if one was not provided, `CommandLink.text` is used. If CommonButtons are clicked `TaskDialog.TaskDialogResult` name is returned ie('Close', 'Retry', 'Yes', etc).

**class** rpw.ui.forms.**CommandLink**(*text, subtext='', return_value=None*)
Command Link Helper Class

**Usage:**

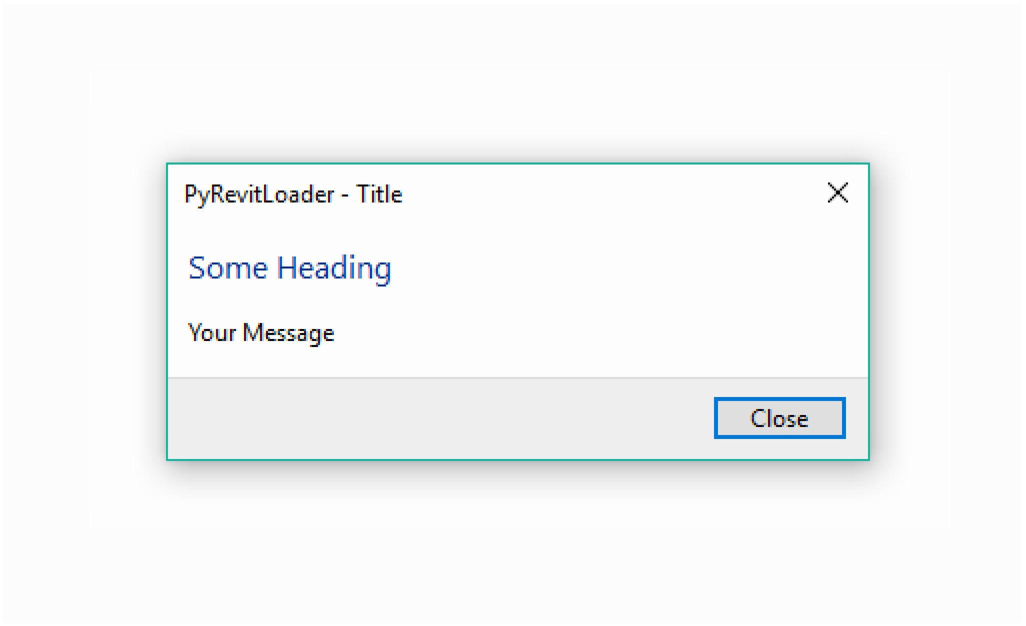
```
>>> from rpw.ui.forms import CommandLink, TaskDialog
>>> CommandLink('Open Dialog', return_value=func_to_open)
>>> TaskDialog('Title', commands=[CommandLink])
```

**Parameters**

- **text** (*str*) – Command Text
- **subtext** (*str, optional*) – Button Subtext
- **return_value** (*any, optional*) – Value returned if button is clicked. If none is provided, text is returned.

**__init__**(*text, subtext='', return_value=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

### Alert



Some Heading

Your Message

**class** `rpw.ui.forms.`**`Alert`**(*content*, *title='Alert'*, *header=''*, *exit=False*)

A Simple Revit TaskDialog for displaying quick messages

**Usage:**

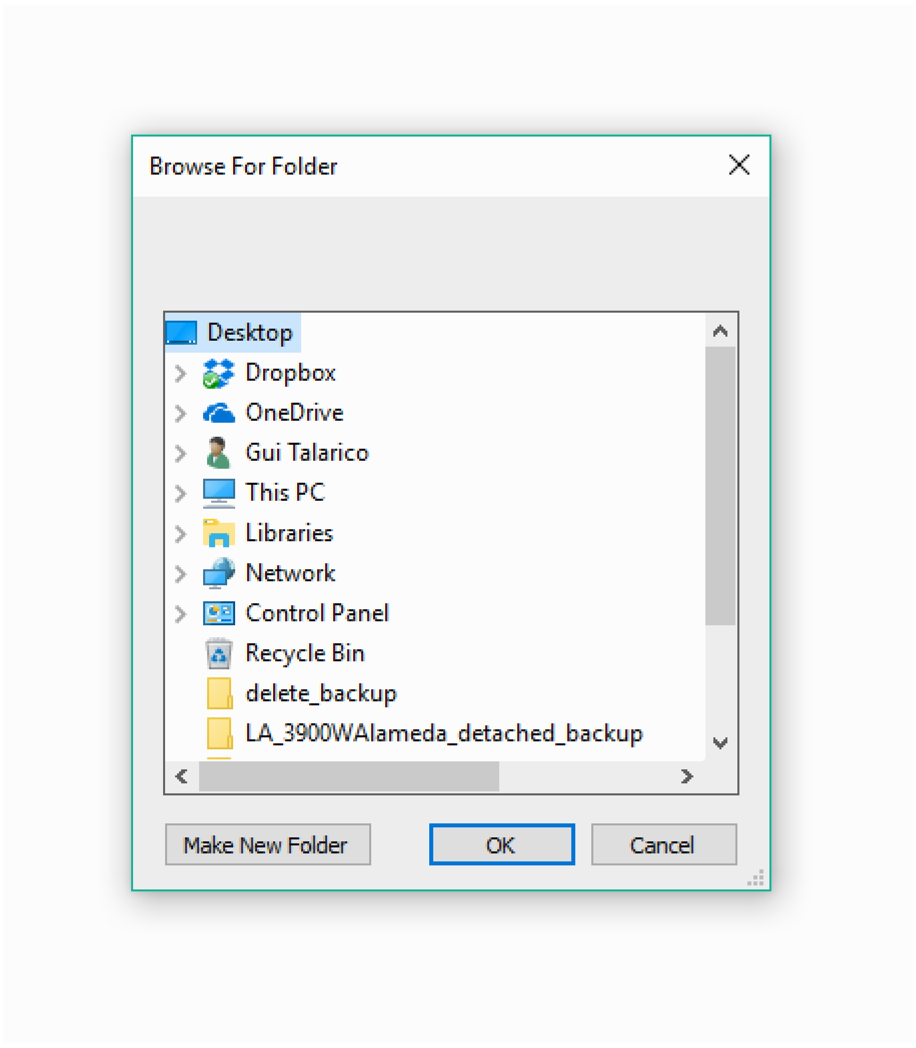
```
>>> from rpw.ui.forms import Alert
>>> Alert('Your Message', title="Title", header="Header Text")
>>> Alert('You need to select Something', exit=True)
```

**Parameters**

- **message** (*str*) – TaskDialog Content
- **title** (*str, optional*) – TaskDialog Title
- **header** (*str, optional*) – TaskDialog content header
- **exit** (*bool, optional*) – Exit Script after Dialog. Useful for displayin Errors. Default is False

### OS Dialogs

#### Select Folder

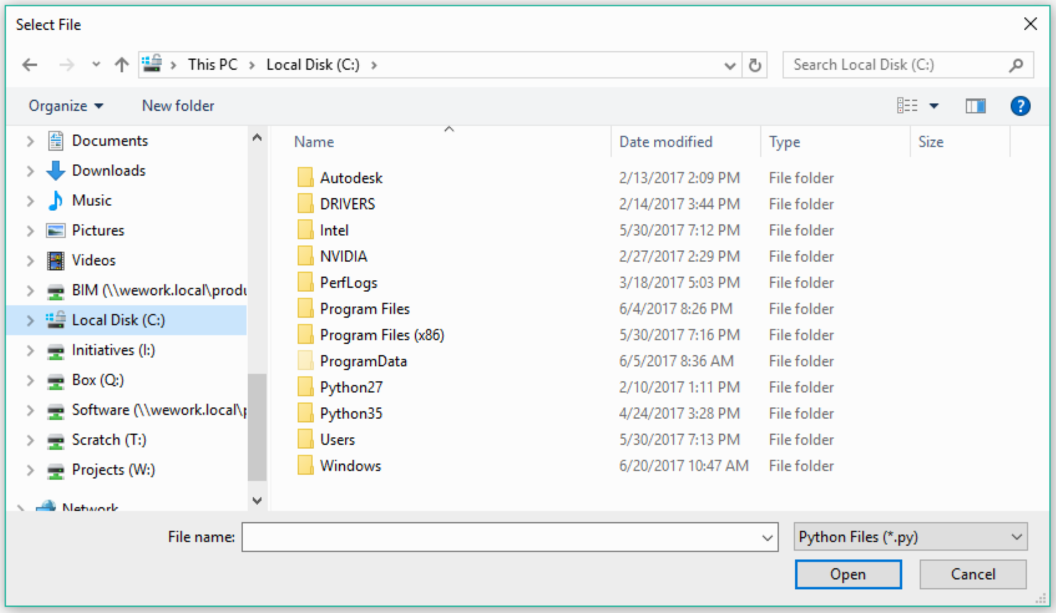


`rpw.ui.forms.`**`select_folder`**`()`

Selects a Folder Path using the standard OS Dialog. Uses Forms.FolderBrowserDialog(). For more information see: https://msdn.microsoft.com/en-us/library/system.windows.forms.openfiledialog.

```
>>> from rpw.ui.forms import select_folder
>>> folderpath = select_folder()
'C:\folder\path'
```

## Select File



| Name | Date modified | Type | Size |
|---|---|---|---|
| Autodesk | 2/13/2017 2:09 PM | File folder | |
| DRIVERS | 2/14/2017 3:44 PM | File folder | |
| Intel | 5/30/2017 7:12 PM | File folder | |
| NVIDIA | 2/27/2017 2:29 PM | File folder | |
| PerfLogs | 3/18/2017 5:03 PM | File folder | |
| Program Files | 6/4/2017 8:26 PM | File folder | |
| Program Files (x86) | 5/30/2017 7:16 PM | File folder | |
| ProgramData | 6/5/2017 8:36 AM | File folder | |
| Python27 | 2/10/2017 1:11 PM | File folder | |
| Python35 | 4/24/2017 3:28 PM | File folder | |
| Users | 5/30/2017 7:13 PM | File folder | |
| Windows | 6/20/2017 10:47 AM | File folder | |

`rpw.ui.forms.`**`select_file`**(*extensions='All Files (*.*)|*.*', title='Select File', multiple=False, restore_directory=True*)

Selects a File Path using the standard OS Dialog. Uses Forms.OpenFileDialog https://msdn.microsoft.com/en-us/library/system.windows.forms.filedialog.filter

```
>>> from rpw.ui.forms import select_file
>>> filepath = select_file('Revit Model (*.rvt)|*.rvt')
'C:\folder\file.rvt'
```

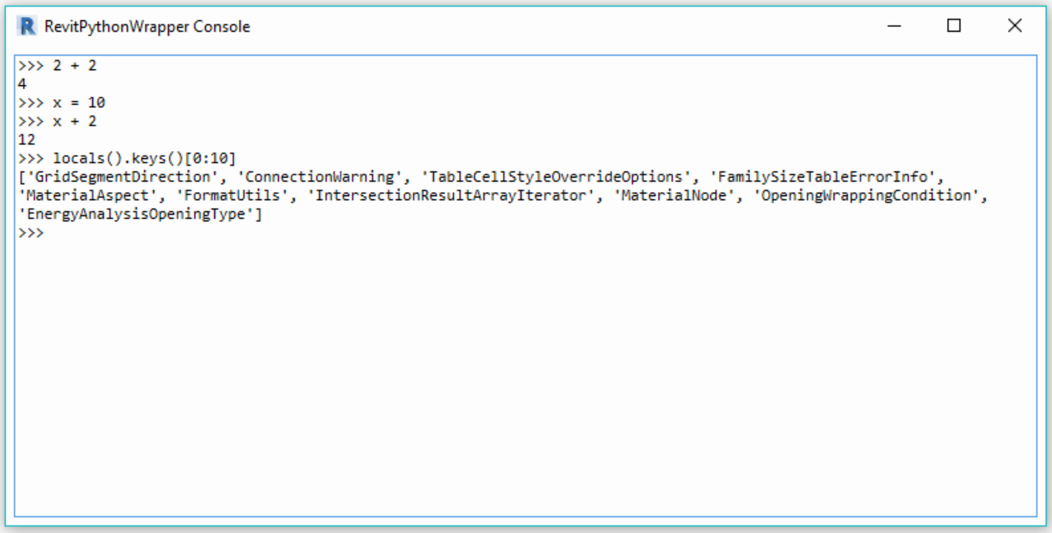
**Parameters**

- **`extensions`** (*`str, optional`*) – File Extensions Filtering options. Default is All Files (.)|*.*
- **`title`** (*`str, optional`*) – File Extensions Filtering options
- **`multiple`** (*`bool`*) – Allow selection of multiple files. Default is *False*
- **`restore_directory`** (*`bool`*) – Restores the directory to the previously selected directory before closing

**Returns** filepath string if `multiple=False` otherwise list of filepath strings

**Return type** filepath (list, string)

### Console



```
>>> 2 + 2
4
>>> x = 10
>>> x + 2
12
>>> locals().keys()[0:10]
['GridSegmentDirection', 'ConnectionWarning', 'TableCellStyleOverrideOptions', 'FamilySizeTableErrorInfo',
'MaterialAspect', 'FormatUtils', 'IntersectionResultArrayIterator', 'MaterialNode', 'OpeningWrappingCondition',
'EnergyAnalysisOpeningType']
>>>
```

REPL Console for Inspecting Stack

```
>>> from rpw.ui.forms import Console
>>> Console()
# Console is launched with locally defined variables injected into context.
```

**Keyboard Shortcuts:**

- `UP` Iterate history up
- `Down` Iterate history down
- `Tab` Iterate possible autocomplete options (works for dotted lookup)

**Note:** The last stack frame is automatically injected is the context of the evaluation loop of the console: the local and global variables from where the Console was called from should be available.

Inspection of the stack requires *stack frames* to be enabled. If an exception is raised stating `` `object has no attribute '_getframe' `` it means IronPython stack frames is not enabled. You can enable it by running with the `-X` argument: `ipy.exe -X: FullFrames file.py`.

If you are trying to use it from within Dynamo, stack inspection is currently not available due to how the engine is setup, but you can still use it by manually passing the context you want to inspect:

```
>>> Console(context=locals())  # or
>>> Console(context=globals())
```

**class** `rpw.ui.forms.`**`Console`**(*stack_level=1*, *stack_info=True*, *context=None*, *msg=''*)

## FlexForm

**class** rpw.ui.forms.**FlexForm**(*title*, *components*, ***kwargs*)

Flex Form Usage

```
>>> from rpw.ui.forms import (FlexForm, Label, ComboBox, TextBox, TextBox,
...                           Separator, Button)
>>> components = [Label('Pick Style:'),
...               ComboBox('combobox1', {'Opt 1': 10.0, 'Opt 2': 20.0}),
...               Label('Enter Name:'),
...               TextBox('textbox1', Text="Default Value"),
...               CheckBox('checkbox1', 'Check this'),
...               Separator(),
...               Button('Select')]
>>> form = FlexForm('Title', components)
>>> form.show()
>>> # User selects `Opt 1`, types 'Wood' in TextBox, and select Checkbox
>>> form.values
{'combobox1': 10.0, 'textbox1': 'Wood', 'checkbox': True}
```

**__init__**(*title*, *components*, ***kwargs*)

**Parameters**

- **title** (str) – Form Title
- **components** (list) – List of Form Components.
- **top_offset** (float) – Optional top offset.
- **options** (kwargs) – WPF Parameters Options

**values**

dict – Dictionary of selected values

**close**()

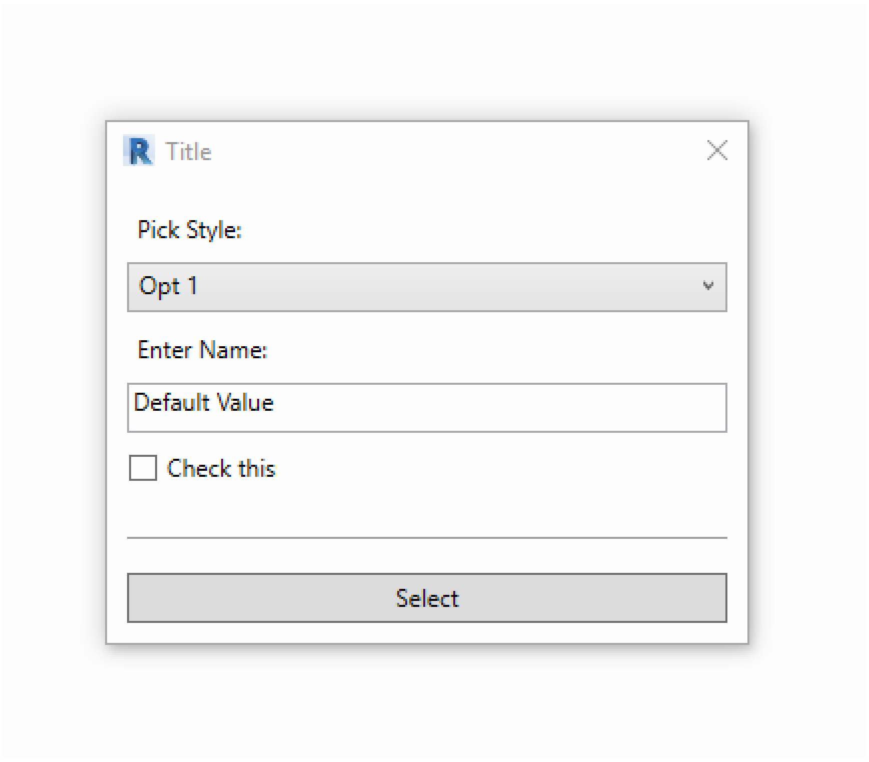
Exits Form. Returns `True` to `show()` method

**static get_values**(*sender*, *e*)

Default Get Values. Set form.values attribute with values from controls and closes form.

**show**()

Initializes Form. Returns `True` or `False` if form was exited.



## FlexForm Controls

**class** rpw.ui.forms.**Label**(*label_text*, ***kwargs*)
Windows.Controls.Label Wrapper

```
>>> Label('Label Text')
```

**__init__**(*label_text*, ***kwargs*)

**Parameters**

- **label_text** (str) – Label Text
- **wpf_params** (*kwargs*) – Additional WPF attributes

**class** rpw.ui.forms.**TextBox**(*name*, *default=”*, ***kwargs*)
Windows.Controls.TextBox Wrapper

```
>>> TextBox()
```

**__init__**(*name*, *default=”*, ***kwargs*)

**Parameters**

- **name** (str) – Name of control. Will be used to return value
- **default** (bool) – Sets `Text` attribute of textbox [Default: ‘’]
- **wpf_params** (*kwargs*) – Additional WPF attributes

**class** rpw.ui.forms.**CheckBox**(*name*, *checkbox_text*, *default=False*, ***kwargs*)
Windows.Controls.Checkbox Wrapper

```
>>> CheckBox('Label')
```

**__init__**(*name*, *checkbox_text*, *default=False*, ***kwargs*)

**Parameters**

- **name** (str) – Name of control. Will be used to return value
- **checkbox_text** (str) – Checkbox label Text
- **default** (bool) – Sets IsChecked state [Default: False]
- **wpf_params** (*kwargs*) – Additional WPF attributes

**class** rpw.ui.forms.**ComboBox**(*name*, *options*, *default=None*, *sort=True*, ***kwargs*)
Windows.Controls.ComboBox Wrapper

```
>>> ComboBox({'Option 1': Element, 'Option 2', 'Elemnet'})
>>> ComboBox({'Option 1': Element, 'Option 2', 'Elemnet'}, sort=False)
```

**__init__**(*name*, *options*, *default=None*, *sort=True*, ***kwargs*)

**Parameters**

- **name** (str) – Name of control. Will be used to return value
- **options** (list, dict) – If dict, selected value is returned
- **default** (str) – Sets SelectedItem attribute [Default: first]
- **wpf_params** (*kwargs*) – Additional WPF attributes

**class** rpw.ui.forms.**Button**(*button_text*, *on_click=None*, ***kwargs*)
Windows.Controls.Button Wrapper

```
>>> Button('Select')
```

**__init__**(*button_text*, *on_click=None*, ***kwargs*)

**Parameters**

- **button_text** (str) – Button Text
- **on_click** (func) – Registers Click event Function [Default: *FlexForm.get_values*]
- **wpf_params** (*kwargs*) – Additional WPF attributes

**class** rpw.ui.forms.**Separator**(***kwargs*)
WPF Separator

## Implementations

FlexForm

```
from itertools import count

from rpw.utils.dotnet import Enum
from rpw.ui.forms.resources import *
```

(continues on next page)

(continued from previous page)

```
class FlexForm(Window):
    """
    Flex Form Usage

    >>> from rpw.ui.forms import (FlexForm, Label, ComboBox, TextBox, TextBox,
    ...                           Separator, Button)
    >>> components = [Label('Pick Style:'),
    ...               ComboBox('combobox1', {'Opt 1': 10.0, 'Opt 2': 20.0}),
    ...               Label('Enter Name:'),
    ...               TextBox('textbox1', Text="Default Value"),
    ...               CheckBox('checkbox1', 'Check this'),
    ...               Separator(),
    ...               Button('Select')]
    >>> form = FlexForm('Title', components)
    >>> form.show()
    >>> # User selects `Opt 1`, types 'Wood' in TextBox, and select Checkbox
    >>> form.values
    {'combobox1': 10.0, 'textbox1': 'Wood', 'checkbox': True}

    """
    layout = """
            <Window
                xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
                xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
                xmlns:local="clr-namespace:WpfApplication1" mc:Ignorable="d"
                ResizeMode="NoResize"
                WindowStartupLocation="CenterScreen"
                Topmost="True"
                SizeToContent="WidthAndHeight">

                <Grid Name="MainGrid" Margin="10,10,10,10">
                </Grid>
            </Window>
            """

    def __init__(self, title, components, **kwargs):
        """
        Args:
            title (``str``): Form Title
            components (``list``): List of Form Components.
            top_offset (``float``): Optional top offset.
            options (``kwargs``): WPF Parameters Options

        Attributes:
            values (``dict``): Dictionary of selected values
        """

        self.ui = wpf.LoadComponent(self, StringReader(self.layout))
        self.ui.Title = title
        self.values = {}

        for key, value in kwargs.iteritems():
            setattr(self, key, value)
```

(continues on next page)

(continued from previous page)

```
        for n, component in enumerate(components):
            self.MainGrid.Children.Add(component)
            if hasattr(component, 'on_click'):
                component.Click += component.on_click

            V_SPACE = 5
            if n > 0:
                prev_comp = components[n - 1]
                top = prev_comp.Margin.Top + prev_comp.Height + V_SPACE
                top += getattr(component, 'top_offset', 0)
                component.Margin = Thickness(0, top, 0, 0)

    def show(self):
        """
        Initializes Form. Returns ``True`` or ``False`` if form was exited.
        """
        return self.ShowDialog()

    @staticmethod
    def get_values(sender, e):
        """
        Default Get Values. Set form.values attribute with values from controls
        and closes form.
        """
        component_values = {}
        window = Window.GetWindow(sender)
        for component in window.MainGrid.Children:
            try:
                component_values[component.Name] = component.value
            except AttributeError:
                pass
        window.values = component_values
        window.close()

    def close(self):
        """ Exits Form. Returns ``True`` to ``show()`` method """
        self.DialogResult = True
        self.Close()


class RpwControlMixin():
    """ Control Mixin """
    _index = count(0)

    def __init__(self, **kwargs):
        self.set_attrs(**kwargs)

    def set_attrs(self, **kwargs):
        """ Parses kwargs, sets default values where appropriate. """
        self.index = next(self._index)  # Counts Instatiation to control Height

        # Default Values
        control_type = self.__class__.__name__
        if not self.Name:
            self.Name = kwargs.get('Name', '{}_{}'.format(control_type, self.index))
```

(continues on next page)

