Automating PowerWorld with Python

SimAuto and Python Basics

PowerWorld Corporation

2001 South First Street
Champaign, Illinois 61820
+1 (217) 384.6330

Python Basics

- Interactive mode

```
D:\> python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World!')
Hello World!
>>>
```

- Scripts

```
D:\> type HelloWorld.py
print('Hello World!')
D:\> python HelloWorld.py
Hello World!
D:\>
```

Python Basics

- Keywords
  - None, False, True
  - and, not, or, is
  - for, in, while, if, elif, else
  - break, continue, pass, return, yield
  - try, except, finally, raise
  - import, from
  - del

Python Basics

- Indentation defines code blocks

```
for i in range(5):
    x += i
    if (i == 4):
        print(x)
```

- Comments

```
# This is a comment

"""
This is a
comment block
"""
```

Python Basics

- Input and Output

```
str = input('Enter a string: ')
print(str)
```

- Building strings

```
>>> x = 10
>>> name = 'Bob'
>>> print('{} ({})'.format(name, x))
Bob (10)

>>> arr = ['spam', 'eggs', 'coconuts']
>>> print(', '.join(arr))
spam, eggs, coconuts
```

Python Basics

- Conditionals

```
if (5 == 10):
    print('Math is broken!')
else:
    print('Order is restored.')
```

- Looping

```
for i in range(5):
    print(i)
```

Python Basics

- Importing modules

```
import os
import win32com.client
from pythoncom import VT_VARIANT
```

- Handling exceptions

```
try:
    print(unknown_identifier)
except:
    print('Danger, Will Robinson!')


try:
   x = 10/0
finally:
    print('Do this regardless')
```

Python Basics

- Lists, Dictionaries, Tuples

```
>>> my_list = [1,2,5]
>>> my_dict = {'a':1, 'b':2, 'c':3}
>>> my_tuple = ('a', 'b', 'c')
>>> my_list[1]
2
>>> my_dict['c'] = my_list
>>> my_list[0] = my_tuple[2]
>>> my_dict
{'a': 1, 'b': 2, 'c': ['c', 2, 5]}
>>> my_tuple[1] = 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Python Data Types

- Python does dynamic typing based on the value
  - Programmer does not explicitly declare a variable type
- Everything is an object – variables, functions, modules, etc.
- Variables are just names that reference an object
  - Variables themselves do not have types
  - It is the object that has the type
  - Implication is that the same variable name can be used to reference objects of different types

Mutable & Immutable

- Immutable types
  - Memory will be allocated once and re-used thereafter
  - Cannot be changed once it is created
    - String, int, float, complex, byte, tuple, set
    - Tuples cannot be changed once created but can contain a mutable object like a list than can be changed
- Mutable types
  - Objects can be modified after creation
  - Content can be changed without changing the identity
    - List, dictionary

Python Data Types

- Use `id(`*`object`*`)` function to return the unique identity of an object
  - Identity is an integer that is guaranteed to be unique and constant for this object during its lifetime
- Use `type(`*`object`*`)` function to return an object's type
- = will assign a variable to reference the same object as another variable
- Determine if objects or values are the same
  - == checks if the values of the operands are equal
    - `!=` checks if the values of the operands are not equal
  - `is` checks whether both operands reference the same object
    - `is not` checks whether both operands do not reference the same object
- *Object*`.copy()` will make a shallow copy of an object
  - Be careful with this if the object being copied contains mutable objects
  - Do you want a copy that will change if the original object changes or do you want the copy to be independent of the original?

Python Data Types Examples

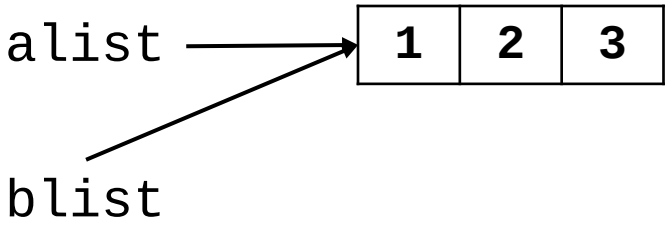
- The following shows that integer objects have unique identities
  - Different variables referencing the same value will have the same identity
  - The object being referenced determines the unique identity and not the variable itself
  - The same variable can reference different objects
  - The same object, e.g., 12 or 22, retains the same unique identifier

```
>>> a = 12
>>> id(a)
140712815456864
>>> a = 22
>>> id(a)
140712815457184
>>> b = 12
>>> id(b)
140712815456864
>>> b = a
>>> id(b)
140712815457184
>>> id(a)
140712815457184
```

Python Data Types Examples

- Assigning a variable to another variable using = will reference the same object

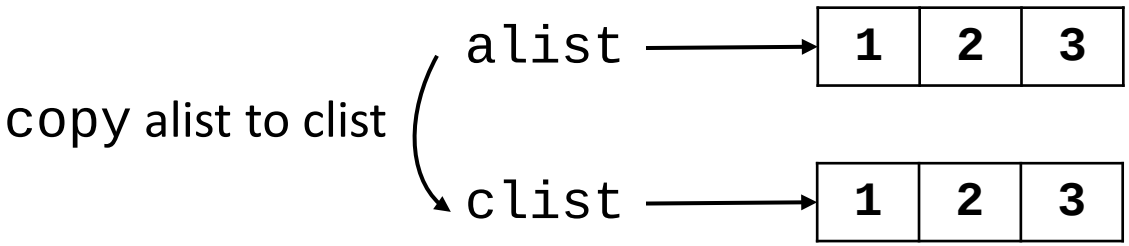
```
>>> alist = [1, 2, 3]
>>> id(alist)
2394224392712
>>> blist = alist
>>> id(blist)
2394224392712
>>> alist[0] = 11
>>> print(alist)
[11, 2, 3]
>>> print(blist)
[11, 2, 3]
>>> id(alist)
2394224392712
>>> id(blist)
2394224392712
```



Python Data Types Examples

- Copying a list from another list containing numbers will make a new object containing the same values
  - Objects have different identifiers but initially contain the same values
  - Because the list contains only immutable objects, changing the values in one list will not change the other list

