OPC UA Client

User Manual

Version 2.3.3

11.9.2017

Prosys OPC UA Client – User Manual

Version: 2.3.3

Contents

1. OPC UA Client Overview

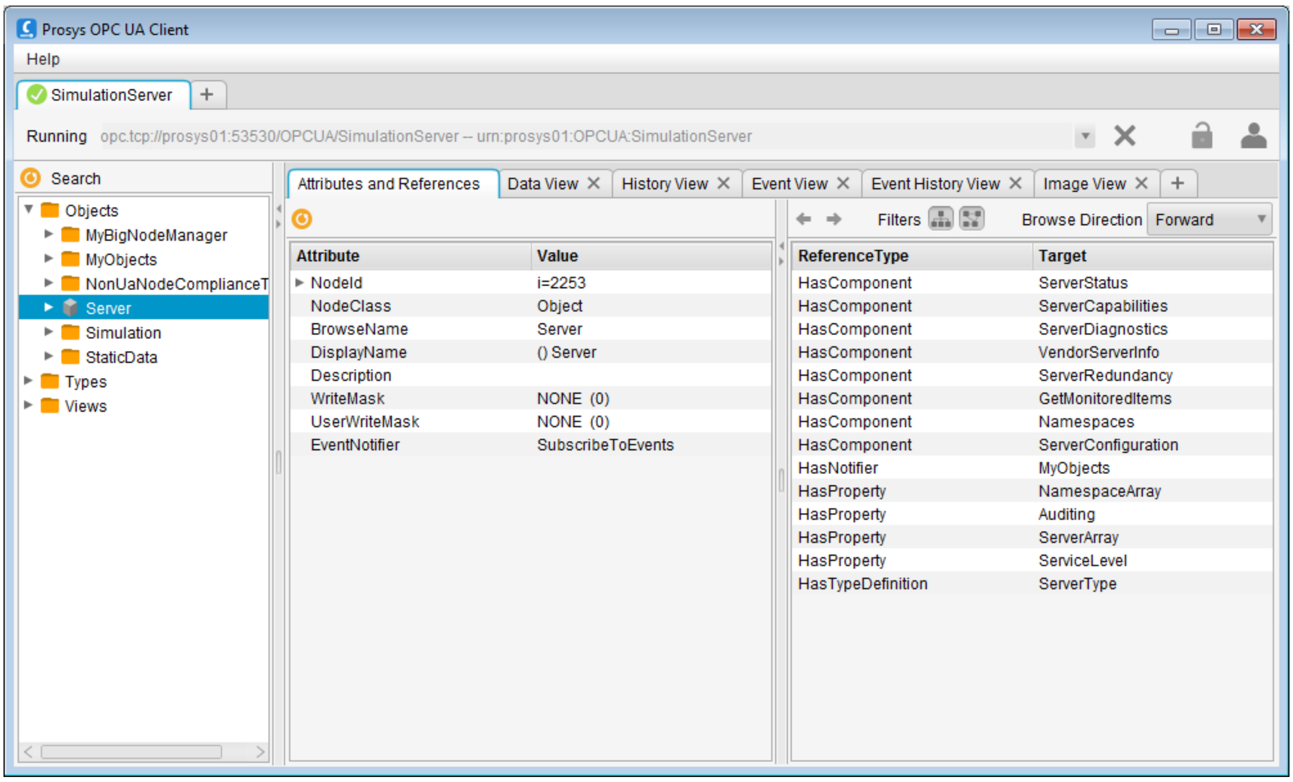
Prosys OPC UA Client is a generic purpose OPC UA client. You can test connections and view data from any OPC UA server. The main functions include browsing the server address space, reading and writing variables, monitoring variables and events - and reading history of variables and events.

Prosys OPC UA Client implements the three OPC UA information models:

1. Data Access
2. Historical Access
3. Alarms & Conditions

With the OPC UA Client you can have multiple secure server connections and manage them using tabbed pages. The address space of each server is represented in the *Address Space Browser* on the left side of the user interface window. For each server, you can also use several tabbed pages for different data visualizations.

Attributes and References is a default view for every server and in addition, you can add any number of the other views: *Data View* for monitoring variables, *History View* to explore historical data of variables, *Event View* to monitor events and alarms and *Event History View* for requesting history of events from the server. *Image View* provides a special option for monitoring of image data. Figure 1 illustrates the overview of the OPC UA Client.



Running opc.tcp://prosys01:53530/OPCUA/SimulationServer -- urn:prosys01:OPCUA:SimulationServer

Browse Direction Forward

Attribute	Value
NodeId	i=2253
NodeClass	Object
BrowseName	Server
DisplayName	() Server
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
EventNotifier	SubscribeToEvents

ReferenceType	Target
HasComponent	ServerStatus
HasComponent	ServerCapabilities
HasComponent	ServerDiagnostics
HasComponent	VendorServerInfo
HasComponent	ServerRedundancy
HasComponent	GetMonitoredItems
HasComponent	Namespaces
HasComponent	ServerConfiguration
HasNotifier	MyObjects
HasProperty	NamespaceArray
HasProperty	Auditing
HasProperty	ServerArray
HasProperty	ServiceLevel
HasTypeDefinition	ServerType

Figure 1 OPC UA Client Overview

2. OPC UA Servers

OPC UA Servers are connected with a server address (aka Server URL). To connect, you will need to find out the address of the server that you wish to connect to.

The address always takes the form:

<Protocol>://<Hostname>:<Port>/<ServerName>

The address starts with a Protocol identifier, which is most often “opc.tcp://” (corresponding to OPC UA TCP). The protocol identifier is followed by the hostname of the computer in which the server is running (‘localhost’ always works locally as well) and the TCP port which the server is listening. OPC UA defines a standard port number 4840, but this is often reserved for a Discovery Server and the actual servers use custom port numbers. The server address may also contain a Server-Name part. If the ServerName is not defined, the ‘/’ character before it may be omitted.

If you don’t have any actual OPC UA servers available to connect to, you can use the Prosys OPC UA Simulation Server to play around with the Client application

You can download Prosys OPC UA Simulation Server from *www.prosysopc.com* and run it locally. You can then use the server address:

opc.tcp://localhost:53530/OPCUA/SimulationServer

Alternatively, you can connect to a publicly available server via Internet. The address of that server is:

opc.tcp://uademo.prosysopc.com:53530/OPCUA/SimulationServer

2.1 Discovery Servers

OPC UA defines two types of Discovery Servers:

- Local Discovery Server (LDS) is often installed in the PC to keep a list of installed servers in the local computer. OPC Foundation provides a standard implementation of the LDS.
- Global Discovery Server (GDS) is an application that is designed to keep a list of servers installed in a site-wide network. The GDS can also manage application instance certificates as a central Certificate Authority (CA). OPC Foundation provides a sample implementation of the GDS.

In addition, each OPC UA server contains an internal discovery server that can provide similar information about the server application itself. The LDS is typically listening at TCP Port 4840 (having address *opc.tcp://localhost:4840*). Several embedded devices and PLCs, which do not have any LDS are also using Port 4840 for the actual server address.

If you have a GDS available, you will need to find its address, similar to other OPC UA server applications.

3. Connecting to a Server

Connecting to an OPC UA server is based on the server address (or Server URL) that can be typed in the address bar at the top of the window (see Figure 2). The previously used addresses are available from the dropdown menu so that it is easy to reselect them. On the right hand side of the address bar, there is a button for connecting and disconnecting with the server. If the client cannot connect to the server, it shows an error message indicating the reason for that.

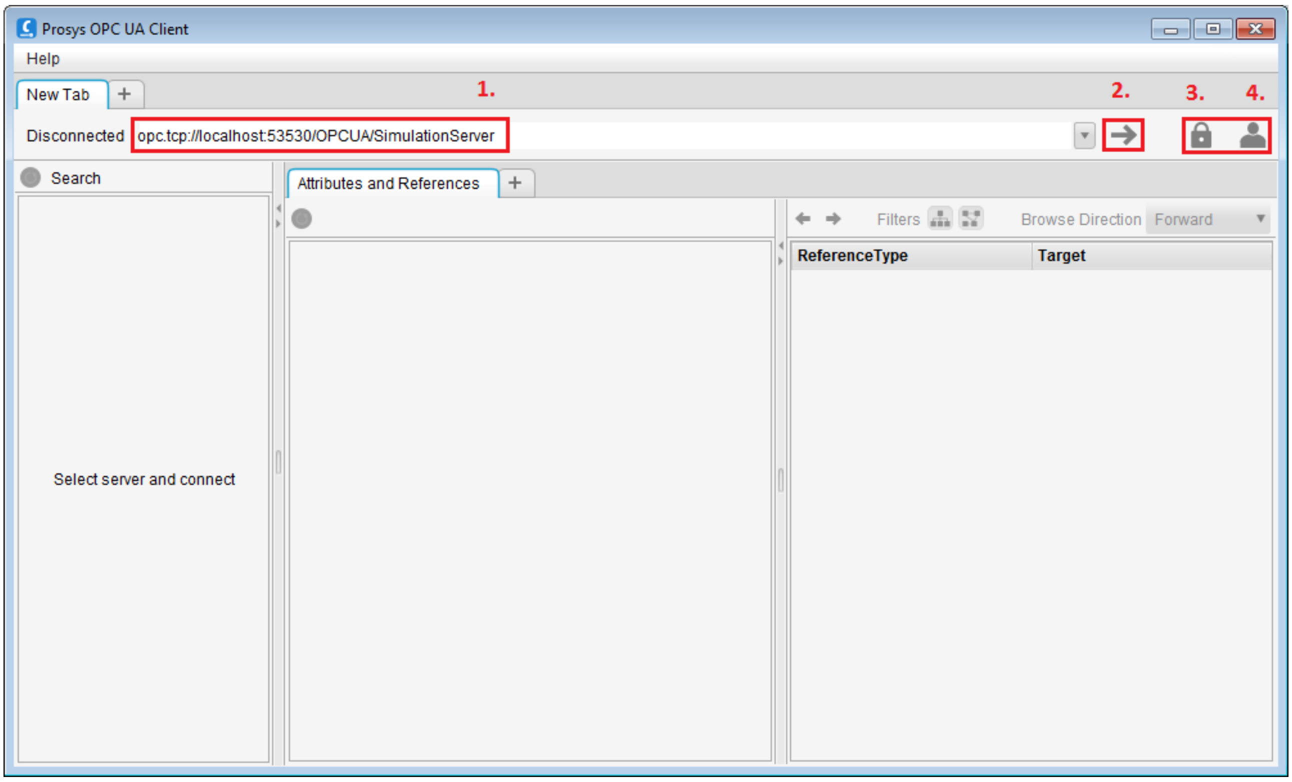


Figure 2 Server Address Bar (1.) Connect button (2.) Security Settings (3.) and User Authentication Settings (4.)

