1. General

The iQunet UNIX Server runs an embedded OPC UA (unified architecture) Server. OPC UA extends the standard, and highly successful, OPC communication protocol. It enables data acquisition and allows that data to be modelled and communicated between the plant floor and the enterprise reliably and securely. OPC UA is future-ready, easy to configure, and maintain.

2. How to access the OPC UA Server?

2.1. OPC UA Server address

All recorded data can be extracted via the built-in OPC UA Server. The OPC UA Server always listens on port 4840 regardless if the connection is made via cable, hotspot or WiFi.

If you use the hotspot connection, this will be 192.168.42.1:4840.

If you use another connection type, you need to use the IP address of the iQunet Unix server (xxx.xxx.xx.xx:4840). The server’s IP address is handed out by your DHCP server. The recommended way of operation is to setup your DHCP server to provide a static lease. The current IP address of the server in the network can easily been found by clicking the 3 bars below the iQunet logo in the sensor dashboard. Select “Ethernet-802.3” in the left pane and the IP address will appear.

It is also possible to set a static IP address in the “Ethernet-802.3” panel. The server will then start a virtual network interface and will operate from 2 simultaneous IP addresses (the static IP address and the regular DHCP lease). You can then either use the static IP address or the DHCP address for the connection to the OPC UA Server.

2.2 OPC UA Server monitoring

After a power restart of the iQunet system it will take a few minutes to allow the server to startup and populate the address space from the database. Thereafter an internal process monitor becomes active which surveys the internal daemons every few seconds. When a problem is detected, a cleanup of the affected processes is performed. This cleanup is reported to the supervisor which can then restart the faulty component. The OPC UA Server is one of the 11 monitored subcomponents.

The supervisor has a local frontend running on port 9001 (http://xxx.xxx.xx.xx:9001 where xxx.xxx.xx.xx is the server’s IP address). Username and password are ‘admin’. Please note that only the highest-level master processes are reported here.

2.3 OPC UA variable nodes addressing

There are 3 ways to access variables in OPC:

- via string ID’s,
- via numeric node-ID’s
- and via browsing.

All 3 options will be explained in further detail below. Which variable nodes are available depends on the type of sensor since they are added dynamically to the address space. Due to the collision of variable names when multiple sensors are present in the OPC UA server, we only support browsing and numeric node-ID’s.

String ID’s

The (deprecated) OPC DA way to access variables is through string ID’s, where the variable path is encoded into a string with dot separators like for example “ns=2;s=my_sensor.tree.encoded.variablename”. We don’t support this old style of accessing any more.
**Numeric node-ID’s**

An alternative way is direct access via the UINT32 numeric node identifier like for example “ns=2;i=’12345678’”. In practice only 31 bits are used to prevent overflow errors in some client software systems. The exact representation depends on your client software. The client extracts the namespace and 32-bit ID and sends this to the OPC server.

**Browsing**

The preferred way of accessing variables is via browsing. iQunet’s OPC UA server supports browsing which means that your client software can browse the address space as follows:

\[
\text{[get node 'root' (ns=0)]} \rightarrow \text{[Objects (ns=0)]} \rightarrow \text{[get sensor children with ns=2]} \rightarrow \text{[get Sensor 'ab:cd:de:ef']} \rightarrow \text{[get Variable 'mmsRmsX']}
\]

For example, in UA Expert browsing looks like this:

![Address Space](image)

The server will send back an ua-node struct object containing the numeric node-ID (as described above). Your client can cache and use this numeric node-ID for fast direct access in all future requests in the same session.

In the figure below, you can see that our server supports node identifiers of the type “Numeric”.

The node below can thus be accessed as follows:

- via browsing: Root->Objects->'15:1a:62:7c'->'mmsRmsX'
- or via direct access: ns=2;i=1113085726.