(continued from previous page)

```
        self.Width = kwargs.get('Width', 300)
        self.Height = kwargs.get('Height', 25)

        h_align = Enum.Parse(HorizontalAlignment, kwargs.get('h_align', 'Left'))
        self.HorizontalAlignment = h_align
        v_align = Enum.Parse(VerticalAlignment, kwargs.get('v_align', 'Top'))
        self.VerticalAlignment = v_align

        # Inject Any other Custom Values into Component
        # Updating __dict__ fails due to how .NET inheritance/properties works
        for key, value in kwargs.iteritems():
            setattr(self, key, value)


class Label(RpwControlMixin, Controls.Label):
    """
    Windows.Controls.Label Wrapper

    >>> Label('Label Text')
    """
    def __init__(self, label_text, **kwargs):
        """
        Args:
            label_text (``str``): Label Text
            wpf_params (kwargs): Additional WPF attributes
        """
        self.Content = label_text
        self.set_attrs(**kwargs)


class TextBox(RpwControlMixin, Controls.TextBox):
    """
    Windows.Controls.TextBox Wrapper

    >>> TextBox()
    """
    def __init__(self, name, default='', **kwargs):
        """
        Args:
            name (``str``): Name of control. Will be used to return value
            default (``bool``): Sets ``Text`` attribute of textbox [Default: '']
            wpf_params (kwargs): Additional WPF attributes
        """
        self.Name = name
        self.Text = default
        self.set_attrs(**kwargs)
        if 'Height' not in kwargs:
            self.Height = 25

    @property
    def value(self):
        return self.Text


class Button(RpwControlMixin, Controls.Button):
    """
    Windows.Controls.Button Wrapper
```

(continues on next page)

(continued from previous page)

```
    >>> Button('Select')
    """
    def __init__(self, button_text, on_click=None, **kwargs):
        """
        Args:
            button_text (``str``): Button Text
            on_click (``func``): Registers Click event Function [Default:␣
↪:any:`FlexForm.get_values`]
            wpf_params (kwargs): Additional WPF attributes
        """
        self.Content = button_text
        self.on_click = on_click or FlexForm.get_values
        self.set_attrs(**kwargs)


class CheckBox(RpwControlMixin, Controls.CheckBox):
    """
    Windows.Controls.Checkbox Wrapper

    >>> CheckBox('Label')
    """
    def __init__(self, name, checkbox_text, default=False, **kwargs):
        """
        Args:
            name (``str``): Name of control. Will be used to return value
            checkbox_text (``str``): Checkbox label Text
            default (``bool``): Sets IsChecked state [Default: False]
            wpf_params (kwargs): Additional WPF attributes
        """
        self.Name = name
        self.Content = checkbox_text
        self.IsChecked = default
        self.set_attrs(top_offset=5, **kwargs)

    @property
    def value(self):
        return self.IsChecked


class ComboBox(RpwControlMixin, Controls.ComboBox):
    """
    Windows.Controls.ComboBox Wrapper

    >>> ComboBox({'Option 1': Element, 'Option 2', 'Elemnet'})
    >>> ComboBox({'Option 1': Element, 'Option 2', 'Elemnet'}, sort=False)
    """
    def __init__(self, name, options, default=None, sort=True, **kwargs):
        """
        Args:
            name (``str``): Name of control. Will be used to return value
            options (``list``, ``dict``): If ``dict``, selected value is returned
            default (``str``): Sets SelectedItem attribute [Default: first]
            wpf_params (kwargs): Additional WPF attributes
        """
        self.Name = name
        self.set_attrs(**kwargs)
```

(continues on next page)

(continued from previous page)

```python
        index = 0

        self.options = options
        if hasattr(options, 'keys'):
            options = options.keys()
        if sort:
            options.sort()
        if default is None:
            index = 0
        else:
            index = options.index(default)

        self.Items.Clear()
        self.ItemsSource = options
        self.SelectedItem = options[index]

    @property
    def value(self):
        selected = self.SelectedItem
        if isinstance(self.options, dict):
            selected = self.options[selected]
        return selected

class Separator(RpwControlMixin, Controls.Separator):
    """ WPF Separator """


if __name__ == '__main__':
    """ TESTS """
    components = [
                  Label('Pick Style:'),
                  ComboBox('combobox1', {'Opt 1': 10.0, 'Opt 2': 20.0}),
                  Label('Enter Name:'),
                  TextBox('textbox1', Text="Default Value"),
                  CheckBox('checkbox1', 'Check this:'),
                  Separator(),
                  Button('Select')]

    form = FlexForm('Title', components)
    form.show()

    print(form.values)
```

QuickForm

```python

import sys

from rpw.ui.forms.flexform import FlexForm, Label, ComboBox, TextBox, Button

def SelectFromList(title, options, description=None, sort=True, exit_on_close=True):
    """ Simple FlexForm wrapped function with ComboBox and button

    Args:
        title (str): Title of form
        options (dict,list[str]): Dictionary (string keys) or List[strings]
        description (str): Optional Description of input requested  [default: None]
```

(continues on next page)

(continued from previous page)

```
        sort (bool): Optional sort flag - sorts keys [default: True]
        exit_on_close (bool): Form will call sys.exit() if Closed on X. [default:␣
↪True]

    Usage:

        >>> from rpw.ui.forms import SelectFromList
        >>> value = SelectFromList('Test Window', ['1','2','3'])
        >>> # Dropdown shows '1', '2' ,'3'. User clicks Select '1'
        >>> print(value)
        '1'
        >>> # Dictionary
        >>> value = SelectFromList('Test Window', {'Text':str, 'Number':int})
        >>> # User clicks Text
        >>> print(value)
        str
    """
    components = []
    if description:
        components.append(Label(description))
    components.append(ComboBox('combobox', options, sort=sort))
    components.append(Button('Select'))
    form = FlexForm(title, components)
    ok = form.show()
    if ok:
        return form.values['combobox']
    if exit_on_close:
        sys.exit()


def TextInput(title, default=None, description=None, sort=True, exit_on_close=True):
    """ Simple FlexForm wrapped function with TextBox and button

    Args:
        title (str): Title of form
        default (str): Optional default value for text box [default: None]
        description (str): Optional Description of input requested  [default: None]
        exit_on_close (bool): Form will call sys.exit() if Closed on X. [default:␣
↪True]

    Usage:

        >>> from rpw.ui.forms import TextInput
        >>> value = TextInput('Title', default="3")
        >>> print(value)
        3
    """
    components = []
    if description:
        components.append(Label(description))
    if default:
        textbox = TextBox('textbox', default=default)
    else:
        textbox = TextBox('textbox')
    components.append(textbox)
    components.append(Button('Select'))
    form = FlexForm(title, components)
```

(continues on next page)

(continued from previous page)

```
    ok = form.show()
    if ok:
        return form.values['textbox']
    if exit_on_close:
        sys.exit()


if __name__ == '__main__':
    rv = SelectFromList('Title', ['A','B'], description="Your Options",
                        exit_on_close=True)
    print(rv)

    rv = SelectFromList('Title', {'A':5, 'B':10}, description="Your Options",
                        exit_on_close=True)
    print(rv)

    rv = TextInput('Title', default="3", exit_on_close=True)
    print(rv)
    print('forms.py ran')
```

TaskDialog

```

import sys
from rpw import UI
from rpw.exceptions import RpwValueError
from rpw.base import BaseObjectWrapper, BaseObject

class Alert():
    """
    A Simple Revit TaskDialog for displaying quick messages

    Usage:
        >>> from rpw.ui.forms import Alert
        >>> Alert('Your Message', title="Title", header="Header Text")
        >>> Alert('You need to select Something', exit=True)

    Args:
        message (str): TaskDialog Content
        title (str, optional): TaskDialog Title
        header (str, optional): TaskDialog content header
        exit (bool, optional): Exit Script after Dialog.
            Useful for displayin Errors. Default is False

    """
    def __init__(self, content, title='Alert', header='', exit=False):
        dialog = UI.TaskDialog(title)
        dialog.TitleAutoPrefix = False
        dialog.MainInstruction = header
        dialog.MainContent = content
        self.result = dialog.Show()

        if exit:
            sys.exit(1)

class CommandLink(BaseObject):
    """
```

(continues on next page)

(continued from previous page)

```
    Command Link Helper Class

    Usage:
        >>> from rpw.ui.forms import CommandLink, TaskDialog
        >>> CommandLink('Open Dialog', return_value=func_to_open)
        >>> TaskDialog('Title', commands=[CommandLink])

    Args:
        text (str): Command Text
        subtext (str, optional): Button Subtext
        return_value (any, optional): Value returned if button is clicked.
            If none is provided, text is returned.

    """
    def __init__(self, text, subtext='', return_value=None):
        self._id = None  # This will later be set to TaskDialogCommandLinkId(n)
        self.text = text
        self.subtext = subtext
        self.return_value = return_value if return_value is not None else text

    def __repr__(self):
        return super(CommandLink, self).__repr__(data={'id': self._id,
                                                        'text':self.text})


class TaskDialog(BaseObjectWrapper):
    """
    Task Dialog Wrapper

    >>> from rpw.ui.forms import CommandLink, TaskDialog
    >>> commands= [CommandLink('Open Dialog', return_value='Open'),
    >>> ...           CommandLink('Command', return_value=lambda: True)]
    >>> ...
    >>> dialog = TaskDialog('This TaskDialog has Buttons ',
    >>> ...                 title_prefix=False,
    >>> ...                 content="Further Instructions",
    >>> ...                 commands=commands,
    >>> ...                 buttons=['Cancel', 'OK', 'RETRY'],
    >>> ...                 footer='It has a footer',
    >>> ...                 # verification_text='Add Verification Checkbox',
    >>> ...                 # expanded_content='Add Expanded Content',
    >>> ...                 show_close=True)
    >>> dialog.show()
    'Open'

    Wrapped Element:
        self._revit_object = `Revit.UI.TaskDialog`

    Args:
        content (str): Main text of TaskDialog.
        commands (list, optional): List of CommandLink Instances.
            Default is no buttons.
        buttons (list, optional): List of TaskDialogCommonButtons names.
            Default is no buttons. 'Close' is shown if no commands are passed.
        title (str, optional): Title of TaskDialog. Default is 'Task Dialog'.p
        instruction (str, optional): Main Instruction.
        footer (str, optional): Footer Text. Default is ``blank``.
```

(continues on next page)

(continued from previous page)

```
        expanded_content (str, optional): Expandable Text. Default is ``blank``.
        verification_text (str, optional): Checkbox text. Default is ``blank``.
        title_prefix (bool, optional): Prefix Title with app name.
            Default is ``False``
        show_close (bool, optional): Show X to close. Default is False.

    """
    _revit_object_class = UI.TaskDialog
    _common_buttons = ['Ok', 'Yes', 'No', 'Cancel', 'Retry', 'Close']

    def __init__(self, instruction, commands=None, buttons=None,
                 title='Task Dialog', content='',
                 title_prefix=False, show_close=False,
                 footer='', expanded_content='', verification_text=''
                 ):

        super(TaskDialog, self).__init__(UI.TaskDialog(title))
        self.dialog = self._revit_object

        # Settings
        self.dialog.TitleAutoPrefix = title_prefix
        self.dialog.AllowCancellation = show_close

        # Properties
        self.dialog.Title = title
        self.dialog.MainInstruction = instruction
        self.dialog.MainContent = content
        self.dialog.FooterText = footer
        self.dialog.ExpandedContent = expanded_content
        self.dialog.VerificationText = verification_text
        self.verification_checked = None if not verification_text else False

        # Add Buttons
        self.buttons = buttons or []
        common_buttons_names = []
        for button_name in [b.capitalize() for b in self.buttons]:
            if button_name not in self._common_buttons:
                raise RpwValueError('TaskDialogCommonButtons member', button_name)
            button_full_name = 'UI.TaskDialogCommonButtons.' + button_name
            common_buttons_names.append(button_full_name)

        if common_buttons_names:
            common_buttons = eval('|'.join(common_buttons_names))
            self.dialog.CommonButtons = common_buttons

        # Set Default Button
        self.dialog.DefaultButton = UI.TaskDialogResult.None

        # Validate Commands
        commands = commands or []
        if len(commands) > 4:
            raise RpwValueError('4 or less command links', len(commands))

        # Process Commands
        self.commands = {}
        for link_index, command_link in enumerate(commands, 1):
            command_id = 'CommandLink{}'.format(link_index)
```

(continues on next page)

(continued from previous page)

```
            command_link._id = getattr(UI.TaskDialogCommandLinkId, command_id)
            self.commands[command_id] = command_link
            self.dialog.AddCommandLink(command_link._id,
                                       command_link.text,
                                       command_link.subtext)

    def show(self, exit=False):
        """
        Show TaskDialog

        Args:
            exit (bool, optional): Exit Script after Dialog. Useful for
                displaying Errors. Default is False.

        Returns:
            Returns is ``False`` if dialog is Cancelled (X or Cancel button).
            If CommandLink button is clicked, ``CommandLink.return_value``
            is returned - if one was not provided, ``CommandLink.text`` is used.
            If CommonButtons are clicked ``TaskDialog.TaskDialogResult`` name is
            returned ie('Close', 'Retry', 'Yes', etc).
        """
        self.result = self.dialog.Show()

        try:
            self.verification_checked = self.dialog.WasVerificationChecked()
        except:
            self.verification_checked = None

        # Handle Cancel
        if self.result == UI.TaskDialogResult.Cancel:
            if exit:
                sys.exit(1)
            return None

        # If result is a CommandLink, return Return Value else Result
        command_link = self.commands.get(str(self.result))
        if command_link:
            return command_link.return_value
        else:
            return self.result.ToString()



if __name__ == '__main__':
    Alert('Test Alert!')

    def sample_callback():
        print('Calling B')
        d = UI.TaskDialog("Revit Build Information")
        d.MainInstruction = "Button 1"
        d.Show()


    from rpw.ui.forms.taskdialog import *
    commands = [
                CommandLink('TestTitle', return_value=sample_callback, subtext='test␣
↪subtext'),
```

(continues on next page)

(continued from previous page)

```
                CommandLink('TestTitle2', return_value=lambda: 'Empty', subtext='test␣
↪subtext2')
                ]

    t = TaskDialog(commands=commands, buttons=['Yes'])
```

OS Dialogs

```

from rpw.ui.forms.resources import *

def select_folder():
    """
    Selects a Folder Path using the standard OS Dialog.
    Uses Forms.FolderBrowserDialog(). For more information see:
    https://msdn.microsoft.com/en-us/library/system.windows.forms.openfiledialog.

    >>> from rpw.ui.forms import select_folder
    >>> folderpath = select_folder()
    'C:\\folder\\path'
    """

    form = Forms.FolderBrowserDialog()
    if form.ShowDialog() == Forms.DialogResult.OK:
        return form.SelectedPath


def select_file(extensions='All Files (*.*)|*.*',
                title="Select File",
                multiple=False,
                restore_directory=True):
    """
    Selects a File Path using the standard OS Dialog.
    Uses Forms.OpenFileDialog
    https://msdn.microsoft.com/en-us/library/system.windows.forms.filedialog.filter

    >>> from rpw.ui.forms import select_file
    >>> filepath = select_file('Revit Model (*.rvt)|*.rvt')
    'C:\\folder\\file.rvt'

    Args:
        extensions (str, optional): File Extensions Filtering options. Default is All␣
↪Files (*.*)|*.*
        title (str, optional): File Extensions Filtering options
        multiple (bool): Allow selection of multiple files. Default is `False`
        restore_directory (bool): Restores the directory to the previously selected␣
↪directory before closing

    Returns:
        filepath (list, string): filepath string if ``multiple=False`` otherwise list␣
↪of filepath strings

    """
    form = Forms.OpenFileDialog()
    form.Filter = extensions
    form.Title = title
    form.Multiselect = multiple
```

(continues on next page)

(continued from previous page)

```
    form.RestoreDirectory = restore_directory
    if form.ShowDialog() == Forms.DialogResult.OK:
        return form.FileName if not multiple else list(form.FileNames)

# Tests
if __name__ == '__main__':
    # print(select_folder())
    # print(select_file('Python Files|*.py'))
    print(select_file('Python Files|*.py', multiple=False))
    print(select_file('Python Files|*.py', multiple=True))
```

Console

```

import os
import inspect
import logging
import tempfile
from collections import defaultdict
import traceback

from rpw.utils.rlcompleter import Completer
from rpw.ui.forms.resources import Window
from rpw.ui.forms.resources import *
# logger.verbose(False)


class Console(Window):
    LAYOUT = """
                <Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
↪presentation"
                        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                        Title="DeployWindow" Height="400" Width="800"␣
↪SnapsToDevicePixels="True"
                        UseLayoutRounding="True" WindowState="Normal"
                        WindowStartupLocation="CenterScreen">
                <Window.Resources>
                    <Style TargetType="{x:Type MenuItem}">
                        <Setter Property="FontFamily" Value="Consolas"/>
                        <Setter Property="FontSize" Value="12.0"/>
                    </Style>
                </Window.Resources>
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*"></ColumnDefinition>
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="0"></RowDefinition>
                        <RowDefinition Height="*"></RowDefinition>
                    </Grid.RowDefinitions>
                    <TextBox Grid.Column="1" Grid.Row="1"  HorizontalAlignment=
↪"Stretch"
                             Name="tbox" Margin="6,6,6,6" VerticalAlignment="Stretch"
                             AcceptsReturn="True" VerticalScrollBarVisibility="Auto"
                             TextWrapping="Wrap"
                             FontFamily="Consolas" FontSize="12.0"
                             />
```

(continues on next page)

(continued from previous page)

```
                </Grid>
                </Window>
    """
    # <Button Content="Terminate" Margin="6,6,6,6" Height="30"
    #     Grid.Column="1" Grid.Row="1" VerticalAlignment="Bottom"
    #     Click="terminate"></Button>

    CARET = '>>> '

    def __init__(self, stack_level=1, stack_info=True, context=None, msg=''):
        """
        Args:
            stack_level (int): Default is 1. 0 Is the Console stack, 1 is the
                               caller; 2 is previous to that, etc.
            stack_info (bool): Display info about where call came from.
                               Will print filename name,  line no. and Caller
                               name.
           msg (str): Message to display on start.
                      Only available if using context
            context (dict): Optional Dictionary for when inspection is not
                            possible.
        """

        # History Helper
        tempdir = tempfile.gettempdir()
        filename = 'rpw-history'
        self.history_file = os.path.join(tempdir, filename)

        self.stack_locals = {}
        self.stack_globals = {}
        self.stack_level = stack_level

        if context:
            self.stack_locals.update(context)
            # Allows to pass context manually, so it can be used in Dynamo
            # Where inspection does not work
        else:
            # Stack Info
            # stack_frame = inspect.currentframe().f_back
            stack_frame = inspect.stack()[stack_level][0]  # Finds Calling Stack

            self.stack_locals.update(stack_frame.f_locals)
            self.stack_globals.update(stack_frame.f_globals)
            # Debug Console
            self.stack_globals.update({'stack': inspect.stack()})

            stack_code = stack_frame.f_code
            stack_filename = os.path.basename(stack_code.co_filename)
            stack_lineno = stack_code.co_firstlineno
            stack_caller = stack_code.co_name

        self._update_completer()

        # Form Setup
        self.ui = wpf.LoadComponent(self, StringReader(Console.LAYOUT))
        self.ui.Title = 'RevitPythonWrapper Console'
        self.PreviewKeyDown += self.KeyPressPreview
```

(continues on next page)

(continued from previous page)

```
        self.KeyUp += self.OnKeyUpHandler
        self.is_loaded = False

        # Form Init
        self.ui.tbox.Focus()
        if not context and stack_info:
            self.write_line('Caller: {} [ Line:{}] | File: {}'.format(
                                                        stack_caller,
                                                        stack_lineno,
                                                        stack_filename))
        elif msg:
            self.write_line(msg)
        else:
            self.tbox.Text = Console.CARET

        self.ui.tbox.CaretIndex = len(self.tbox.Text)

        # Vars
        self.history_index = 0
        self.ac_options = defaultdict(int)

        self.ShowDialog()

    def force_quit(self, sender, e):
        raise SystemExit('Force Quit')

    def _update_completer(self):
        # Updates Completer. Used at start, and after each exec loop
        context = self.stack_locals.copy()
        context.update(self.stack_globals)
        # context.update(vars(__builtins__))
        self.completer = Completer(context)

    def get_line(self, index):
        line = self.tbox.GetLineText(index).replace('\r\n', '')
        if line.startswith(Console.CARET):
            line = line[len(Console.CARET):]
        logger.debug('Get Line: {}'.format(line))
        return line

    def get_last_line(self):
        try:
            last_line = self.get_lines()[-1]
        except IndexError:
            last_line = self.get_line(0)
        logger.debug('Last Line: {}'.format(last_line))
        return last_line

    def get_last_entered_line(self):
        try:
            last_line = self.get_lines()[-2]
        except IndexError:
            last_line = self.get_line(0)
        logger.debug('Last Line: {}'.format(last_line))
        return last_line

    def get_lines(self):
```

(continues on next page)

(continued from previous page)

```
        last_line_index = self.tbox.LineCount
        lines = []
        for index in range(0, last_line_index):
            line = self.get_line(index)
            lines.append(line)
        logger.debug('Lines: {}'.format(lines))
        return lines

    def OnKeyUpHandler(self, sender, args):
        # Need to use this to be able to override ENTER
        if not self.is_loaded:
            return
        if args.Key == Key.Enter:
            entered_line = self.get_last_entered_line()
            if entered_line == '':
                self.write_line(None)
                return
            output = self.evaluate(entered_line)
            self.append_history(entered_line)
            self.history_index = 0
            self.write_line(output)
            self.tbox.ScrollToEnd()

    def format_exception(self):
        """ Formats Last Exception"""
        exc_type, exc_value, exc_traceback = sys.exc_info()
        tb = traceback.format_exception(exc_type, exc_value, exc_traceback)
        last_exception = tb[-1]
        output = 'Traceback:\n' + last_exception[:-1]
        return output

    def evaluate(self, line):
        try:
            output = eval(line, self.stack_globals, self.stack_locals)
        except SyntaxError as exc:
            try:
                exec(line, self.stack_globals, self.stack_locals)
                self._update_completer()  # Update completer with new locals
                return
            except Exception as exc:
                output = self.format_exception()
        except Exception as exc:
            output = self.format_exception()
        return str(output)

    def OnKeyDownHandler(self, sender, args):
        pass

    @property
    def last_caret_start_index(self):
        return self.tbox.Text.rfind(Console.CARET)

    @property
    def last_caret_end_index(self):
        return self.last_caret_start_index + len(Console.CARET)

    @property
```

(continues on next page)

(continued from previous page)

```
    def last_caret_line_start_index(self):
        return self.last_caret_start_index - len(Console.CARET)

    def reset_caret(self):
        self.tbox.CaretIndex = self.last_caret_end_index

    def KeyPressPreview(self, sender, e):
        # This Happens before all other key handlers
        # If e.Handled = True, stops event propagation here.
        e.Handled = False
        if self.tbox.CaretIndex < self.last_caret_start_index:
            self.tbox.CaretIndex = len(self.tbox.Text)
        if e.Key == Key.Up:
            self.history_up()
            e.Handled = True
        if e.Key == Key.Down:
            self.history_down()
            e.Handled = True
        if e.Key == Key.Left or e.Key == Key.Back:
            if self.ui.tbox.CaretIndex == self.last_caret_end_index:
                e.Handled = True
        if e.Key == Key.Home:
            self.reset_caret()
            e.Handled = True
        if e.Key == Key.Tab:
            self.autocomplete()
            e.Handled = True
        if e.Key == Key.Enter:
            self.is_loaded = True
            self.tbox.CaretIndex = len(self.tbox.Text)

    def autocomplete(self):
        text = self.tbox.Text[self.last_caret_end_index:self.tbox.CaretIndex]
        logger.debug('Text: {}'.format(text))

        # Create Dictionary to Track iteration over suggestion
        index = self.ac_options[text]
        suggestion = self.completer.complete(text, index)

        logger.debug('ac_options: {}'.format(self.ac_options))
        logger.debug('Sug: {}'.format(suggestion))

        if not suggestion:
            self.ac_options[text] = 0
        else:
            self.ac_options[text] += 1
            if suggestion.endswith('('):
                suggestion = suggestion[:-1]

        if suggestion is not None:
            caret_index = self.tbox.CaretIndex
            self.write_text(suggestion)
            self.tbox.CaretIndex = caret_index

    def write(self, text):
        """ Make Console usable as File Object """
        self.write_line(line=text)
```

