

# EPICS Support for OPC UA – Cheat Sheet

*This document version applies to release 0.6.*

## Concepts

A *Session* is a named connection between client and server (*0..n per IOC*).

Any piece of data on the server is 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

```
opcuaCreateSession <name> <server URL> [<debug> [<autoconnect>]]
```

Create a new session.

|             |  |
|-------------|--|
| name        | Name of the session to create                      |
| server URL  | URL of the OPC UA server to connect to             |
| debug       | Initial level of debugging [def: 0 = off]          |
| autoconnect | Automatically connect/reconnect to server [def: y] |

```
opcuaSetOption <session> <option name> <value>
```

Set session specific options.

|             |                                  |
|-------------|----------------------------------|
| session     | Name of the session to configure |
| option name | Name of the option to configure  |
| value       | Value for the specified option   |

Currently supported options for the Unified Automation SDK client:

|             |  |
|-------------|--|
| batch-nodes | Maximum number of nodes per low-level service call |
|-------------|--|

```
opcuaCreateSubscription <name> <session> <interval> [<priority> [<debug>]]
```

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] |
| priority | Priority of the subscription [0..255; def: 0 = lowest]  |
| debug    | Initial level of debugging [def: 0 = off]               |

## 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 fields 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]   |
| 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]     |

## 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.

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<br>["." = hierarchy separator; "" = root element in case of no structure] |
| timestamp | Timestamp source [server/source; 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:

```
opcuaCreateSession PLC opc.tcp://192.168.1.145:4840
opcuaCreateSubscription FAST PLC 100
opcuaCreateSubscription SLOW PLC 500
```

creates the session and two subscriptions 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.