![Attributes](image)

The OPC UA standard requires this numeric node-ID to remain static only for 1 single server session. The client software must then re-request the node-ID mapping for each session. Our server will however remap the same ID’s to the same endpoints to support client software with incorrect caching.
2.4 iQunet OPC UA variable nodes

The OPC UA variable nodes used by iQunet are described below per sensor type. For comparison it is described and shown on figures what these nodes represent on the iQunet sensor dashboard.

General sensor nodes

- **batteryVoltage**: indication of the battery power level ("Power" value in the System Information pane)
- **beatperiod**: wakeup interval of the sensor ("WakeUp Interval" setting in the Network Interface pane). The wakeup interval is an internal parameter of the iQunet sensors and indicates the maximum time interval at which the sensor must exit sleep mode and contact the base station to collect its scheduled tasks. If no task has been scheduled for this sensor or if the base station could not be contacted (for example because the base station is out of range), the sensor will return to sleep mode for the wakeup interval period. The purpose of the wakeup interval is to extend the battery life time.
- **boardTemperature**: temperature measured with the build in temperature sensors ("Temperature" value in the System Information pane)
- **firmware**: current firmware version of the sensor ("Firmware" version in the System Information pane)
- **googlesheets**: internally used identifier string
- **hardware**: hardware version ("Hardware" version in the System Information pane)
- **lastseen**: last moment in time a sensor was seen live for the iQunet server in the local sensor network ("Last Seen" value in the Network Interface pane)
- **rssi**: current wireless signal strength between the device and the base station ("Signal Strength" value in the Network Interface pane)

---

**SENSOR STATUS**

<table>
<thead>
<tr>
<th>Network Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rssi</strong> Signal Strength: <strong>-67 dBm</strong></td>
</tr>
<tr>
<td>MAC Address: <strong>68:90:43:13</strong></td>
</tr>
<tr>
<td>PAN Address: <strong>192.168.1.150</strong></td>
</tr>
<tr>
<td>WakeUp Interval: <strong>45 sec</strong></td>
</tr>
<tr>
<td><strong>lastseen</strong> Last Seen: Thu Feb 14 2019 15:18:47 GMT+0100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>firmware</strong> Firmware: <strong>B36B79C1</strong></td>
</tr>
<tr>
<td><strong>hardware</strong> Hardware: <strong>SERN-322-9943</strong></td>
</tr>
<tr>
<td>Temperature: <strong>25.7 °C</strong></td>
</tr>
<tr>
<td>Power: <strong>2.92V [80%]</strong></td>
</tr>
</tbody>
</table>

---

iQunet.
queueEnabled: are automatic measurements enabled or not (slider in the Auto Measurements pane)

queueInterval: time between two automatic measurements in seconds (“Queue Interval” value in the Auto Measurements pane)

Hall sensor nodes

hField: measured magnetic field (first box of the “Hall” values in the Hall Sensor Control pane)

hVoltage: measuring voltage used for calibration purposes (second box of the “Hall” values in the Hall Sensor Control pane)

Tilt sensor nodes

activityThreshold: required activity level to wake up the sensor (“Activity Level” setting in the Tilt Sensor Control pane)

burstSamples: number of samples in 1 burst measurement (“Burst Samples” value in the Tilt Sensor Control pane)

guardRoll: maximum roll value allowed before an alarm is initiated (“Guard Roll” setting in the Tilt Sensor Control pane)

pitch: pitch value of the inclination (first box of the “Position” values in the Tilt Sensor Control pane)

roll: roll value of the inclination (second box of the “Position” values in the Tilt Sensor Control pane)

yaw: yaw value of the inclination (not used)

Reed sensor nodes

ReedCount: the counted number of magnetic pulses (“Count” value in the Reed Sensor Control pane)
ReedRPM: revolutions per minute based on the magnetic pulses count ("RPM" value in the Reed Sensor Control pane)

