



OPC UA **Browser**

User Manual
Version 4.3.0

Table of Contents

Introduction	1
Installing the Application	1
Windows	1
Linux	1
macOS	1
Uninstalling the Application	2
Windows	2
Linux	2
macOS	2
How to start using the application	3
OPC UA Servers	6
Discovery Servers	6
Client/Server Connection	7
Connecting to a Server	7
Reverse Connection	7
Security Options	8
Application Instance Certificates	9
User Authentication	11
Address Space Browser	12
Nodes	13
Searching for a Node	13
Copying browse path	14
Viewing DataTypeDictionary data	15
Writing to a Variable	16
Calling a Method	18
Attributes and References View	19
Data View	20
History View	25
Event View	27
Basic Events and Conditions	28
Condition Refresh	29
Event History View	30
Image View	31
PubSub Connection	32
Connecting to a PubSub Network	32
MQTT Network	32
UDP Network	34
UDP Multicast	35
UDP Unicast	35
Network Interface	36
PubSub Connection View	36
PubSub View	36
PubSub Data View	37
PubSub Event View	38
PubSub Log View	39

Introduction

Prosystech OPC UA Browser is a general-purpose OPC UA Client and OPC UA Subscriber. You can test connections and view data from any OPC UA server. The main functions of the OPC UA Browser include browsing the server address space, reading and writing variables, monitoring variables and events, and reading a history of variables and events. Additionally, you can connect and subscribe to a PubSub Network and receive and observe messages from PubSub Publishers.

The newest versions of the OPC UA Browser can be installed with an auto-update feature offered during the installation process. If you do not want to enable the auto-update feature, you can find the option to check for updates manually from the application's *Help* menu whenever you choose.

Installing the Application



If upgrading from version 3.x.x or earlier, you should note that the locations for the installation and the settings have changed. All your previous settings will be lost. The older version of Prosystech OPC UA Client (the previous name) is not automatically removed but can be uninstalled manually.

The installation includes a complete "embedded" Java Runtime Environment (JRE). This ensures that although the application runs in a Java environment, you do not need to install Java on your computer and worry about the Java updates. The embedded Java is only used for this application.

The application install packages are available from <http://www.prosysopc.com> upon your request. You should receive the correct package, depending on your target environment.

Windows

On Windows, run the installer executable `prosys-opc-ua-browser-windows-x64-4.3.0-xxx.exe` and follow the instructions. By default, the application is installed to the folder *Program Files/ProsystechOPC/Prosystech OPC UA Browser*.

Linux



The application requires a GUI (Linux Desktop Environment) in order to run.

On Linux, first open the terminal and navigate to the directory of the downloaded `.sh` file.

Then run the installation with the command

```
sudo sh prosys-opc-ua-browser-linux-x64-4.3.0-xxx.sh
```

This will open the installer where you can follow the steps to complete the installation. By default, the application is installed to the folder `opt/prosys-opc-ua-browser`.

macOS

On macOS, run the installer application and follow the instructions. By default, the application is installed to the folder `/Applications`.

Uninstalling the Application

Windows

On Windows the application can be uninstalled through the Control Panel or the *Apps & features* menu, or optionally with the uninstaller that is located in the installation folder.

Linux

On Linux, open the terminal and navigate to the installation folder (default folder is *opt/prosys-opc-ua-browser*) and use the command `sudo ./uninstall`.

macOS

On macOS you can just remove the application from the Applications folder.

How to start using the application

This chapter briefly explains how you can get started with the OPC UA Browser. If you prefer to watch a video tutorial, you can find a four-part series on our YouTube channel: https://www.youtube.com/watch?v=M3rBjffmmUk&list=PLZpqH2EeqVW3P4NQGgXn9tsulWh_p5cMN

The OPC UA Browser remembers the server and PubSub connections you have made. The connections are listed for you in the starting view of the application (see [Figure 1](#)). Your favorite connections are listed first, followed by the rest of the connections. You can add and remove favorite connections by clicking the star icon on the right side of the *Quick Links* in the starting view or on the left side of the address bar during an active session. You can remove connections from Quick Links by pressing the 'X' next to each connection.