In case the client manages to connect, but determines that the target is an OPC UA Discovery Server (see 2.1, above), it will prompt for a selection from the available server addresses provided by the discovery server. You can then choose one of the actual server address to make a new connection attempt.

Prosys OPC UA Client accepts partial addresses to be entered as well: if you only enter the host-name, it will assume ‘opc.tcp’ as the protocol and port number 4840.

3.1 Security Options

Whenever you make a connection to the server, you must choose the level of security to use. The security options include Security Mode and Security Policy. The alternative Security Modes are:

1. None
2. Sign
3. Sign & Encrypt

If the selected Security Mode is *None*, the connection is not secured at all. If *Sign* is selected, the messages used in the communication are signed to protect them from alteration during transfer. Choosing Sign & Encrypt gives the highest level of security, protecting the contents of the communication to be seen by any third parties.

Security policy determines the algorithm for signing and encrypting. There are multiple security policies available as defined by the OPC UA Specification v1.02 (see Figure 3), each one of them defining a set of security algorithms and parameters to use for the OPC UA TCP communication:

1. None
2. Basic128RSA15
3. Basic256
4. Basic256SHA256

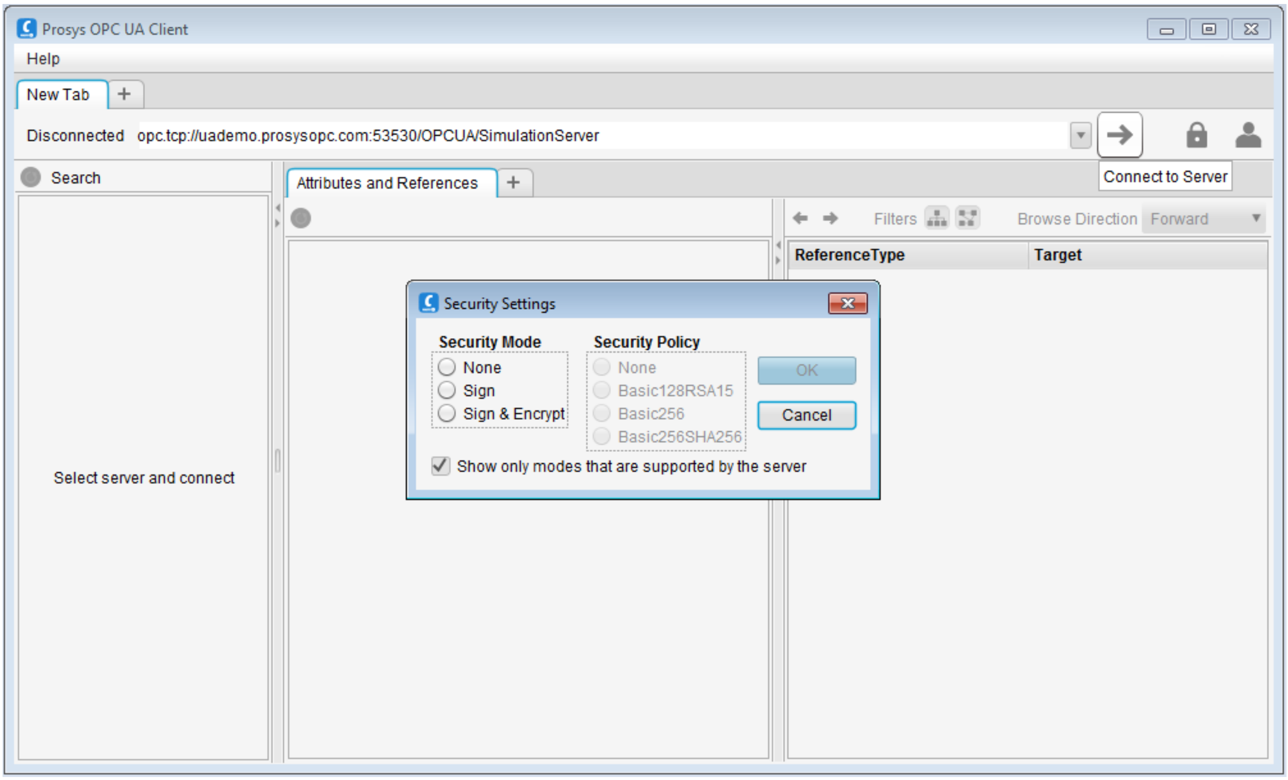


Figure 3 Security Settings

In order to be able to connect, the selected mode and policy must match the ones supported by the server application. To help with finding a usable mode, you can use the *Show only modes that are supported by the server* option.

Also, if you wish to use a secure connection (Sign or Sign&Encrypt), the client and server application must trust to each other. In order to build the trust ship, the applications use Application Instance Certificates.

3.2 Application Instance Certificates

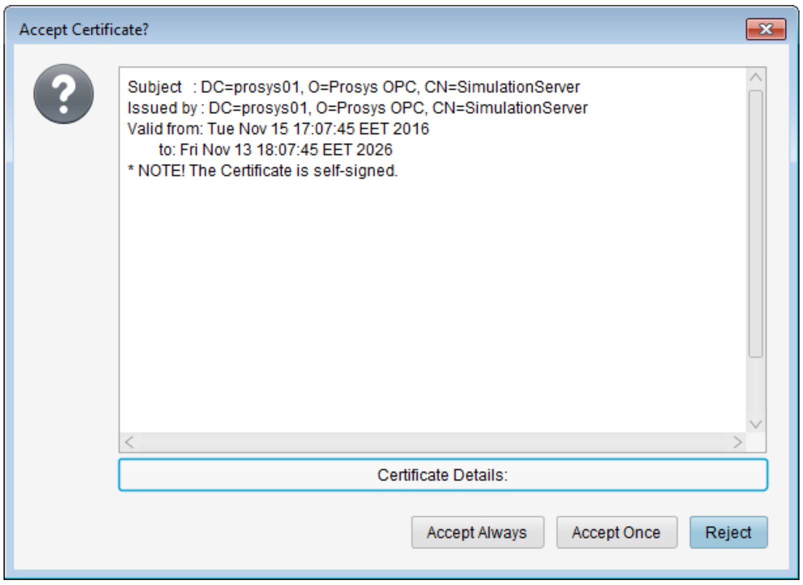
OPC UA applications are identified using Application Instance Certificates. All applications that wish to communicate securely will need to have a certificate. The certificates enable identifying the applications reliably. The certificates are based on Public Key Infrastucture (PKI) principles as defined by the X.509 standard.

In order to build the trust relationship between each other, the applications use a Certificate Store where they keep the certificates of other known applications and where they can define which certificates (and applications) are trusted.

By default, all OPC UA applications can create the certificate for themselves. In this case, we talk about *Self-signed certificates*.

In case of self-signed certificates, each application must trust to each other separately. Therefore, by default when you try to connect securely to a new server for the first time, you will get a *Bad_SecurityChecksFailed* error, indicating that the server does not trust your client application. You must then go to the server configuration, locate the certificate of the client application and define that it can be trusted.

After the server trusts the client certificate, you still need to make the client trust the server: the client will prompt you, accordingly:



```
Subject   : DC=prosys01, O=Prosys OPC, CN=SimulationServer
Issued by : DC=prosys01, O=Prosys OPC, CN=SimulationServer
Valid from: Tue Nov 15 17:07:45 EET 2016
        to: Fri Nov 13 18:07:45 EET 2026
* NOTE! The Certificate is self-signed.
```

Figure 4 Certificate verification dialog in the client application

If you select *Accept Always*, the certificate will be moved to the list of trusted certificates in the client. From now on, this client and server trust each other and can open secure communication with each other, as long as the certificates are valid.