(continues on next page)

(continued from previous page)

```
    def write_line(self, line=None):
        # Used for Code Output
        # Writes line with no starting caret, new line + caret
        if line:
            self.tbox.AppendText(line)
            self.tbox.AppendText(NewLine)
        self.tbox.AppendText(Console.CARET)

    def write_text(self, line):
        # Used by Autocomplete and History
        # Adds text to line, including Caret
        self.tbox.Text = self.tbox.Text[0:self.last_caret_start_index]
        self.tbox.AppendText(Console.CARET)
        self.tbox.AppendText(line)
        self.ui.tbox.CaretIndex = len(self.ui.tbox.Text)

    def get_all_history(self):
        # TODO: Add clean up when history > X
        with open(self.history_file) as fp:
            lines = fp.read().split('\n')
            return [line for line in lines if line != '']

    def history_up(self):
        self.history_index += 1
        line = self.history_iter()
        if line is not None:
            self.write_text(line)

    def history_down(self):
        self.history_index -= 1
        line = self.history_iter()
        if line is not None:
            self.write_text(line)

    def append_history(self, line):
        logger.debug('Adding Line to History:' + repr(line))
        with open(self.history_file, 'a') as fp:
            fp.write(line + '\n')

    def history_iter(self):
        lines = self.get_all_history()
        logger.debug('Lines: {}'.format(lines))
        try:
            line = lines[::-1][self.history_index - 1]
        # Wrap around lines to loop and up down infinetly.
        except IndexError:
            if len(lines) == 0:
                return None
            if len(lines) < 0:
                self.history_index += len(lines)
            if len(lines) > 0:
                self.history_index -= len(lines)
            line = lines[0]
        return line

    def __repr__(self):
```

(continues on next page)

(continued from previous page)

```
        '<rpw:Console stack_level={}>'.format(self.stack_level)


if __name__ == '__main__':
    def test():
        x = 1
        # Console()
        Console(context=locals())
    test()
    z = 2
```

Resources

```

import sys

from abc import ABCMeta

from rpw import revit
from rpw.utils.dotnet import clr
from rpw.utils.logger import logger

# WPF/Form Imports
clr.AddReference("PresentationFramework")  # System.Windows: Controls, ?
clr.AddReference("WindowsBase")            # System.Windows.Input
clr.AddReference("System.Drawing")         # FontFamily
clr.AddReference('System.Windows.Forms') # Forms

import System.Windows
from System.Windows import Window
from System.IO import StringReader

# Console
from System.Environment import Exit, NewLine
from System.Drawing import FontFamily
from System.Windows.Input import Key

# FlexForm Imports
from System.Windows import Controls, Window
from System.Windows import HorizontalAlignment, VerticalAlignment, Thickness

# OS Dialogs
from System.Windows import Forms

if revit.host == 'Dynamo':
    # IronPython 2.7.3 - Dynamo + RPS w/out pyRevit
    # Conflicts with PyRevit. Must Ensure exact path is specified
    # https://github.com/architecture-building-systems/revitpythonshell/issues/46
    clr.AddReferenceToFileAndPath(r'C:\Program Files (x86)\IronPython 2.
↪7\Platforms\Net40\IronPython.Wpf.dll')
    import wpf
    # on 2.7.7 this raises wpf import error
else:
    # IronPython 2.7.7 - pyRevit
    # clr.AddReference('IronPython')    # Works W/Out
    clr.AddReference('IronPython.Wpf')  # 2.7.
    from IronPython.Modules import Wpf as wpf
```

(continues on next page)

(continued from previous page)

```
    # on 2.7.7 this works. On 2.7.3 you get a LoadComponent 3 args error
```

## 4.4.2 Selection

*uidoc.Selection* Wrapper

### Selection Wrapper

**class** rpw.ui.**Selection**(*elements_or_ids=None*, *uidoc=ActiveUIDocument*)

Bases: *rpw.base.BaseObjectWrapper*, rpw.db.collection.ElementSet

```
>>> from rpw import ui
>>> selection = ui.Selection()
>>> selection[0]
FirstElement
>>> selection.element_ids
[ SomeElementId, SomeElementId, ...]
>>> selection.elements
[ SomeElement, SomeElement, ...]
>>> len(selection)
2
```

**Wrapped Element:** _revit_object = *Revit.UI.Selection*

**__bool__**()

**Returns** *False* if selection is empty, *True* otherwise

**Return type** bool

```
>>> len(selection)
2
>>> Selection() is True
True
```

**__getitem__**(*index*)

Overrides ElementSet __getitem__ to retrieve from selection based on index.

**__init__**(*elements_or_ids=None*, *uidoc=ActiveUIDocument*)

Initializes Selection. Elements or ElementIds are optional. If no elements are provided on intiialization, selection handler will be created with selected elements.

**Parameters** **elements** (*[DB.Element or DB.ElementID]*) – Elements or ElementIds

```
>>> selection = Selection(SomeElement)
>>> selection = Selection(SomeElementId)
>>> selection = Selection([Element, Element, Element, ...])
```

**add**(*elements_or_ids*, *select=True*)

Adds elements to selection.

**Parameters** **elements** (*[DB.Element or DB.ElementID]*) – Elements or ElementIds

```
>>> selection = Selection()
>>> selection.add(SomeElement)
>>> selection.add([elements])
>>> selection.add([element_ids])
```

**clear**()
Clears Selection

```
>>> selection = Selection()
>>> selection.clear()
```

**Returns** None

**update**()
Forces UI selection to match the Selection() object

## Pick

**class** rpw.ui.**Pick**(**args*, ***kwargs*)
Bases: `rpw.base.BaseObject`

Pick Class

Handles all pick* methods in the Seletion Class

```
>>> from rpw import ui
>>> ui.Pick.pick_element()
<rpw:reference>
>>> ui.Pick.pick_element(multiple=True)
[<rpw:reference>, ...]
```

**classmethod pick_box**(*msg*, *style='directional'*)
PickBox

**Returns** Min and Max Points of Box

**Return type** XYZ Points (`XYZ`)

**classmethod pick_by_rectangle**(*msg*)
PickBox

**Returns** List of wrapped Elements

**Return type** Elements (`List`)

**classmethod pick_edge**(*msg='Pick Edge(s)'*, *multiple=False*)
Pick Edge

**Parameters**

- **msg** (*str*) – Message to show
- **multiple** (*bool*) – False to pick single edge, True for multiple

**Returns** *Reference* Class

**Return type** reference (`Reference`)

**classmethod pick_element**(*msg='Pick Element(s)'*, *multiple=False*)
Pick Element

**Parameters**

- **msg** (*str*) – Message to show
- **multiple** (*bool*) – False to pick single element, True for multiple

**Returns** *Reference* Class

**Return type** reference (Reference)

**classmethod pick_face**(*msg='Pick Face(s)'*, *multiple=False*)
Pick Face

**Parameters**

- **msg** (*str*) – Message to show
- **multiple** (*bool*) – False to pick single face, True for multiple

**Returns** *Reference* Class

**Return type** reference (Reference)

**classmethod pick_linked_element**(*msg='Pick Linked Element'*, *multiple=False*)
Pick Linked Element

**Parameters**

- **msg** (*str*) – Message to show
- **multiple** (*bool*) – False to pick single element, True for multiple

**Returns** *Reference* Class

**Return type** reference (Reference)

**classmethod pick_pt**(*msg='Pick Point'*, *snap=None*)
Pick Point location

**Parameters**

- **msg** (*str*) – Message to show
- **snap** (*str*) – Snap Options: [endpoints, midpoints, nearest, workplanegrid, intersections, centers, perpendicular, tangents, quadrants, points]

**Returns** Rpw XYZ Point

**Return type** XYZ (*Xyz*)

**classmethod pick_pt_on_element**(*msg='Pick Pt On Element(s)'*, *multiple=False*)
Pick Point On Element

**Parameters**

- **msg** (*str*) – Message to show
- **multiple** (*bool*) – False to pick single point, True for multiple

**Returns** *Reference* Class

**Return type** reference (Reference)

### Implementation

```
"""
`uidoc.Selection` Wrapper
"""
import sys

import rpw
from rpw import revit, DB, UI
from rpw.utils.dotnet import List
from rpw.base import BaseObjectWrapper, BaseObject
from rpw.exceptions import RpwTypeError, RevitExceptions
from rpw.utils.logger import logger
from rpw.utils.coerce import to_element_ids, to_elements, to_iterable
from rpw.db.collection import ElementSet
from rpw.db.reference import Reference
from rpw.db.xyz import XYZ
from rpw.db.element import Element

if revit.host and revit.doc:
    ObjectType = UI.Selection.ObjectType
    ObjectSnapTypes = UI.Selection.ObjectSnapTypes
    PickObjects = revit.uidoc.Selection.PickObjects
    PickObject = revit.uidoc.Selection.PickObject
    PickElementsByRectangle = revit.uidoc.Selection.PickElementsByRectangle
    PickBox = revit.uidoc.Selection.PickBox
    PickPoint = revit.uidoc.Selection.PickPoint


class Selection(BaseObjectWrapper, ElementSet):
    """
    >>> from rpw import ui
    >>> selection = ui.Selection()
    >>> selection[0]
    FirstElement
    >>> selection.element_ids
    [ SomeElementId, SomeElementId, ...]
    >>> selection.elements
    [ SomeElement, SomeElement, ...]
    >>> len(selection)
    2

    Wrapped Element:
        _revit_object = `Revit.UI.Selection`
    """

    _revit_object_class = UI.Selection.Selection

    def __init__(self, elements_or_ids=None, uidoc=revit.uidoc):
        """
        Initializes Selection. Elements or ElementIds are optional.
        If no elements are provided on intiialization,
        selection handler will be created with selected elements.

        Args:
            elements ([DB.Element or DB.ElementID]): Elements or ElementIds
```

(continues on next page)

(continued from previous page)

```
        >>> selection = Selection(SomeElement)
        >>> selection = Selection(SomeElementId)
        >>> selection = Selection([Element, Element, Element, ...])

        """

        BaseObjectWrapper.__init__(self, uidoc.Selection)
        self.uidoc = uidoc

        if not elements_or_ids:
            # Is List of elements is not provided, uses uidoc selection
            elements_or_ids = [e for e in uidoc.Selection.GetElementIds()]

        ElementSet.__init__(self, elements_or_ids, doc=self.uidoc.Document)

    def add(self, elements_or_ids, select=True):
        """ Adds elements to selection.

        Args:
            elements ([DB.Element or DB.ElementID]): Elements or ElementIds

        >>> selection = Selection()
        >>> selection.add(SomeElement)
        >>> selection.add([elements])
        >>> selection.add([element_ids])
        """
        # Call Set for proper adding into set.
        ElementSet.add(self, elements_or_ids)
        if select:
            self.update()

    def update(self):
        """ Forces UI selection to match the Selection() object """
        self._revit_object.SetElementIds(self.get_element_ids(as_list=True))

    def clear(self):
        """ Clears Selection

        >>> selection = Selection()
        >>> selection.clear()

        Returns:
            None
        """
        ElementSet.clear(self)
        self.update()

    def __getitem__(self, index):
        """
        Overrides ElementSet __getitem__ to retrieve from selection
        based on index.
        """
        # https://github.com/gtalarico/revitpythonwrapper/issues/32
        for n, element in enumerate(self.__iter__()):
            if n ==index:
                return element
        else:
```

(continues on next page)

(continued from previous page)

```
            raise IndexError('Index is out of range')

    def __bool__(self):
        """
        Returns:
            bool: `False` if selection  is empty, `True` otherwise

        >>> len(selection)
        2
        >>> Selection() is True
        True
        """
        return super(Selection, obj).__bool__()

    def __repr__(self):
        return super(Selection, self).__repr__(data={'count': len(self)})


class Pick(BaseObject):
    """ Pick Class

    Handles all pick* methods in the Seletion Class

    >>> from rpw import ui
    >>> ui.Pick.pick_element()
    <rpw:reference>
    >>> ui.Pick.pick_element(multiple=True)
    [<rpw:reference>, ...]
    """

    @classmethod
    def _pick(cls, obj_type, msg='Pick:', multiple=False, linked=False):
        """ Note: Moved Reference Logic to Referenc Wrapper."""

        try:
            if multiple:
                references = PickObjects(obj_type, msg)
            else:
                reference = PickObject(obj_type, msg)
        except RevitExceptions.OperationCanceledException:
            logger.debug('ui.Pick aborted by user')
            sys.exit(0)

        if multiple:
            return [Reference(ref, linked=linked) for ref in references]
        else:
            return Reference(reference, linked=linked)

    @classmethod
    def pick_box(cls, msg, style='directional'):
        """
        PickBox

        Returns:
            XYZ Points (``XYZ``): Min and Max Points of Box
        """
        # This seems kind of usless right now.
```

(continues on next page)

(continued from previous page)

```
        PICK_STYLE = {'crossing': UI.Selection.PickBoxStyle.Crossing,
                      'enclosing': UI.Selection.PickBoxStyle.Enclosing,
                      'directional': UI.Selection.PickBoxStyle.Directional,
                      }

        pick_box = PickBox(PICK_STYLE[style])
        return (XYZ(pick_box.Min), XYZ(pick_box.Max))

    @classmethod
    def pick_by_rectangle(cls, msg):
        """
        PickBox

        Returns:
            Elements (``List``): List of wrapped Elements
        """
        # TODO: Implement ISelectFilter overload
        # NOTE: This is the only method that returns elements
        refs = PickElementsByRectangle(msg)
        return [Element(ref) for ref in refs]

    @classmethod
    def pick_element(cls, msg='Pick Element(s)', multiple=False):
        """
        Pick Element

        Args:
            msg (str): Message to show
            multiple (bool): False to pick single element, True for multiple

        Returns:
            reference (``Reference``): :any:`Reference` Class
        """
        return cls._pick(ObjectType.Element, msg=msg, multiple=multiple)

    @classmethod
    def pick_pt_on_element(cls, msg='Pick Pt On Element(s)', multiple=False):
        """
        Pick Point On Element

        Args:
            msg (str): Message to show
            multiple (bool): False to pick single point, True for multiple

        Returns:
            reference (``Reference``): :any:`Reference` Class
        """
        return cls._pick(ObjectType.PointOnElement, msg=msg, multiple=multiple)

    @classmethod
    def pick_edge(cls, msg='Pick Edge(s)', multiple=False):
        """
        Pick Edge

        Args:
            msg (str): Message to show
            multiple (bool): False to pick single edge, True for multiple
```

(continues on next page)

(continued from previous page)

```
        Returns:
            reference (``Reference``): :any:`Reference` Class
        """
        return cls._pick(ObjectType.Edge, msg=msg, multiple=multiple)

    @classmethod
    def pick_face(cls, msg='Pick Face(s)', multiple=False):
        """
        Pick Face

        Args:
            msg (str): Message to show
            multiple (bool): False to pick single face, True for multiple

        Returns:
            reference (``Reference``): :any:`Reference` Class
        """
        return cls._pick(ObjectType.Face, msg=msg, multiple=multiple)

    @classmethod
    def pick_linked_element(cls, msg='Pick Linked Element', multiple=False):
        """
        Pick Linked Element

        Args:
            msg (str): Message to show
            multiple (bool): False to pick single element, True for multiple

        Returns:
            reference (``Reference``): :any:`Reference` Class
        """
        return cls._pick(ObjectType.LinkedElement, msg=msg, multiple=multiple,␣
↪linked=True)

    @classmethod
    def pick_pt(cls, msg='Pick Point', snap=None):
        """
        Pick Point location

        Args:
            msg (str): Message to show
            snap (str): Snap Options: [endpoints, midpoints, nearest,
                                       workplanegrid, intersections,
                                       centers, perpendicular,
                                       tangents, quadrants, points]

        Returns:
            XYZ (`Xyz`): Rpw XYZ Point
        """

        SNAPS = {
                 # 'none': ObjectSnapTypes.None,
                 'endpoints': ObjectSnapTypes.Endpoints,
                 'midpoints': ObjectSnapTypes.Midpoints,
                 'nearest': ObjectSnapTypes.Nearest,
                 'workplanegrid': ObjectSnapTypes.WorkPlaneGrid,
```

(continues on next page)

(continued from previous page)

```
                 'intersections': ObjectSnapTypes.Intersections,
                 'centers': ObjectSnapTypes.Centers,
                 'perpendicular': ObjectSnapTypes.Perpendicular,
                 'tangents': ObjectSnapTypes.Tangents,
                 'quadrants': ObjectSnapTypes.Quadrants,
                 'points': ObjectSnapTypes.Points,
                 }

        if snap:
            return XYZ(PickPoint(SNAPS[snap], msg))
        else:
            return XYZ(PickPoint(msg))


class SelectionFilter(UI.Selection.ISelectionFilter):
    # http://www.revitapidocs.com/2017.1/d552f44b-221c-0ecd-d001-41a5099b2f9f.htm
    # Also See Ehsan's implementation on pyrevit
    # TODO: Implement ISelectFilter overload
    def __init__(self):
        raise NotImplemented
```

## 4.5 rpw.base

### 4.5.1 Base Wrappers

Base Object Wrapper Class

Most other wrappers inherit from this base class, which has 4 primary responsibilities:

- Instantiates Class and stores wrapped element.
- Provides a `unwrap()` method to return the wrapped object.
- Provides access to all original methods and attributes of the wrapped object through a pass through `__getattr__`
- Implements a `__repr__()` for consistent object representation

Because access to original methods and properties is maintained, you can keep the elements wrapped throughout your code. You would only need to unwrap when when passing the element into function where the original Type is expected.

```
>>> wrapped = BaseObjectWrapper(SomeObject)
>>> wrapped
<RPW_BaseOBjectWrapper:>
>>> wrapped.unwrap()
SomeObject
>>> wrapped.Pinned
False
>>> wrapped.AnyRevitPropertyOrMethod
```

**Warning:** This class is primarily for internal use. If you plan on creating your own wrappers using this base class make sure you read through the documentation first. Misusing this class can cause easily cause Max Recursion Crashes.

**class** `rpw.base.`**`BaseObjectWrapper`**(*revit_object*, *enforce_type=True*)

Bases: `rpw.base.BaseObject`

**Parameters** **`element`** (*`APIObject`*) – Revit Element to store

**`__init__`**(*revit_object*, *enforce_type=True*)

Child classes can use self._revit_object to refer back to Revit Element

> **Warning:** Any Wrapper that inherits and overrides __init__ class MUST ensure `_revit_object` is created by calling super().__init__ before setting any self attributes. Not doing so will cause recursion errors and Revit will crash. BaseObjectWrapper should define a class variable _revit_object_class to define the object class being wrapped.

**`unwrap`**()

Returns the Original Wrapped Element

## Implementation

```
"""
Base Object Wrapper Class

Most other wrappers inherit from this base class,
which has 4 primary responsibilities:

* Instantiates Class and stores wrapped element.
* Provides a ``unwrap()`` method to return the wrapped object.
* Provides access to all original methods and attributes of the
  wrapped object through a pass through ``__getattr__``
* Implements a ``__repr__()`` for consistent object representation

Because access to original methods and properties is maintained, you can keep
the elements wrapped throughout your code. You would only need to unwrap when
when passing the element into function where the original Type is expected.

>>> wrapped = BaseObjectWrapper(SomeObject)
>>> wrapped
<RPW_BaseOBjectWrapper:>
>>> wrapped.unwrap()
SomeObject
>>> wrapped.Pinned
False
>>> wrapped.AnyRevitPropertyOrMethod

Warning:
    This class is primarily for internal use. If you plan on creating your
    own wrappers using this base class make sure you read through the
    documentation first. Misusing this class can cause easilly cause
    Max Recursion Crashes.

"""

import rpw
from rpw.utils.logger import logger
```

(continues on next page)

(continued from previous page)

```python
class BaseObject(object):

        def __init__(self, *args, **kwargs):
            pass

        def ToString(self, *args, **kwargs):
            # Show correct repr on Dynamo
            return self.__repr__(*args, **kwargs)

        # def __dir__(self):
        # TODO: Implement Dir on BaseOBject and BaseObjectWrapper for proper AC
            # return list(self.__dict__)

        # TODO: Clean up repr. remove wraps, add brackets to data
        def __repr__(self, data=''):
            if data:
                data = ' '.join(['{0}:{1}'.format(k, v) for k, v in data.iteritems()])
            return '<rpw:{class_name} | {data}>'.format(
                                                class_name=self.__class__.__name__,
                                                data=data)


class BaseObjectWrapper(BaseObject):
    """
    Arguments:
        element(APIObject): Revit Element to store
    """

    def __init__(self, revit_object, enforce_type=True):
        """
        Child classes can use self._revit_object to refer back to Revit Element

        Warning:
            Any Wrapper that inherits and overrides __init__ class MUST
            ensure ``_revit_object`` is created by calling super().__init__
            before setting any self attributes. Not doing so will
            cause recursion errors and Revit will crash.
            BaseObjectWrapper should define a class variable _revit_object_class
            to define the object class being wrapped.

        """
        _revit_object_class = self.__class__._revit_object_class

        if enforce_type and not isinstance(revit_object, _revit_object_class):
            raise rpw.exceptions.RpwTypeError(_revit_object_class, type(revit_object))

        object.__setattr__(self, '_revit_object', revit_object)

    def __getattr__(self, attr):
        """
        Getter for original methods and properties or the element.
        This method is only called if the attribute name does not
        already exists.
        """
        try:
```

(continues on next page)

(continued from previous page)

~~~
            return getattr(self.__dict__['_revit_object'], attr)
        # except AttributeError:
            # This lower/snake case to be converted.
            # This automatically gives acess to all names in lower case format
            # x.name (if was not already defined, will get x.Name)
            # Note: will not Work for setters, unless defined by wrapper
            # attr_pascal_case = rpw.utils.coerce.to_pascal_case(attr)
            # return getattr(self.__dict__['_revit_object'], attr_pascal_case)
        except KeyError:
            raise rpw.exceptions.RpwException('BaseObjectWrapper is missing _revit_
↪object')

    def __setattr__(self, attr, value):
        """
        Setter allows setting of wrapped object properties, for example
        ```WrappedWall.Pinned = True``
        """
        if hasattr(self._revit_object, attr):
            self._revit_object.__setattr__(attr, value)
        else:
            object.__setattr__(self, attr, value)

    def unwrap(self):
        """ Returns the Original Wrapped Element """
        return self._revit_object

    def __repr__(self, data={}, to_string=None):
        """ ToString can be overriden for objects in which the method is
        not consistent - ie. XYZ.ToString returns pt tuple not Class Name """
        class_name = self.__class__.__name__

        revit_object_name = to_string or self._revit_object.ToString()
        revit_class_name = revit_object_name.split('.')[-1]
        if class_name != revit_class_name:
            class_name = '{} % {}'.format(class_name, revit_class_name)

        data = ''.join([' [{0}:{1}]'.format(k, v) for k, v in data.iteritems()])
        return '<rpw:{class_name}{data}>'.format(class_name=class_name,
                                                 data=data
                                                 )
~~~

## 4.6 rpw.utils

### 4.6.1 Coerce / Type Casting

Type Casting Utilities

rpw.utils.coerce.**to_category**(*category_reference*, *fuzzy=True*)

Coerces a category, category name or category id to a BuiltInCategory.

```
>>> from rpw.utils.coerce import to_category
>>> to_category('OST_Walls')
BuiltInCategory.OST_Walls
>>> to_category('Walls')
```

(continues on next page)

(continued from previous page)

```
BuiltInCategory.OST_Walls
>>> to_category(BuiltInCategory.OST_Walls)
BuiltInCategory.OST_Walls
```

**Parameters** **cateagory_reference** ([`DB.BuiltInCategory`, `str`, `CategoryId`]) – Category Reference or Name

**Returns** BuiltInCategory

**Return type** [`BuiltInCategory`]

`rpw.utils.coerce.`**`to_category_id`**(*category_reference*, *fuzzy=True*)
Coerces a category, category name or category id to a Category Id.

```
>>> from rpw.utils.coerce import to_category_id
>>> to_category_id('OST_Walls')
<ElementId>
>>> to_category_id('Wall')
<ElementId>
>>> to_category_id(BuiltInCategory.OST_Walls)
<ElementId>
```