```
>>> alist = [1, 2, 3]
>>> clist = alist.copy()
>>> id(alist)
2394193376136
>>> id(clist)
2394193320456
>>> print(alist)
[1, 2, 3]
>>> print(clist)
[1, 2, 3]
>>> alist[0] = 33
>>> id(alist)
2394193376136
>>> id(clist)
2394193320456
>>> print(alist)
[33, 2, 3]
>>> print(clist)
[1, 2, 3]
```



Python Data Types Examples

- Copying a list from another list that contains other lists is a different story
  - List being copied contains mutable objects, i.e., other lists
  - List objects contained in outer list are the same object after the copy
  - Changing a value in one inner list will change the value in the copied list
  - To truly make an independent copy, must copy the content of the lists and not the list objects themselves

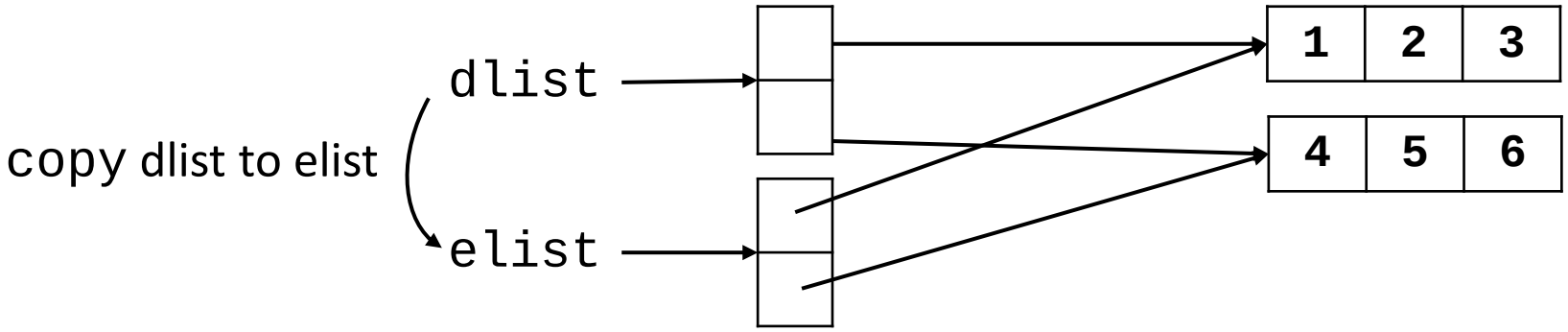
```
>>> dlist = []
>>> dlist.append([1, 2, 3])
>>> dlist.append([4, 5, 6])
>>> id(dlist)
2394192827464
>>> print(dlist)
[[1, 2, 3], [4, 5, 6]]
>>> elist = dlist.copy()
>>> id(elist)
2394224417736
>>> print(elist)
[[1, 2, 3], [4, 5, 6]]
>>> dlist[0][0] = 11
>>> print(dlist)
[[11, 2, 3], [4, 5, 6]]
>>> print(elist)
[[11, 2, 3], [4, 5, 6]]
>>> id(dlist[0])
2394224417928
>>> id(elist[0])
2394224417928
```

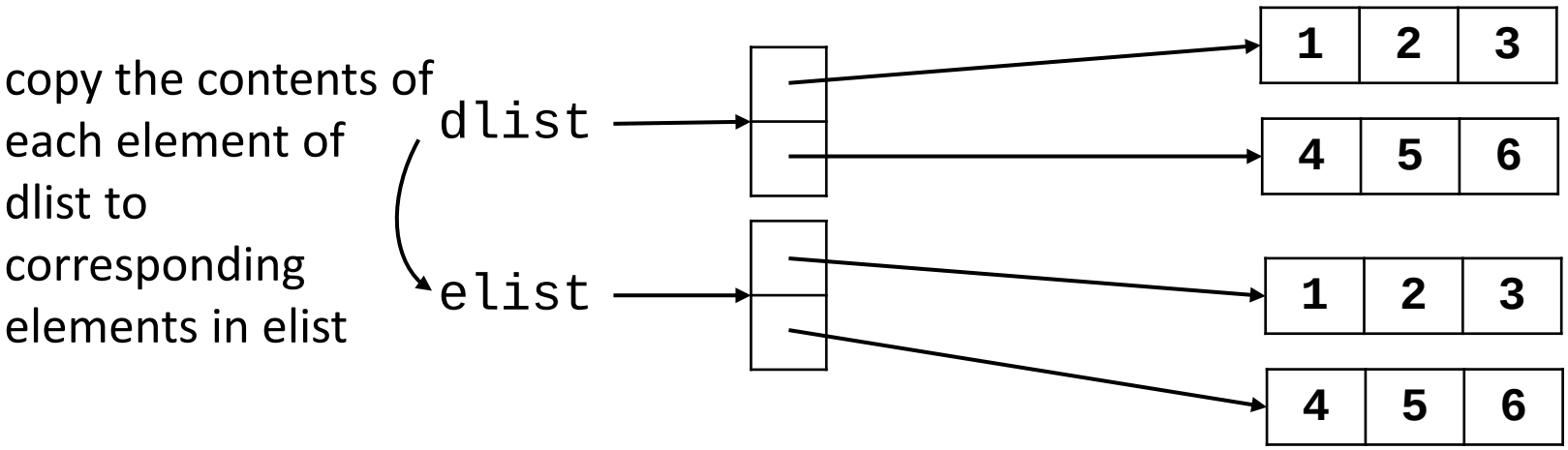
Python Data Types Examples

Copy the contents of one list containing other lists by making new list objects