Vibration sensor nodes

accelerationPack: the last recorded raw vibration data

The accelerationPack format is as follows:
1/ numSamples: n = #samples
2/ accelArray: rawSample[0:n-1]
3/ sampleRate: e.g. 400 = 400Hz
4/ formatRange: e.g. 4 = +/-4g (hardware setting of the accelerometer IC)
5/ offset: unused, 0 (hardware offset of the accelerometer IC)
6/ encoded_axis: X = 0, Y = 1, Z = 2
7/ prescaler: unused (only used when no compression in debug mode)
8/ compression: unused (0 = no compression in debug mode, 1 = compression)

You will see that the first 7 samples of the accelArray (at the start of each measurement) show a transient response due to the start-up behavior of the compression algorithm. Since a Hanning window is used for the calculation of the DFT and RMS, this behavior will be automatically suppressed and has thus no further impact.

The conversion of the accelArray to g units is as follows:
Conversion of rawSample[0:n-1] to [g]:

\[
gSample = \frac{\text{rawSample}[0:n-1]}{512.0} \times \text{formatRange} \ [g]
gTimes = \frac{[0:n-1]}{\text{sampleRate}} \ [\text{sec}]
\]

axis: measurement axis ("Axis" setting in the MEMS Vibration Setup pane)

formatRange: dynamic range of the sensor ("Limit" setting in the MEMS Vibration Setup pane)

gKurtX: last calculated kurtosis value on X axis in g units (see Statistics Lab)
gKurtY: last calculated kurtosis value on Y axis in g units (see Statistics Lab)
gKurtZ: last calculated kurtosis value on Z axis in g units (see Statistics Lab)
gRmsX: last calculated rms value on X axis in g units (see Statistics Lab)
gRmsY: last calculated rms value on Y axis in g units (see Statistics Lab)
gRmsZ: last calculated rms value on Z axis in g units (see Statistics Lab)
mmsKurtX: last calculated kurtosis value on X axis in mm/s units (see Statistics Lab)
mmsKurtY: last calculated kurtosis value on Y axis in mm/s units (see Statistics Lab)
mmsKurtZ: last calculated kurtosis value on Z axis in mm/s units (see Statistics Lab)
mmsRmsX: last calculated rms value on X axis in mm/s units (see Statistics Lab)

mmsRmsY: last calculated rms value on Y axis in mm/s units (see Statistics Lab)

mmsRmsZ: last calculated rms value on Z axis in mm/s units (see Statistics Lab)

numSamples: number of measurement samples ("Samples" setting in the MEMS Vibration Setup pane)

defetch: number of prefetch samples ("Prefetch" setting in the Vibration Download pane)

sampleRate: sampling rate of the sensor ("Rate" setting in the MEMS Vibration Setup pane)

threshold: threshold for full vibration download ("Threshold" setting in the Vibration Download pane)

vibration: this is a custom OPC UA data container structure containing the last recorded vibration data on the X, Y, and Z axis. The data structure contains the acceleration (accel) and velocity (veloc) data in the time (xAccelTime, xVelocTime) and frequency domain (xAccelFreq, xVelocFreq) for each axis (X, Y, and Z).

vibrationStat: selected vibration statistic in the Statistics Lab ("Statistic" value in the Statistics Lab under the Units tab)

vibrationavg: number of averages used in the DFT averaging ("DFT Averaging" setting in the Vibration Lab under the "1X" averaging tab)

vibrationdetrend: 1/f flicker noise detrending activated or not ("1/f detrend" slider in the Vibration Lab under the "6Hz" High Pass Filter tab)

vibrationhpf: high pass cut off frequency ("Highpass" setting in the Vibration Download pane)

vrange_accel: viewport range in g units in the Vibration Lab ("Viewport" setting in the Vibration Lab under the Units tab)

vrange_veloc: viewport range in mm/s units in the Vibration Lab ("Viewport" setting in the Vibration Lab under the Units tab)
vunits: selected display units for the vibration lab or for the vibration statistics in the Statistics Lab ("Units" value in the Vibration Lab or in the Statistics Lab under the Units tab)
2.5 Example using UA Expert

To extract data via OPC, you can use UAExpert for example. Open UA Expert and click on Server → Add.