**Parameters** **cateagory_reference** ([`DB.BuiltInCategory`, `str`, `CategoryId`]) – Category Reference or Name

**Returns** ElementId of Category

**Return type** [`DB.ElementId`]

`rpw.utils.coerce.`**`to_class`**(*class_reference*)
Coerces a class or class reference to a Class.

```
>>> from rpw.utils.coerce import to_class
>>> to_class('Wall')
[ DB.Wall ]
>>> to_class(Wall)
[ DB.Wall ]
```

**Parameters** **class_reference** ([`DB.Wall`, `str`]) – Class Reference or class name

**Returns** Class

**Return type** [`type`]

`rpw.utils.coerce.`**`to_element`**(*element_reference*, *doc=Document*)
Same as to_elements but for a single object

`rpw.utils.coerce.`**`to_element_id`**(*element_reference*)
Coerces Element References (Element, ElementId, . . . ) to Element Id

```
>>> from rpw.utils.coerce import to_element_id
>>> to_element_id(SomeElement)
<Element Id>
```

`rpw.utils.coerce.`**`to_element_ids`**(*element_references*)
Coerces an element or list of elements into element ids. Elements remain unchanged. This will always return a list, even if only one element is passed.

```
>>> from rpw.utils.coerce import to_element_ids
>>> to_element_ids(DB.Element)
[ DB.ElementId ]
>>> to_element_ids(20001)
[ DB.ElementId ]
>>> to_element_ids([20001, 20003])
[ DB.ElementId, DB.ElementId ]
```

**Parameters** **elements** (`DB.Element`) – Iterable list (`list` or `set`) or single of `Element`, `int`.

**Returns** List of Element Ids.

**Return type** [`DB.ElementId`, ... ]

`rpw.utils.coerce.`**`to_elements`**(*element_references*, *doc=Document*)

Coerces element reference (`int`, or `ElementId`) into `DB.Element`. Remains unchanged if it's already `DB.Element`. Accepts single object or lists.

```
>>> from rpw.utils.coerce import to_elements
>>> to_elements(DB.ElementId)
[ DB.Element ]
>>> to_elements(20001)
[ DB.Element ]
>>> to_elements([20001, 20003])
[ DB.Element, DB.Element ]
```

**Parameters** **element_references** ([`DB.ElementId`, `int`, `DB.Element`]) – Element Reference, single or list

**Returns** Elements

**Return type** [`DB.Element`]

`rpw.utils.coerce.`**`to_iterable`**(*item_or_iterable*)

Ensures input is iterable

```
>>> from rpw.utils.coerce import to_iterable
>>> to_iterable(SomeElement)
[SomeElement]
```

**Parameters** **any** (*`iterable, non-iterable`*) –

**Returns** Same as input

**Return type** (*iterable*)

`rpw.utils.coerce.`**`to_pascal_case`**(*snake_str*)

Converts Snake Case to Pascal Case

```
>>> to_pascal_case('family_name')
'FamilyName'
```

### 4.6.2 Mixins

Collection of Class Mixins

**class** rpw.utils.mixins.**ByNameCollectMixin**

Adds name, by_name(), and by_name_or_element_ref() methods. This is for class inheritance only, used to reduce duplication

**classmethod by_name**(*name*)

Mixin to provide instantiating by a name for classes that are collectible. This is a mixin so specifi usage will vary for each for. This method will call the *rpw.db.Element.collect* method of the class, and return the first element with a matching .name property.

```
>>> LinePatternElement.by_name('Dash')
<rpw:LinePatternElement name:Dash>
```

```
>>> FillPatternElement.by_name('Solid')
<rpw:FillPatternElement name:Solid>
```

**classmethod by_name_or_element_ref**(*reference*)

Mixin for collectible elements. This is to help cast elements from name, elemente, or element_id

**name**

Returns object's Name attribute

**class** rpw.utils.mixins.**CategoryMixin**

Adds category and get_category methods.

**_category**

Default Category Access Parameter. Overwrite on wrapper as needed. See Family Wrapper for an example.

**category**

Wrapped DB.Category

**get_category**(*wrapped=True*)

Wrapped DB.Category

### 4.6.3 Logger

Rpw Logger

Usage:

```
>>> from rpw.utils.logger import logger
>>> logger.info('My logger message')
>>> logger.error('My error message')
```

**class** rpw.utils.logger.**LoggerWrapper**

Logger Wrapper to extend loggers functionality. The logger is called in the same as the regular python logger, but also as a few extra features.

```
>>> logger.info('Message')
[INFO]  Message
```

Log Title

```
>>> logger.title('Message')
=========
Message
=========
```

Disable logger

```
>>> logger.disable()
```

Log Errors: This method appends errmsg to self.errors. This allows you to check if an error occured, and if it did not, close console window.

```
>>> logger.error('Message')
[ERROR]  Message
>>> print(logger.errors)
['Message']
```

**critical**(*msg*)
Log Message on logging.CRITICAL level

**debug**(*msg*)
Log Message on logging.DEBUG level

**disable**()
Sets logger level to logging.CRICITAL

**error**(*msg*)
Log Message on logging.ERROR level

**info**(*msg*)
Log Message on logging.INFO level

**title**(*msg*)
Log Message on logging.INFO level with lines above and below

**verbose**(*verbose*)
Sets logger to Verbose.

**Parameters** **(bool)** – True to set *logger.DEBUG*, False to set to *logging.INFO*.

**Usage:**

```
>>> logger.verbose(True)
```

**warning**(*msg*)
Log Message on logging.WARNING level

### 4.6.4 .Net

.NET imports

This module ensures most commonly used .NET classes are loaded for you.for

```
>>> from rpw.utils.dotnet import List, Enum, Process
```

---

### 4.6.5 Implementation

Coerce

```
"""
Type Casting Utilities

"""

import rpw
from rpw import revit, DB
from rpw.base import BaseObjectWrapper
from rpw.db.builtins import BicEnum
from rpw.utils.dotnet import List
from rpw.exceptions import RpwTypeError


def to_element_id(element_reference):
    """
    Coerces Element References (Element, ElementId, ...) to Element Id

    >>> from rpw.utils.coerce import to_element_id
    >>> to_element_id(SomeElement)
    <Element Id>

    """
    if hasattr(element_reference, 'Id'):
        element_id = element_reference.Id
    elif isinstance(element_reference, DB.Reference):
        element_id = element_reference.ElementId
    elif isinstance(element_reference, int):
        element_id = DB.ElementId(element_reference)
    elif isinstance(element_reference, DB.ElementId):
        element_id = element_reference
    elif element_reference == DB.ElementId.InvalidElementId:
        element_id = element_reference
    else:
        raise RpwTypeError('Element, ElementId, or int', type(element_reference))
    return element_id


def to_element_ids(element_references):
    """
    Coerces an element or list of elements into element ids.
    Elements remain unchanged.
    This will always return a list, even if only one element is passed.

    >>> from rpw.utils.coerce import to_element_ids
    >>> to_element_ids(DB.Element)
    [ DB.ElementId ]
    >>> to_element_ids(20001)
    [ DB.ElementId ]
    >>> to_element_ids([20001, 20003])
    [ DB.ElementId, DB.ElementId ]

    Args:
        elements (``DB.Element``): Iterable list (``list`` or ``set``)
                                   or single of ``Element``, ``int``.

    Returns:
        [``DB.ElementId``, ... ]: List of Element Ids.
```

(continues on next page)

(continued from previous page)

```
    """
    element_references = to_iterable(element_references)
    return [to_element_id(e_ref) for e_ref in element_references]

# TODO: Add case to unwrap rpw elements
def to_element(element_reference, doc=revit.doc):
    """ Same as to_elements but for a single object """
    if isinstance(element_reference, DB.Element):
        element = element_reference
    elif isinstance(element_reference, DB.ElementId):
        element = doc.GetElement(element_reference)
    elif isinstance(element_reference, DB.Reference):
        element = doc.GetElement(element_reference)
    elif isinstance(element_reference, int):
        element = doc.GetElement(DB.ElementId(element_reference))
    elif hasattr(element_reference, 'unwrap'):
        element = element_reference.unwrap()
    else:
        raise RpwTypeError('Element, ElementId, or int', type(element_reference))
    return element


def to_elements(element_references, doc=revit.doc):
    """
    Coerces element reference (``int``, or ``ElementId``) into ``DB.Element``.
    Remains unchanged if it's already ``DB.Element``.
    Accepts single object or lists.

    >>> from rpw.utils.coerce import to_elements
    >>> to_elements(DB.ElementId)
    [ DB.Element ]
    >>> to_elements(20001)
    [ DB.Element ]
    >>> to_elements([20001, 20003])
    [ DB.Element, DB.Element ]

    Args:
        element_references ([``DB.ElementId``, ``int``, ``DB.Element``]): Element␣
↪Reference,
                                                                            single or␣
↪list

    Returns:
        [``DB.Element``]: Elements
    """
    element_references = to_iterable(element_references)
    return [to_element(e_ref) for e_ref in element_references]


def to_class(class_reference):
    """ Coerces a class or class reference to a Class.

    >>> from rpw.utils.coerce import to_class
    >>> to_class('Wall')
    [ DB.Wall ]
    >>> to_class(Wall)
    [ DB.Wall ]
```

(continues on next page)

(continued from previous page)

```
    Args:
        class_reference ([``DB.Wall``, ``str``]): Class Reference or class name

    Returns:
        [``type``]: Class
    """
    if isinstance(class_reference, str):
        return getattr(DB, class_reference)
    if isinstance(class_reference, type):
        return class_reference
    raise RpwTypeError('Class Type, Class Type Name', type(class_reference))


def to_category(category_reference, fuzzy=True):
    """ Coerces a category, category name or category id to a BuiltInCategory.

    >>> from rpw.utils.coerce import to_category
    >>> to_category('OST_Walls')
    BuiltInCategory.OST_Walls
    >>> to_category('Walls')
    BuiltInCategory.OST_Walls
    >>> to_category(BuiltInCategory.OST_Walls)
    BuiltInCategory.OST_Walls

    Args:
        cateagory_reference ([``DB.BuiltInCategory``, ``str``, ``CategoryId``]):␣
↪Category Reference
                                                                              or␣
↪Name

    Returns:
        [``BuiltInCategory``]: BuiltInCategory
    """
    if isinstance(category_reference, DB.BuiltInCategory):
        return category_reference
    if isinstance(category_reference, str):
        if fuzzy:
            return BicEnum.fuzzy_get(category_reference)
        else:
            return BicEnum.get(category_reference)
    if isinstance(category_reference, DB.ElementId):
        return BicEnum.from_category_id(category_reference)
    raise RpwTypeError('Category Type, Category Type Name',
                       type(category_reference))


def to_category_id(category_reference, fuzzy=True):
    """
    Coerces a category, category name or category id to a Category Id.

    >>> from rpw.utils.coerce import to_category_id
    >>> to_category_id('OST_Walls')
    <ElementId>
    >>> to_category_id('Wall')
    <ElementId>
    >>> to_category_id(BuiltInCategory.OST_Walls)
```

(continues on next page)

(continued from previous page)

```
    <ElementId>

    Args:
        cateagory_reference ([``DB.BuiltInCategory``, ``str``, ``CategoryId``]):␣
↪Category Reference
                                                                                or␣
↪Name

    Returns:
        [``DB.ElementId``]: ElementId of Category
    """
    category_enum = to_category(category_reference)
    return DB.ElementId(category_enum)


def to_iterable(item_or_iterable):
    """
    Ensures input is iterable

    >>> from rpw.utils.coerce import to_iterable
    >>> to_iterable(SomeElement)
    [SomeElement]

    Args:
        any (iterable, non-iterable)

    Returns:
        (`iterable`): Same as input
    """
    if hasattr(item_or_iterable, '__iter__'):
        return item_or_iterable
    else:
        return [item_or_iterable]


def to_pascal_case(snake_str):
    """ Converts Snake Case to Pascal Case

    >>> to_pascal_case('family_name')
    'FamilyName'
    """
    components = snake_str.split('_')
    return "".join(x.title() for x in components)


# def dictioary_to_string(dictionary):
#     """ Makes a string with key:value pairs from a dictionary

#     >>> dictionary_to_string({'name': 'value'})
#     'name:value'
#     >>> dictionary_to_string({'name': 'value', 'id':5})
#     'name:value id:5'
#     """
#     return ' '.join(['{0}:{1}'.format(k, v) for k, v in dictionary.iteritems()])
```

Mixins

```
""" Collection of Class Mixins """

import rpw
# from rpw import revit, db, DB # Fixes Circular Import
from rpw.exceptions import RpwCoerceError
from rpw.utils.logger import deprecate_warning


class ByNameCollectMixin():

    """ Adds name, by_name(), and by_name_or_element_ref() methods.
    This is for class inheritance only, used to reduce duplication
    """

    @property
    def name(self):
        """ Returns object's Name attribute """
        return self._revit_object.Name

    @classmethod
    def by_name(cls, name):
        """
        Mixin to provide instantiating by a name for classes that are
        collectible. This is a mixin so specifi usage will vary for each for.
        This method will call the :any:`rpw.db.Element.collect`
        method of the class, and return the first element with a
        matching ``.name`` property.

        >>> LinePatternElement.by_name('Dash')
        <rpw:LinePatternElement name:Dash>

        >>> FillPatternElement.by_name('Solid')
        <rpw:FillPatternElement name:Solid>

        """
        e = cls.collect(where=lambda e: e.name.lower() == name.lower()).get_first()
        if e:
            return e
        raise RpwCoerceError('by_name({})'.format(name), cls)

    @classmethod
    def by_name_or_element_ref(cls, reference):
        """
        Mixin for collectible elements.
        This is to help cast elements from name, elemente, or element_id
        """
        if isinstance(reference, str):
            return cls.by_name(reference)
        elif isinstance(reference, rpw.DB.ElementId):
            return rpw.db.Element.from_id(reference)
        else:
            return cls(reference)



class CategoryMixin():
```

(continues on next page)

(continued from previous page)

```
    """ Adds category and get_category methods.
    """

    @property
    def _category(self):
        """
        Default Category Access Parameter. Overwrite on wrapper as needed.
        See Family Wrapper for an example.
        """
        return self._revit_object.Category

    @property
    def category(self):
        """ Wrapped ``DB.Category`` """
        deprecate_warning('.category', 'get_category()')
        return rpw.db.Category(self._category)

    def get_category(self, wrapped=True):
        """ Wrapped ``DB.Category``"""
        return rpw.db.Category(self._category) if wrapped else self._category
```

Logger

```
"""
Rpw Logger

Usage:

    >>> from rpw.utils.logger import logger
    >>> logger.info('My logger message')
    >>> logger.error('My error message')

"""

import sys


class mockLoggerWrapper():
    def __init__(*args, **kwargs):
        pass

    def __getattr__(self, *args, **kwargs):
        return mockLoggerWrapper(*args, **kwargs)

    def __call__(self, *args, **kwargs):
        pass


class LoggerWrapper():
    """
    Logger Wrapper to extend loggers functionality.
    The logger is called in the same as the regular python logger,
    but also as a few extra features.

    >>> logger.info('Message')
    [INFO]  Message
```

(continues on next page)

(continued from previous page)

```
    Log Title

    >>> logger.title('Message')
    =========
    Message
    =========

    Disable logger

    >>> logger.disable()

    Log Errors: This method appends errmsg to self.errors.
    This allows you to check  if an error occured, and if it did not,
    close console window.

    >>> logger.error('Message')
    [ERROR]  Message
    >>> print(logger.errors)
    ['Message']

    """

    def __init__(self):

        handler = logging.StreamHandler(sys.stdout)
        formatter = logging.Formatter("[%(levelname)s] %(message)s")
        # TODO: Show Module
        # formatter = logging.Formatter("[%(levelname)s] %(message)s [%(module)s:
↪%(lineno)s]")
        handler.setFormatter(formatter)

        logger = logging.getLogger('rpw_logger')
        logger.addHandler(handler)
        logger.setLevel(logging.INFO)

        handler_title = logging.StreamHandler(sys.stdout)
        formatter_title = logging.Formatter("%(message)s")
        handler_title.setFormatter(formatter_title)

        logger_title = logging.getLogger('rpw_logger_title')
        logger_title.addHandler(handler_title)
        logger_title.setLevel(logging.INFO)

        self._logger = logger
        self._logger_title = logger_title
        self.errors = []

    def disable(self):
        """ Sets logger level to logging.CRICITAL """
        self._logger.setLevel(logging.CRITICAL)

    def verbose(self, verbose):
        """
        Sets logger to Verbose.

        Args:
            (bool): True to set `logger.DEBUG`, False to set to `logging.INFO`.
```

(continues on next page)

(continued from previous page)

```
        Usage:
            >>> logger.verbose(True)

        """
        if verbose:
            self._logger.setLevel(logging.DEBUG)
        else:
            self._logger.setLevel(logging.INFO)

    def title(self, msg):
        """ Log Message on logging.INFO level with lines above and below """
        print('=' * 100)
        self._logger_title.info(msg)
        print('=' * 100)

    def info(self, msg):
        """ Log Message on logging.INFO level """
        self._logger.info(msg)

    def debug(self, msg):
        """ Log Message on logging.DEBUG level """
        self._logger.debug(msg)

    def warning(self, msg):
        """ Log Message on logging.WARNING level """
        self._logger.warning(msg)

    def error(self, msg):
        """ Log Message on logging.ERROR level """
        self._logger.error(msg)
        self.errors.append(msg)

    def critical(self, msg):
        """ Log Message on logging.CRITICAL level """
        self._logger.critical(msg)

    def setLevel(self, level):
        self._logger.setLevel(level)

def deprecate_warning(depracated, replaced_by=None):
    msg = '{} has been deprecated and will be removed soon.'.format(depracated)
    if replaced_by:
        msg += ' Use {} instead'.format(replaced_by)
    logger.warning(msg)

try:
    import logging
except ImportError:
    # In Dynamo, Use Mock Logger
    logger = mockLoggerWrapper()
else:
    # In PyRevit, Use Logger
    logger = LoggerWrapper()
```

.NET

```
"""
.NET imports

This module ensures most commonly used .NET classes are loaded for you.for

>>> from rpw.utils.dotnet import List, Enum, Process

"""

import sys

from rpw.utils.logger import logger
from rpw.utils.sphinx_compat import MockImporter

# Attempt to Import clr
try:
    import clr
except ImportError:
    # Running Sphinx. Import MockImporter
    logger.warning('Error Importing CLR. Loading Mock Importer')
    sys.meta_path.append(MockImporter())

################
# .NET IMPORTS #
################

import clr
clr.AddReference('System')                 # Enum, Diagnostics
clr.AddReference('System.Collections')     # List

# Core Imports
from System import Enum
from System.Collections.Generic import List
from System.Diagnostics import Process
```

## 4.7 rpw.extras

The Extras module allow you to conveniently load 3rd party .NET assemblies.

### 4.7.1 Rhino

Rhino - Rhino3dmIO

Rhino3dmIO is a subset of RhinoCommon and it gives you access to openNurbs, allowing you to, amongst other things, read and write 3dm files.

Usage:

```
>>> from rpw.extras.rhino import Rhino as rc
>>> pt1 = rc.Geometry.Point3d(0,0,0)
>>> pt2 = rc.Geometry.Point3d(10,10,0)
>>> line1 = rc.Geometry.Line(pt1, pt2)
>>> line1.Length
14.142135623730951
```

(continues on next page)

(continued from previous page)

```
>>>
>>> pt1 = rc.Geometry.Point3d(10,0,0)
>>> pt2 = rc.Geometry.Point3d(0,10,0)
>>> line2 = rc.Geometry.Line(pt1, pt2)
>>>
>>> rc.Geometry.Intersect.Intersection.LineLine(line1, line2)
(True, 0.5, 0.5)
>>>
>>> file3dm = f = rc.FileIO.File3dm()
>>> file3md_options = rc.FileIO.File3dmWriteOptions()
>>> file3dm.Objects.AddLine(line1)
>>> filepath = 'c:/folder/test.3dm'
>>> file3dm.Write(filepath, file3md_options)
```

**Note:** Although the openNURBS toolkit appears to be a full-featured geometry library, it is not. The toolkit does not include a number of important features, including:

- Closest point calculations
- Intersection calculations
- Surface tessellation (meshing)
- Interpolation
- Booleans
- Area and mass property calculations
- Other miscellaneous geometry calculations

[ Note from McNeel's website on openNURBS ]

**More Information about openNURBES in the links below:**

- Github Repo
- RhinoCommon API
- openNURBS

## 4.8 rpw.exceptions

### 4.8.1 Exceptions

Use these exceptions to *try* against specific Rpw Exceptions.

```
>>> from rpw.exceptions import RpwWrongStorageType
>>> try:
...     element.parameters['Height'].value = 'String'
... except RpwWrongStorageType:
...     print('Height Parameter cannot be a string')
...     raise
```

This module also provides easy access to the `Autodesk.Revit.Exceptions` namespaces:

```
>>> from rpw.exceptions import RevitExceptions
>>> try:
...     doc.Delete(ElementId)
... except RevitExceptions.InvalidObjectException:
...     print('This element is no longer valid ')
```

**exception** rpw.exceptions.**RpwCoerceError**(*value*, *target_type*)
Bases: *rpw.exceptions.RpwException*, exceptions.ValueError

Coerce Error

**exception** rpw.exceptions.**RpwException**
Bases: exceptions.Exception

Revit Python Wrapper Base Exception

**exception** rpw.exceptions.**RpwParameterNotFound**(*element*, *param_name*)
Bases: *rpw.exceptions.RpwException*, exceptions.KeyError

Revit Python Wrapper Parameter Error

**exception** rpw.exceptions.**RpwTypeError**(*type_expected*, *type_received='not reported'*)
Bases: exceptions.TypeError

Revit Python Wrapper Type Exception

**exception** rpw.exceptions.**RpwValueError**(*value_expected*, *value_received='not reported'*)
Bases: exceptions.ValueError

Revit Python Wrapper Value Error Exception

**exception** rpw.exceptions.**RpwWrongStorageType**(*storage_type*, *value*)
Bases: *rpw.exceptions.RpwException*, exceptions.TypeError

Wrong Storage Type

### 4.8.2 Implementation

```

from rpw.utils.logger import logger
# Added on 1.7.4
# Since adding Autodesk.Revit, it became impossible to run any non-revit
# tools such as forms.os_forms, etc using run_forms.BaseException
# This needs clean up
from rpw.utils.sphinx_compat import MockObject
try:
    from Autodesk.Revit import Exceptions as RevitExceptions
except ImportError:
    RevitExceptions = MockObject()


class RpwException(Exception):
    """ Revit Python Wrapper Base Exception """


class RpwTypeError(TypeError):
    """ Revit Python Wrapper Type Exception """
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, type_expected, type_received='not reported'):
        msg = 'expected [{}], got [{}]'.format(type_expected, type_received)
        super(RpwTypeError, self).__init__(msg)


class RpwValueError(ValueError):
    """ Revit Python Wrapper Value Error Exception """
    def __init__(self, value_expected, value_received='not reported'):
        msg = 'expected [{}], got [{}]'.format(value_expected, value_received)
        super(RpwValueError, self).__init__(msg)


class RpwParameterNotFound(RpwException, KeyError):
    """ Revit Python Wrapper Parameter Error """
    def __init__(self, element, param_name):
        msg = 'parameter not found [element:{}]:[param_name:{}]'.format(
                                                    element.Id, param_name)
        super(RpwParameterNotFound, self).__init__(msg)


class RpwWrongStorageType(RpwException, TypeError):
    """ Wrong Storage Type """
    def __init__(self, storage_type, value):
        msg = 'Wrong Storage Type: [{}]:[{}:{}]'.format(storage_type,
                                                    type(value), value)
        super(RpwWrongStorageType, self).__init__(msg)


class RpwCoerceError(RpwException, ValueError):
    """ Coerce Error """
    def __init__(self, value, target_type):
        msg = 'Could not cast value:{} to target_type:{}'.format(value,
                                                             target_type)
        super(RpwCoerceError, self).__init__(msg)
```

## 4.9 Known Issues

- **Inconsisten Returns** The library is not consistent with its return types. Sometimes a wrapped element is returned, sometimes is unwrapped. Since fixing this would be a breaking change, I am panning on fixing this on the next major release (2.0) The main change will be that attributes that were previously properties, will become a get method with an optional kwarg for wrapped:

  ```
  >>> # Previous
  >>> instance.family
  >>> # 2.0
  >>> instance.get_family()
  ```

- Case Sensitive ElementParameterFilter

  The boolean argument for case_sensitive string comparison has no effect
  I achieved the same result using the RevitPythonShell, so this could
  be either a RevitAPI bug, or a IronPython/API issue.

