EPICS Support for OPC UA – Cheat Sheet

This document version applies to release 0.11.

Concepts

A *Session* is a named connection between client and server *(0..n per IOC)*.
Any variable on the server is also called an *Item*, which is accessed through its *NodeID*, consisting of a numerical *namespace index* and a *name* or *numerical ID*.
To monitor (subscribe to) items, they are added to a *Subscription (0..n per Session)*.

Initialization from startup script

`opcuaSession <name> <server URL> [options…]`

Create a new session.

name	Name of the session to create
server URL	URL of the OPC UA server to connect to
options	Option list (series of ‘key=value’ pairs)

`opcuaSubscription <name> <session> <interval> [options…]`

Create a new subscription.

name	Name of the subscription to create
session	Name of the session this subscription is related to
interval	Publishing interval for the subscription in ms [double]
options	Option list (series of ‘key=value’ pairs)

`opcuaOptions <pattern> <options…>`

Set session/subscription specific options.

pattern	Name pattern for the session/subscription to configure (* and ? are supported for batch operations)
options	Option list (series of ‘key=value’ pairs)

Valid session options for the Unified Automation SDK client:

debug	Verbosity level of debugging [default: 0 = off]
autoconnect	Automatically connect/reconnect to server [default: y]
nodes-max	Maximum number of nodes used in any low-level service call (client will split larger requests into multiple batches)
read-nodes-max	Maximum number of nodes per read service call (client will split larger read requests into multiple batches)

read-timeout-min	Timeout (holdoff period) after read service call [ms] (with one node if read-timeout-max is set)
read-timeout-max	Timeout (holdoff period) after read service call with max number of nodes [ms]
write-nodes-max	Maximum number of nodes per write service call (client will split larger write requests into multiple batches)
write-timeout-min	Timeout (holdoff period) after write service call [ms] (with one node if write-timeout-max is set)
write-timeout-max	Timeout (holdoff period) after write service call with max number of nodes [ms]
sec-mode	Requested security mode [default: 'best'; must use 'None' for running with no security]
sec-policy	Requested security policy [default: use best available]
sec-id	Set file to read identity credentials from

Valid subscription options for the Unified Automation SDK client:

debug	Verbosity level of debugging [default: 0 = off]
priority	Priority of the subscription [0..255; default: 0 = lowest]

`opcuaMapNamespace <session> <index> <uri>`

Map a locally used namespace index to a unique namespace URI on the server.

session	Name of the session to configure
index	Locally used namespace index to map
uri	Server side namespace URI to map to

OPC UA Security Setup

Important: see the separate security related README for details and instructions.

A minimal setup will include creating a self-signed X.509 certificate for the IOC client (and the matching private key) and trusting the certificate that the server presents by adding it to the list of explicitly trusted certificates inside the PKI file store.

`opcuaSetupPKI <PKI location>`

Set up the PKI file store of the IOC client, where certificates and revocation lists are stored.

PKI location	Path to the file based PKI store (certificates and revocation lists in `trusted/certs`, `trusted/crl`, `issuers/certs`, `issuers/crl`)

`opcuaClientCertificate <public key> <private key>`

Set up the OPC UA client certificates to use for the IOC client.

public key	Path to the file containing the certificate (public key)
private key	Path to the file containing the private key

`opcuaSaveRejected <rejected cert location>`

Set the location where the IOC client will save rejected certificates.

rejected cert location	Path where rejected certificates will be saved

It is not recommended to use OPC UA without the Security features. Doing so requires explicit configuration by setting the option `sec-mode=None` for the session.

Database configuration

All OPC UA related records use the setting `DTYP = "OPCUA"`.
Periodic `SCAN` settings will create a "polling" behavior, which is not suggested for OPC UA. Instead, items should be monitored for changes by connecting them to a subscription and setting `SCAN = "I/O Intr"`.