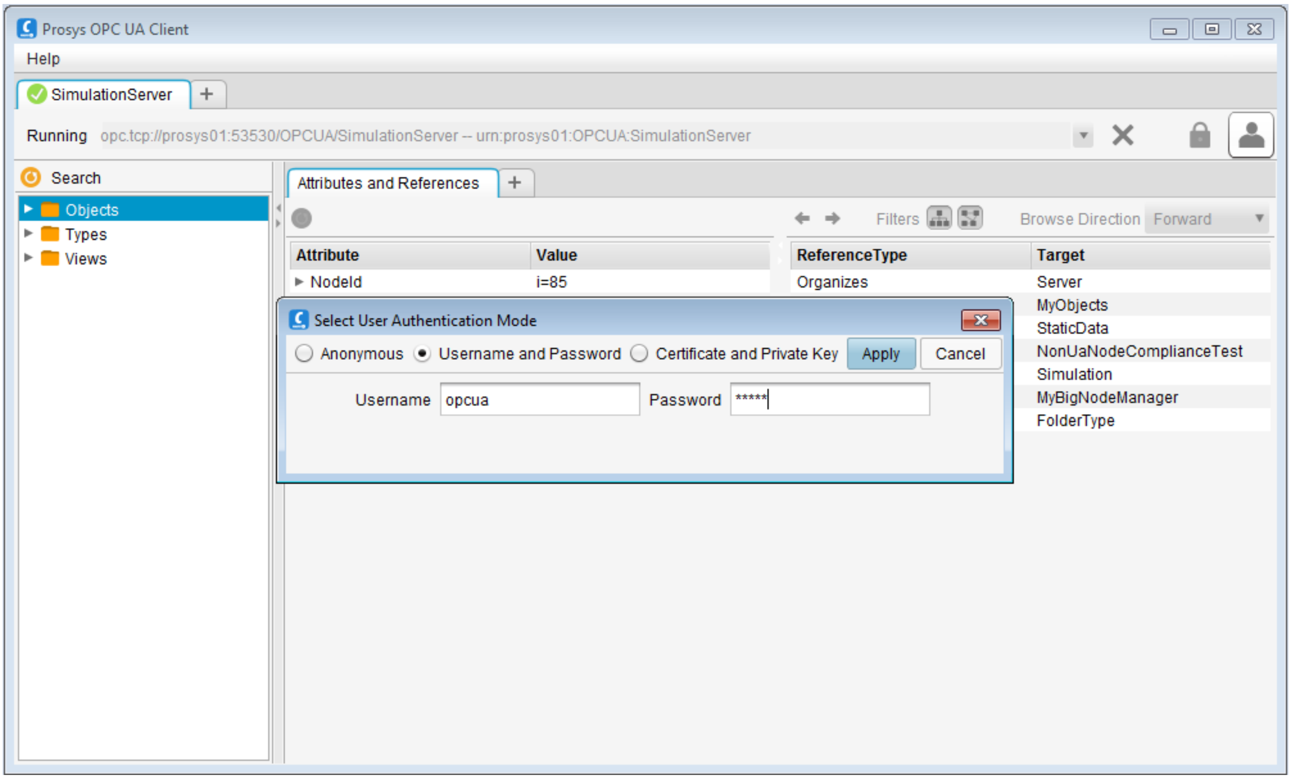
The Certificate contains information about the application, but also a cryptographic Public Key. It is accompanied with a separate Private Key. These two keys enable *asymmetric encryption*, which is the basis of the OPC UA security mechanism. In order to keep things secure, the private key must be kept secret from other people and applications. It should be available only for the application itself.

3.3 User Authentication

In addition to application level authentication, OPC UA also supports user authentication. Users can be identified with standard Username & Password combination or X.509 certificates (similar to the applications) or by some external user authentication system. OPC UA also enables anonymous connections without any user identification. Prosys OPC UA Client currently supports all but the external authentication systems.

The user identity may be defined before establishing the connection or while the connection is open. In the latter case, the existing session will be *impersonated* to a new user without closing the connection.

You can select the user authentication setting by clicking the tool button in the upper right corner of the user interface (see Figure 2 & Figure 5). It can be applied also when the connection is established to impersonate the session to a new user account.



Running opc.tcp://prosys01:53530/OPCUA/SimulationServer -- urn:prosys01:OPCUA:SimulationServer

Browse Direction: Forward

Attribute	Value
NodeId	i=85

ReferenceType	Target
Organizes	Server
	MyObjects
	StaticData
	NonUaNodeComplianceTest
	Simulation
	MyBigNodeManager
	FolderType

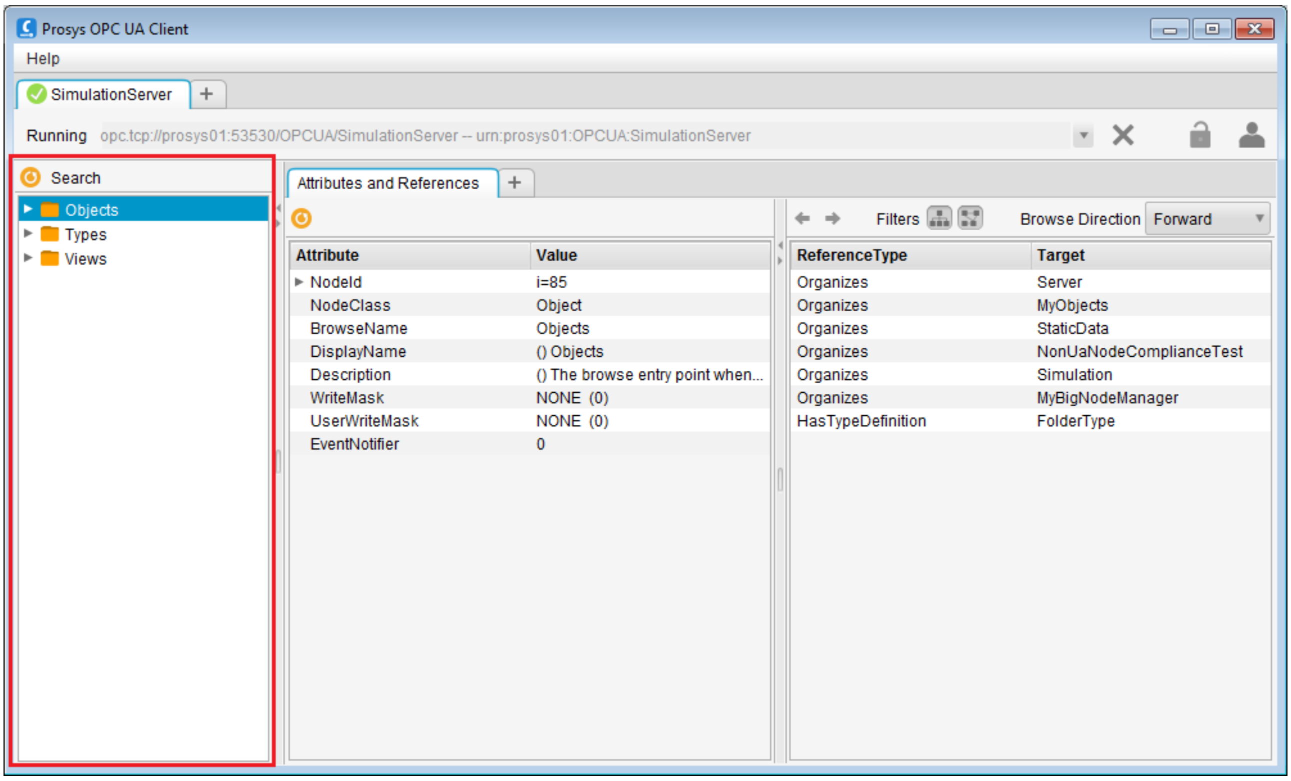
Anonymous [✓ Username and Password] Certificate and Private Key

Username: opcua
Password: *****

Figure 5 Defining the user authentication mode and user account. You can change the user account before or during a connection. Prosys OPC UA Simulation Server enables you to define your own usernames and passwords to test with.

4. Address Space Browser

Once you are connected to a server, you will be able to browse the *Address Space* of the server, to locate the data and information that you are interested in. You can do this with the Address Space Browser, which is on the left hand side of the application window. The address space always contains three main folders (as defined by the OPC UA Specification): Objects, Types and Views.


Prosys OPC UA Client

Help

SimulationServer

Running opc.tcp://prosys01:53530/OPCUA/SimulationServer -- urn:prosys01:OPCUA:SimulationServer

Search

- Objects
- Types
- Views

Attributes and References

Attribute	Value
NodeId	i=85
NodeClass	Object
BrowseName	Objects
DisplayName	() Objects
Description	() The browse entry point when...
WriteMask	NONE (0)
UserWriteMask	NONE (0)
EventNotifier	0

Filters Browse Direction Forward

ReferenceType	Target
Organizes	Server
Organizes	MyObjects
Organizes	StaticData
Organizes	NonUaNodeComplianceTest
Organizes	Simulation
Organizes	MyBigNodeManager
HasTypeDefinition	FolderType

Figure 6 The Address Space Browser is used to locate instances (in Objects folder) and types (in Types folder) from the server. Views folder is used only very seldom.

The *Objects* folder contains all data of the server: objects and variables. We also call these instances.

The *Types* folder contains all meta data of the server: object types, variable types, data types, event types and reference types. The meta data helps to understand what the instances are; the semantics of the information model that the server uses to group the data.

The *Views* folder is meant to be used to define alternate layouts to the servers information model, but it is not used very often.

4.1 Nodes

All items in the address space are called nodes: *Objects*, *Variables*, types, etc. The nodes have a number of *Attributes* that define specific details of each. For example, Variables have a *Value* attribute.