  To Reproduce:

```
>>> Assumes an element with a parameter with string value: "Test"
>>> param_id = SomeElementId
>>> value = 'test'
>>> case_sensitive = True
```

```
>>> rule = DB.ParameterFilterRuleFactory.CreateBeginsWithRule(param_id,␣
↪value, case_sensitive)
>>> filter_rule = DB.ElementParameterFilter(rule)
>>> col = FilteredElementCollector(doc).WherePasses(filter_rule)
```

Expected:

*col* would not include element with parameter with value 'Test' with case_sensitive is True.

**Result:** *col* always is always case insensitive.

## 4.10 Tests

The Test Suite below is used to verify functionalities, as well as to validate compatibility across different platforms. These tests below have been executed without failures on:

- Revit 2015
- Revit 2016
- Revit 2017
- Dynamo 1.2 / 1.3

The Test Suite also provides a many examples of how the library is intended to be used.

### 4.10.1 Test Suite

```
"""
Globals

Passes:
 * 2017.1

Revit Python Wrapper
github.com/gtalarico/revitpythonwrapper
revitpythonwrapper.readthedocs.io

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
```

(continues on next page)

(continued from previous page)

```
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

Copyright 2017 Gui Talarico

"""

import sys
import unittest
import os


######################
# Globals
######################


class Globals(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        import rpw

    def setUp(self):
        pass

    def tearDown(self):
        pass

    def test_doc(self):
        from rpw import revit
        self.assertEqual(revit.doc.__class__.__name__, 'Document')

    def test_db(self):
        from rpw import revit, DB
        Wall = getattr(DB, 'Wall', None)
        self.assertIsInstance(Wall, type)

    def test_ui(self):
        from rpw import revit, UI
        TaskDialog = getattr(UI, 'TaskDialog', None)
        self.assertIsInstance(TaskDialog, type)

    def test_uidoc(self):
        from rpw import revit
        self.assertEqual(revit.uidoc.Application.__class__.__name__, 'UIApplication')

    def test_logger(self):
        from rpw.utils.logger import logger
        from rpw.utils.logger import LoggerWrapper
        self.assertIsInstance(logger, LoggerWrapper)

    #TODO: test version
```

(continues on next page)

(continued from previous page)

```
    #TODO: test built
```

```
""" Revit Python Wrapper Tests - Forms

Passes:
2017

"""

import sys
import unittest
import os

test_dir = os.path.dirname(__file__)
root_dir = os.path.dirname(test_dir)
sys.path.append(root_dir)

import rpw
from rpw import revit, DB, UI
doc, uidoc = rpw.revit.doc, rpw.revit.uidoc

from rpw.ui.forms.taskdialog import TaskDialog, CommandLink, Alert
from rpw.exceptions import RpwParameterNotFound, RpwWrongStorageType
from rpw.utils.logger import logger

######################
# Task Dialog
######################


class TaskDialogTests(unittest.TestCase):

    def test_basic(self):
        commands = [CommandLink('Select This Option', return_value='XXX',
                                subtext='Subtext 1'),
                    CommandLink('Option 2 - Should see subtext',
                                subtext='Subtext 2')
                    ]

        dialog = TaskDialog('Test 1 - Full',
                            title='My Title - Footer is ON. No Close Btn',
                            content='X Close IS SHOWING',
                            footer='Foot Text',
                            verification_text='Check This',
                            show_close=True,
                            commands=commands)
        result = dialog.show()
        self.assertEqual(result, 'XXX')
        self.assertEqual(dialog.Title, 'My Title - Footer is ON. No Close Btn')
        self.assertEqual(dialog.MainInstruction, 'Test 1 - Full')
        self.assertEqual(dialog.MainContent, 'X Close IS SHOWING')
        self.assertEqual(dialog.FooterText, 'Foot Text')
        self.assertEqual(dialog.verification_checked, True)
```

(continues on next page)

(continued from previous page)

```
    def test_func_return_value(self):
        commands = [CommandLink('Select This Option',
                                return_value=lambda: 'ZZZ',
                                subtext='Subtext 1'),
                    CommandLink('Option 2', subtext='Subtext 2')
                    ]
        dialog = TaskDialog('Test 2 - Simple',
                            verification_text='Leave this off',
                            commands=commands,
                            content='X Close Should NOT be Showing',
                            show_close=False)
        result = dialog.show()
        self.assertEqual(result(), 'ZZZ')
        self.assertEqual(dialog.verification_checked, False)

    def test_func_defaultvalue_button(self):
        commands = [CommandLink('Press This Button')]
        dialog = TaskDialog('Test 3',
                            content='Press Button Below',
                            commands=commands,
                            buttons=['Cancel'])
        result = dialog.show()
        self.assertEqual(result, 'Press This Button')
        self.assertEqual(dialog.verification_checked, None)

    def test_func_all_buttons_retry(self):
        dialog = TaskDialog('Test 4',
                            content='Press Retry',
                            buttons=['Ok', 'Yes', 'No',
                                     'Cancel', 'Retry', 'Close'])
        result = dialog.show()
        self.assertEqual(result, 'Retry')

    def test_func_all_buttons_close(self):
        dialog = TaskDialog('Test 5',
                            content='Press Close',
                            buttons=['Ok', 'Yes', 'No',
                                     'Cancel', 'Retry', 'Close'])
        result = dialog.show()
        self.assertEqual(result, 'Close')

    def test_func_all_buttons_cancel(self):
        dialog = TaskDialog('Test 6',
                            content='Press Cancel',
                            buttons=['Ok', 'Yes', 'No',
                                     'Cancel', 'Retry', 'Close'])
        result = dialog.show()
        self.assertEqual(result, None)

    def test_close(self):
        dialog = TaskDialog('Test 7 - Exit',
                            content="Close Using X",
                            buttons=[],
                            show_close=True)
        with self.assertRaises(SystemExit):
            result = dialog.show(exit=True)
```

(continues on next page)

(continued from previous page)

```
    def test_close_cancel(self):
            dialog = TaskDialog('Test 8 - Exit',
                                content="Close Using Cancel",
                                buttons=['Cancel'],
                                show_close=False)
            with self.assertRaises(SystemExit):
                result = dialog.show(exit=True)


    # def test_close_for_docs(self):
    #         commands= [CommandLink('Open Dialog', return_value='Open'),
    #                    CommandLink('Command', return_value=lambda: True)]

    #         dialog = TaskDialog('This TaskDialog has Buttons ',
    #                             title_prefix=False,
    #                             content="Further Instructions",
    #                             commands=commands,
    #                             buttons=['Cancel', 'OK', 'RETRY'],
    #                             footer='It has a footer',
    #                             # verification_text='And Verification Text',
    #                             # expanded_content='And Expanded Content',
    #                             show_close=True)
    #         dialog.show()

class AlertTests(unittest.TestCase):

    def test_alert(self):
        alert = Alert('my message - press close',
                      title='My Title',
                      header='My Header',
                      exit=False)
        self.assertIsInstance(alert, Alert)
        self.assertIsInstance(alert.result, UI.TaskDialogResult)


    def test_alert_exit_on_close(self):
        with self.assertRaises(SystemExit):
            Alert('my message - press close - will exit',
                  title='My Title',
                  header='My Header',
                  exit=True)
```

```
"""
Collector Tests

Passes:
 * 2017.1

Revit Python Wrapper
github.com/gtalarico/revitpythonwrapper
revitpythonwrapper.readthedocs.io
```

(continues on next page)

(continued from previous page)

```
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

Copyright 2017 Gui Talarico

"""

import sys
import unittest
import os

parent = os.path.dirname

script_dir = parent(__file__)
panel_dir = parent(script_dir)
sys.path.append(script_dir)

import rpw
from rpw import revit, DB, UI

doc, uidoc = revit.doc, revit.uidoc

from rpw.utils.dotnet import List
from rpw.exceptions import RpwParameterNotFound, RpwWrongStorageType
from rpw.utils.logger import logger

import test_utils

def setUpModule():
    logger.title('SETTING UP COLLECTOR TESTS...')
    logger.title('REVIT {}'.format(revit.version))
    uidoc.Application.OpenAndActivateDocument(os.path.join(panel_dir, 'collector.rvt
↪'))
    test_utils.delete_all_walls()
    test_utils.make_wall()

def tearDownModule():
    test_utils.delete_all_walls()

######################
# COLLECTOR
```

(continues on next page)

(continued from previous page)

```
######################


class CollectorTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING COLLECTOR...')
        collector = DB.FilteredElementCollector(doc)
        cls.family_loaded = collector.OfCategory(DB.BuiltInCategory.OST_Furniture).
↪ToElements()

    @staticmethod
    def collector_helper(filters):
        logger.debug('{}'.format(filters))
        collector = rpw.db.Collector(**filters)
        elements = collector.elements
        logger.debug(collector)
        if collector:
            logger.debug(collector[0])
        return collector

    def setUp(self):
        self.collector_helper = CollectorTests.collector_helper

    def test_collector_elements(self):
        x = self.collector_helper({'of_class': DB.View})
        assert isinstance(x.elements[0], DB.View)

    def test_collector_elements_view_element(self):
        x = self.collector_helper({'of_class': DB.Wall, 'view': uidoc.ActiveView})
        self.assertEqual(len(x), 1)

    def test_collector_elements_view_element_another(self):
        # Id of view where everything is hidden
        view_hidden = doc.GetElement(DB.ElementId(12531))
        x = self.collector_helper({'of_class': DB.Wall, 'view': view_hidden})
        self.assertEqual(len(x), 0)

    def test_collector_elements_view_id(self):
        x = self.collector_helper({'of_class': DB.Wall, 'view': uidoc.ActiveView.Id})
        self.assertEqual(len(x), 1)

    def test_collector_len(self):
        x = self.collector_helper({'of_class': DB.View})
        assert len(x) > 1

    def test_collector_first(self):
        x = self.collector_helper({'of_class': DB.View})
        assert isinstance(x.get_first(wrapped=False), DB.View)

    def test_collector_caster(self):
        x = self.collector_helper({'of_class': DB.Wall}).elements[0]
        assert isinstance(x, DB.Wall)
        y = self.collector_helper({'of_class': 'Wall'}).elements[0]
        assert isinstance(y, DB.Wall)
```

(continues on next page)

(continued from previous page)

```
    def test_collector_is_element(self):
        walls = self.collector_helper({'of_category': 'OST_Walls',
                                       'is_not_type': True})
        assert all([isinstance(x, DB.Wall) for x in walls.elements])

    def test_collector_is_element_false(self):
        walls = self.collector_helper({'of_category': 'OST_Walls',
                                       'is_not_type': False})
        assert all([isinstance(x, DB.WallType) for x in walls.elements])

    def test_collector_is_element_type(self):
        walls = self.collector_helper({'of_category': 'OST_Walls',
                                       'is_type': True})
        assert all([isinstance(x, DB.WallType) for x in walls.elements])

    def test_collector_is_element_type_false(self):
        walls = self.collector_helper({'of_category': 'OST_Walls',
                                       'is_type': False})
        assert all([isinstance(x, DB.Wall) for x in walls.elements])

    def test_collector_is_view_dependent(self):
        fregions = self.collector_helper({'of_category': 'OST_FilledRegion'})
        assert all([f.ViewSpecific for f in fregions.elements])
        view_dependent = self.collector_helper({'is_view_independent': True})
        assert not all([f.ViewSpecific for f in view_dependent.elements])

    # def test_collector_chained_calls(self):
    #     wall_collector = self.collector_helper({'of_category': DB.BuiltInCategory.
↪OST_Walls})
    #     walls_category = len(wall_collector)
    #     wall_collector.filter(is_not_type=True)
    #     walls_elements = len(wall_collector)
    #     wall_collector.filter(is_type=True)
    #     walls_element_type = len(wall_collector)
    #     assert walls_category > walls_elements > walls_element_type

    def tests_collect_rooms(self):
        collector = rpw.db.Collector(of_category='OST_Rooms')
        if collector:
            self.assertIsInstance(collector.get_first(wrapped=False), DB.
↪SpatialElement)
            collector = rpw.db.Collector(of_class='SpatialElement')
            self.assertIsInstance(collector.get_first(wrapped=False), DB.Architecture.
↪Room)

    def test_collector_scope_elements(self):
        """ If Collector scope is list of elements, should not find View"""
        wall = rpw.db.Collector(of_class='Wall').get_first(wrapped=False)
        collector = rpw.db.Collector(elements=[wall], of_class='View')
        self.assertEqual(len(collector), 0)

    def test_collector_scope_element_ids(self):
        wall = rpw.db.Collector(of_class='Wall').get_first(wrapped=False)
        collector = rpw.db.Collector(element_ids=[wall.Id], of_class='View')
        self.assertEqual(len(collector), 0)

    def test_collector_symbol_filter(self):
```

(continues on next page)

(continued from previous page)

```
        desk_types = rpw.db.Collector(of_class='FamilySymbol',
                                      of_category="OST_Furniture").elements
        self.assertEqual(len(desk_types), 3)

        all_symbols = rpw.db.Collector(of_class='FamilySymbol').elements
        self.assertGreater(len(all_symbols), 3)
        all_symbols = rpw.db.Collector(of_class='FamilySymbol').elements

        #Placed Twice
        first_symbol = rpw.db.Collector(symbol=desk_types[0]).elements
        self.assertEqual(len(first_symbol), 2)

        #Placed Once
        second_symbol = rpw.db.Collector(symbol=desk_types[1]).elements
        self.assertEqual(len(second_symbol), 1)

        second_symbol = rpw.db.Collector(of_class='Wall', symbol=desk_types[1]).
↪elements
        self.assertEqual(len(second_symbol), 0)

##############################
# Built in Element Collector #
##############################

class BuiltInCollectorTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING ELEMENT COLLECTOR...')

    def test_element_collector_wall(self):
        walls = rpw.db.Wall.collect()
        self.assertEqual(len(walls), 1)
        self.assertIsInstance(walls.get_first(wrapped=False), DB.Wall)

    def test_element_collector_wallsymbols(self):
        wallsymbols = rpw.db.WallType.collect()
        self.assertEqual(len(wallsymbols), 4)
        self.assertIsInstance(wallsymbols.get_first(wrapped=False), DB.WallType)

    def test_element_collector_Room(self):
        rooms = rpw.db.Room.collect()
        self.assertEqual(len(rooms), 2)
        self.assertIsInstance(rooms.get_first(wrapped=False), DB.Architecture.Room)

    def test_element_collector_Area(self):
        areas = rpw.db.Area.collect()
        self.assertEqual(len(areas), 1)
        self.assertIsInstance(areas.get_first(wrapped=False), DB.Area)

    def test_element_collector_AreaScheme(self):
        areas = rpw.db.AreaScheme.collect()
        self.assertEqual(len(areas), 2)
        self.assertIsInstance(areas.get_first(wrapped=False), DB.AreaScheme)


############################
```

(continues on next page)

(continued from previous page)

```
# COLLECTOR PARAMETER FILTER
############################

class ParameterFilterTests(unittest.TestCase):

    @classmethod
    def setUpClass(self):
        logger.title('TESTING PARAMETER FILTER...')

    def setUp(self):
        self.wall = rpw.db.Collector(of_class='Wall').get_first(wrapped=False)
        self.wrapped_wall = rpw.db.Element(self.wall)
        with rpw.db.Transaction('Set Comment'):
            self.wrapped_wall.parameters['Comments'].value = 'Tests'
            self.wrapped_wall.parameters['Unconnected Height'].value = 12.0

        # BIP Ids

        self.param_id_height = rpw.db.builtins.BipEnum.get_id('WALL_USER_HEIGHT_PARAM
↪')
        self.param_id_location = rpw.db.builtins.BipEnum.get_id('WALL_KEY_REF_PARAM')
        self.param_id_comments = rpw.db.builtins.BipEnum.get_id('ALL_MODEL_INSTANCE_
↪COMMENTS')
        self.param_id_level_name = rpw.db.builtins.BipEnum.get_id('DATUM_TEXT')

    def tearDown(self):
        pass

    def test_param_filter_float_less_no(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_height, less=10.0)
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 0)

    def test_param_filter_float_less_yes(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_height, less=15.0)
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 1)

    def test_param_filter_float_equal(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_height, equals=12.0)
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 1)

    def test_param_filter_float_not_equal(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_height, not_equals=12.
↪0)
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 0)

    def test_param_filter_float_greater(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_height, greater=10.0)
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 1)

    def test_param_filter_float_multi_filter(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_height, greater=10.0,␣
↪less=14.0)
```

(continues on next page)

(continued from previous page)

```
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 1)

    def test_param_filter_float_multi_filter(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_height, greater=10.0,␣
↪not_less=14.0)
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 0)

    def test_param_filter_int_equal(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_location, equals=0)
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 1)

    def test_param_filter_int_less(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_location, less=3)
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)

        self.assertEqual(len(col), 1)

    def test_param_comments_equals(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_comments, equals=
↪'Tests')
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 1)

    def test_param_comments_not_equals(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_comments, equals='Blaa
↪')
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 0)

    def test_param_comments_begins(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_comments, begins='Tes
↪')
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 1)

    def test_param_comments_not_begins(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_comments, equals='Bla␣
↪bla')
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 0)

    def test_param_comments_not_begins(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_comments, not_begins=
↪'Bla bla')
        col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        self.assertEqual(len(col), 1)

    # FAILS - CASE SENSITIVE FLAG IS NOT WORKING
    # def test_param_comments_equal_case(self):
        # parameter_filter = rpw.db.ParameterFilter(self.param_id_comments, contains=
↪'tests')
        # col = rpw.db.Collector(of_class="Wall", parameter_filter=parameter_filter)
        # self.assertEqual(len(col), 0)
```

(continues on next page)

(continued from previous page)

```
    def tests_param_name_contains(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_level_name, contains=
↪'1')
        col = rpw.db.Collector(of_category="OST_Levels", parameter_filter=parameter_
↪filter)
        self.assertEqual(len(col), 1)

    def tests_param_name_ends(self):
        parameter_filter = rpw.db.ParameterFilter(self.param_id_level_name, ends='1')
        col = rpw.db.Collector(of_category="OST_Levels", parameter_filter=parameter_
↪filter)
        self.assertEqual(len(col), 1)

    def tests_param_id_coerce(self):
        """ Uses Param Name instead of Param Id. Works only for BIP """
        param_name = 'DATUM_TEXT'
        parameter_filter = rpw.db.ParameterFilter(param_name, ends='1')
        col = rpw.db.Collector(of_category="OST_Levels", parameter_filter=parameter_
↪filter)
        self.assertEqual(len(col), 1)

    def test_from_parameter_name(self):
        """ Uses LooksUp Parameter from sample element """
        level = rpw.db.Collector(of_category="OST_Levels", is_type=False).get_
↪first(wrapped=False)
        parameter_filter = rpw.db.ParameterFilter.from_element_and_parameter(level,
↪'Name', ends='1')
        col = rpw.db.Collector(of_category="OST_Levels", parameter_filter=parameter_
↪filter)
        self.assertEqual(len(col), 1)


class FilteredCollectorCompareTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING COLLECTOR...')

    def test_category(self):
        rv = DB.FilteredElementCollector(doc).OfCategory(DB.BuiltInCategory.OST_
↪Levels).WhereElementIsElementType().ToElements()
        rv2 = rpw.db.Collector(of_category="Levels", is_type=True)
        self.assertEqual(len(rv), len(rv2))

    def test_category2(self):
        rv = DB.FilteredElementCollector(doc).OfCategory(DB.BuiltInCategory.OST_
↪Levels).WhereElementIsNotElementType().ToElements()
        rv2 = rpw.db.Collector(of_category="Levels", is_type=False)
        self.assertEqual(len(rv), len(rv2))

    def test_class(self):
        rv = DB.FilteredElementCollector(doc).OfClass(DB.View).ToElements()
        rv2 = rpw.db.Collector(of_class="View")
        self.assertEqual(len(rv), len(rv2))

    def test_excludes(self):
        e = DB.FilteredElementCollector(doc).OfClass(DB.View).FirstElement()
```

(continues on next page)

(continued from previous page)

```
        e = List[DB.ElementId]([e.Id])
        rv = DB.FilteredElementCollector(doc).OfClass(DB.View).Excluding(e).
↪ToElements()

        e = rpw.db.Collector(of_class="View").wrapped_elements[0]
        rv2 = rpw.db.Collector(of_class="View", exclude=e)
        rv3 = rpw.db.Collector(of_class="View", exclude=[e])
        rv4 = rpw.db.Collector(of_class="View", exclude=e.unwrap())
        rv5 = rpw.db.Collector(of_class="View", exclude=[e.unwrap()])
        rv6 = rpw.db.Collector(of_class="View", exclude=e.Id)
        rv7 = rpw.db.Collector(of_class="View", exclude=[e.Id])

        self.assertEqual(len(rv), len(rv2))
        self.assertEqual(len(rv), len(rv3))
        self.assertEqual(len(rv), len(rv4))
        self.assertEqual(len(rv), len(rv5))
        self.assertEqual(len(rv), len(rv6))
        self.assertEqual(len(rv), len(rv7))

    def test_and(self):
        col1 = DB.FilteredElementCollector(doc).OfClass(DB.FamilySymbol)
        col2 = DB.FilteredElementCollector(doc).OfCategory(DB.BuiltInCategory.OST_
↪Furniture).IntersectWith(col1)
        rv = col2.ToElements()

        e = rpw.db.Collector(of_class="FamilySymbol")
        rv2 = rpw.db.Collector(of_category='Furniture', and_collector=e)

        self.assertEqual(len(rv), len(rv2))
        self.assertEqual(rv[0].Id, rv2[0].Id)
        self.assertEqual(rv[1].Id, rv2[1].Id)
        self.assertEqual(rv[2].Id, rv2[2].Id)

    def test_or(self):
        col1 = DB.FilteredElementCollector(doc).OfClass(DB.View)
        col2 = DB.FilteredElementCollector(doc).OfCategory(DB.BuiltInCategory.OST_
↪Furniture).UnionWith(col1)
        rv = col2.ToElements()

        e = rpw.db.Collector(of_class="View")
        rv2 = rpw.db.Collector(of_category='Furniture', or_collector=e)

        self.assertEqual(len(rv), len(rv2))
        self.assertEqual(rv[0].Id, rv2[0].Id)
        self.assertEqual(rv[1].Id, rv2[1].Id)
        self.assertEqual(rv[2].Id, rv2[2].Id)

    # TODO: Fo all FilteredElementCollector
```

```
"""
Collector Tests

Passes:
 * 2017.1
```

(continues on next page)

(continued from previous page)

```
Revit Python Wrapper
github.com/gtalarico/revitpythonwrapper
revitpythonwrapper.readthedocs.io

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

Copyright 2017 Gui Talarico

"""

import sys
import unittest
import os

parent = os.path.dirname

script_dir = parent(__file__)
panel_dir = parent(script_dir)
sys.path.append(script_dir)

import rpw
from rpw import revit, DB, UI

doc, uidoc = revit.doc, revit.uidoc

from rpw.utils.dotnet import List
from rpw.db.xyz import XYZ
from rpw.exceptions import RpwParameterNotFound, RpwWrongStorageType
from rpw.utils.logger import logger

import test_utils

def setUpModule():
    logger.title('SETTING UP COLLECTION TESTS...')

def tearDownModule():
    test_utils.delete_all_walls()

######################
# ElementSet
######################
```

(continues on next page)