Simple setup: one item = one record, no structures

In addition to `DTYP` and `SCAN`, the link in the `INP/OUT` field must be set.

```
@<name> ns=<namespace>;s=<identifierString> [<option>=<value>…]
@<name> ns=<namespace>;i=<identifierNumber> [<option>=<value>…]
```

name	Name of the subscription or session
namespace	Namespace index number of the NodeID
identifierString	Name (string identifier) of the node
identifierNumber	Numerical identifier of the node

Supported options are:

sampling	Sampling interval in ms [double; def: -1 = use publishing interval]
qsize	Size of server side queue [def: 1 = no queueing]
cqsize	Size of client side queue [def: 1.5 * qsize; minimum 3]
discard	Discard policy on queue overrun [old/new; def: old = drop oldest]
deadband	Deadband filter for subscriptions [double, def: 0.0 = no filter]
register	Register item with the server for better performance [y/n; def: n]
timestamp	Timestamp source [server/source; def: server]
monitor	Set up monitor (output record gets bidirectional) [y/n; def: y]
bini	Behavior at initialization [read/ignore/write; def: read]

Full setup: one itemRecord, multiple connected data records

For each item, an `itemRecord` instance is created. Its `INP` link follows the definition from the simple setup shown above.
One additional option is supported:

timestamp	Set the name of the top level data element that the timestamp is to be taken taken from, e.g. `timestamp=@myTS`

All connected data records define their `INP`/`OUT` links to point to the itemRecord.

`@<item> [<option>=<value>…]`

item	Name of the itemRecord this data record is connected to

Supported options are:

element	Path of the data record's element inside the item's structure [“.” = hierarchy separator; “” = root element in case of no structure]
timestamp	Timestamp source [server/source/data; def: server]
monitor	Set up monitor (output record gets bidirectional) [y/n; def: y]

In all string values (identifierString, element), double quotes can be escaped by preceding them with one backslash and separators can be escaped by preceding them with two backslashes.

Example

The Example is taken from a test setup connecting to a Siemens S7-1500 series PLC.

Startup Script

The following setup in the startup script:

```
opcuaSession PLC opc.tcp://192.168.1.145:4840
opcuaOptions PLC nodes-max=1000 sec-mode=None
opcuaSubscription FAST PLC 100
opcuaSubscription SLOW PLC 500
```

creates the session, sets a limit of 1000 nodes per batch for low-level service calls (the Siemens S7 server has that limitation), disables OPC UA Security and creates two subscriptions on that session with different publishing periods.

Database Records

Records working with this setup might look like this.

```
record(longin, "OPC:li1") {
    field(DTYP, "OPCUA")
    field(INP,  "@SLOW ns=3;s=\"DB_1\".\"myInt\" sampling=100 qsize=5")
    field(TSE,  "-2")
    field(SCAN, "I/O Intr")
}
record(ao, "OPC:ao1") {
    field(DTYP, "OPCUA")
    field(OUT,  "@PLC ns=3;s=\"DB_1\".\"myDouble\" monitor=n")
}
record(ao, "OPC:ao2") {
    field(DTYP, "OPCUA")
    field(OUT,  "@SLOW ns=3;s=\"DB_1\".\"myDouble\"")
}

record(opcuaItem, "OPC:item1") {
    field(INP, "@FAST ns=3;s=\"DB_1\".\"myStruct\"")
    field(SCAN, "I/O Intr")
}
record(longin, "OPC:sli1") {
    field(DTYP, "OPCUA")
    field(INP,  "@OPC:item1 element=Int1")
    field(TSE,  "-2")
    field(SCAN, "I/O Intr")
}
record(ai, "OPC:sai1") {
    field(DTYP, "OPCUA")
    field(INP,  "@OPC:item1 element=sub2.sub21.Double1")
    field(SCAN, "I/O Intr")
}
```

The first three records use the simple setup. While `OPC:ao1` is bound to the session and is only writing, `OPC:ao2` is monitored, i.e. bound to a subscription and being updated if the value changes on the PLC. `OPC:li1` is sampling the value from the PLC with a higher frequency (every 100ms) and sets up a server side queue to not lose any updates.

The lower three records show a full setup. The itemRecord `OPC:item1` is connecting to a structured item, and the two data records are connected the itemRecord, pointing to elements of its structure at different hierarchy levels.
The unusual node identifiers in the example (containing quote characters that have to be escaped) is the naming style used by TIA Portal for an S7-1500 series PLC.

The itemRecord is bidirectional. Setting its field `DEFACTN` to `read` or `write` selects which operation the record initiates when triggered through forward link processing or by writing to its `PROC` field. Writing to its `READ` or `WRITE` fields will explicitly force a read or write operation.