Double click on "Double click to Add Server" and fill out the IP address behind opc.tcp://. Click OK.
Select the added server in the server list. All sensors connected to this server will appear in the Address Space.

Click on the macId of the sensor to see all possible attributes of the sensor.
Add a document to inspect for example the board temperature data (Document → Add). Select ‘History Trend View’ as the document type and click ‘Add’.

Drag the boardTemperature attribute of the sensor to the configuration window.

Temperature read-out is possible via either a single update that extracts all data values in between two points of time at once or via a cyclic update that extracts all data over the set timespan every set time interval (update interval).

The accelerationPack attribute contains the raw vibration data. The accelerationPack format is as follows:

1/ numSamples: n = #samples
2/ accelArray: rawSample[0:n-1]
3/ sampleRate: e.g. 400 = 400Hz
4/ formatRange: e.g. 4 = +/-4g (hardware setting of the accelerometer IC)
5/ offset: unused, 0 (hardware offset of the accelerometer IC)
6/ encoded_axis: X = 0, Y = 1, Z = 2
7/ prescaler: unused (only used when no compression in debug mode)
You will see that the first 7 samples of the accelArray (at the start of each measurement) show a transient response due to the start-up behavior of the compression algorithm. Since a Hanning window is used for the calculation of the DFT and RMS, this behavior will be automatically suppressed and has thus no further impact.

The conversion of the accelArray to g units is as follows:

Conversion of rawSample[0:n-1] to [g]:

\[
gSample = \frac{\text{rawSample}[0:n-1]}{512.0\cdot\text{formatRange [g]}} \\
gTimes = \frac{[0:n-1]}{\text{sampleRate [sec]}}
\]

### 2.6 Example using TIA Portal V15 WinCC RT Advanced

Connect to the iQunet server. In this example, the connection is set-up via VPN with IP address 10.50.29.1. Note that other connection methods can be used as well. See the support page on the iQunet website (FAQ) for more information.

Open TIA Portal V15 and select “Create new project”.
Select “Motion & technology” and click on the button on the right to add a new device.

Select PC systems → SIMATIC HMI application → WinCC RT Advanced and click OK.
Open the project view.

Select PC-System_1 → HMI_RT_1 → Connections.

Select “Add new” and add a new connection. Select “OPC UA” as the communication driver.
Fill out the UA server discovery URL and the security settings. 
url = opc.tcp:// 10.50.29.1:4840 
policy = Basic128Rsa15 
mode = Sign & encrypt

Accept the server certificates. Move the iQunet server certificate from the ‘rejected’ folder to the ‘certs’ folder. The procedure is explained in the following link: 

Move the certificate from C:\ProgramData\Siemens\CoRtHmiRTm\OPC\PKI\CA\default\rejected

to

C:\ProgramData\Siemens\CoRtHmiRTm\OPC\PKI\CA\default\certs.
Set-up the HMI tags. Select “Default tag table [1]” under HMI tags.

Add a new tag → Connection iQunet_X (as set above).
Choose the Address. Browse to the Objects node and select the sensor’s macId.

Select the sensor attribute or tag you want to observe.
Add a new screen using the Screens node.

Add a new I/O widget from the Elements groupbox to the layout pane.
Select the HMI tag.

Compile and run.

Check if the OPC UA connection works.
Remark: please note that there is a certificate bug in older TIA WinCC Advanced Version V15 versions (and OPC Scout). You need service pack 1 for this to be fixed in TIA WinCC.

2.7 Python/Matlab

For OPC UA communication in Python the OPCUA library can be used (https://python-opcua.readthedocs.io/en/latest/index.html). You can find 3 example Python scripts on our Github page (https://github.com/iqunet/sern).

Matlab also offers an extension or toolbox to read data directly from OPC UA (https://nl.mathworks.com/products/opc.html).