(continued from previous page)

```python
class ElementSetTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING ElementSetTests...')
        collector = DB.FilteredElementCollector(doc)
        cls.views = collector.OfClass(DB.View).ToElements()

    # def setUp(self):
        # self.collector_helper = CollectorTests.collector_helper


    def test_element_set_element_add(self):
        rv = rpw.db.ElementSet()
        rv.add(self.views)
        self.assertEqual(len(rv), len(self.views))

    def test_element_set_unique(self):
        rv = rpw.db.ElementSet()
        rv.add(self.views)
        rv.add(self.views)
        self.assertEqual(len(rv), len(self.views))

    def test_element_set_init__bool(self):
        x = rpw.db.ElementSet(self.views)
        self.assertTrue(x)

    def test_element_set_elements(self):
        x = rpw.db.ElementSet(self.views)
        self.assertIsInstance(x.elements[0], DB.View)

    def test_element_set_element_ids(self):
        x = rpw.db.ElementSet(self.views)
        self.assertIsInstance(x.element_ids[0], DB.ElementId)

    def test_element_set_len(self):
        rv = len(rpw.db.ElementSet(self.views))
        self.assertGreater(rv, 2)

    def test_element_set_element_clear(self):
        rv = rpw.db.ElementSet(self.views)
        rv.clear()
        self.assertEqual(len(rv), 0)

    def test_element_set_as_element_list(self):
        rv = rpw.db.ElementSet(self.views)
        l = rv.as_element_list
        self.assertTrue(hasattr(l, 'Count'))
        self.assertEqual(len(l), len(self.views))

    def test_element_set_as_element_id_list(self):
        rv = rpw.db.ElementSet(self.views)
        l = rv.as_element_id_list
        self.assertTrue(hasattr(l, 'Count'))
        self.assertEqual(len(l), len(self.views))
```

(continues on next page)

(continued from previous page)

```
    def test_element_set_select(self):
        rv = rpw.db.ElementSet(self.views)
        rv.select()

    def test_element_set_get_item(self):
        rv = rpw.db.ElementSet(self.views)
        key = self.views[0]
        self.assertIsInstance(rv[key].unwrap(), DB.View)

    def test_element_set_iter(self):
        rv = rpw.db.ElementSet(self.views)
        self.assertTrue(all([isinstance(v.unwrap(), DB.View) for v in rv]))

    def test_element_set_pop(self):
        rv = rpw.db.ElementSet(self.views)
        id_ = self.views[0].Id
        poped = rv.pop(id_)
        self.assertNotIn(id_, rv)
        self.assertEqual(poped.Id, id_)
        self.assertIsInstance(poped.unwrap(), DB.View)

    def test_element_set_wrapped_elements(self):
        rv = rpw.db.ElementSet(self.views).wrapped_elements
        self.assertIsInstance(rv[0], rpw.db.Element)



######################
# ElementCollection
######################

class ElementCollectionTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING ElementCollection...')
        collector = DB.FilteredElementCollector(doc)
        cls.views = collector.OfClass(DB.View).ToElements()

    def test_element_collection_element_add(self):
        rv = rpw.db.ElementCollection()
        rv.append(self.views)
        self.assertEqual(len(rv), len(self.views))

    def test_element_collection_unique(self):
        rv = rpw.db.ElementCollection()
        rv.append(self.views)
        rv.append(self.views)
        self.assertEqual(len(rv), len(self.views)*2)

    def test_element_collection_init__bool(self):
        x = rpw.db.ElementCollection(self.views)
        self.assertTrue(x)

    def test_element_collection_elements(self):
        x = rpw.db.ElementCollection(self.views)
        self.assertIsInstance(x.elements[0].unwrap(), DB.View)
```

(continues on next page)

(continued from previous page)

```
    def test_element_collection_element_ids(self):
        x = rpw.db.ElementCollection(self.views)
        self.assertIsInstance(x.element_ids[0], DB.ElementId)

    def test_element_collection_len(self):
        rv = len(rpw.db.ElementCollection(self.views))
        self.assertGreater(rv, 2)

    def test_element_collection_element_clear(self):
        rv = rpw.db.ElementCollection(self.views)
        rv.clear()
        self.assertEqual(len(rv), 0)

    def test_element_collection_as_element_list(self):
        rv = rpw.db.ElementCollection(self.views)
        l = rv.as_element_list
        self.assertTrue(hasattr(l, 'Count'))
        self.assertEqual(len(l), len(self.views))

    def test_element_collection_as_element_id_list(self):
        rv = rpw.db.ElementCollection(self.views)
        l = rv.as_element_id_list
        self.assertTrue(hasattr(l, 'Count'))
        self.assertEqual(len(l), len(self.views))

    def test_element_collection_select(self):
        rv = rpw.db.ElementCollection(self.views)
        rv.select()

    def test_element_collection_first(self):
        rv = rpw.db.ElementCollection(self.views)
        self.assertEqual(rv.get_first(wrapped=False).Id, self.views[0].Id)

    def test_element_collection_get_item(self):
        rv = rpw.db.ElementCollection(self.views)
        self.assertIsInstance(rv[0].unwrap(), DB.View)

    def test_element_collection_iter(self):
        rv = rpw.db.ElementCollection(self.views)
        self.assertTrue(all([isinstance(v.unwrap(), DB.View) for v in rv]))

    def test_element_collection_pop(self):
        col = rpw.db.ElementCollection(self.views)
        size = len(col)
        e = col.pop(0, wrapped=False)

        self.assertIsInstance(e, DB.View)
        self.assertEqual(len(col), size - 1)

    def test_element_collection_wrapped_elements(self):
        rv = rpw.db.ElementSet(self.views).wrapped_elements
        self.assertIsInstance(rv[0], rpw.db.Element)


######################
# XYZCollection
```

(continues on next page)

(continued from previous page)

```
######################

class XyzCollectionTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING XYZ Collection...')
        cls.points = [XYZ(0,0,0), XYZ(10,10,0), XYZ(5,5,0)]

    def test_xyz_add_len(self):
        xyz_collection = rpw.db.XyzCollection(self.points)
        self.assertEqual(len(xyz_collection), 3)

    def test_xyz_max(self):
        xyz_collection = rpw.db.XyzCollection(self.points)
        mx = xyz_collection.max
        self.assertEqual(mx, XYZ(10,10,0))

    def test_xyz_min(self):
        xyz_collection = rpw.db.XyzCollection(self.points)
        mn = xyz_collection.min
        self.assertEqual(mn, XYZ(0,0,0))

    def test_xyz_average(self):
        xyz_collection = rpw.db.XyzCollection(self.points)
        av = xyz_collection.average
        self.assertEqual(av, XYZ(5,5,0))

    def test_xyz_sorted_by(self):
        xyz_collection = rpw.db.XyzCollection(self.points)
        rv = xyz_collection.sorted_by('x')
        self.assertEqual(rv[0], XYZ(0,0,0))
        self.assertEqual(rv[1], XYZ(5,5,0))
        self.assertEqual(rv[2], XYZ(10,10,0))
```

```
"""
Curve Tests

Passes:
 * 2017.1

Revit Python Wrapper
github.com/gtalarico/revitpythonwrapper
revitpythonwrapper.readthedocs.io

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
```

(continues on next page)

(continued from previous page)

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

Copyright 2017 Gui Talarico

"""

import sys
import unittest
import os

parent = os.path.dirname

script_dir = parent(__file__)
panel_dir = parent(script_dir)
sys.path.append(script_dir)

import rpw
from rpw import revit, DB, UI, db

doc, uidoc = revit.doc, revit.uidoc

from rpw.db.xyz import XYZ
from rpw.exceptions import RpwParameterNotFound, RpwWrongStorageType
from rpw.utils.logger import logger


def setUpModule():
    logger.title('SETTING UP Curve TESTS...')

def tearDownModule():
    pass

class Line(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING Line...')
        pt1 = DB.XYZ(0,0,0)
        pt2 = DB.XYZ(10,10,0)
        cls.Line = DB.Line.CreateBound(pt1, pt2)
        cls.line = db.Line.new(pt1, pt2)

    def test_line(self):
        Line, line = self.Line, self.line
        self.assertIsInstance(line.unwrap(), DB.Line)
        self.assertTrue(Line.GetEndPoint(1).IsAlmostEqualTo(line.end_point.unwrap()))

    def test_line_start_point(self):
        Line, line = self.Line, self.line
        self.assertTrue(Line.GetEndPoint(0).IsAlmostEqualTo(line.start_point.
↪unwrap()))
```

(continues on next page)

(continued from previous page)

```
    def test_line_end_point(self):
        Line, line = self.Line, self.line
        self.assertTrue(Line.GetEndPoint(1).IsAlmostEqualTo(line.end_point.unwrap()))

    def test_line_end_point(self):
        Line, line = self.Line, self.line
        self.assertTrue(Line.GetEndPoint(0.5).IsAlmostEqualTo(line.mid_point.
↪unwrap()))

    def test_line_end_points(self):
        Line, line = self.Line, self.line
        self.assertIsInstance(line.end_points, tuple)
        self.assertTrue(Line.GetEndPoint(0).IsAlmostEqualTo(line.end_points[0].
↪unwrap()))


class Ellipse(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING Line...')
        pt1 = DB.XYZ(0,0,0)
        pt2 = DB.XYZ(10,10,0)
        cls.Line = DB.Line.CreateBound(pt1, pt2)
        cls.line = db.Line.new(pt1, pt2)

    # def test_line(self):
    #     Line, line = self.Line, self.line
    #     self.assertIsInstance(line.unwrap(), DB.Line)
    #     self.assertTrue(Line.GetEndPoint(1).IsAlmostEqualTo(line.end_point.
↪unwrap()))
    #
    # def test_line_start_point(self):
    #     Line, line = self.Line, self.line
    #     self.assertTrue(Line.GetEndPoint(0).IsAlmostEqualTo(line.start_point.
↪unwrap()))
    #
    # def test_line_end_point(self):
    #     Line, line = self.Line, self.line
    #     self.assertTrue(Line.GetEndPoint(1).IsAlmostEqualTo(line.end_point.
↪unwrap()))
    #
    # def test_line_end_point(self):
    #     Line, line = self.Line, self.line
    #     self.assertTrue(Line.GetEndPoint(0.5).IsAlmostEqualTo(line.mid_point.
↪unwrap()))
    #
    # def test_line_end_points(self):
    #     Line, line = self.Line, self.line
    #     self.assertIsInstance(line.end_points, tuple)
    #     self.assertTrue(Line.GetEndPoint(0).IsAlmostEqualTo(line.end_points[0].
↪unwrap()))


class CurveCreate(unittest.TestCase):
```

(continues on next page)

(continued from previous page)

```
    @classmethod
    def setUpClass(cls):
        logger.title('TESTING Curve Create...')

    def setUp(self):
        line = db.Line.new([0,0], [10,10])
        with rpw.db.Transaction():
            self.detail_line = line.create_detail()

    def tearDown(self):
        with rpw.db.Transaction():
            revit.doc.Delete(self.detail_line.Id)

    def test_detail_line(self):
        self.assertIsInstance(self.detail_line, DB.DetailLine)
        curve = self.detail_line.GeometryCurve
        self.assertTrue(curve.GetEndPoint(1).IsAlmostEqualTo(DB.XYZ(10,10,0)))
```

```
"""
Collector Tests

Passes:
 * 2017.1

Revit Python Wrapper
github.com/gtalarico/revitpythonwrapper
revitpythonwrapper.readthedocs.io

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

Copyright 2017 Gui Talarico

"""

import sys
import unittest
import os
```

(continues on next page)

(continued from previous page)

```
parent = os.path.dirname

script_dir = parent(__file__)
panel_dir = parent(script_dir)
sys.path.append(script_dir)

import rpw
from rpw import revit, DB, UI
from rpw.utils.dotnet import List
from rpw.exceptions import RpwParameterNotFound, RpwWrongStorageType, RpwCoerceError
from rpw.utils.logger import logger

import test_utils

def setUpModule():
    logger.title('SETTING UP ELEMENTS TESTS...')
    revit.uidoc.Application.OpenAndActivateDocument(os.path.join(panel_dir,
↪'collector.rvt'))
    test_utils.delete_all_walls()
    test_utils.make_wall()

def tearDownModule():
    test_utils.delete_all_walls()


######################
# ELEMENT
######################


class ElementTests(unittest.TestCase):

    @classmethod
    def setUpClass(self):
        logger.title('TESTING ELEMENT...')

    def setUp(self):
        self.wall = DB.FilteredElementCollector(revit.doc).OfClass(DB.Wall).
↪ToElements()[0]
        self.wrapped_wall = rpw.db.Element(self.wall)
        # param = self.wall.LookupParameter('Comments')
        # t = DB.Transaction(doc)
        # t.Start('Clear Comment Param')
        # param.Set('')
        # t.Commit()

    def tearDown(self):
        collector = rpw.db.Collector()
        levels = rpw.db.Collector(of_class=DB.Level).elements
        with rpw.db.Transaction('Delete Test Levels'):
            for level in levels[1:]:
                revit.doc.Delete(level.Id)

    def test_element_repr(self):
        self.assertIn('<RPW_Element:<Autodesk.Revit.DB.Wall', self.wrapped_wall.__
↪repr__())
```

(continues on next page)

(continued from previous page)

```
    def test_element_repr(self):
        self.assertIsInstance(self.wrapped_wall, rpw.db.Element)
        self.assertIsInstance(self.wrapped_wall.unwrap(), DB.Wall)

    def test_element_id(self):
        assert isinstance(self.wrapped_wall.Id, DB.ElementId)

    def test_element_from_id(self):
        element = rpw.db.Element.from_id(self.wall.Id)
        self.assertIsInstance(element, rpw.db.Element)

    def test_element_from_int(self):
        element = rpw.db.Element.from_int(self.wall.Id.IntegerValue)
        self.assertIsInstance(element, rpw.db.Element)

    def test_element_id(self):
        self.assertIsInstance(self.wrapped_wall, rpw.db.Element)

    def test_element_get_parameter_type(self):
        rv = self.wrapped_wall.parameters['Comments'].type
        self.assertEqual(rv, str)
        rv = self.wrapped_wall.parameters['Base Constraint'].type
        self.assertEqual(rv, DB.ElementId)
        rv = self.wrapped_wall.parameters['Unconnected Height'].type
        self.assertEqual(rv, float)
        rv = self.wrapped_wall.parameters['Room Bounding'].type
        self.assertEqual(rv, int)

    def test_element_get_parameter_name(self):
        rv = self.wrapped_wall.parameters['Comments'].name
        self.assertEqual(rv, 'Comments')

    def test_element_get_parameter(self):
        rv = self.wrapped_wall.parameters['Comments'].value
        self.assertEqual(rv, None)

    def tests_element_set_get_parameter_string(self):
        with rpw.db.Transaction('Set String'):
            self.wrapped_wall.parameters['Comments'].value = 'Test String'
        rv = self.wrapped_wall.parameters['Comments'].value
        self.assertEqual(rv, 'Test String')

    def tests_element_set_get_parameter_coerce_string(self):
        with rpw.db.Transaction('Set String'):
            self.wrapped_wall.parameters['Comments'].value = 5
        rv = self.wrapped_wall.parameters['Comments'].value
        self.assertEqual(rv, '5')

    def tests_element_set_get_parameter_float(self):
        with rpw.db.Transaction('Set Integer'):
            self.wrapped_wall.parameters['Unconnected Height'].value = 5.0
        rv = self.wrapped_wall.parameters['Unconnected Height'].value
        self.assertEqual(rv, 5.0)

    def tests_element_set_get_parameter_coerce_int(self):
        with rpw.db.Transaction('Set Coerce Int'):
            self.wrapped_wall.parameters['Unconnected Height'].value = 5
```

(continues on next page)

(continued from previous page)

```
        rv = self.wrapped_wall.parameters['Unconnected Height'].value
        self.assertEqual(rv, 5.0)

    def tests_element_set_get_parameter_element_id(self):
        active_view = revit.uidoc.ActiveView
        wrapped_view = rpw.db.Element(active_view)
        with rpw.db.Transaction('Create and Set Level'):
            try:
                new_level = DB.Level.Create(revit.doc, 10)
            except:
                new_level = revit.doc.Create.NewLevel(10)
            self.wrapped_wall.parameters['Top Constraint'].value = new_level.Id
        self.assertEqual(self.wrapped_wall.parameters['Top Constraint'].value.
↪IntegerValue,
                         new_level.Id.IntegerValue)

    def test_element_get_builtin_parameter_by_strin(self):
        bip = self.wrapped_wall.parameters.builtins['WALL_KEY_REF_PARAM'].value
        self.assertIsInstance(bip, int)

    def test_element_set_get_builtin_parameter_by_strin(self):
        bip = self.wrapped_wall.parameters.builtins['WALL_KEY_REF_PARAM']
        with rpw.db.Transaction('Set Value'):
            bip.value = 0
        bip = self.wrapped_wall.parameters.builtins['WALL_KEY_REF_PARAM']
        self.assertEqual(bip.value, 0)

    def test_element_get_builtin_parameter_caster(self):
        bip = self.wrapped_wall.parameters.builtins['WALL_KEY_REF_PARAM'].value
        BIP_ENUM = DB.BuiltInParameter.WALL_KEY_REF_PARAM
        bip2 = self.wrapped_wall.parameters.builtins[BIP_ENUM].value
        self.assertEqual(bip, bip2)

    def tests_wrong_storage_type(self):
        with self.assertRaises(RpwWrongStorageType) as context:
            with rpw.db.Transaction('Set String'):
                self.wrapped_wall.parameters['Unconnected Height'].value = 'Test'

    def test_parameter_does_not_exist(self):
        with self.assertRaises(RpwParameterNotFound) as context:
            self.wrapped_wall.parameters['Parameter Name']

    def test_built_in_parameter_exception_raised(self):
        with self.assertRaises(RpwCoerceError) as context:
            self.wrapped_wall.parameters.builtins['PARAMETERD_DOES_NOT_EXIST']


#########################
# Parameters / Isolated #
#########################

    def tests_param_class(self):
        param = self.wall.LookupParameter('Comments')
        self.assertIsInstance(param, DB.Parameter)
        wrapped_param = rpw.db.Parameter(param)
        self.assertIs(wrapped_param.type, str)
        self.assertEqual(wrapped_param.builtin, DB.BuiltInParameter.ALL_MODEL_
↪INSTANCE_COMMENTS)
```

(continues on next page)

(continued from previous page)

```
################################### INSTANCES / Symbols / Families #
##################################

class InstanceTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING INSTANCES...')

    def setUp(self):
        instance = rpw.db.Collector(of_category='OST_Furniture', is_not_type=True).
↪get_first(wrapped=False)
        self.instance = rpw.db.FamilyInstance(instance)

    def tearDown(self):
        logger.debug('SELECTION TEST PASSED')

    def test_instance_wrap(self):
        self.assertIsInstance(self.instance, rpw.db.FamilyInstance)
        self.assertIsInstance(self.instance.unwrap(), DB.FamilyInstance)

    def test_instance_symbol(self):
        symbol = self.instance.symbol
        self.assertIsInstance(symbol, rpw.db.FamilySymbol)
        self.assertIsInstance(symbol.unwrap(), DB.FamilySymbol)
        self.assertEqual(symbol.name, '60" x 30"')
        self.assertEqual(len(symbol.instances), 2)
        self.assertEqual(len(symbol.siblings), 3)

    def test_instance_family(self):
        family = self.instance.symbol.family
        self.assertIsInstance(family, rpw.db.Family)
        self.assertEqual(family.name, 'desk')
        self.assertIsInstance(family.unwrap(), DB.Family)
        self.assertEqual(len(family.instances), 3)
        self.assertEqual(len(family.siblings), 1)
        self.assertEqual(len(family.symbols), 3)

    def test_instance_category(self):
        category = self.instance.symbol.family.category
        self.assertIsInstance(category, rpw.db.Category)
        self.assertIsInstance(category.unwrap(), DB.Category)
        self.assertEqual(category.name, 'Furniture')
        self.assertEqual(len(category.instances), 3)
        self.assertEqual(len(category.symbols), 3)
        self.assertEqual(len(category.families), 1)

    def test_element_factory_class(self):
        instance = self.instance
        symbol = instance.symbol
        family = instance.family
        category = instance.category
        self.assertIsInstance(rpw.db.Element.Factory(instance.unwrap()), rpw.db.
↪FamilyInstance)
        self.assertIsInstance(rpw.db.Element.Factory(symbol.unwrap()), rpw.db.
↪FamilySymbol)
```

(continues on next page)

(continued from previous page)

```
        self.assertIsInstance(rpw.db.Element.Factory(family.unwrap()), rpw.db.Family)

        # TODO: Move this. Category No Longer Element
        # self.assertIsInstance(rpw.db.Element.Factory(category.unwrap()), rpw.db.
↪Category)


##################################################
# Wall / Wall Types / Wall Kind / Wall Category  #
##################################################

class WallTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING WALL...')

    def setUp(self):
        test_utils.delete_all_walls()
        test_utils.make_wall()
        wall = rpw.db.Collector(of_class='Wall', is_not_type=True).get_
↪first(wrapped=False)
        self.wall = rpw.db.wall.Wall(wall)

    def tearDown(self):
        test_utils.delete_all_walls()

    def test_wall_instance_wrap(self):
        self.assertIsInstance(self.wall, rpw.db.wall.Wall)
        self.assertIsInstance(self.wall.unwrap(), DB.Wall)

    def test_wall_factory(self):
        wrapped = rpw.db.Element.Factory(self.wall.unwrap())
        self.assertIsInstance(wrapped, rpw.db.wall.Wall)
        wrapped = rpw.db.Element.Factory(self.wall.symbol.unwrap())
        self.assertIsInstance(wrapped, rpw.db.wall.WallType)
        # TODO: MOVE THESE > No Longer Element
        wrapped = rpw.db.WallKind(self.wall.family.unwrap())
        self.assertIsInstance(wrapped, rpw.db.WallKind)

    def test_wall_instance_symbol(self):
        wall_symbol = self.wall.symbol
        self.assertIsInstance(wall_symbol, rpw.db.wall.WallType)
        self.assertIsInstance(wall_symbol.unwrap(), DB.WallType)
        self.assertEqual(wall_symbol.name, 'Wall 1')
        self.assertEqual(len(wall_symbol.instances), 1)
        self.assertEqual(len(wall_symbol.siblings), 2)

    def test_wall_instance_family(self):
        wall_family = self.wall.family
        self.assertIsInstance(wall_family, rpw.db.wall.WallKind)
        self.assertEqual(wall_family.unwrap(), DB.WallKind.Basic)
        self.assertEqual(wall_family.name, 'Basic')
        self.assertEqual(len(wall_family.instances), 1)
        self.assertEqual(len(wall_family.symbols), 2)

    def test_wall_instance_category(self):
```

(continues on next page)

(continued from previous page)

```
        wall_category = self.wall.category
        self.assertIsInstance(wall_category, rpw.db.wall.WallCategory)
        self.assertIsInstance(wall_category.unwrap(), DB.Category)
        self.assertEqual(wall_category.name, 'Walls')

    def test_wall_instance_category(self):
        wall_category = self.wall.category
        self.assertIsInstance(wall_category, rpw.db.wall.WallCategory)
        self.assertIsInstance(wall_category.unwrap(), DB.Category)
        self.assertEqual(wall_category.name, 'Walls')

    def test_wall_change_type_by_name(self):
        wall = self.wall
        with rpw.db.Transaction():
            wall.change_type('Wall 2')
        self.assertEqual(wall.wall_type.name, 'Wall 2')

    def test_wall_change_type(self):
        wall = self.wall
        wall_type = rpw.db.Collector(of_class='WallType', where=lambda w: w.name ==
↪'Wall 2').get_first(wrapped=False)
        with rpw.db.Transaction():
            wall.change_type('Wall 2')
        self.assertEqual(wall.wall_type.name, 'Wall 2')