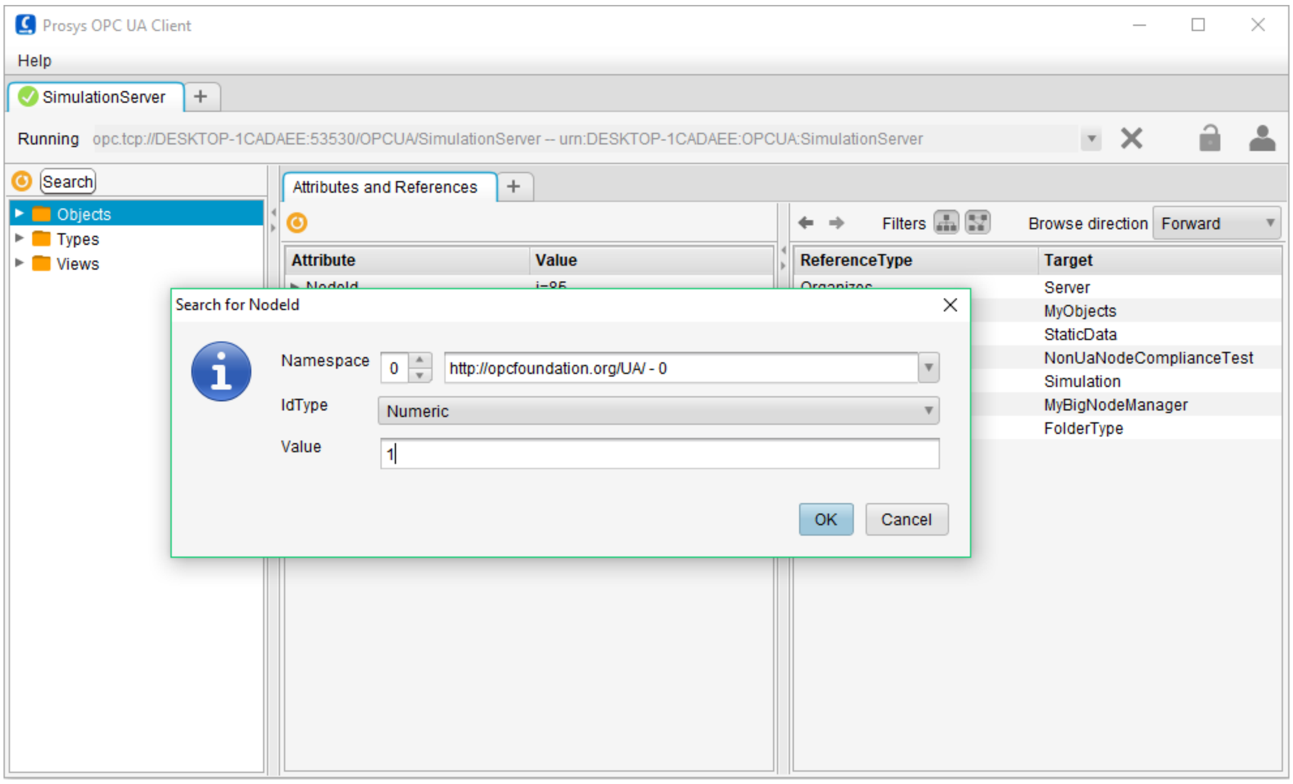
The main attribute of each node is the *NodeId*, which is a unique identifier that is used to identify the node in the server.

In addition to attributes, the nodes also have references. The references are used to define relations between the nodes: hierarchical references enable the address space to be viewed as a tree and additional non-hierarchical references can define other relations, such as the type of an object.

See chapter 5. Attributes and References View for more about the Attributes and References View.

4.2 Searching for a Node

At the top of the Address Space Browser, you will find a *Search* button, which helps you to locate a node with a particular *NodeId*. In the search dialog (see Figure 7), you can specify the *Namespace*, *IdType* and *Value* of the *NodeId*. If the respective node exists, it is activated in the Address Space Browser.


Search for NodeId

Namespace 0 http://opcfoundation.org/UA/ - 0

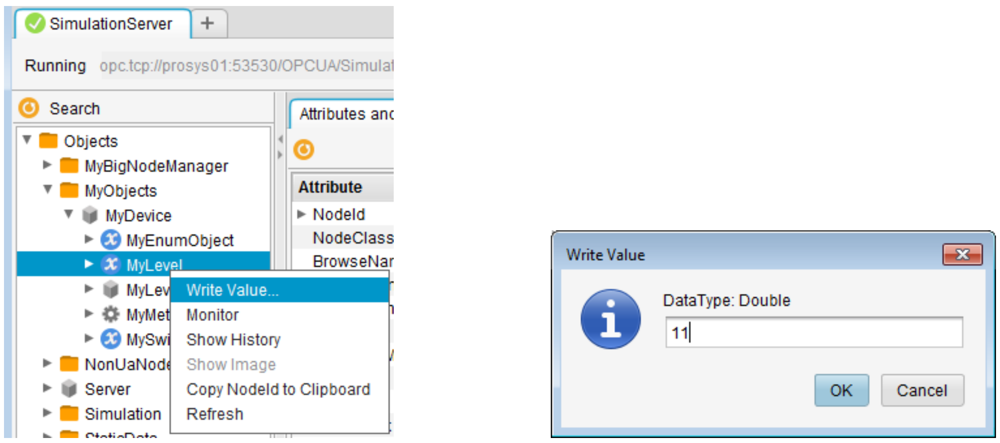
IdType Numeric

Value 1

Figure 7 The node search dialog.

4.3 Writing to a Variable

The Address Space Browser has a context menu, which you can use to perform specific operations on the nodes; for example to Write Values of variables.



DataType: Double

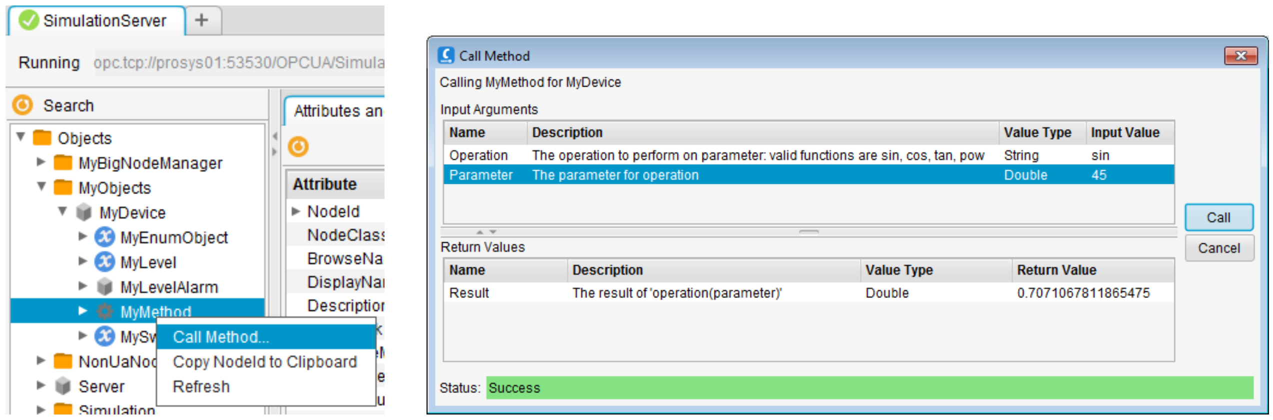
11

Figure 8 You can Write the Value of a Variable from the Address Space Browser context menu.

4.4 Calling a Method

OPC UA includes a standard way for the server to define methods and their input and output arguments. This enables the client to use a generic method call dialog that gets the information about the arguments from the server.

Call Method function is available for *Method* nodes. In the dialog box, you can set input values and then press *Call* to execute the method in the server. If the call succeeds, you will see the return values in the lower table. Possible errors in parameters, for example, are shown in the *Status* field.



Calling MyMethod for MyDevice

Input Arguments

Name	Description	Value Type	Input Value
Operation	The operation to perform on parameter: valid functions are sin, cos, tan, pow	String	sin
Parameter	The parameter for operation	Double	45

Return Values

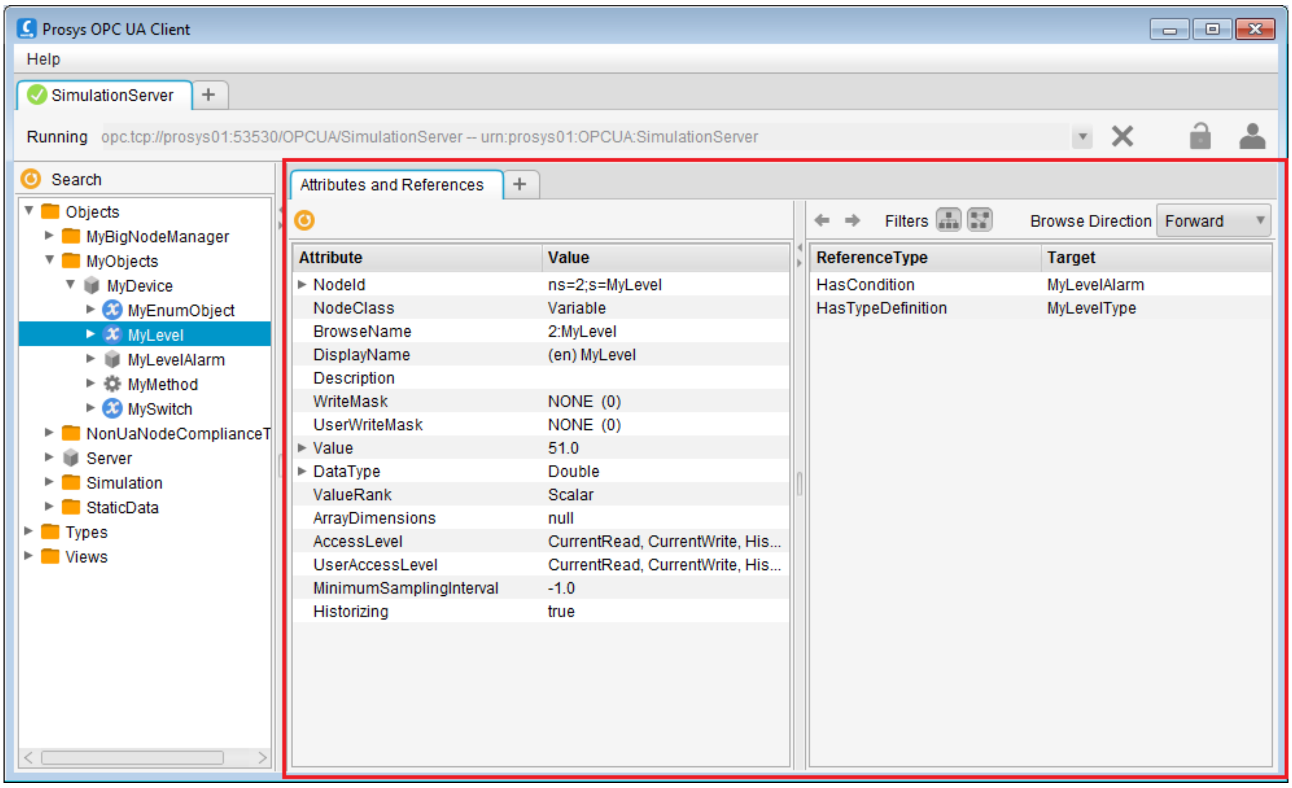
Name	Description	Value Type	Return Value
Result	The result of 'operation(parameter)'	Double	0.7071067811865475

Status: Success

Figure 9 Calling a Method. MyMethod of the Prosys OPC UA Simulation Server enables you to define an operation (sin/cos/tan/pow) and a parameter (in degrees for the trigonometric functions).