```
>>> dlist
[[1, 2, 3], [4, 5, 6]]
>>> elist = []
>>> elist.append([dlist[0][i] for i in range(len(dlist[0]))])
>>> elist.append([d for d in dlist[1]])
>>> elist
[[1, 2, 3], [4, 5, 6]]
>>> elist == dlist
True
>>> elist is dlist
False
>>> id(elist)
2394224423304
>>> id(dlist)
2394224417672
>>> id(elist[0])
2394192827464
>>> id(dlist[0])
2394193376136
>>> id(elist[1])
2394224423048
>>> id(dlist[1])
2394224418760
>>> dlist[0][0] = 33
>>> dlist
[[33, 2, 3], [4, 5, 6]]
>>> elist
[[1, 2, 3], [4, 5, 6]]
```

Python Data Types Examples





SimAuto Add-on

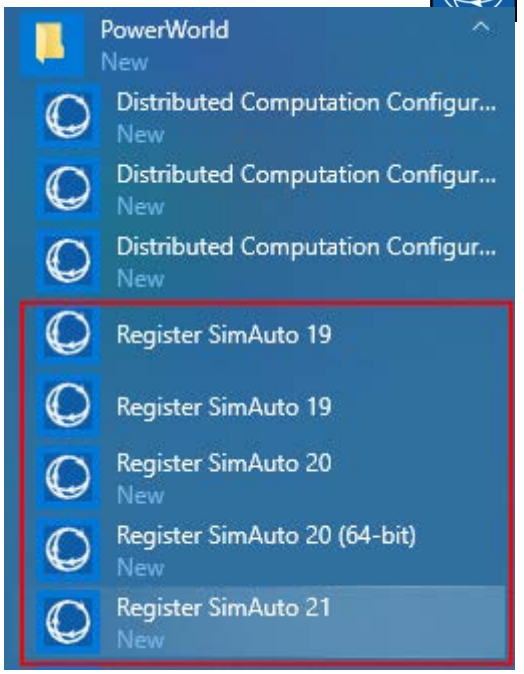
- SimAuto is an add-on and is not included with the base package of Simulator
- SimAuto is a Component Object Model (COM) automation server that allows Simulator to be controlled from an external application
- Microsoft documentation on COM: https://docs.microsoft.com/en-us/windows/win32/com/component-object-model--com--portal
- A client must be written in your programming language of choice, e.g., Python, to interact with Simulator

Registering SimAuto Automatically

- The Simulator installation MSI file will register SimAuto for use as a COM server
  - Simulator Version 20 installation will only register the 32-bit version
  - Simulator Version 21 installation will register the 64-bit (there is only one version)
  - Each installation will unregister EVERY version that is already registered and register according to the Simulator version being installed

Registering SimAuto After Installation

- Register SimAuto *X* tool found in the PowerWorld folder in the Start menu for installed versions of Simulator
  - Tool can be used if any problems encountered with SimAuto not being available or changing versions
  - Can also be run directly from the folder in which the pwrworld.exe is contained
  - Both 32-bit and 64-bit versions can be registered simultaneously, but this can be confusing
    - This should not be an issue going forward with Version 21 because only 64-bit version is available



Registering SimAuto After Installation

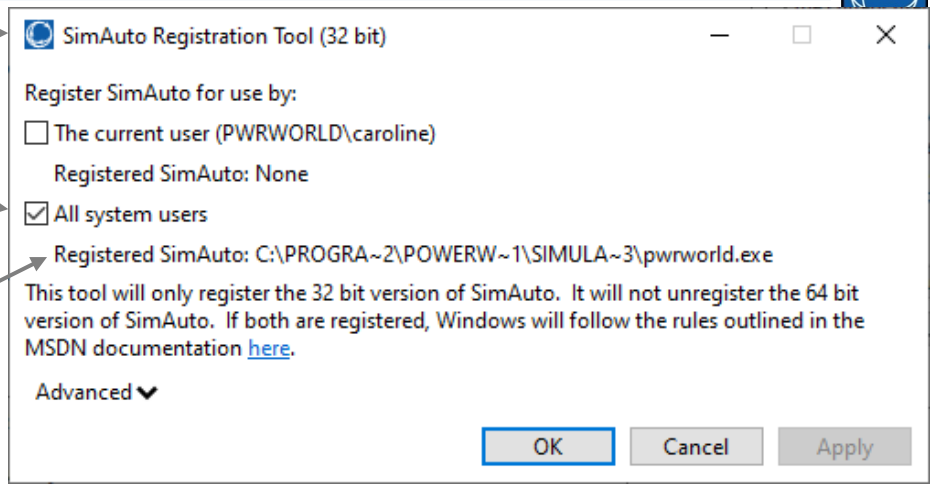
- Microsoft documentation describes how clients select the version of a COM server to use when different versions are registered
  - https://docs.microsoft.com/en-us/windows/win32/api/wtypesbase/ne-wtypesbase-clsctx?redirectedfrom=MSDN#CLSCTX_ACTIVATE_32_BIT_SERVER_and_CLSCTX_ACTIVATE_64_BIT_SERVER
- Best practice
  - Stick with one version of Simulator
  - Use the Register SimAuto tool and do not use manual registration

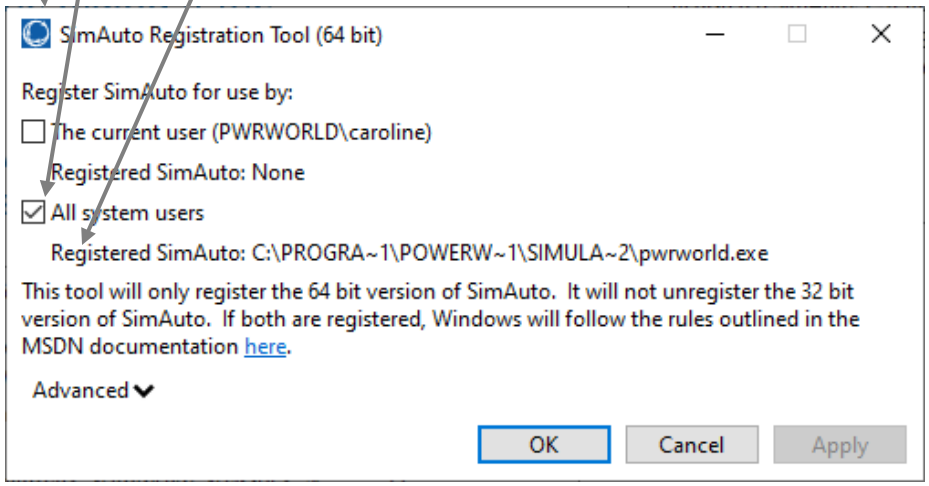
Register SimAuto Tool

Bit version of Simulator included in the caption

Check to register the version. This will be checked when opening the dialog if the version is already registered. Uncheck the box to unregister a version.

Shows the location of registered version

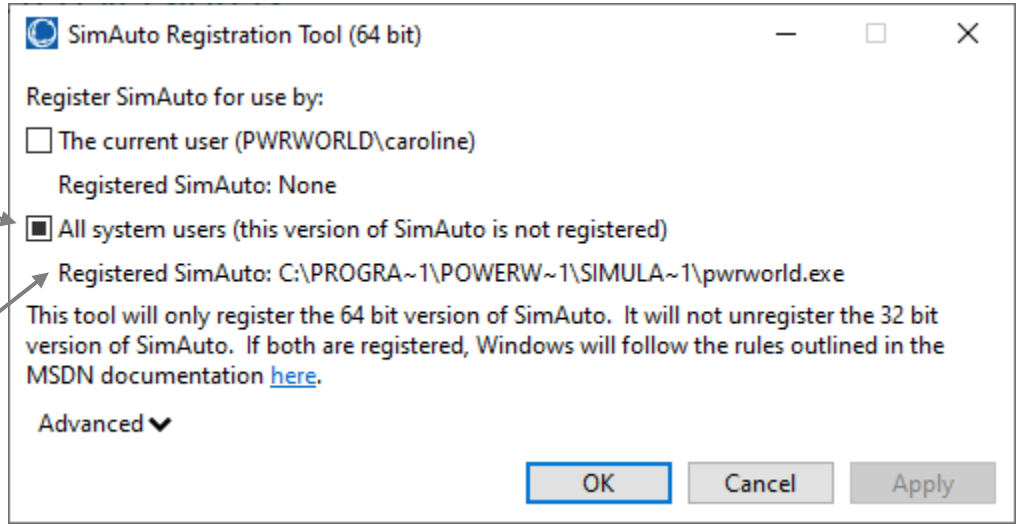




Register SimAuto Tool

Indicates that a 64-bit version is registered but it is not the version for which the Register SimAuto tool was run

Must look at the registered version to determine if the registered version is the correct version



Registering Desired Simulator Version

- Use the Register SimAuto tool to unregister the Simulator and bit version that you do not want
  - It is best practice to not have multiple bit versions registered in the first place
  - Registering a 64-bit version will not unregister a 32-bit version and registering a 32-bit version will not unregister a 64-bit version
- Use the Register SimAuto tool to register the Simulator and bit version that you do want

Connecting Python to SimAuto

- Requires pywin32 Python package that is not included with standard Python installation
  - https://pypi.org/project/pywin32/
    - `pip install pywin32`
- Code needed for connecting and disconnecting