##################
# Rooms / Areas  #
##################

class RoomTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        pass
        # t = DB.Transaction(doc)
        # t.Start('Add Room')


    def setUp(self):
        room = rpw.db.Collector(os_category='OST_Rooms', is_not_type=True).get_
↪first(wrapped=False)
        self.wall = rpw.db.wall.Wall(wall)
    #
    # def test_wall_instance_wrap(self):
    #     self.assertIsInstance(self.wall, rpw.db.wall.Wall)
    #     self.assertIsInstance(self.wall.unwrap(), DB.Wall)
```

```
"""
Selection Tests

Passes:
 * 2017.1
```

(continues on next page)

(continued from previous page)

```
Revit Python Wrapper
github.com/gtalarico/revitpythonwrapper
revitpythonwrapper.readthedocs.io

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

Copyright 2017 Gui Talarico

"""

import sys
import unittest
import os

parent = os.path.dirname
script_dir = parent(__file__)
panel_dir = parent(script_dir)
sys.path.append(script_dir)

import rpw
from rpw import DB, UI
doc, uidoc = rpw.revit.doc, rpw.revit.uidoc
from rpw.utils.logger import logger

import test_utils

def setUpModule():
    logger.title('SETTING UP SELECTION TESTS...')
    # uidoc.Application.OpenAndActivateDocument(os.path.join(panel_dir, 'collector.rvt
↪'))

def tearDownModule():
    pass


######################
# SELECTION
######################


class SelectionTests(unittest.TestCase):
```

(continues on next page)

(continued from previous page)

```
    @classmethod
    def setUpClass(cls):
        logger.title('TESTING SELECTION...')
        test_utils.delete_all_walls()
        wall = test_utils.make_wall()
        cls.wall = wall

    @classmethod
    def tearDownClass(cls):
        test_utils.delete_all_walls()

    def setUp(self):
        self.wall = SelectionTests.wall
        self.selection = rpw.ui.Selection([self.wall.Id])

    def tearDown(self):
        self.selection.clear()
        logger.debug('SELECTION TEST PASSED')

    def test_selection_element_ids(self):
        ids = self.selection.element_ids
        self.assertTrue(all(
                        [isinstance(eid, DB.ElementId) for eid in ids]
                        ))

    def test_selection_elements(self):
        elements = self.selection.elements
        self.assertTrue(all(
                        [isinstance(e, DB.Element) for e in elements]
                        ))

    def test_selection_by_index(self):
        wall = self.selection.get_elements(wrapped=False)[0]
        self.assertIsInstance(wall, DB.Wall)
        wall2 = self.selection.get_elements(wrapped=True)[0]
        self.assertTrue(hasattr(wall2, 'unwrap'))

    def test_selection_length(self):
        self.assertEqual(len(self.selection), 1)

    def test_selection_boolean(self):
        self.assertTrue(self.selection)

    def test_selection_boolean_false(self):
        self.selection.clear()
        self.assertFalse(self.selection)

    def test_selection_clear(self):
        self.selection.clear()
        self.assertEqual(len(self.selection), 0)
        self.selection = rpw.ui.Selection([self.wall.Id])

    def test_selection_add(self):
        selection = rpw.ui.Selection()
        selection.add([self.wall])
        wall = self.selection.get_elements(wrapped=False)[0]
```

(continues on next page)

(continued from previous page)

```
        self.assertIsInstance(wall, DB.Wall)

    def test_selection_contains(self):
        selection = rpw.ui.Selection()
        selection.add([self.wall])
        self.assertIn(self.wall, selection)

    def test_selection_updates_does_not_lose(self):
        selection = rpw.ui.Selection([self.wall])
        selection2 = rpw.ui.Selection([self.wall])
        selection2.update()
        self.assertEqual(selection.elements[0].Id, selection2.elements[0].Id)

    def test_selection_update(self):
        selection = rpw.ui.Selection()
        selection.update()
```

```
"""
Utils Tests

Passes:
 * 2017.1

Revit Python Wrapper
github.com/gtalarico/revitpythonwrapper
revitpythonwrapper.readthedocs.io

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

Copyright 2017 Gui Talarico

"""

import sys
import unittest
import os

parent = os.path.dirname
```

(continues on next page)

(continued from previous page)

```
script_dir = parent(__file__)
panel_dir = parent(script_dir)
sys.path.append(script_dir)

import rpw
from rpw import revit, DB, UI
from rpw.db import Element
from rpw.db import View, ViewPlan, ViewSection
from rpw.db import ViewSheet, ViewSchedule, View3D
from rpw.db import ViewFamilyType
from rpw.db import ViewType, ViewPlanType

from rpw.utils.logger import logger

# from rpw.utils.dotnet import List
# from rpw.exceptions import RpwParameterNotFound, RpwWrongStorageType

import test_utils

def setUpModule():
    logger.title('SETTING UP VIEW TESTS...')
    # test_utils.delete_all_walls()
    # test_utils.make_wall()

def tearDownModule():
    pass
    # test_utils.delete_all_walls()

class TestViewWrappers(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING View Classes...')
        cls.view = DB.FilteredElementCollector(revit.doc).OfClass(DB.View).
↪FirstElement()
        cls.view_plan = DB.FilteredElementCollector(revit.doc).OfClass(DB.ViewPlan).
↪FirstElement()
        cls.view_sheet = DB.FilteredElementCollector(revit.doc).OfClass(DB.ViewSheet).
↪FirstElement()
        cls.view_schedule = DB.FilteredElementCollector(revit.doc).OfClass(DB.
↪ViewSchedule).FirstElement()
        cls.view_section = DB.FilteredElementCollector(revit.doc).OfClass(DB.
↪ViewSection).FirstElement()
        cls.view_3d = DB.FilteredElementCollector(revit.doc).OfClass(DB.View3D).
↪FirstElement()
        cls.view_family_type = DB.FilteredElementCollector(revit.doc).OfClass(DB.
↪ViewFamilyType).FirstElement()

    def setUp(self):
        pass

    def tearDown(self):
        pass

    def test_view_wrapper(self):
        wrapped_view = Element(self.view)
        self.assertIsInstance(wrapped_view, View)
```

(continues on next page)

(continued from previous page)

```
        wrapped_view = View(self.view)
        self.assertIsInstance(wrapped_view, View)

    def test_view_plan_wrapper(self):
        wrapped_view_plan = Element(self.view_plan)
        self.assertIsInstance(wrapped_view_plan, ViewPlan)
        wrapped_view_plan = ViewPlan(self.view_plan)
        self.assertIsInstance(wrapped_view_plan, ViewPlan)

    def test_view_section_wrapper(self):
        wrapped_view_section = Element(self.view_section)
        self.assertIsInstance(wrapped_view_section, ViewSection)
        wrapped_view_section = ViewSection(self.view_section)
        self.assertIsInstance(wrapped_view_section, ViewSection)

    def test_view_sheet_wrapper(self):
        wrapped_view_sheet = Element(self.view_sheet)
        self.assertIsInstance(wrapped_view_sheet, ViewSheet)
        wrapped_view_sheet = ViewSheet(self.view_sheet)
        self.assertIsInstance(wrapped_view_sheet, ViewSheet)

    def test_view_schedule_wrapper(self):
        wrapped_view_schedule = Element(self.view_schedule)
        self.assertIsInstance(wrapped_view_schedule, ViewSchedule)
        wrapped_view_schedule = ViewSchedule(self.view_schedule)
        self.assertIsInstance(wrapped_view_schedule, ViewSchedule)

    def test_view_3D(self):
        wrapped_view_3d = Element(self.view_3d)
        self.assertIsInstance(wrapped_view_3d, View3D)
        wrapped_view_3d = View3D(self.view_3d)
        self.assertIsInstance(wrapped_view_3d, View3D)

    def test_view_family_type(self):
        wrapped_view_family_type = Element(self.view_family_type)
        self.assertIsInstance(wrapped_view_family_type, ViewFamilyType)
        wrapped_view_family_type = ViewFamilyType(self.view_family_type)
        self.assertIsInstance(wrapped_view_family_type, ViewFamilyType)

class TestViewRelationships(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING View Classes...')
        cls.view = DB.FilteredElementCollector(revit.doc).OfClass(DB.View).
↪FirstElement()
        cls.view_plan = DB.FilteredElementCollector(revit.doc).OfClass(DB.ViewPlan).
↪FirstElement()
        cls.view_sheet = DB.FilteredElementCollector(revit.doc).OfClass(DB.ViewSheet).
↪FirstElement()
        cls.view_schedule = DB.FilteredElementCollector(revit.doc).OfClass(DB.
↪ViewSchedule).FirstElement()
        cls.view_section = DB.FilteredElementCollector(revit.doc).OfClass(DB.
↪ViewSection).FirstElement()
        cls.view_3d = DB.FilteredElementCollector(revit.doc).OfClass(DB.View3D).
↪FirstElement()
        cls.view_family_type = DB.FilteredElementCollector(revit.doc).OfClass(DB.
↪ViewFamilyType).FirstElement()
```

(continues on next page)

(continued from previous page)

```
    def setUp(self):
        pass

    def tearDown(self):
        pass

    def test_view_type(self):
        wrapped_view = Element(self.view_3d)
        view_type = wrapped_view.view_type
        self.assertIsInstance(view_type.unwrap(), DB.ViewType)
        self.assertEqual(view_type.unwrap(), DB.ViewType.ThreeD)
        self.assertEqual(view_type.name, 'ThreeD')

    def test_view_plan_level(self):
        wrapped_view = Element(self.view_plan)
        level = wrapped_view.level
        self.assertIsInstance(level, DB.Level)

    def test_view_family_type(self):
        wrapped_view = Element(self.view_3d)
        view_type = wrapped_view.view_family_type
        self.assertIsInstance(view_type.unwrap(), DB.ViewFamilyType)

    def test_view_family(self):
        wrapped_view = Element(self.view_3d)
        view_family = wrapped_view.view_family
        self.assertIsInstance(view_family.unwrap(), DB.ViewFamily)

    def test_view_type_aggregator(self):
        wrapped_view_plan = Element(self.view_plan)
        same_view_type_views = wrapped_view_plan.view_type.views
        for view in same_view_type_views:
            self.assertEqual(view.view_type.unwrap(), wrapped_view_plan.view_type.
↪unwrap())

    def test_view_family_aggregator(self):
        wrapped_view_plan = Element(self.view_plan)
        same_family_views = wrapped_view_plan.view_family.views
        for view in same_family_views:
            self.assertEqual(view.view_family.unwrap(), wrapped_view_plan.view_family.
↪unwrap())

    def test_view_family_aggregator(self):
        wrapped_view_plan = Element(self.view_plan)
        same_view_family_type_views = wrapped_view_plan.view_family_type.views
        for view in same_view_family_type_views:
            self.assertEqual(view.view_family_type.unwrap(), wrapped_view_plan.view_
↪family_type.unwrap())

    def test_view_family_type_name(self):
        wrapped_view = rpw.db.ViewPlan.collect(where=lambda x: x.view_family_type.
↪name == 'Floor Plan').wrapped_elements[0]
        self.assertEqual(wrapped_view.view_family_type.name, 'Floor Plan')

    # def test_view_family_type_name_get_setter(self):
    #     wrapped_view = rpw.db.ViewPlan.collect(where=lambda x: x.view_family_type.
↪name == 'My Floor Plan').wrapped_elements[0]
```

(continues on next page)

(continued from previous page)

```
    #     # self.assertEqual(wrapped_view.view_family_type.name, 'My Floor Plan')
    #     with rpw.db.Transaction('Set Name'):
    #         wrapped_view.view_family_type.name = 'ABC'
    #     self.assertEqual(wrapped_view.view_family_type.name, 'ABC')
        # with rpw.db.Transaction('Set Name'):
          # wrapped_view.view_family_type.name = 'My Floor Plan'
        # rpw.ui.forms.Console()

class TestViewOverrides(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING View Classes...')
        cls.view_plan = revit.active_view.unwrap()
        # cls.view_plan = DB.FilteredElementCollector(revit.doc).OfClass(DB.ViewPlan).
↪FirstElement()
        cls.wrapped_view = revit.active_view
        cls.element = DB.FilteredElementCollector(revit.doc).OfClass(DB.
↪FamilyInstance).WhereElementIsNotElementType().FirstElement()

        linepattern = rpw.db.Collector(of_class='LinePatternElement', where=lambda x:␣
↪x.Name == 'Dash').get_first()
        cls.line_pattern_id = linepattern.Id
        fillpattern = rpw.db.Collector(of_class='FillPatternElement', where=lambda x:␣
↪x.Name == 'Horizontal').get_first()
        cls.fillpattern_id = fillpattern.Id

    def tearDown(cls):
        """ Resets Element after each test """
        with rpw.db.Transaction():
            cls.view_plan.SetElementOverrides(cls.element.Id, DB.
↪OverrideGraphicSettings())

    def test_match(self):
        e1 = DB.FilteredElementCollector(revit.doc).OfClass(DB.FamilyInstance).
↪WhereElementIsNotElementType().ToElements()[0]
        e2 = DB.FilteredElementCollector(revit.doc).OfClass(DB.FamilyInstance).
↪WhereElementIsNotElementType().ToElements()[1]
        o = DB.OverrideGraphicSettings()
        o.SetHalftone(True)
        o.SetSurfaceTransparency(30)
        with rpw.db.Transaction():
            self.view_plan.SetElementOverrides(e1.Id, o)

        with rpw.db.Transaction():
            self.wrapped_view.override.match_element(e2, e1)
        rv = self.view_plan.GetElementOverrides(e2.Id)
        self.assertTrue(rv.Halftone)
        self.assertEqual(rv.Transparency, 30)

    def test_halftone(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.halftone(self.element, True)
        rv = self.view_plan.GetElementOverrides(self.element.Id)
        self.assertTrue(rv.Halftone)

    def test_halftone(self):
```

(continues on next page)

(continued from previous page)

```
        with rpw.db.Transaction():
            self.wrapped_view.override.halftone(self.element, True)
        rv = self.view_plan.GetElementOverrides(self.element.Id)
        self.assertTrue(rv.Halftone)

    def test_transparency(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.transparency(self.element, 40)
        rv = self.view_plan.GetElementOverrides(self.element.Id)
        self.assertEqual(rv.Transparency, 40)

    def test_detail_level_by_enum(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.detail_level(self.element, DB.ViewDetailLevel.
↪Fine)
        rv = self.view_plan.GetElementOverrides(self.element.Id)
        self.assertEqual(rv.DetailLevel, DB.ViewDetailLevel.Fine)

    def test_detail_level_by_name(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.detail_level(self.element, 'Fine')
        rv = self.view_plan.GetElementOverrides(self.element.Id)
        self.assertEqual(rv.DetailLevel, DB.ViewDetailLevel.Fine)

    def test_projection_line(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.projection_line(self.element,
                                                       color=(0,120,255),
                                                       weight=5,
                                                       pattern=self.line_pattern_id)

        rv = self.view_plan.GetElementOverrides(self.element.Id)
        self.assertEqual(rv.ProjectionLineColor.Red, 0)
        self.assertEqual(rv.ProjectionLineColor.Green, 120)
        self.assertEqual(rv.ProjectionLineColor.Blue, 255)
        self.assertEqual(rv.ProjectionLineWeight, 5)
        self.assertEqual(rv.ProjectionLinePatternId, self.line_pattern_id)

    def test_projection_line_pattern_by_name(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.projection_line(self.element, pattern='Dash')
            rv = self.view_plan.GetElementOverrides(self.element.Id)
            self.assertEqual(rv.ProjectionLinePatternId, self.line_pattern_id)

    def test_cut_line(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.cut_line(self.element,
                                                color=(0,80,150),
                                                weight=7,
                                                pattern=self.line_pattern_id)

        rv = self.view_plan.GetElementOverrides(self.element.Id)
        self.assertEqual(rv.CutLineColor.Red, 0)
        self.assertEqual(rv.CutLineColor.Green, 80)
        self.assertEqual(rv.CutLineColor.Blue, 150)
        self.assertEqual(rv.CutLineWeight, 7)
        self.assertEqual(rv.CutLinePatternId, self.line_pattern_id)
```

(continues on next page)

(continued from previous page)

```
    def test_cut_line_pattern_by_name(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.cut_line(self.element, pattern='Dash')
            rv = self.view_plan.GetElementOverrides(self.element.Id)
            self.assertEqual(rv.CutLinePatternId, self.line_pattern_id)

    def test_projection_fill(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.projection_fill(self.element,
                                                       color=(0,40,190),
                                                       pattern=self.fillpattern_id,
                                                       visible=False)

        rv = self.view_plan.GetElementOverrides(self.element.Id)
        self.assertEqual(rv.ProjectionFillColor.Red, 0)
        self.assertEqual(rv.ProjectionFillColor.Green, 40)
        self.assertEqual(rv.ProjectionFillColor.Blue, 190)
        self.assertEqual(rv.IsProjectionFillPatternVisible, False)
        self.assertEqual(rv.ProjectionFillPatternId, self.fillpattern_id)

    def test_projection_fill_pattern_by_name(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.projection_fill(self.element, pattern=
↪'Horizontal')
            rv = self.view_plan.GetElementOverrides(self.element.Id)
            self.assertEqual(rv.ProjectionFillPatternId, self.fillpattern_id)

    def test_cut_fill(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.cut_fill(self.element,
                                                color=(0,30,200),
                                                pattern=self.fillpattern_id,
                                                visible=False)

        rv = self.view_plan.GetElementOverrides(self.element.Id)
        self.assertEqual(rv.CutFillColor.Red, 0)
        self.assertEqual(rv.CutFillColor.Green, 30)
        self.assertEqual(rv.CutFillColor.Blue, 200)
        self.assertEqual(rv.IsCutFillPatternVisible, False)
        self.assertEqual(rv.CutFillPatternId, self.fillpattern_id)


    def test_cut_fill_pattern_by_name(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.cut_fill(self.element, pattern='Horizontal')
            rv = self.view_plan.GetElementOverrides(self.element.Id)
            self.assertEqual(rv.CutFillPatternId, self.fillpattern_id)

    def test_halftone_category(self):
        with rpw.db.Transaction():
            self.wrapped_view.override.halftone('Furniture', True)
        rv = self.view_plan.GetCategoryOverrides(DB.ElementId(DB.BuiltInCategory.OST_
↪Furniture))
        self.assertTrue(rv.Halftone)

    def test_halftone_category_bi(self):
```

(continues on next page)

(continued from previous page)

```
        with rpw.db.Transaction():
            self.wrapped_view.override.halftone(DB.BuiltInCategory.OST_Furniture,␣
↪True)
        rv = self.view_plan.GetCategoryOverrides(DB.ElementId(DB.BuiltInCategory.OST_
↪Furniture))
        self.assertTrue(rv.Halftone)
```

```
"""
XYZ Tests

Passes:
 * 2017.1

Revit Python Wrapper
github.com/gtalarico/revitpythonwrapper
revitpythonwrapper.readthedocs.io

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

Copyright 2017 Gui Talarico

"""

import sys
import unittest
import os

parent = os.path.dirname

script_dir = parent(__file__)
panel_dir = parent(script_dir)
sys.path.append(script_dir)

import rpw
from rpw import revit, DB, UI

doc, uidoc = revit.doc, revit.uidoc
```

(continues on next page)

(continued from previous page)

```
from rpw.db.xyz import XYZ
from rpw.exceptions import RpwParameterNotFound, RpwWrongStorageType
from rpw.utils.logger import logger

# import test_utils

def setUpModule():
    logger.title('SETTING UP COLLECTION TESTS...')

def tearDownModule():
    pass
    # test_utils.delete_all_walls()

######################
# XYZTests
######################

class XYZInitTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING XYZ...')

    def test_xyz_from_2args(self):
        pt = XYZ(2,4)
        self.assertEqual(pt.X, 2)
        self.assertEqual(pt.Y, 4)
        self.assertEqual(pt.Z, 0)

    def test_xyz_from_3args(self):
        pt = XYZ(2,4,6)
        self.assertEqual(pt.X, 2)
        self.assertEqual(pt.Y, 4)
        self.assertEqual(pt.Z, 6)

    def test_xyz_from_tuple2(self):
        pt = XYZ([2,4])
        self.assertEqual(pt.X, 2)
        self.assertEqual(pt.Y, 4)
        self.assertEqual(pt.Z, 0)

    def test_xyz_from_tuple3(self):
        pt = XYZ([2,4,6])
        self.assertEqual(pt.X, 2)
        self.assertEqual(pt.Y, 4)
        self.assertEqual(pt.Z, 6)

    def test_xyz_from_DB_XYZ(self):
        pt = XYZ(DB.XYZ(2,4,6))
        self.assertEqual(pt.X, 2)
        self.assertEqual(pt.Y, 4)
        self.assertEqual(pt.Z, 6)

    def test_xyz_from_XYZ(self):
        pt = XYZ(XYZ(2,4,6))
        self.assertEqual(pt.X, 2)
        self.assertEqual(pt.Y, 4)
```

(continues on next page)

(continued from previous page)

```
        self.assertEqual(pt.Z, 6)

class XYZUsageTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING XYZ Usage...')
        cls.pt = XYZ(1,2,3)
        cls.pt2 = XYZ(4,5,6)

    def test_xyz_get_properties(self):
        pt = XYZ(1,2,3)
        self.assertEqual(pt.x, 1)
        self.assertEqual(pt.y, 2)
        self.assertEqual(pt.z, 3)

    def test_xyz_set_properties(self):
        pt = XYZ(1,2,3)
        pt.x = 5
        pt.y = 6
        pt.z = 7
        self.assertEqual(pt.x, 5)
        self.assertEqual(pt.y, 6)
        self.assertEqual(pt.z, 7)

    def test_xyz_at_z(self):
        pt = XYZ(1,2,3).at_z(10)
        self.assertEqual(pt.z, 10)

    def test_xyz_as_tuple(self):
        pt_tuple = XYZ(1,2,3).as_tuple
        self.assertEqual(pt_tuple, (1,2,3))
        self.assertIsInstance(pt_tuple, tuple)

    def test_xyz_as_dict(self):
        pt_dict = XYZ(1,2,3).as_dict
        self.assertIsInstance(pt_dict, dict)
        self.assertEqual(pt_dict, {'x':1, 'y':2, 'z':3})

    def test_xyz_repr(self):
        self.assertIn('<rpw:XYZ', XYZ(0,0,0).__repr__())

    def test_xyz_add(self):
        pt = XYZ(1,2,3) + XYZ(4,5,6)
        self.assertEqual(pt.x, 5)
        self.assertEqual(pt.y, 7)
        self.assertEqual(pt.z, 9)

    def test_xyz_sub(self):
        pt = XYZ(1,2,3) - XYZ(1,1,1)
        self.assertEqual(pt.x, 0)
        self.assertEqual(pt.y, 1)
        self.assertEqual(pt.z, 2)

    def test_xyz_mul(self):
        pt = XYZ(1,2,3) * 2
        self.assertEqual(pt.x, 2)
```

(continues on next page)

(continued from previous page)

```
        self.assertEqual(pt.y, 4)
        self.assertEqual(pt.z, 6)

    def test_xyz_eq(self):
        self.assertEqual(XYZ(1,2,3), XYZ(1,2,3))
        self.assertNotEqual(XYZ(1,2,3), XYZ(2,2,3))

    def test_xyz_rotate_90(self):
        pt = XYZ(1,0,0)
        rotate_pt = (0,1,0)
        self.assertEqual(pt.rotate(90), rotate_pt)

    def test_xyz_rotate_180(self):
        pt = XYZ(1,0,0)
        rotate_pt = (-1,0,0)
        self.assertEqual(pt.rotate(180), rotate_pt)

    def test_xyz_rotate_radians(self):
        import math
        pt = XYZ(1,0,0)
        rotate_pt = (-1,0,0)
        self.assertEqual(pt.rotate(math.pi, radians=True), rotate_pt)

    def test_xyz_rotate_radians(self):
        import math
        pt = XYZ(1,0,0)
        rotate_pt = (-1,0,0)
        self.assertEqual(pt.rotate(math.pi, radians=True), rotate_pt)

    def test_xyz_rotate_axis(self):
        import math
        pt = XYZ(1,0,0)
        axis = XYZ(0,-1,0)
        rotate_pt = (0,0,1)
        self.assertEqual(pt.rotate(90, axis=axis), rotate_pt)
```