5. Attributes and References View

The *Attributes and References View* is the default view that is displayed on the right hand side of the main window. It consists of two tables: one to represent node attributes and another one to represent the node references. Both tables show the data from the node that is currently selected in the Address Space Browser. The *Attribute Table* shows the attributes of the node and their respective values. The attributes depend on the *NodeClass* of the selected node.



opc.tcp://prosys01:53530/OPCUA/SimulationServer -- urn:prosys01:OPCUA:SimulationServer

- Objects
  - MyBigNodeManager
  - MyObjects
    - MyDevice
      - MyEnumObject
      - MyLevel
      - MyLevelAlarm
      - MyMethod
      - MySwitch
  - NonUaNodeComplianceT
  - Server
  - Simulation
  - StaticData
- Types
- Views

Attribute	Value
NodeId	ns=2;s=MyLevel
NodeClass	Variable
BrowseName	2:MyLevel
DisplayName	(en) MyLevel
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	51.0
DataType	Double
ValueRank	Scalar
ArrayDimensions	null
AccessLevel	CurrentRead, CurrentWrite, His...
UserAccessLevel	CurrentRead, CurrentWrite, His...
MinimumSamplingInterval	-1.0
Historizing	true

ReferenceType	Target
HasCondition	MyLevelAlarm
HasTypeDefinition	MyLevelType

Figure 10 Attributes and References View showing the attributes and references of a Variable node.

The *Reference Table* shows the references of the node: *ReferenceType* represents the type of the reference, and *Target* represents the node to which the reference points to.

You can filter the Reference Table with the controls on top of the table. With the buttons in the middle, you can select whether the table should show hierarchical, non-hierarchical, neither or both types of nodes. The *Browse Direction* selection has three options, to display only *Forward* or *Backward* references or *Both*.

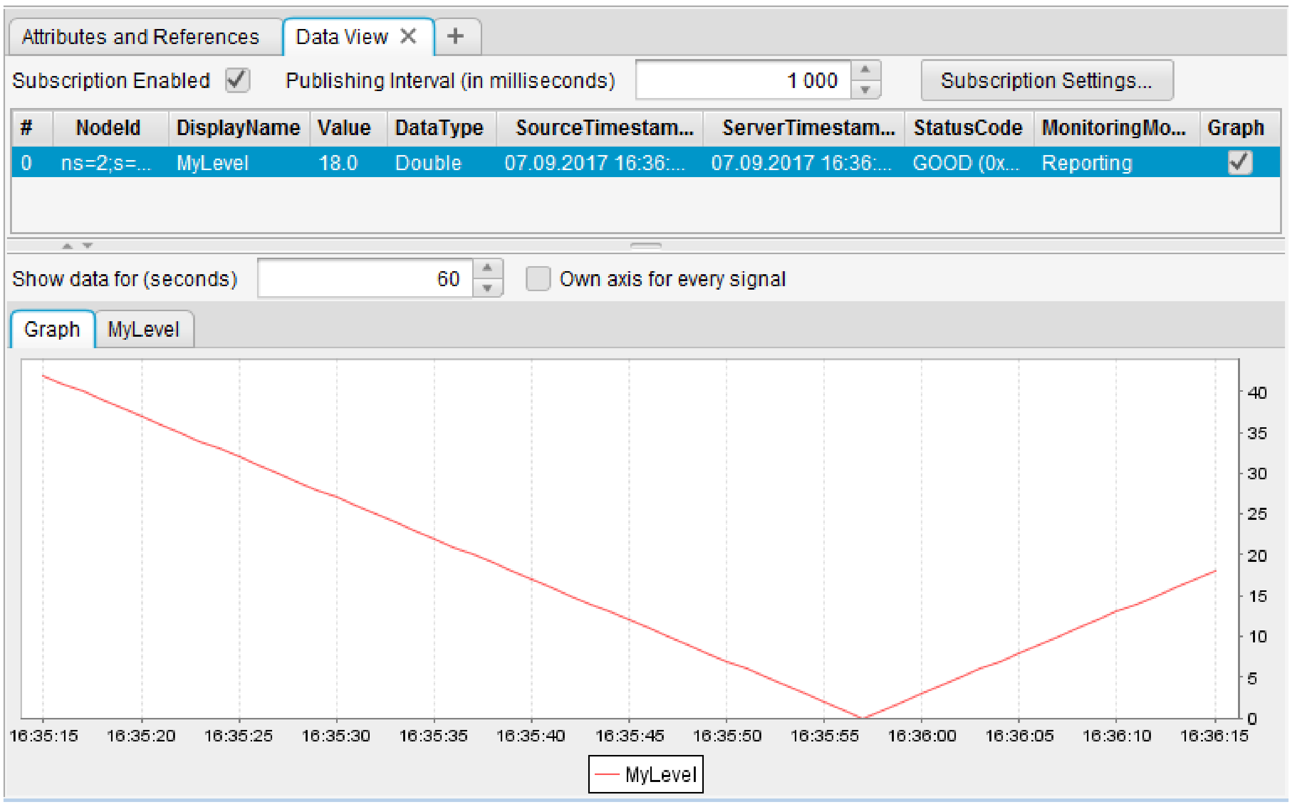
Note that you can see more complete reference information of each reference if you move the mouse over the table.

5.1 Browsing via References

It is possible to navigate from node to node also from the References View. If you double click a reference, the target node will be located from the Address Space. The arrows at the top left of the Reference Table let you follow the references back and forth.

6. Data View

The *Data View* corresponds to an OPC UA subscription, which can be used to monitor data changes in the OPC UA Server. You can add variables that you wish to monitor by dragging them from the Address Space Browser to the *Monitored Items Table* of the Data View. You can also use the *Monitor* action in the context menu of the Address Space Browser to add variables to the Data View.



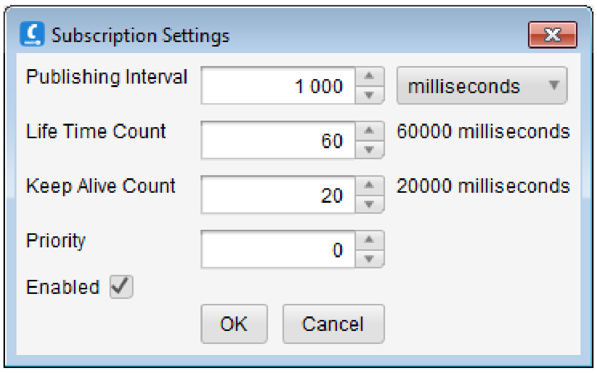
Subscription Enabled ✓ Publishing Interval (in milliseconds) 1 000

#	NodeId	DisplayName	Value	DataType	SourceTimestam...	ServerTimestam...	StatusCode	MonitoringMo...	Graph
0	ns=2;s=...	MyLevel	18.0	Double	07.09.2017 16:36:...	07.09.2017 16:36:...	GOOD (0x...	Reporting	✓

Show data for (seconds) 60 Own axis for every signal

Figure 11 The Data View corresponds to an OPC UA subscription. It contains a Monitored Items Table and Trend Graph.

The subscription has a few parameters that may be modified at the top of the screen (*Subscription Enabled* and *Publishing Interval*). You can also open the *Subscription Settings* dialog (see Figure 12), which enables more advanced configuration of standard OPC UA parameters.



Publishing Interval	1 000	milliseconds
Life Time Count	60	60000 milliseconds
Keep Alive Count	20	20000 milliseconds
Priority	0	
Enabled ✓		

Figure 12 Subscription Settings dialog in Data View

Publishing Interval defines how often the server should send notification messages. This is the maximum rate that the server may send notifications. If there are no changes to send, the server will nevertheless send an empty *Keep Alive Message* after a number of intervals defined by the

Keep Alive Count. If the client does not receive anything from the server up to a respectively long period, it may determine that the subscription has timed out. Likewise, the server is monitoring the life time of the client application: if the client is not sending any acknowledgements back to the server for the time corresponding to the *Life Time Count*, the server may stop the subscription and remove the all resources related to it. *Priority* may be used to define the relative importance of each subscription: bigger numbers correspond to higher priority. Finally, each subscription may be enabled or disabled without removing it from the server.

OPC UA Client will create a *Monitored Item* for each variable that is added to the subscription. The following information is displayed for each Monitored Item (see Figure 11):

1. NodeId
2. DisplayName
3. Value
4. DataType
5. SourceTimestamp
6. ServerTimestamp
7. StatusCode
8. MonitoringMode

NodeId uniquely identifies the measurement variable in the server.

Display Name is the name given for the variable in the server to be displayed in user interfaces. The server may have different names for different locales (i.e. languages) and each client application defines the locale of choice.

Value field is simply the current value of the measurement.

DataType represents the data type of the value.

SourceTimestamp is used to reflect the timestamp that was applied to a variable value by the data source. It should indicate the last change of the value or status code, and it should always be generated by the same physical clock.

ServerTimestamp is used to reflect the time that the server received a variable value or knew it to be accurate.

StatusCode represents the quality of the measurement: if the value is not available due to any errors in the device or connection to the device, the status may be *Bad* with a more specific code explaining the reason for the problem. When status is Bad, then Value will not be available.

MonitoringMode defines whether sampling of the node and reporting of notifications are enabled in the server.

You can use the checkboxes in the *Graph* column to select the variables that should be displayed in a trend graph. The length of the time axis is by default 60 seconds and it can be changed between 1 and 3600 seconds. By default, the variables are sharing one common Y axis. Use the *Own axis for every signal* option to create a separate Y axis for each variable.