```
import win32com.client

# The following will open the connection

simauto_obj = win32com.client.Dispatch("pwrworld.SimulatorAuto")

# Do something here

# The following will close the connection
del simauto_obj
```

Checking Simulator Version Using SimAuto

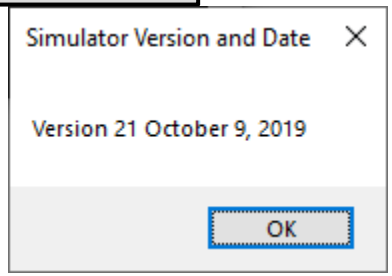
PW_Determine_Simulator_Version.py

```
# This code connects to SimAuto, determines the Simulator version and patch
# date in use, and shows a dialog with this information.
import win32com.client
import ctypes

simauto_obj = win32com.client.Dispatch("pwrworld.SimulatorAuto")
# To run script commands a case must be open, but the case does not have to
# contain anything.  NewCase will open a blank case, which will allow the
# check on the version of Simulator that is running.
simauto_obj.RunScriptCommand('NewCase;')
(err, output)= simauto_obj.GetParametersSingleElement('PowerWorldSession',
                                                      ['Version',
                                                       'EXEBuildDate'],
                                                       ['', ''])

ctypes.windll.user32.MessageBoxW(
          0, 'Version {} {}'.format(output[0], output[1]),
          'Simulator Version and Date', 0)
# The following will close the connection
del simauto_obj
```

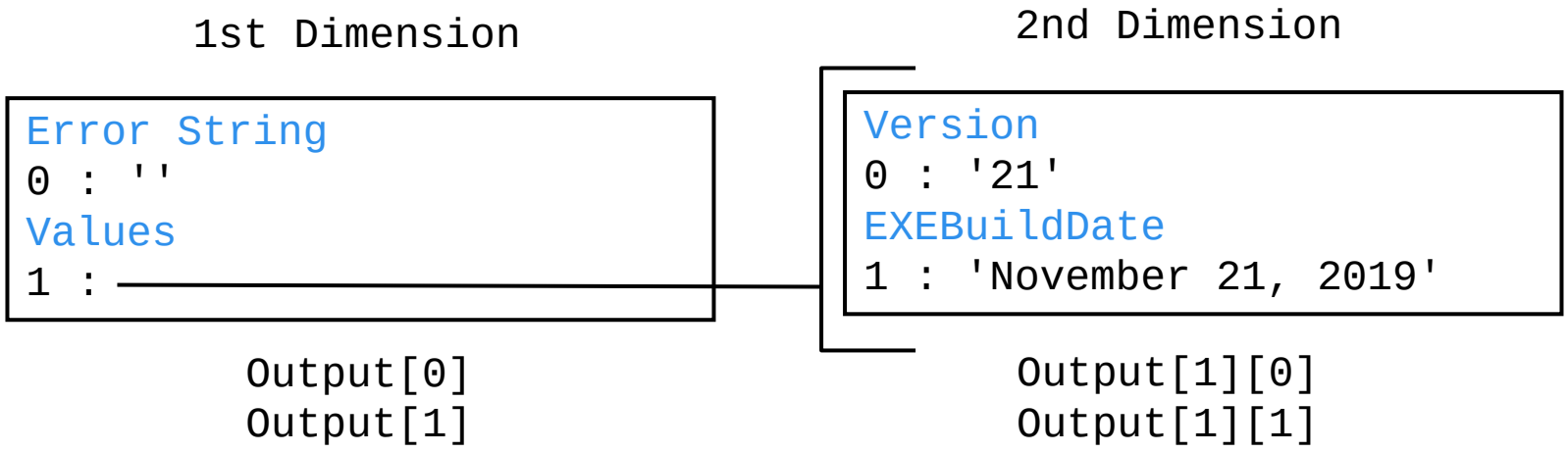
Result of this code is this informational message box



SimAuto Return Object

```
>>> simauto_output = simauto_obj.GetParametersSingleElement(
                                                    'PowerWorldSession',
                                                    ['Version', 'EXEBuildDate'],
                                                    ['', ''])

>>> print(simauto_output)
('', ('21', 'November 21, 2019'))
>>> print(type(simauto_output))
<class 'tuple'>
```



SimAuto Output Processing Function

sa_utility.py

```
# Function used to catch and display errors passed back from SimAuto
# SimAuto return object format:
# [0] = Error message, if any
# [1+] = Return data

def sa_chk(SimAutoOutput, SuccessMessage=''):

    if SimAutoOutput[0] != '':      # Error occurred – alert user and return nothing
        print('Error: ' + SimAutoOutput[0])
        return None
    else:                           # Success – display message if specified
        if (SuccessMessage != ''):
            print(SuccessMessage)

    if len(SimAutoOutput) == 1:     # No return value
        return None
    elif len(SimAutoOutput) == 2:   # Singular return value
        return SimAutoOutput[1]
    else:                           # Array of return values
        return SimAutoOutput[1:]
```

SimAuto Functions

- Documentation of SimAuto and available functions found online at https://www.powerworld.com/WebHelp/#MainDocumentation_HTML/Simulator_Automation_Server.htm
- Minimal set of functions with most used for data manipulation
- RunScriptCommand is used for all analysis
  - Allows any script command available for use in auxiliary files to be used with SimAuto
- Syntax used in following slides is valid for Python

ChangeParametersSingleElement (ObjectType, ParamList, Values)

- Input
  - ObjectType : String
    - The type of object for which parameters are being changed, e.g., "Bus"
  - ParamList : Variant
    - A variant array storing strings that are Simulator object field variables, e.g., "Number"
    - Must contain the key fields for the object type
  - Values : Variant
    - A variant array storing variants (integer, string, single, etc.) that are the values for each of the fields in the ParamList
- Output
  - Returns error string in Output[0]

ChangeParamatersSingleElement Python Example

PW_ChangeParametersSingleElement.py

```
object_type = 'Load'                          #ObjectType
param_list = ['BusNum', 'ID', 'MW']           #ParamList
# Update the MW values for load at bus 11009 with ID 1.
value_list = [11009,'1', 50]                  #Values
simauto_output = simauto_obj.ChangeParametersSingleElement(object_type, param_list,
    value_list)
```

- ObjectType is a string
- ParamList is a list of strings
- Values is a list of varying types

ChangeParametersMultipleElement (ObjectType, ParamList, ValueList)

- Input
  - ObjectType : String
    - The type of object for which parameters are being changed, e.g., "Bus"
  - ParamList : Variant
    - A variant array storing strings that are Simulator object field variables, e.g., "Number"
    - Must contain the key fields for the object type
  - ValueList : Variant
    - A variant array storing arrays of variants
    - Create variant arrays (one for each element being changed) with values corresponding to the fields in ParamList
    - Insert each of these variant arrays into ValueList
- Output
  - Returns error string in Output[0]

ChangeParametersMultipleElement Python Example

PW_ChangeParametersMultipleElement.py