```
"""
Transaction Tests

Passes:
 * 2017.1

Revit Python Wrapper
github.com/gtalarico/revitpythonwrapper
revitpythonwrapper.readthedocs.io

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
```

(continues on next page)

(continued from previous page)

```
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

Copyright 2017 Gui Talarico

"""

import sys
import unittest
import os

parent = os.path.dirname

script_dir = parent(__file__)
panel_dir = parent(script_dir)
sys.path.append(script_dir)

import rpw
from rpw import revit, DB, UI
from rpw.utils.dotnet import List
from rpw.utils.logger import logger
doc = rpw.revit.doc

import test_utils

def setUpModule():
    logger.title('SETTING UP TRANSACTION TESTS...')

def tearDownModule():
    pass


class TransactionsTest(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        logger.title('TESTING TRANSACTIONS...')
        test_utils.delete_all_walls()
        wall = test_utils.make_wall()
        cls.wall = wall

    @classmethod
    def tearDownClass(cls):
        test_utils.delete_all_walls()

    def setUp(self):
        wall = DB.FilteredElementCollector(doc).OfClass(DB.Wall).ToElements()[0]
        self.wall = rpw.db.Wall(wall)
        with rpw.db.Transaction('Reset Comment') as t:
            self.wall.parameters['Comments'] = ''
```

(continues on next page)

(continued from previous page)

```
    def test_transaction_instance(self):
        with rpw.db.Transaction('Test Is Instance') as t:
            self.wall.parameters['Comments'].value = ''
            self.assertIsInstance(t.unwrap(), DB.Transaction)

    def test_transaction_started(self):
        with rpw.db.Transaction('Has Started') as t:
            self.wall.parameters['Comments'].value = ''
            self.assertTrue(t.HasStarted())

    def test_transaction_has_ended(self):
        with rpw.db.Transaction('Add Comment') as t:
            self.wall.parameters['Comments'].value = ''
            self.assertFalse(t.HasEnded())

    def test_transaction_get_name(self):
        with rpw.db.Transaction('Named Transaction') as t:
            self.assertEqual(t.GetName(), 'Named Transaction')

    def test_transaction_commit_status_success(self):
        with rpw.db.Transaction('Set String') as t:
            self.wall.parameters['Comments'].value = ''
            self.assertEqual(t.GetStatus(), DB.TransactionStatus.Started)
        self.assertEqual(t.GetStatus(), DB.TransactionStatus.Committed)

    def test_transaction_commit_status_rollback(self):
        with self.assertRaises(Exception):
            with rpw.db.Transaction('Set String') as t:
                self.wall.parameters['Top Constraint'].value = DB.ElementId('a')
        self.assertEqual(t.GetStatus(), DB.TransactionStatus.RolledBack)

    def test_transaction_group(self):
        with rpw.db.TransactionGroup('Multiple Transactions') as tg:
            self.assertEqual(tg.GetStatus(), DB.TransactionStatus.Started)
            with rpw.db.Transaction('Set String') as t:
                self.assertEqual(t.GetStatus(), DB.TransactionStatus.Started)
                self.wall.parameters['Comments'].value = '1'
            self.assertEqual(t.GetStatus(), DB.TransactionStatus.Committed)
        self.assertEqual(tg.GetStatus(), DB.TransactionStatus.Committed)

    def test_transaction_decorator(self):
        @rpw.db.Transaction.ensure('Transaction Name')
        def somefunction():
            param = self.wall.parameters['Comments'].value = '1'
            return param
        self.assertTrue(somefunction())
```

```
"""
Utils Tests

Passes:
 * 2017.1

Revit Python Wrapper
```

(continues on next page)

(continued from previous page)

```
github.com/gtalarico/revitpythonwrapper
revitpythonwrapper.readthedocs.io

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

Copyright 2017 Gui Talarico

"""

import sys
import unittest
import os

parent = os.path.dirname

script_dir = parent(__file__)
panel_dir = parent(script_dir)
sys.path.append(script_dir)

import rpw
from rpw import revit, DB, UI
from rpw.utils.dotnet import List
from rpw.exceptions import RpwParameterNotFound, RpwWrongStorageType
from rpw.utils.logger import logger

import test_utils

def setUpModule():
    logger.title('SETTING UP UTILS TESTS...')
    test_utils.delete_all_walls()
    test_utils.make_wall()

def tearDownModule():
    test_utils.delete_all_walls()

class CoerceTests(unittest.TestCase):

    @classmethod
    def setUpClass(self):
        logger.title('TESTING COERCE FUNCITONS...')
```

(continues on next page)

(continued from previous page)

```
    def setUp(self):
        self.wall = rpw.db.Collector(of_class='Wall').get_first(wrapped=False)

    def tearDown(self):
        pass

    def test_corce_into_id(self):
        id_ = rpw.utils.coerce.to_element_id(self.wall)
        self.assertIsInstance(id_, DB.ElementId)

    def test_corce_into_ids(self):
        ids = rpw.utils.coerce.to_element_ids([self.wall])
        all_id = all([isinstance(i, DB.ElementId) for i in ids])
        self.assertTrue(all_id)

    def test_corce_element_ref_int(self):
        element = rpw.utils.coerce.to_element(self.wall.Id.IntegerValue)
        self.assertIsInstance(element, DB.Element)

    def test_corce_element_ref_id(self):
        wall_id = DB.ElementId(self.wall.Id.IntegerValue)
        elements = rpw.utils.coerce.to_elements([wall_id])
        self.assertTrue(all([isinstance(e, DB.Element) for e in elements]))

    def test_corce_to_element_diverse(self):
        elements = rpw.utils.coerce.to_elements([self.wall, self.wall.Id, self.wall.
↪Id.IntegerValue])
        self.assertTrue(all([isinstance(e, DB.Element) for e in elements]))

    def test_to_class_wall(self):
        self.assertIs(rpw.utils.coerce.to_class('Wall'), DB.Wall)

    def test_to_class_view(self):
        self.assertIs(rpw.utils.coerce.to_class('View'), DB.View)

    def test_to_category_walls(self):
        self.assertIs(rpw.utils.coerce.to_category('Walls'), DB.BuiltInCategory.OST_
↪Walls)
        self.assertIs(rpw.utils.coerce.to_category('walls'), DB.BuiltInCategory.OST_
↪Walls)
        self.assertIs(rpw.utils.coerce.to_category('ost_walls'), DB.BuiltInCategory.
↪OST_Walls)

    def test_to_category_id_walls(self):
        self.assertEqual(rpw.utils.coerce.to_category_id('Walls'), DB.ElementId(DB.
↪BuiltInCategory.OST_Walls))
        self.assertEqual(rpw.utils.coerce.to_category_id('walls'), DB.ElementId(DB.
↪BuiltInCategory.OST_Walls))
        self.assertEqual(rpw.utils.coerce.to_category_id('ost_walls'), DB.
↪ElementId(DB.BuiltInCategory.OST_Walls))

    def test_to_category_stacked_walls(self):
        self.assertIs(rpw.utils.coerce.to_category('ost_StackedWalls'), DB.
↪BuiltInCategory.OST_StackedWalls)
        self.assertIs(rpw.utils.coerce.to_category('StackedWalls'), DB.
↪BuiltInCategory.OST_StackedWalls)
        self.assertIs(rpw.utils.coerce.to_category('stackedwalls'), DB.
↪BuiltInCategory.OST_StackedWalls)
```

(continues on next page)

(continued from previous page)

```
        self.assertIs(rpw.utils.coerce.to_category('stacked walls'), DB.
↪BuiltInCategory.OST_StackedWalls)

    def test_to_iterable(self):
        self.assertTrue([w for w in rpw.utils.coerce.to_iterable(self.wall)])

    def test_to_iterable_element_id(self):
        self.assertTrue([w for w in rpw.utils.coerce.to_element_ids(self.wall)])

    def test_to_iterable_element(self):
        self.assertTrue([w for w in rpw.utils.coerce.to_elements(self.wall)])


    # TODO: Add BuiltInCategory Tests
    # CATEGORY COERCE
    # >>> with rpw.db.Transaction():
    # ...         rpw.revit.active_view.override.projection_line(BuiltInCategory.OST_
↪Furniture, color=[255,0,255])
    # ...
    # >>> with rpw.db.Transaction():
    # ...         rpw.revit.active_view.override.projection_
↪line(ElementId(BuiltInCategory.OST_Furniture), color=[255,0,255])
    # ...
    # >>> with rpw.db.Transaction():
    # ...         rpw.revit.active_view.override.projection_
↪line(ElementId(BuiltInCategory.OST_Furniture), color=[255,0,120])
    # ...
    # >>>
```

```
"""
Selection Tests

Passes:
 * 2017.1

Revit Python Wrapper
github.com/gtalarico/revitpythonwrapper
revitpythonwrapper.readthedocs.io

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
```

(continues on next page)

(continued from previous page)

```
OTHER DEALINGS IN THE SOFTWARE.

Copyright 2017 Gui Talarico

"""

import sys
import unittest
import os

parent = os.path.dirname
script_dir = parent(__file__)
panel_dir = parent(script_dir)
sys.path.append(script_dir)

import rpw
from rpw import DB, UI
doc, uidoc = rpw.revit.doc, rpw.revit.uidoc
from rpw.utils.logger import logger
from rpw.ui.selection import Pick
from rpw.db.reference import Reference
from rpw.db.xyz import XYZ
from rpw.db.element import Element

import test_utils

def setUpModule():
    logger.title('SETTING UP PICK TESTS...')

def tearDownModule():
    pass


######################
# SELECTION
######################


class PickTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        logger.title('TESTING PICK...')
        test_utils.delete_all_walls()
        wall = test_utils.make_wall()
        cls.wall = wall

    @classmethod
    def tearDownClass(cls):
        test_utils.delete_all_walls()

    def setUp(self):
        self.wall = PickTests.wall
        # Pick().clear()

    def tearDown(self):
        # Pick().clear()
```

(continues on next page)

(continued from previous page)

```
        logger.debug('SELECTION TEST PASSED')

    def test_pick_element(self):
        selection = Pick()
        desk = selection.pick_element('Pick a Desk')
        self.assertIsInstance(desk, Reference)

    def test_pick_elements(self):
        selection = Pick()
        desks = selection.pick_element('Pick 2 Desks', multiple=True)
        self.assertIsInstance(desks[0], Reference)

    def test_pick_element_point(self):
        selection = Pick()
        rv = selection.pick_pt_on_element('pick_pt_on_element')
        self.assertIsInstance(rv, Reference)
        rv = selection.pick_pt_on_element('pick_pt_on_element', multiple=True)
        self.assertIsInstance(rv[0], Reference)

    def test_pick_element_edge(self):
        selection = Pick()
        rv = selection.pick_edge('pick_edge')
        self.assertIsInstance(rv, Reference)
        rv = selection.pick_edge('pick_edges', multiple=True)
        self.assertIsInstance(rv[0], Reference)

    def test_pick_element_face(self):
        selection = Pick()
        rv = selection.pick_face('pick_face')
        self.assertIsInstance(rv, Reference)
        rv = selection.pick_face('pick_faces', multiple=True)
        self.assertIsInstance(rv[0], Reference)

    def test_pick_pt(self):
        selection = Pick()
        rv = selection.pick_pt('pick_pt')
        self.assertIsInstance(rv, XYZ)

    def test_pick_snaps(self):
        selection = Pick()
        rv = selection.pick_pt('pick_pt', snap='endpoints')
        self.assertIsInstance(rv, XYZ)

    def test_pick_box(self):
        selection = Pick()
        rv = selection.pick_box('PickBox')
        self.assertIsInstance(rv[0], XYZ)

    def test_pick_by_rectangle(self):
        selection = Pick()
        rv = selection.pick_by_rectangle('Pick By Rectangle')
        self.assertIsInstance(rv[0], Element)

    # def test_pick_linked(self):
    #     selection = Pick()
    #     rv = selection.pick_linked_element('pick_linked_element')
    #     rpw.ui.Console()
```

(continues on next page)

(continued from previous page)

```python
""" Revit Python Wrapper Tests - Forms

Passes:
2017

"""

import sys
import unittest
import os

test_dir = os.path.dirname(__file__)
root_dir = os.path.dirname(test_dir)
sys.path.append(root_dir)

import rpw
from rpw import revit, DB, UI
doc, uidoc = rpw.revit.doc, rpw.revit.uidoc

from rpw.utils.dotnet import List
from rpw.exceptions import RpwParameterNotFound, RpwWrongStorageType
from rpw.utils.logger import logger

data = ['A', 'B', 'C']

######################
# FORMS
######################


class FormSelectFromListTests(unittest.TestCase):

    def test_get_value(self):
        value = rpw.ui.forms.SelectFromList('Select From List Test', data,
                                            description='Select A and click select',
                                            exit_on_close=False)
        self.assertEqual(value, 'A')

    def test_get_dict_value(self):
        value = rpw.ui.forms.SelectFromList('Select From List Test', {'A':10},
                                                description='Select A and click select',
                                                exit_on_close=False)
        self.assertEqual(value, 10)

    def test_cancel(self):
        value = rpw.ui.forms.SelectFromList('Test Cancel', data,
                                            description='CLOSE WITHOUT SELECTING',
                                            exit_on_close=False)
        self.assertIsNone(value)

    def test_close_exit(self):
        with self.assertRaises(SystemExit):
```

(continues on next page)

(continued from previous page)

```
            rpw.ui.forms.SelectFromList('Text Exit on Close', data,
                                        description='CLOSE WITHOUT SELECTING',
                                        exit_on_close=True)


class FormTextInputTests(unittest.TestCase):

    def test_get_value(self):
        value = rpw.ui.forms.TextInput('Text Input', default='A',
                                       description='select with letter A',
                                       exit_on_close=False)
        self.assertEqual(value, 'A')

    def test_cancel(self):
        value = rpw.ui.forms.TextInput('Test Cancel', default='A',
                                   description='CLOSE FORM',
                                   exit_on_close=False)
        self.assertIsNone(value)

    def test_close_exit(self):
        with self.assertRaises(SystemExit):
            rpw.ui.forms.TextInput('Test Exit on Close', default='A',
                                    description='CLOSE FORM',
                                    exit_on_close=True)


class FlexFormTests(unittest.TestCase):

    def test_flex_form_launch(self):
        components = [rpw.ui.forms.Label('Test'), rpw.ui.forms.Button('Click Here')]
        form = rpw.ui.forms.FlexForm('Text Input', components)
        form_result = form.show()
        self.assertTrue(form_result)

    def test_flex_form(self):
        components = [rpw.ui.forms.Label('Test'),
                      rpw.ui.forms.TextBox('textbox', default='Default Value'),
                      rpw.ui.forms.ComboBox('combo', {'A':0, 'B':1}, default='B'),
                      rpw.ui.forms.CheckBox('checkbox', 'SELECTED', default=True),
                      rpw.ui.forms.Separator(),
                      rpw.ui.forms.Button('Click Here'),
                      ]
        form = rpw.ui.forms.FlexForm('Text Input', components)
        form_result = form.show()
        self.assertTrue(form_result)
        self.assertEqual(form.values['checkbox'], True)
        self.assertEqual(form.values['combo'], 1)
        self.assertEqual(form.values['textbox'], 'Default Value')
```

## 4.11 Contribute

### 4.11.1 Overview

This projects welcomes contributions in the form of bug reports, suggestions, as well as Pull Requests. If you have any questions about how to contribute just start an issue on the github repo and we will go from there.

### 4.11.2 Bug Reports

Please be as specific as you can when creating an issue, and always try to answer the questions: What are you trying to do? What was the result? What did you expect?

Also, try to pinpoint the cause or error by creating a small reproduceable snippet that isolates the issue in question from other variables.

### 4.11.3 Suggestions

There is no single correct way of writing these wrappers. Many of the design decisions were based on maintainability/scalability, code aesthetics, and trying to create intuitive and friendly design patterns.

If you think something can be improved, or have ideas, please send them our way

### 4.11.4 Pull Requests

Pull Requests are welcome and appreciated. If you are planning on sending PRs, please consider the following:

- Is the code style and patterns cohesive with the existing code base?
- Is the functionality being added belong to a general-purpose wrapper, or is it specific to a single project or used case? In other words, how likely is it to be used by others?

And Lastly, PRs should have the corresponding Tests and Documentation. This can sometimes be more work than writing the wrappers themselves, but they are essential.

#### Documentation

1. Make sure you are familiar with the existing documentation.
2. Write doc strings on your classes and methods as you go. The docstrings in rpw use the Google Style Python Strings. The style is documented here, and you can find some great examples here.
3. Duplicate one of the pages, and re-link the *autodoc* and *literal include* directives to point to your new class.

4. From a terminal on the *revitpythonlibrarywrapper.lib* folder, run the *build_docs.bat* file. This should rebuild the documentation and save it on the folder *docs/_build/html*. The Docs will be build remotely on readthedocs.org once the code is pushed to github.

#### Unit Tests

Testing python code in revit is a bit unconventional due to the need having to have Revit open. Rpw uses standard unit tests packaged as a pyRevit extensions. Take a look at some of the existing tests, and start by duplicate one of the existing tests. Then add the the parent folder of *rpw.extension* to your pyRevit Paths and reload. You should see a new tab with all the tests.

**Caution:** Do not run any of the tests with other revit files open. The current tests require the *collector.rvt* for the tests to execute properly. This is left over from earlier tests and we intend it to fix it at some point.

Tests should be as self-contained as possible in the sense that they should create and destroy objects and not be depend son a specific model state. The best approach is to setup your tests using standard API code, and then verify that the class returns the same response as the API by it self. And if any objects are created, try to clean up and destroy them using the tearDown methods. A simple example of a test for a Collector test might be something like this:

```
>>> from rpw import db, DB
>>> # Tests are generally group into Test Case classes. This part is omitted from␣
↪this example.
>>> def test_collector_of_class(self):
>>>   elements = DB.FilteredElementCollector(doc).OfClass(DB.View).ToElements()
>>>   rv = db.Collector(of_class='View').elements
>>>   # Test Return Value:
>>>   self.assertEqual(len(elements), len(rv))
>>>   self.assertEqual(elements[0].Id, rv[1].Id)
>>>   self.assertIsInstance(rv[0], DB.View)
```

# CHAPTER 5

# Quick Overview and Comparison

The examples below give a basic overview of how the library is used, paired with an example sans-rpw.

## 5.1 rpw and rpw.revit

```
>>> # Handles Document Manager and namespace imports for RevitPythonShell and Dynamo
>>> import rpw
>>> from rpw import revit, db, ui, DB, UI
# That's pretty much all you need
```

Without RPW

```
>>> # Dynamo Example
>>> import clr
>>> clr.AddReference('RevitAPI')
>>> clr.AddReference('RevitAPIUI')
>>> from Autodesk.Revit.DB import *
>>> from Autodesk.Revit.UI import *
>>> # RevitServices
>>> clr.AddReference("RevitServices")
>>> import RevitServices
>>> from RevitServices.Persistence import DocumentManager
>>> from RevitServices.Transactions import TransactionManager
>>> # doc and uiapp
>>> doc = DocumentManager.Instance.CurrentDBDocument
>>> uiapp = DocumentManager.Instance.CurrentUIApplication
>>> app = uiapp.Application
>>> uidoc = DocumentManager.Instance.CurrentUIApplication.ActiveUIDocument
```

## 5.2 Transaction

```
>>> # Using Wrapper - Same code for RevitPythonShell, and Dynamo
>>> from rpw import revit, db
>>> with db.Transaction('Delete Object'):
...     revit.doc.Remove(SomeElementId)
```

Without RPW

```
>>> # Typical Transaction In Dynamo
>>> import clr
>>> clr.AddReference("RevitServices")
>>> import RevitServices
>>> from RevitServices.Persistence import DocumentManager
>>> from RevitServices.Transactions import TransactionManager
>>> doc = DocumentManager.Instance.CurrentDBDocument
>>> TransactionManager.Instance.EnsureInTransaction(doc)
>>> doc.Remove(SomeElementId)
>>> TransactionManager.Instance.TransactionTaskDone()
```

```
>>> # Typical Transaction in Revit Python Shell / pyRevit
>>> import clr
>>> clr.AddReference('RevitAPI')
>>> from Autodesk.Revit.DB import Transaction
>>> doc = __revit__.ActiveUIDocument.Document
>>> transaction = Transaction(doc, 'Delete Object')
>>> transaction.Start()
>>> try:
...     doc.Remove(SomeElementId)
>>> except:
...     transaction.RollBack()
>>> else:
...     transaction.Commit()
```

## 5.3 Selection

```
>>> from rpw import ui
>>> selection = ui.Selection()
>>> selection[0]
< Autodesk.Revit.DB.Element >
>>> selection.elements
[< Autodesk.Revit.DB.Element >]
```

Without RPW

```
>>> # In Revit Python Shell
>>> uidoc = __revit__.ActiveUIDocument # Different for Dynamo
>>> selection_ids = uidoc.Selection.GetElementIds()
>>> selected_elements = [doc.GetElemend(eid) for eid in selection_ids]
```

## 5.4 Element

```
>>> from rpw import revit, db
>>> element = db.Element(SomeRevitElement)
>>> with db.Transaction('Set Comment Parameter'):
...     element.parameters['Comments'].value = 'Some String'
>>> element.parameters['some value'].type
<type: string>
>>> element.parameters['some value'].value
'Some String'
>>> element.parameters.builtins['WALL_LOCATION_LINE'].value
1
```

Access to original attributes, and parameters are provided by the *Element* wrapper.

More Specialized Wrappers also provide additional features based on its type: `DB.FamilyInstace` (*FamilyInstance*), `DB.FamilySymbol` (*FamilySymbol*), `DB.Family` (*Family*), and `DB.Category` (*Category*).

```
>>> instance = db.Element(SomeFamilyInstance)
<rpw:FamilyInstance % DB.FamilyInstance symbol:72" x 36">
>>> instance.symbol
<rpw:FamilySymbol % DB.FamilySymbol symbol:72" x 36">
>>> instance.symbol.name
'72" x 36"'
>>> instance.family
<rpw:Family % DB.Family name:desk>
>>> instance.family.name
'desk'
>>> instance.category
<rpw:Category % DB.Category name:Furniture>
>>> instance.symbol.instances
[<rpw:Instance % DB.FamilyInstance symbol:72" x 36">, ... ]
```

## 5.5 Collector

```
>>> from rpw import db
>>> walls = db.Collector(of_class='Wall')
<rpw:Collector % DB.FilteredElementCollector count:10>
>>> walls.wrapped_elements
[< instance DB.Wall>, < instance DB.Wall>, < instance DB.Wall>, ...]
```

```
>>> view = db.collector(of_category='OST_Views', is_type=False).first
```

The Collector Class is also accessible through the wrappers using the `collect()` method

```
>>> db.Room.collect()
<rpw:Collector % DB.FilteredElementCollector count:8>
>>> db.Room.collect(level='Level 1')
<rpw:Collector % DB.FilteredElementCollector count:2>
```

Without RPW

```
>>> # Typical API Example:
>>> from Autodesk.Revit.DB import FilteredElementCollector, WallType
>>> collector = FilteredElementCollector()
>>> walls = FilteredElementCollector.OfClass(WallType).ToElements()
```

## 5.6 ParameterSet

```
>>> import rpw
>>> filter_rule = db.ParameterFilter(param_id, greater=3)
>>> collector = db.Collector(of_class='WallType', paramter_filter=filter_rule)
```

Without RPW

```
>>> # Typical API Example:
>>> from Autodesk.Revit.DB import FilteredElementCollector, WallType
>>> from Autodesk.Revit.DB import ParameterFilterRuleFactory, ElementParameterFilter
>>>
>>> rule = ParameterFilterRuleFactory.CreateEqualsRule(param_id, some_string, False)
>>> filter_rule = ElementParameterFilter(rule)
>>> collector = FilteredElementCollector.OfClass(WallType).WherePasses(filter_rule)
```

## 5.7 Forms

```
>>> from rpw.ui.forms import SelectFromList
>>> options = ['Option 1','Option 2','Option 3']
>>> form = SelectFromList('Window Title', options)
>>> form.show()
>>> selected_item = form.selected
```

# Python Module Index

# Index