Automation
For Future Industry

o6\Python API OPC UA Client Cheat Sheet

Installation: `pip install o6`

Documentation: https://docs.o6-automation.com/o6-python/latest/

Synchronous Client

```python
import o6                                # Load o6 library
c = o6.Client("opc.tcp://host:4840")     # Create client
c.connect()                              # Connect anonymous
# Object-syntax to browse (for unique node BrowseNames)
temp = c.objects.sensor.temperature() # Variable read
# No variant container, temp has the target DataType
if temp > 20
    c.objects.cooling.power(20.0)        # Variable write
    c.objects.cooling.enable()           # Method call
c.disconnect()                           # Clean up
```

Asynchronous Client

```python
import o6
import asyncio
async def main_async():
    # Client integrates with the current eventloop (can be customized)
    async with o6.Client("opc.tcp://host:4840") as c:
        temp = await c.objects.sensor.temperature() # Variable read
        if temp > 20:
            await c.objects.cooling.power(20.0)      # Variable write
            await c.objects.cooling.enable()         # Method call
asyncio.run(main_async())                            # Run to completion
```

Licensing

- Developer seat / volume license:
  www.o6-automation.com/o6-python
  or contact sales@o6-automation.com
- Request a free non-commercial / research license
- AGPL Dual License for Open Source:
  https://github.com/o6-automation/o6-python-agpl

OPC UA Builtin DataTypes

```python
import o6
import uuid
import numpy as np

# Numerical
o6.Boolean    # <np.bool>
o6.Byte       # <np.uint8>
o6.SByte      # <np.int8>
o6.Int32      # <np.int32> (16/32/64bit)
o6.UInt32     # <np.uint32> (16/32/64bit)
o6.Float      # <np.float>
o6.Double     # <np.double>
o6.DateTime   # Wraps <np.int64>
o6.StatusCode # Wraps <np.uint32>

# Python types
o6.String     # Python <str>
o6.ByteString # Python <bytes>
o6.XmlElement # Wraps <o6.String>
o6.Guid       # <uuid.UUID>

# OPC UA-specific
o6.NodeId
o6.ExpandedNodeId
o6.QualifiedName
o6.LocalizedText
o6.ExtensionObject
o6.DataValue
o6.DiagnosticInfo
```

Automatic Casting
Python <int> casts o6.Int32
Python <float> casts to o6.Double

No Variant Type
Instead of a "variant" container, any valid OPC UA datatype can be used directly. Lists and (multi-dimensional) numpy-arrays are also possible.

Read/Write/Call

```python
# Read
value = c.objects.server.serverStatus()                  # Default: attr='value'
dv    = c.objects.server.serverStatus(attr='dataValue')  # Value as <o6.DataValue>
bn    = c.objects.server(attr='browseName')              # <o6.QualifiedName>
value = c.objects.myMatrix(range='1:2,0:1')              # Returns a 2x2 matrix

# Write
var(1.0)     # Default: attr='value'
obj(o6.QualifiedName('1:MyName'), attr='browseName')
c.objects.myMatrix(np.array([[1.,2.],[3.,4.]]),
                           range='1:2,0:1')

# Call
m = c.objects.myObj.myMethod          # myObj method context
print(m.inputArguments)               # Inspect arguments
status, *results = m(5, "test")       # In-order arguments
status, *results = m(5, object=obj) # Different context
```

Attribute	DataType
nodeId	o6.NodeId
nodeClass	o6.NodeClass
browseName	o6.QualifiedName
displayname	o6.LocalizedText
description	o6.LocalizedText
value	<Any>
dataValue	o6.DataValue
...	...

Nodes and Namespaces

```python
# Global namespace definitions to be reused in client/server
ns0 = o6.ns.ns0 # <o6 namespace 'ns0' version='1.05.07' uri='http://opcfoundation.org/UA/'>
di  = o6.ns.di  # <o6 namespace 'di', version='1.05.0' uri='http://opcfoundation.org/UA/DI/'>

# Global Namespace-Nodes
rr = ns0.datatypes.ReadRequest() # Creates an instance of the datatype
st = ns0.objtypes.ServerType     # Namespace-level ObjectType definition
ss = st.serverStatus             # See /o6/ns/ns0/objtypes.py for the definition

# Client-specific nodes
# Node names are normalized to Python syntax without the namespace index.
# Duplicates in the normalized node names need to be resolved via the unique NodeId.
obj = c.objects.server                  # <o6.node.ObjectNode>
var = c.objects.server.serverStatus     # <o6.node.VariableNode>
```

Subscriptions / Monitoring

```python
# Monitor with the default subscription (publishing interval: 100.0 ms)
print_cb = lambda mon,val: print(f"{mon.id}: {val}") # Callback function
c.monitor(c.objects.myVariable, callback=print_cb, samplingInterval=200.0)

# Monitor with a custom subscription (publish with 500ms, sample with 50ms interval)
sub = c.createSubscription(publishingInterval=500.0)
mon = c.monitor(c.objects.myVariable, callback=print_cb,
                samplingInterval=50.0, subscription=sub)
# c.monitorEvent(...) for event monitoring

mon.delete() # Remove MonitoredItem
sub.delete() # Remove Subscription
```