```
object_type = 'Load'                    #ObjectType
param_list = ['BusNum', 'ID', 'MW']     #ParamList
# Update the MW values for loads at bus 11009 with IDs 1 and 2.
value_list = []                         #ValueList
value_list.append(VARIANT(pythoncom.VT_VARIANT | pythoncom.VT_ARRAY,[11009,'1', 50]))
value_list.append(VARIANT(pythoncom.VT_VARIANT | pythoncom.VT_ARRAY,[11009,'2', 100]))
simauto_output = simauto_obj.ChangeParametersMultipleElement(object_type, param_list,
    value_list)
```

- ObjectType is a string
- ParamList is a list of strings
- ValueList is a list containing variant arrays of variants

References

- Using VARIANT object: http://timgolden.me.uk/pywin32-docs/html/com/win32com/HTML/variant.html
- PyWin32 Documentation: http://timgolden.me.uk/pywin32-docs/contents.html
- *Python Programming On Win32: Help for Windows Programmers* by Mark Hammond
- https://docs.microsoft.com/en-us/windows/win32/api/wtypes/ne-wtypes-varenum

ChangeParametersMultipleElement Python Example

- Input into SimAuto is a variant array of arrays containing variants
- The Python COM interface attempts to take the parameters passed in and interpret them correctly
  - Python list of lists will be interpreted as a rectangular, i.e. two-dimensional, array with resulting error
    - `ChangeParametersMultipleElement: Exception: Variant or safe array index out of bounds`
- Must convert the input list into a variant array of variants
  - `(VARIANT(pythoncom.VT_VARIANT | pythoncom.VT_ARRAY,[11009,'1', 50]))`

ChangeParametersMultipleElementFlatInput (ObjectType,ParamList,NoOfObjects,ValueList)

- Input
  - ObjectType : String
    - Type of object for which parameters are being changed, e.g., "Bus"
  - ParamList : Variant
    - A variant array storing strings that are Simulator object field variables, e.g., "Number"
    - Must contain the key fields for the object type
  - NoOfObjects : Integer
    - Number of devices for which values are being passed
  - ValueList : Variant
    - Single-dimensional variant array storing a list of variants (integer, single, string, etc.) representing the values corresponding to ParamList for all devices being changed
    - All parameters for the first object are listed first followed by all of the parameters for the second object, etc.
      - ValueList = [Obj1Param1, Obj1Param2, …Obj1ParamM, Obj2Param1, Obj2Param2, … Obj2ParamM, Obj3Param1, …ObjNParam1, …ObjNParamM]
- Output
  - Returns error string in Output[0]

ChangeParametersMultipleElementFlatInput Python Example

PW_ChangeParametersMultipleElementFlatInput.py

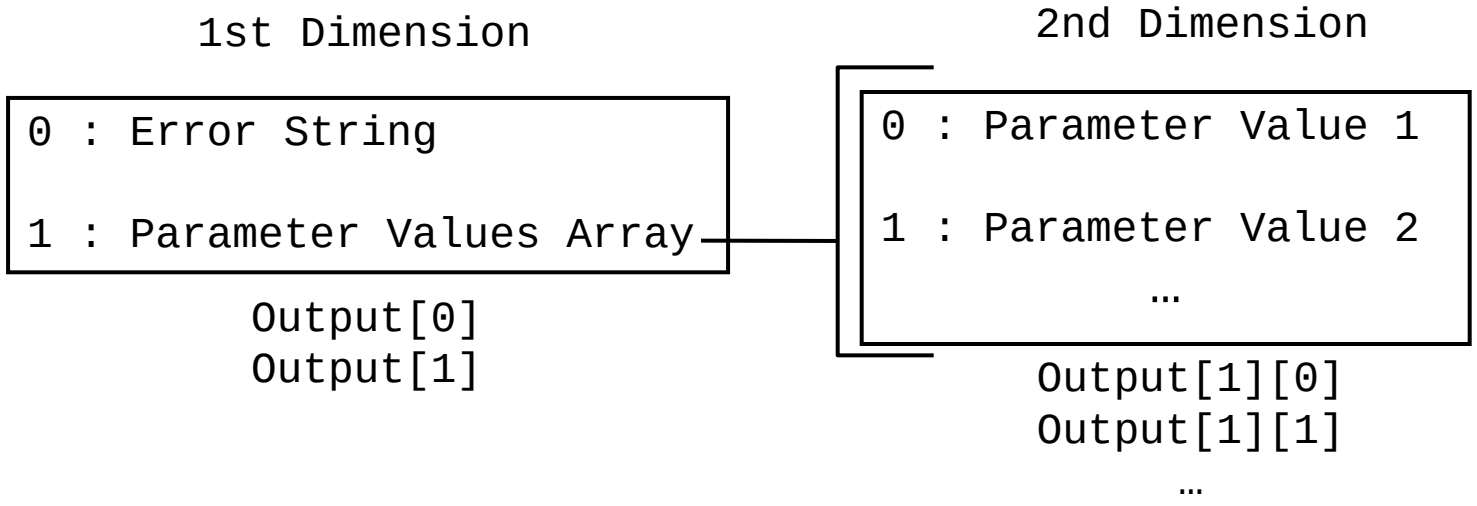
```
object_type = 'Load'                                                    #ObjectType
param_list = ['BusNum', 'ID', 'MW']                                     #ParamList
num_objects = 2                                                         #NoOfObjects
# Update the MW values for loads at bus 11009 with IDs 1 and 2.
value_list = [11009, '1', 50, 11009, '2', 100]                            #ValueList
simauto_output = simauto_obj.ChangeParametersMultipleElementFlatInput(object_type, param_list,
    num_objects, value_list)
```

- ObjectType is a string
- ParamList is a list of strings
- NoOfObjects is an integer
- ValueList is a list of varying types

GetParametersSingleElement (ObjectType, ParamList, Values)

- Input
  - ObjectType : String
    - The type of object for which parameters are being retrieved, e.g., "Bus"
  - ParamList : Variant
    - A variant array storing strings that are Simulator object field variables, e.g., "Number"
    - Must contain the key fields for the object type
  - Values : Variant
    - A variant array storing variants (integer, string, single, etc.) that are the values for each of the fields in the ParamList
    - Values must be passed in for the key fields
    - Values other than key fields should be set to zero
- Output
  - First element, Output[0], contains error string
  - Second element, Output[1], is a one dimensional variant array containing variants corresponding to fields specified in ParamList

GetParametersSingleElement Output



GetParametersSingleElement Python Example

PW_GetParametersSingleElement.py