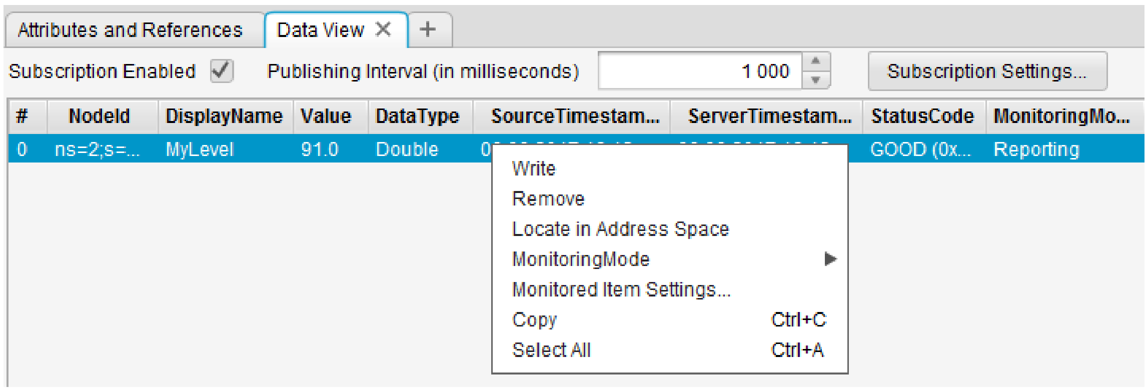
The values that are displayed in the graph are also available on separate tabbed pages.

The Monitored Items Table has a context menu, which enables a few actions. You can *Write* a new value to a variable. You can *Remove* the selected items from the subscription. *Locate in Address Space* selects the variable in the Address Space Browser.

MonitoringMode defines the current state of monitoring in the server:

- *Disabled* means that the server is not monitoring the value
- *Sampling* means that the server should sample the variable, but should not send any notifications
- *Reporting* means that the server should sample and report changes to the client via notifications

Finally, you can configure individual settings for each item from *Monitored Item Settings*.

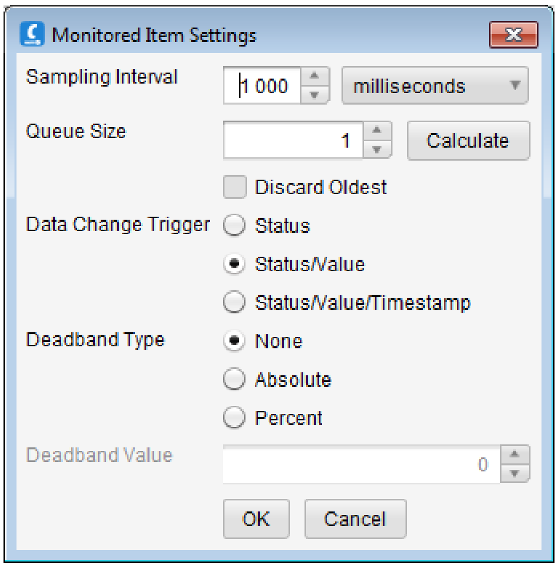


Subscription Enabled ✓ Publishing Interval (in milliseconds) 1 000

#	NodeId	DisplayName	Value	DataType	SourceTimestam...	ServerTimestam...	StatusCode	MonitoringMo...
0	ns=2;s=...	MyLevel	91.0	Double	[illegible]	[illegible]	GOOD (0x...	Reporting

Figure 13 Context menu of the Monitored Items Table.

The settings for Monitored Items include the following (see Figure 14):



Sampling Interval	1 000 milliseconds
Queue Size	1
	☐ Discard Oldest
Data Change Trigger	○ Status
	● Status/Value
	○ Status/Value/Timestamp
Deadband Type	● None
	○ Absolute
	○ Percent
Deadband Value	0

Figure 14 Monitored Item Settings dialog in Data View

Sampling Interval is typically equal to the Publishing Interval of the subscription, but you may also use lower values to request faster sampling in the server. In this case, you will also need to configure *Queue Size* so that it can hold enough samples for the whole Publishing Interval (use the Calculate button to get this filled with a proper number). Anyway, it may be better to configure more space to the queue. In case of communication problems, the server can then record more samples, even if it cannot send the notification back to the client. In case the sampling queue in the server runs out, it will throw old samples out of the queue. If Discard Oldest is set, the oldest values will be removed; otherwise the new values are discarded, except that the latest (i.e. current) value will always be available as the last value in the queue. These settings are only necessary, if you need to record all changes; if you are only interested in the current value, you will do fine with the default sampling.

Data Change Trigger defines when the server should record and send new samples. By default, any change in Status or Value will trigger a change. Alternatively, you can request only Status changes. Or if you select Status/Value/Timestamp, you should get all samples (with new Timestamps) even when the Status or Value do not change.

If you configure a *Deadband Type* for the item, it will request the server to only record new samples when the value changes more than the defined *Deadband Value*. You have two ways to define it: either as an *Absolute* value or as a *Percentage* of the variable range (the range may be defined for the variables with an EURange property. EURange is defined as part of AnalogItemType variables).

7. History View

The purpose of the *History View* is to represent historical data of variables and thus to provide OPC UA History Access functionality. The *History View* consists of a list of variables and a set of tools to define the time interval for which to request the trends from the server. The layout is illustrated in Figure 15.

Variables that have history data, the ones that have *HistoryRead* in the *AccessLevel* and *UserAccessLevel*, can be dragged and dropped from the Address Space Browser into the list of *Variables*.

You can define the history interval by two alternative ways. The default is to define the *Last* X hours (or other time units) Until a specified time (or until Now, which is the default selection). the alternative way is to define the interval between two timestamps – From and To. See Figure 15 and Figure 16 respectively, for examples.

When you click the *Read* button, the client makes a History Read request to the server and displays the received data in a Graph. The values for each variable are also available on separate tabbed pages.

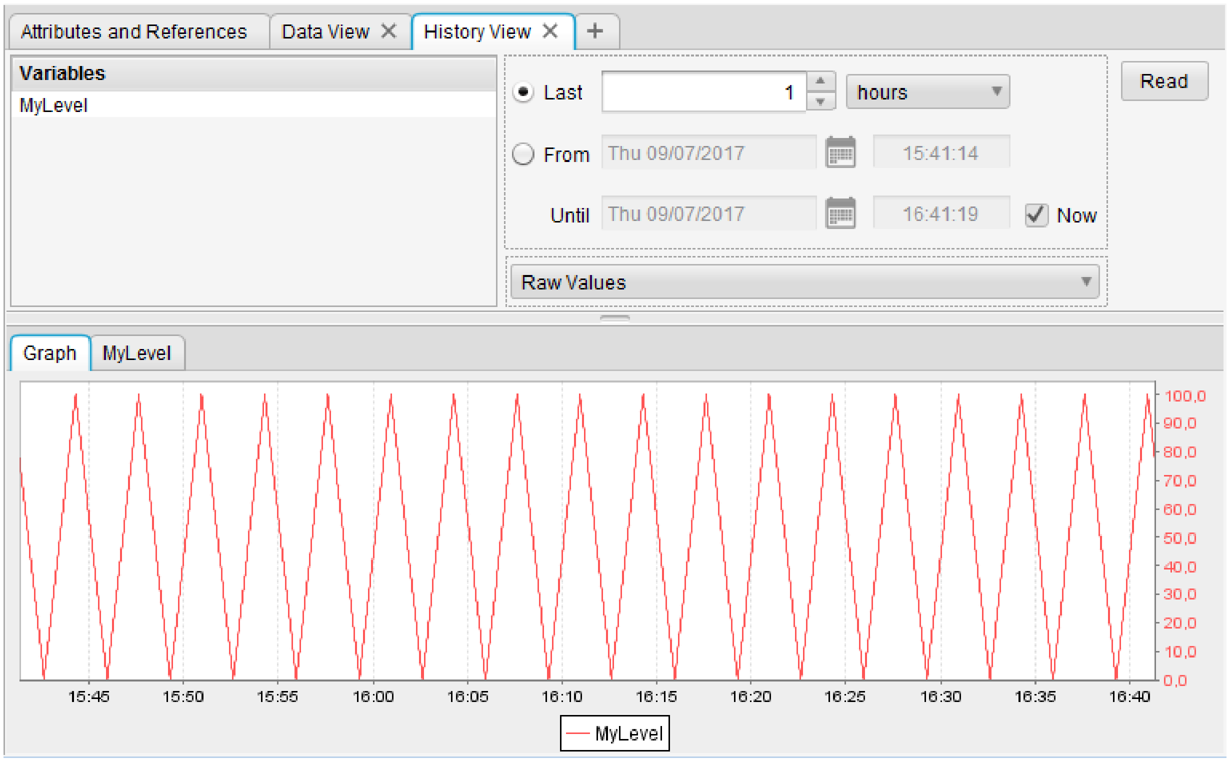


Figure 15 History View of the MyLevel variable for the Last 1 hour until Now.

The drop down list under the time interval definition is used to define optional data processing. The default is to request the *Raw Values* without any processing. Other alternatives depend on the capabilities of the OPC UA Server, which is responsible of the processing. If the server is capable of supporting some History Aggregate functions, these will appear as alternatives in addition to Raw Values. Processing interval can also be configured in this case to accompany the aggregate function. Figure 16 shows an example of a requesting the minimum value for every 5 seconds.

Note that Prosys OPC UA Simulation Server version 2.3.2 does not support aggregate functions, yet.