```
object_type = 'Load'                          #ObjectType
param_list = ['BusNum', 'ID', 'MW']           #ParamList
# Get the MW value for the load at bus 11009 with ID 1.  Pass in 0 for values other than keys.
value_list = [11009,'1', 0]                   #Values
(err, output) = simauto_obj.GetParametersSingleElement(object_type, param_list, value_list)
print(output)     # BusNum, ID, MW
print(output[0])  # BusNum
print(output[1])  # ID
print(output[2])  # MW
```

- ObjectType is a string
- ParamList is a list of strings
- Values is a list of varying types
- Output[1] is a tuple of strings (notice the single quotes below when printing the output)

```
(' 11009', '1 ', '  8.78020003')
 11009
1
  8.78020003
```

Notice the quotes around ALL parameters that are being returned. This means that numeric results cannot be used directly without type conversion.

GetParameters… Results Checking Data Type

- Results are returned as strings even for numeric data types
- Conversion must be done before numeric calculations can be done
- Create Python functions that return whether or not a string is numeric and the converted value in a tuple

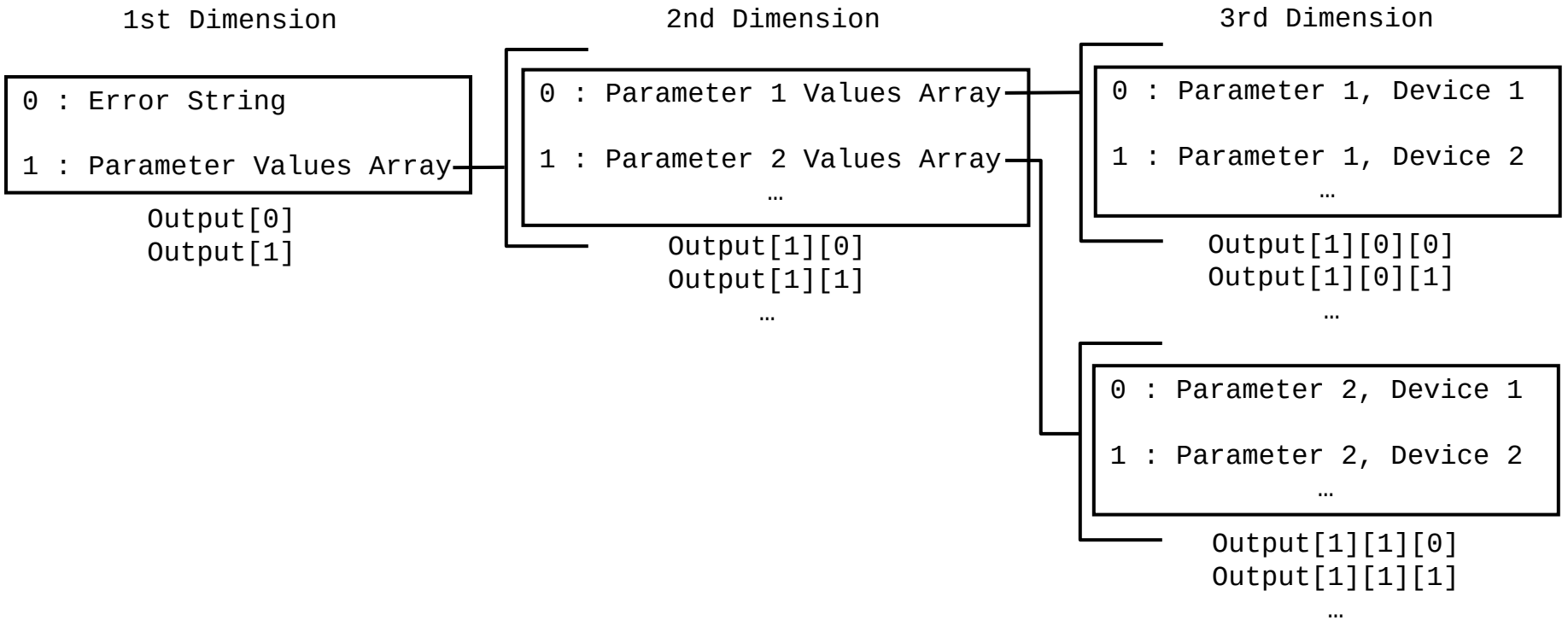
```
def is_float(localstring, defaultfloat):
    try:
        localvalue = float(localstring)
        return(True, localvalue)
    except ValueError:
        return(False, defaultfloat)

def is_integer(localstring, defaultint):
    try:
        localint = int(localstring)
        return(True, localint)
    except ValueError:
        return(False, defaultint)
```

GetParametersMultipleElement (ObjectType, ParamList, FilterName)

- Input
  - ObjectType : String
    - The type of object for which parameters are being retrieved, e.g., "Bus"
  - ParamList : Variant
    - A variant array storing strings that are Simulator object field variables, e.g., "Number"
  - FilterName : String
    - Name of a pre-defined advanced filter that will limit the objects returned
    - All objects of the specified type will be returned if the filter cannot be found
    - Pass an empty string to return all objects of the specified type
- Output
  - First element, Output[0], contains error string
  - Second element, Output[1]
    - Variant array of variant arrays containing variants for the parameter values for the object type requested
    - Number of arrays returned depends on the number of fields in ParamList

GetParametersMultipleElement Output



GetParametersMultipleElement Python Example

PW_GetParametersMultipleElement.py

```
object_type = 'Load'                          #ObjectType
param_list = ['BusNum', 'ID', 'MW']           #ParamList
filter_name = '<DEVICE>Bus 11009'             #FilterName
(err, output) = simauto_obj.GetParametersMultipleElement(object_type, param_list, filter_name)
print(output)
for i in range(len(output)):          # Number of fields
    for j in range(len(output[i])):   # Number of objects
        print(output[i][j])
```

- ObjectType is a string
- ParamList is a list of strings
- FilterName is a string – using this device filter will return all loads at this bus
- Output[1] is a tuple of tuples of strings (notice the single quotes below when printing the output)

```
((' 11009', ' 11009'), ('1 ', '2 '), ('  8.78020003', '  3.50739993'))
 11009
 11009
1
2
  8.78020003
  3.50739993
```

GetParametersMultipleElementRect (ObjectType, ParamList, FilterName)

- Input
  - ObjectType : String
    - The type of object for which parameters are being retrieved, e.g., "Bus"
  - ParamList : Variant
    - A variant array storing strings that are Simulator object field variables, e.g., "Number"
  - FilterName : String
    - Name of a pre-defined advanced filter that will limit the objects returned
    - All objects of the specified type will be returned if the filter cannot be found
    - Pass an empty string to return all objects of the specified type
- Output
  - First element, Output[0], contains error string
  - Second element, Output[1]
    - Two-dimensional variant array containing variants with each row representing an object and the corresponding columns representing each field in ParamList for that object

GetParametersMultipleElementRect Output

1st Dimension

```
0 : Error String

1 : Parameter Values Rectangular Array
    N x M where N = number of objects
                M = number of fields
```

Output[0]
Output[1]

Notice the index order is reversed for these results relative to GetParametersMultipleElement

2nd Dimension

Index	0	1	…	M-1
0	Object 1, Value 1	Object 1, Value 2	…	Object 1, Value M
1	Object 2, Value 1	Object 2, Value 2	…	Object 2, Value M
…	…	…	…	…
N-1	Object N, Value 1	Object N, Value 2	…	Object N, Value M

```
Output[1][0][0]
Output[1][0][1]
…
Output[1][0][M-1]
Output[1][1][0]
Output[1][1][1]
…
Output[1][N-1][M-1]
```

GetParametersMultipleElementRect Python Example

PW_GetParametersMultipleElementRect.py

```
object_type = 'Load'                          #ObjectType
param_list = ['BusNum', 'ID', 'MW']           #ParamList
filter_name = '<DEVICE>Bus 11009'             #FilterName
(err, output) = simauto_obj.GetParametersMultipleElementRect(
                                                  object_type, param_list, filter_name)
print(output)
for i in range(len(output)):          # Number of objects
    for j in range(len(output[i])):  # Number of fields
        print(output[i][j])
```

- ObjectType is a string
- ParamList is a list of strings
- FilterName is a string – using this device filter will return all loads at this bus
- Output[1] is a tuple of tuples of strings (notice the single quotes below when printing the output)

```
((' 11009', '1 ', '  8.78020003'), (' 11009', '2 ', '  3.50739993'))
 11009
1
  8.78020003
 11009
2
  3.50739993
```

GetParametersMultipleElementFlatOutput (ObjectType,ParamList,FilterName)

- Input
  - Handled in the same manner as GetParametersMultipleElement
- Output
  - Single-dimensional variant array of variants instead of nested arrays
    - [errorstring, NumberOfObjectsReturned, NumberOfFieldsPerObject, Ob1Fld1, Ob1Fld2, …, Ob(n)Fld(m-1), Ob(n)Fld(m)]
    - Output[0] still contains error string

GetParametersMultipleElementFlatOutput Python Example

PW_GetParametersMultipleElementFlatOutput.py

```
object_type = 'Load'                        #ObjectType
param_list = ['BusNum', 'ID', 'MW']         #ParamList
filter_name = '<DEVICE>Bus 11009'           #FilterName
simauto_output = simauto_obj.GetParametersMultipleElementFlatOutput(object_type, param_list, filter_name)
print(simauto_output)
num_objects = is_integer(simauto_output[1], 0)
num_fields = is_integer(simauto_output[2], 0)
if (num_objects[0] and (num_objects[1] > 0) and
    num_fields[0] and (num_fields[1] > 0)):
    resultsindex = 0
    for i in range(num_objects[1]):
        for j in range(num_fields[1]):
            print('Object: {0} Field: {1} : {2}'.format(i, j, simauto_output[3+resultsindex]))
            resultsindex += 1
```

- ObjectType is a string
- ParamList is a list of strings
- FilterName is a string – using this device filter will return all loads at this bus
- Output is a tuple of strings (notice the single quotes below when printing the output)

```
('', '2', '3', ' 11009', '1 ', '  8.78020003', ' 11009', '2 ', '  3.50739993')
Object: 0 Field: 0 :  11009
Object: 0 Field: 1 : 1
Object: 0 Field: 2 :   8.78020003
Object: 1 Field: 0 :  11009
Object: 1 Field: 1 : 2
Object: 1 Field: 2 :   3.50739993
```

GetFieldList (ObjectType)

- Returns all fields associated with a given object type
  - ObjectType : String
    - Type of object for which fields are requested, e.g., "Bus
- Output
  - First element, Output[0], contains error string
  - Second element, Output[1], is a n x 5 variant array of variants containing the fields
    - Similar to information obtained from Export Case Object Fields…
    - [n][0] specifies the key and required fields
      - Key - *1*,*2*, etc.
      - Secondary Key – *A*, *B*, etc.
      - Required – **
    - [n][1] variablename of the field
      - This is always the legacy variablename
    - [n][2] type of data stored in the field (integer, string, real)
    - [n][3] field description
    - [n][4] concise variablename of the field

GetFieldList
Python Example

PW_GetFieldList.py

```
object_type = 'Load'                          # ObjectType
(err, output) = simauto_obj.GetFieldList(object_type)
print(output)
```

- ObjectType is a string
- Output[1] is a tuple of tuples of strings

```
(('', 'ABCLoadAngle', 'Real', 'Load Phase Angle A angle during fault ', 'FaultCurAngA'), ('',
'ABCLoadAngle:1', 'Real', 'Load Phase Angle B angle during fault ', 'FaultCurAngB'), ('',
'ABCLoadAngle:2', 'Real', 'Load Phase Angle C angle during fault ', 'FaultCurAngC'), ('',
'ABCLoadI', 'Real', 'Load Phase Current A current (Amps) during fault ', 'FaultCurMagA'), ('',
'ABCLoadI:1', 'Real', 'Load Phase Current B current (Amps) during fault ', 'FaultCurMagB'),
…
('*1*', 'BusNum', 'Integer', 'Number of the bus', 'BusNum')
…
)
```

GetFieldList
Python Example

PW_GetFieldList.py

Store results for GetFieldList in a dictionary to easily look up field parameters by variablename

```
object_type = 'Load'                          # ObjectType
(err, output) = simauto_obj.GetFieldList(object_type)
# Create a dictionary that is keyed by the concise variablename and contains all parameters.
# Force the values to be a list, otherwise they will default to a tuple.
fieldlist_dict = {output[i][4]: list(output[i]) for i in range(len(output))}
myvariable = 'MW'
if myvariable in fieldlist_dict:
    print(fieldlist_dict[myvariable])
```

Contents of values in the dictionary based on the *MW* dictionary key

```
['', 'LoadMW', 'Real', 'Total MW load seen by the system from this bus.  Determined from the
constant power, current, and impedance portion of the loads', 'MW']
```

GetSpecificFieldList (ObjectType, FieldList)

- Returns all specified fields associated with a given object type
  - ObjectType : String
    - Type of object for which fields are requested, e.g., "Bus“
  - FieldList : Variant
    - A variant array storing strings that are Simulator object field variables, e.g., "Number"
- Output
  - First element, Output[0], contains error string
  - Second element, Output[1], is an n x 4 variant array of variants containing the fields
    - [n][0] variablename of the field
      - This is either the legacy or concise variablename depending on how the option to display variablenames is set
    - [n][1] field identifier (identifier that appears when looking at list of available fields)
    - [n][2] column header
    - [n][3] field description

GetSpecificFieldList Python Example

PW_GetFieldList.py

Append extra information contained in GetSpecificFieldList to the dictionary of results from GetFieldList