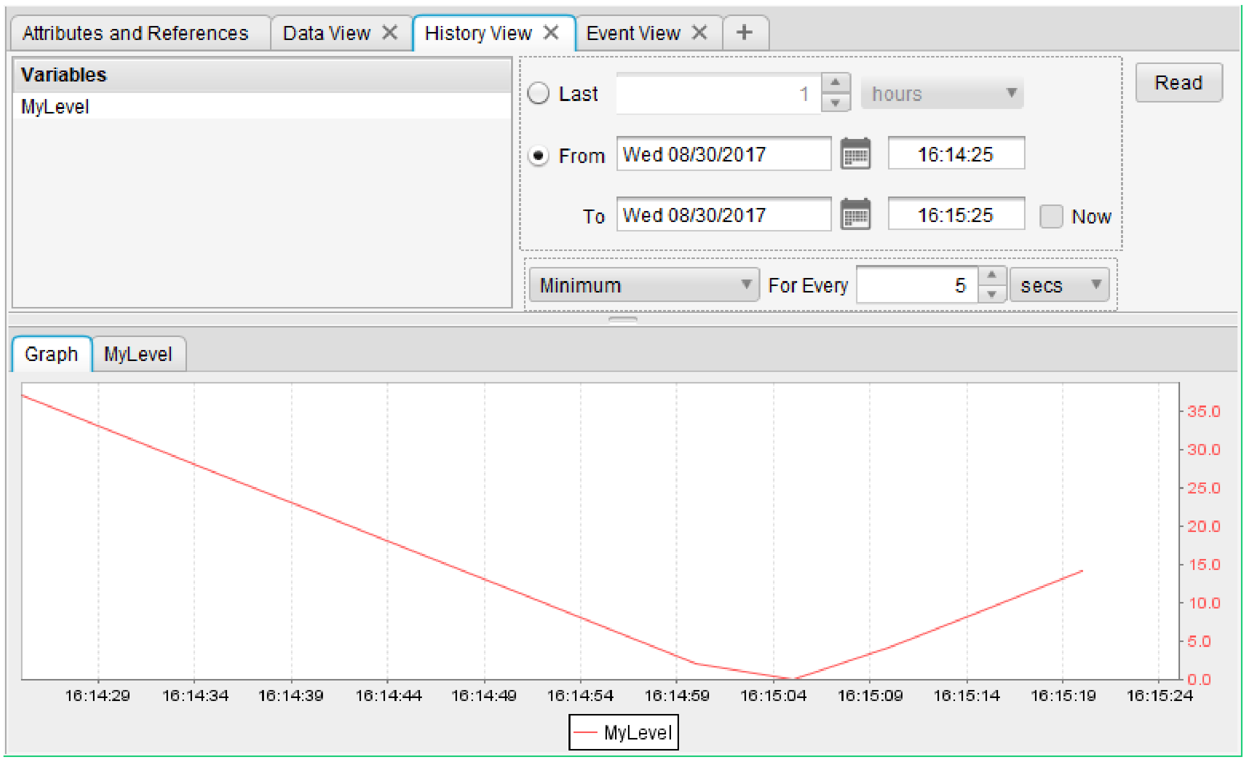
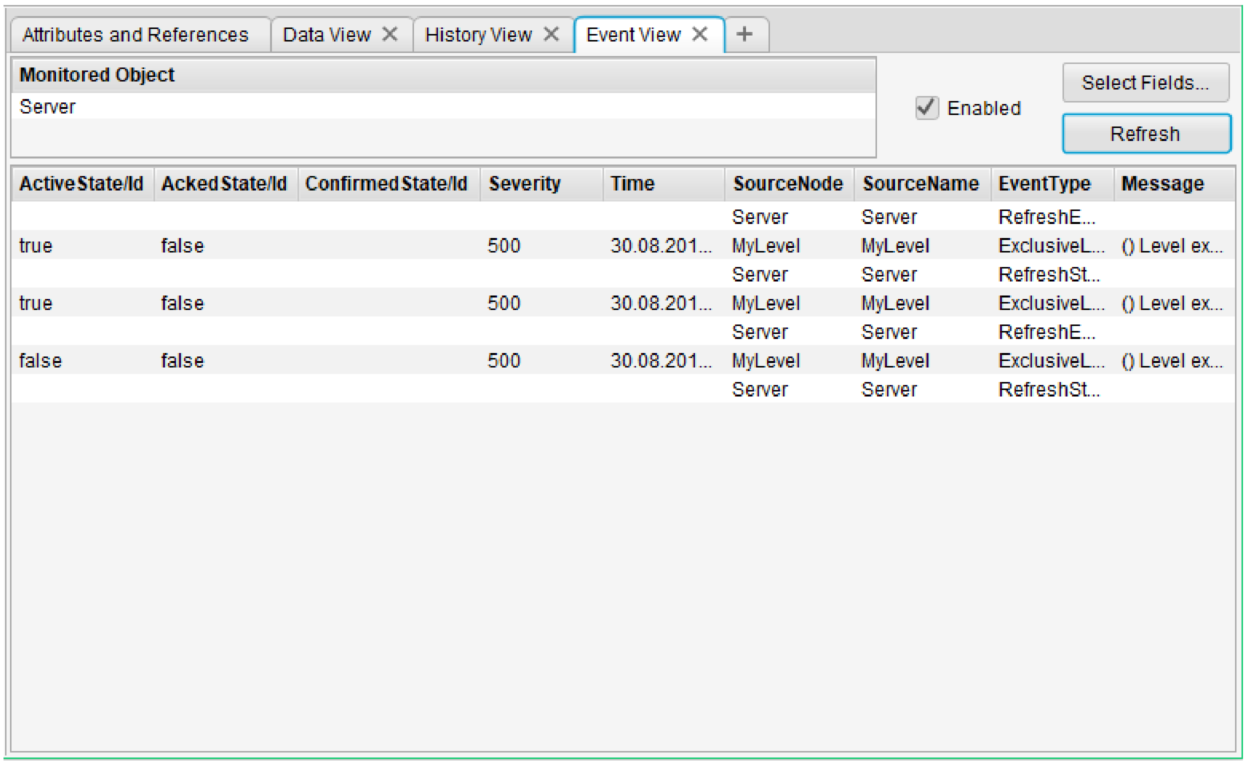


Figure 16 History View with a fixed time period and data processing.

8. Event View

The *Event View* is designed for receiving Event and Alarm notifications from the server. The main component in the Event View is the *Event Table* in which the received events are listed.



Monitored Object: Server

[✓ Enabled]

ActiveState/Id	AckedState/Id	ConfirmedState/Id	Severity	Time	SourceNode	SourceName	EventType	Message
					Server	Server	RefreshE...	
true	false		500	30.08.201...	MyLevel	MyLevel	ExclusiveL...	() Level ex...
					Server	Server	RefreshSt...	
true	false		500	30.08.201...	MyLevel	MyLevel	ExclusiveL...	() Level ex...
					Server	Server	RefreshE...	
false	false		500	30.08.201...	MyLevel	MyLevel	ExclusiveL...	() Level ex...
					Server	Server	RefreshSt...	

Figure 17 Event View

By the default, the Event Table contains nine fields *ActiveState*, *AckedState* and *ConfirmedState* present whether the event or alarm is active, acknowledged and confirmed, respectively. *Severity* shows the severity of each event. Servers can use different severity values for an event, varying from 0 to 1000 to indicate the importance of reacting to an event. Table 1 represents a mapping of five different severity levels to normal client severity levels.

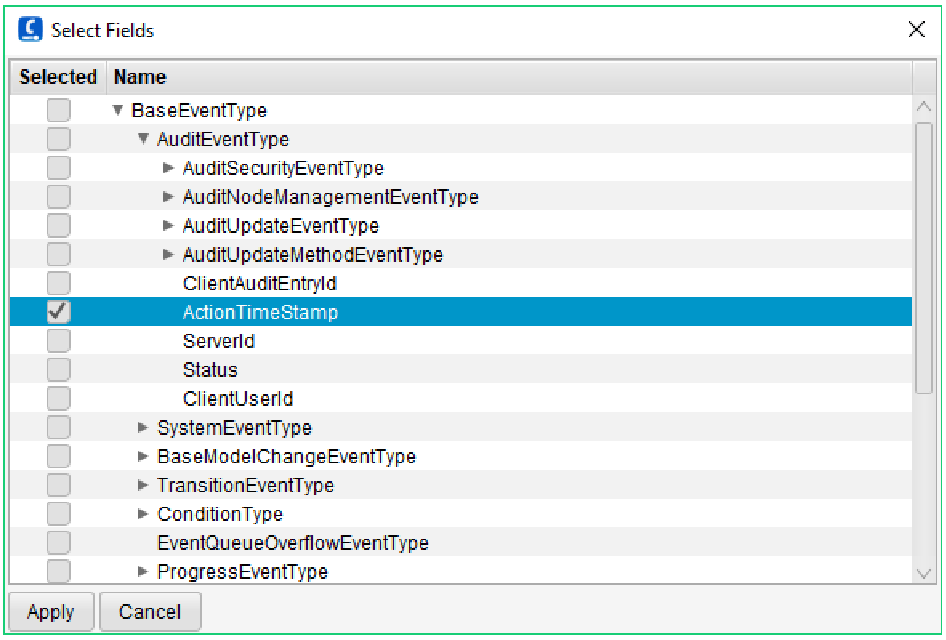
Time is the timestamp of the event as recorded by the server. *SourceNode* and *SourceName* identify the source of the event in two different ways: as a reference to the node (as a *NodeId*, in case of an object or variable) and as a name (usually as the *BrowseName* of the node). *EventType* refers to the type of event (as a NodeId). The last field, *Message*, shows the message text associated with the event as defined by the server.

Client Severity	OPC Severity
HIGH	801-1000
MEDIUM HIGH	601-800
MEDIUM	401-600
MEDIUM LOW	201-400
LOW	1-200

Table 1: Five severity levels

Use the *Select Fields* function to define which fields are requested from the server. You can select from the components and properties of every Event Type. The events may contain a lot of information and the fields that are important depend on your application and on which events you are expecting to receive. The default fields are available for most event types.

Note that the fields are only applied to new events: the events that have already been received from the server will not get any new data any more. Instead, the Event History View (next chapter) may be able to provide that for you.