```
# Append information that is in GetSpecificFieldList to dictionary.
for fieldlistdictkey in fieldlist_dict:
    (err, output) = simauto_obj.GetSpecificFieldList(object_type, [fieldlistdictkey])
    fieldlist_dict[fieldlistdictkey].append(output[0][1])    # field identifier
    fieldlist_dict[fieldlistdictkey].append(output[0][2])    # column header
myvariable = 'MW'
if myvariable in fieldlist_dict:
    print(fieldlist_dict[myvariable])
```

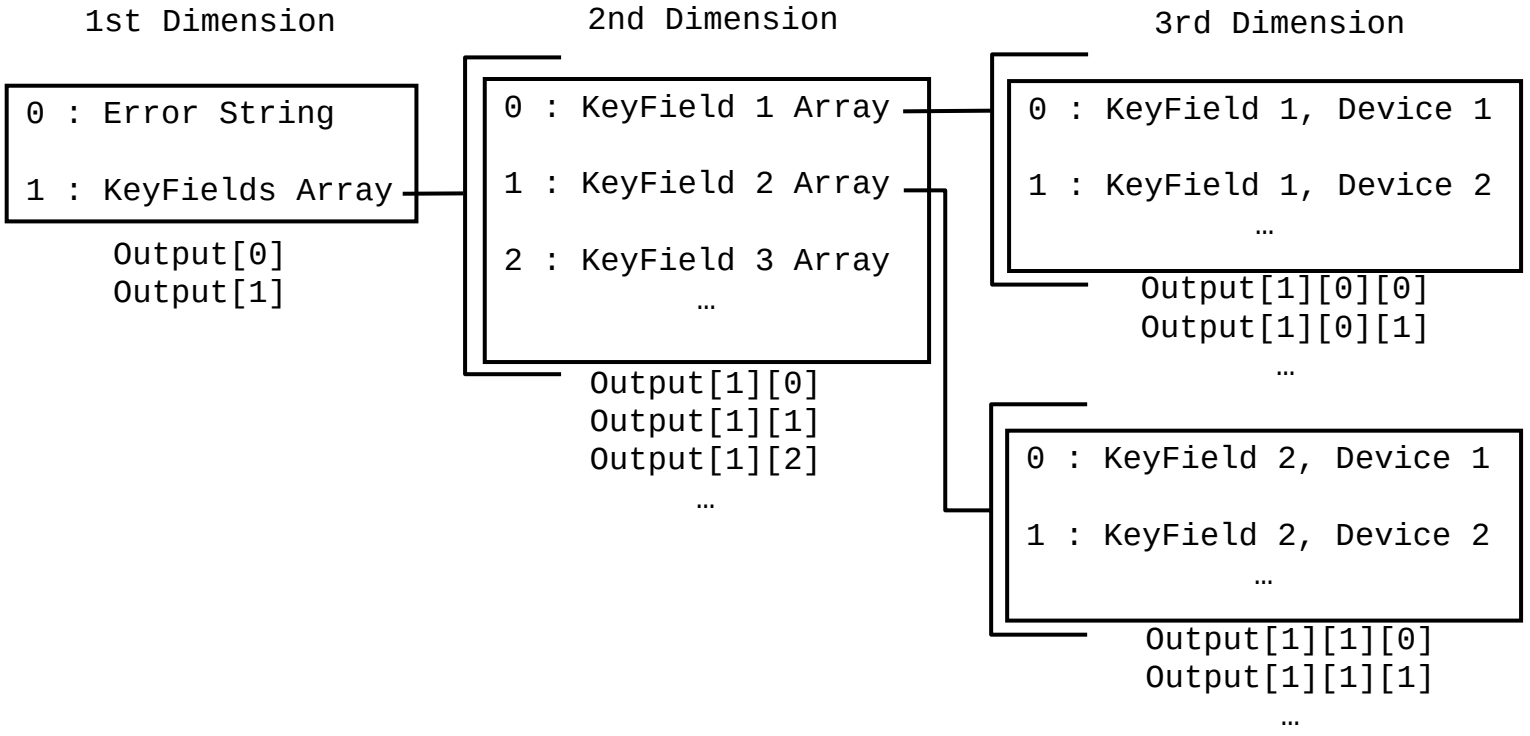
Contents of values in the dictionary based on the *MW* dictionary key after appending information

```
['', 'LoadMW', 'Real', 'Total MW load seen by the system from this bus.  Determined from the
constant power, current, and impedance portion of the loads', 'MW', 'MW\\MW', 'MW']
```

ListOfDevices (ObjectType,FilterName)

- Input
  - ObjectType : String
    - Type of object for which devices are being acquired.
  - FilterName : String
    - Name of a pre-defined advanced filter that will limit the objects returned
    - All objects of the specified type will be returned if the filter cannot be found
    - Pass an empty string to return all objects of the specified type
- Output
  - First element, Output[0], contains error string
  - Second element, Output[1], is a variant array of variant arrays containing the key field values for the type of object requested
    - Number of arrays returned depends on the object type selected
    - Values in the arrays are strongly typed, i.e. bus numbers are returned as integers instead of as a variant
    - Use ListOfDevicesAsVariantStrings to return values as variants

ListOfDevices Output



ListOfDevices
Python Example

PW_ListOfDevices.py

```
object_type = 'Load'                          #ObjectType
filter_name = '<DEVICE>Bus 11009'             #FilterName
(err, output) = simauto_obj.ListOfDevices(object_type, filter_name)
print(output)
for i in range(len(output)):          #Number of key fields
     for j in range(len(output[i])): #Number of objects
         print(output[i][j])
print(type(output[0][1]))  # Show that this is an integer
```

- ObjectType is a string
- FilterName is a string – using this device filter will return all loads at this bus
- Output[1] is a tuple of tuples of particular data types (notice the lack of single quotes below on some of the fields)

```
((11009, 11009), ('1 ', '2 '))
11009
11009
1
2
<class 'int'>
```

RunScriptCommand (Statements)

- Input
  - Statements : String
    - List of script statements to be executed
    - Each script statement must end in a semicolon
    - List of script statements should not be enclosed in curly braces
- Output
  - Returns error string in Output[0]

RunScriptCommand Python Example

```
cmds = [
          'LoadAux(“prep.aux”)',
          'SolvePowerFlow',
          'SaveDataUsingExportFormat(“output.aux”, AUX, “my_results”)’
       ]
cmd = '; '.join(cmds)                    # Pass in a single, semi-colon-delimited string
(err,) = simauto_obj.RunScriptCommand(cmd)
if err != '':
          raise Exception('Error running “{}”: {}'.format(cmd, err))
```

Questions?