Select Fields

Selected	Name
☐	▼ BaseEventType
☐	▼ AuditEventType
☐	▶ AuditSecurityEventType
☐	▶ AuditNodeManagementEventType
☐	▶ AuditUpdateEventType
☐	▶ AuditUpdateMethodEventType
☐	ClientAuditEntryId
✓	ActionTimeStamp
☐	ServerId
☐	Status
☐	ClientUserId
☐	▶ SystemEventType
☐	▶ BaseModelChangeEventType
☐	▶ TransitionEventType
☐	▶ ConditionType
☐	EventQueueOverflowEventType
☐	▶ ProgressEventType

Figure 18: Selecting Event Fields

8.1 Basic Events and Conditions

There are two main varieties of events: basic events and conditions (all descendants of Condition-Type). The main difference is that conditions have a state: for example alarms are active or inactive, unacknowledged or acknowledged, etc. The condition types are often visible in the address space and the current state can also be read and monitored via the components of the condition object (for example ActiveState and AcknowledgedState variables). Whenever the condition changes its state (for example from Inactive to Active) the server will send an event, which can be considered as a notification of state change.

The event types that are not conditions can be used to trigger basic notifications: configuration change, system events, state transitions, etc.

8.2 Condition Refresh

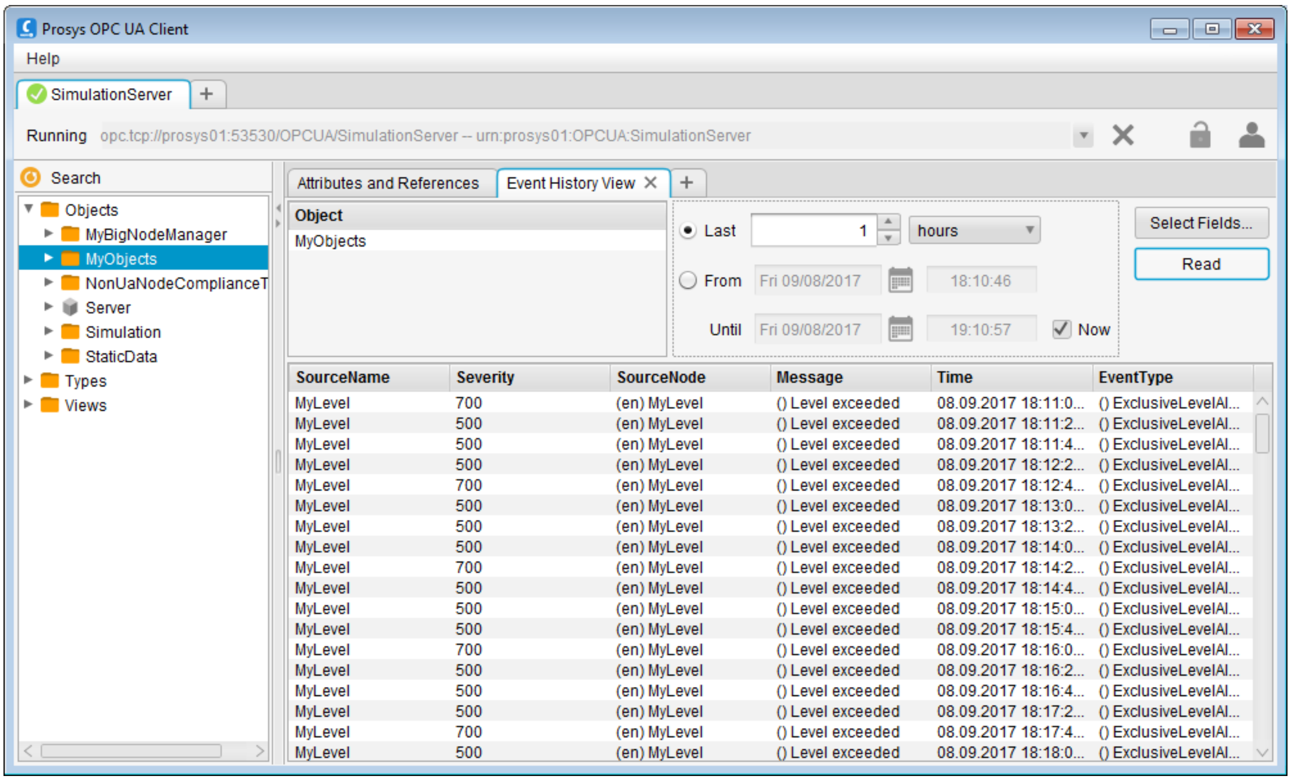
The Refresh button in the Event View will make a ConditionRefresh service call to the server. It requests the server to resend the current state of all active alarms to the client. Prosys OPC UA Client will make an automatic ConditionRefresh to the server when the Event View is initialized.

9. Event History View

OPC UA Objects, which define *HistoryRead* in their *EventNotifier* attribute, may be requested for event history.

The *Event History View* enables you to drag a valid object to the upper left corner of the view, select a suitable time interval (see the chapter 7. History View for details) and *Read* the history. The results are displayed in a table.

You may also use *Select Fields…* to define which information you want to read. See more about the event fields in chapter 8. Event View.



opc.tcp://prosys01:53530/OPCUA/SimulationServer -- urn:prosys01:OPCUA:SimulationServer

Object
MyObjects

● Last 1 hours

○ From Fri 09/08/2017 18:10:46

Until Fri 09/08/2017 19:10:57 ✓ Now

SourceName	Severity	SourceNode	Message	Time	EventType
MyLevel	700	(en) MyLevel	() Level exceeded	08.09.2017 18:11:0...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:11:2...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:11:4...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:12:2...	() ExclusiveLevelAl...
MyLevel	700	(en) MyLevel	() Level exceeded	08.09.2017 18:12:4...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:13:0...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:13:2...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:14:0...	() ExclusiveLevelAl...
MyLevel	700	(en) MyLevel	() Level exceeded	08.09.2017 18:14:2...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:14:4...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:15:0...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:15:4...	() ExclusiveLevelAl...
MyLevel	700	(en) MyLevel	() Level exceeded	08.09.2017 18:16:0...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:16:2...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:16:4...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:17:2...	() ExclusiveLevelAl...
MyLevel	700	(en) MyLevel	() Level exceeded	08.09.2017 18:17:4...	() ExclusiveLevelAl...
MyLevel	500	(en) MyLevel	() Level exceeded	08.09.2017 18:18:0...	() ExclusiveLevelAl...

Figure 19. Requesting Event History for MyObjects of Prosys OPC UA Simulation Server.

10. Image View

In *Image View*, you can view image nodes. Drag a variable, whose DataType is one of the Image types, into the view and the value of the variable will be displayed as an image. If the image is changing (i.e. snap shots from a camera), you will see the changes as they appear. The Image View also makes a Subscription to the variable, similar to the Data View subscriptions.

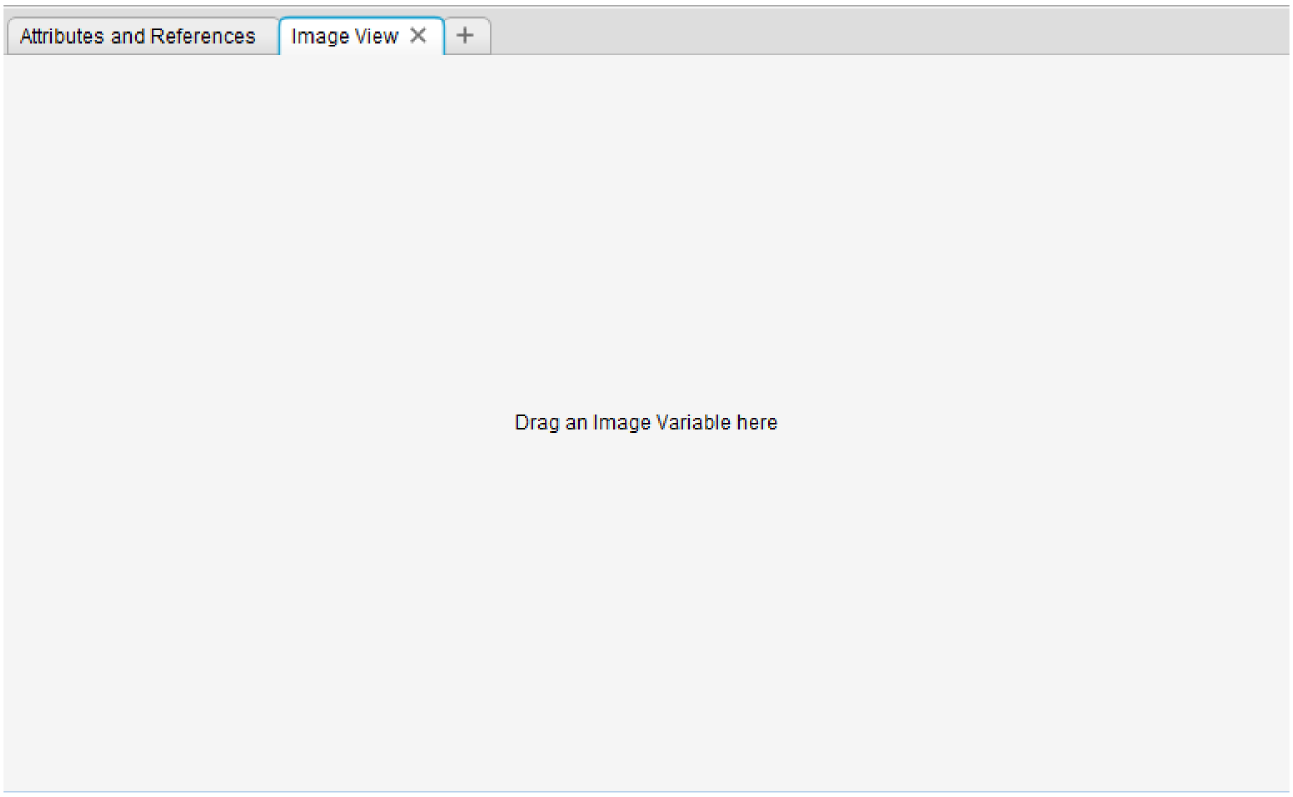


Figure 20: Image View.