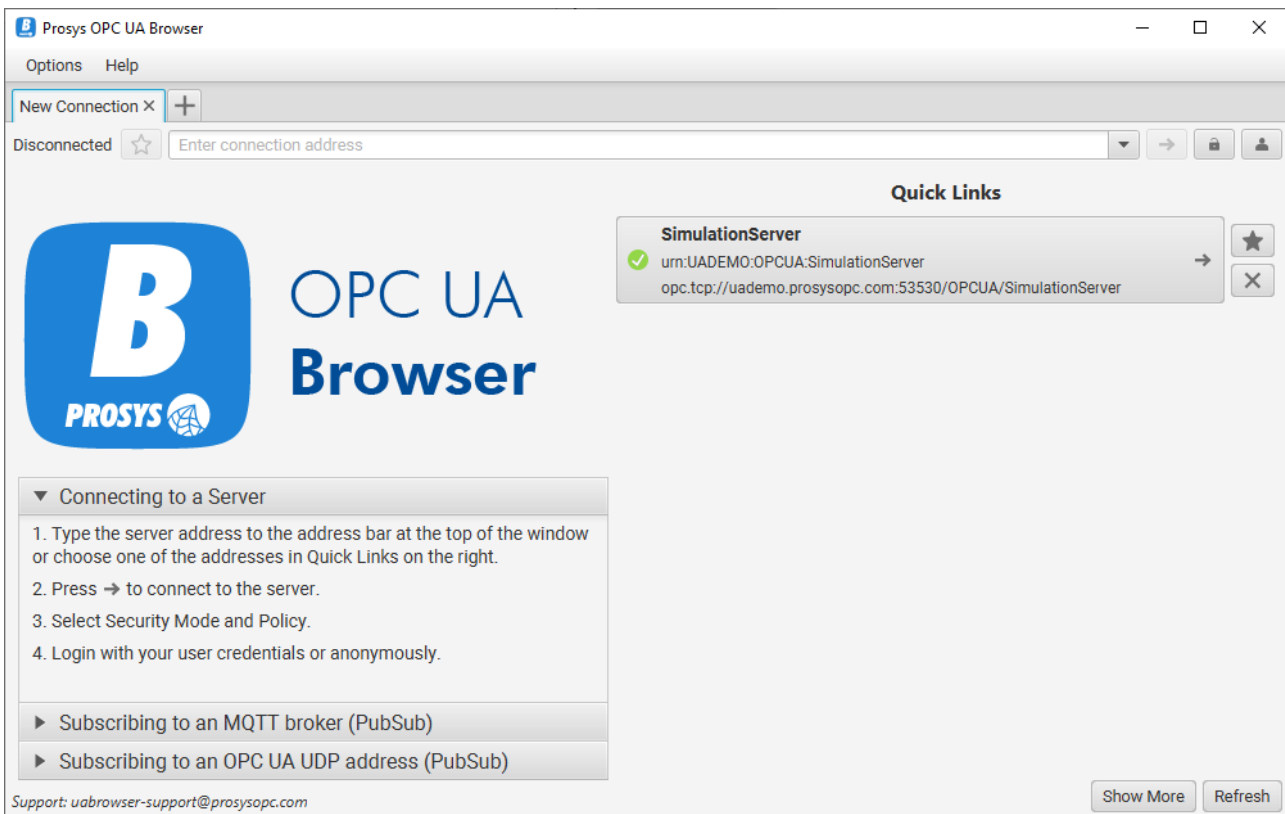


Figure 1. Starting view of the OPC UA Browser

The preferences of the OPC UA Browser can be configured in *Preferences Configuration* dialog (see [Figure 2](#)) opened by selecting **Preferences...** in *Options* menu. Hover over the controls on the right side of the dialog with your cursor to show tooltips of various settings to read more information on them. Changes to preferences are applied immediately after pressing the **Save** button in the dialog.

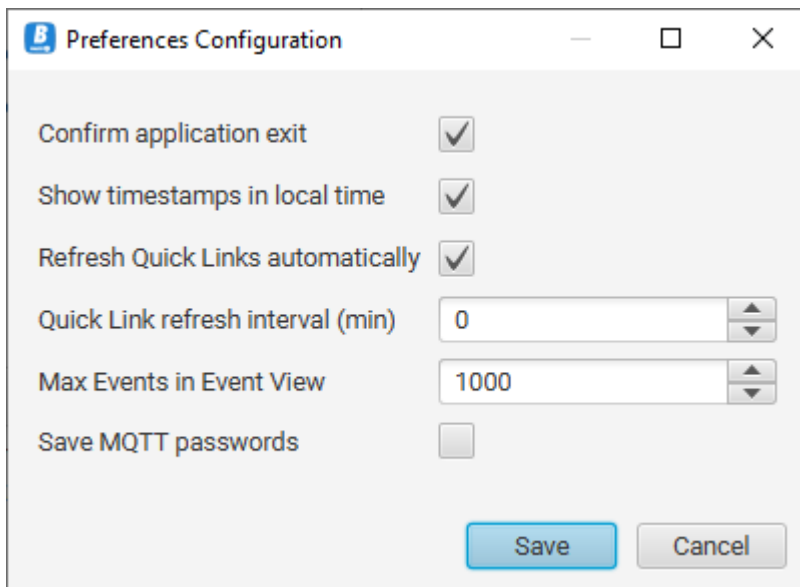


Figure 2. Preferences Configuration dialog.

The OPC UA Browser has the internet-based UADEMO server connection configuration set for you by default. This way, you can get started right away, even if you do not know any other server addresses. Follow the instructions on the left side of the starting view to establish a connection. Note, that you cannot form a secure connection to the UADEMO server. You can read more about security in [Security Options](#). In the latest version of the OPC UA Browser, you can subscribe to PubSub Networks as well. See [PubSub Connection](#) for more details on that.

With the OPC UA Browser, you can have multiple 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.

Attributes and *References* is a default view for every server. By clicking the '+' button highlighted in [Figure 3](#) by the red circle, 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 the history of events from the server. [Image View](#) provides a unique view for monitoring image data. [Figure 3](#) illustrates the layout of the OPC UA Browser.

You can find the most useful Nodes from the *Objects* folder on the left side of the view shown in [Figure 3](#). If you want to monitor a Variable from the Objects folder, you have two ways of achieving the same result. Right-click on a Variable and select *Monitor Data* (see [Figure 3](#)). This will open a new Data View for you with the chosen Variable. Alternatively, you can click on the '+' button mentioned above to open a Data View and drag the Variable into the table (see [Data View](#) for more information).

In [Figure 4](#) you can see that all six Variables under *Simulation* have been selected and dragged to be monitored in the *Data View*.

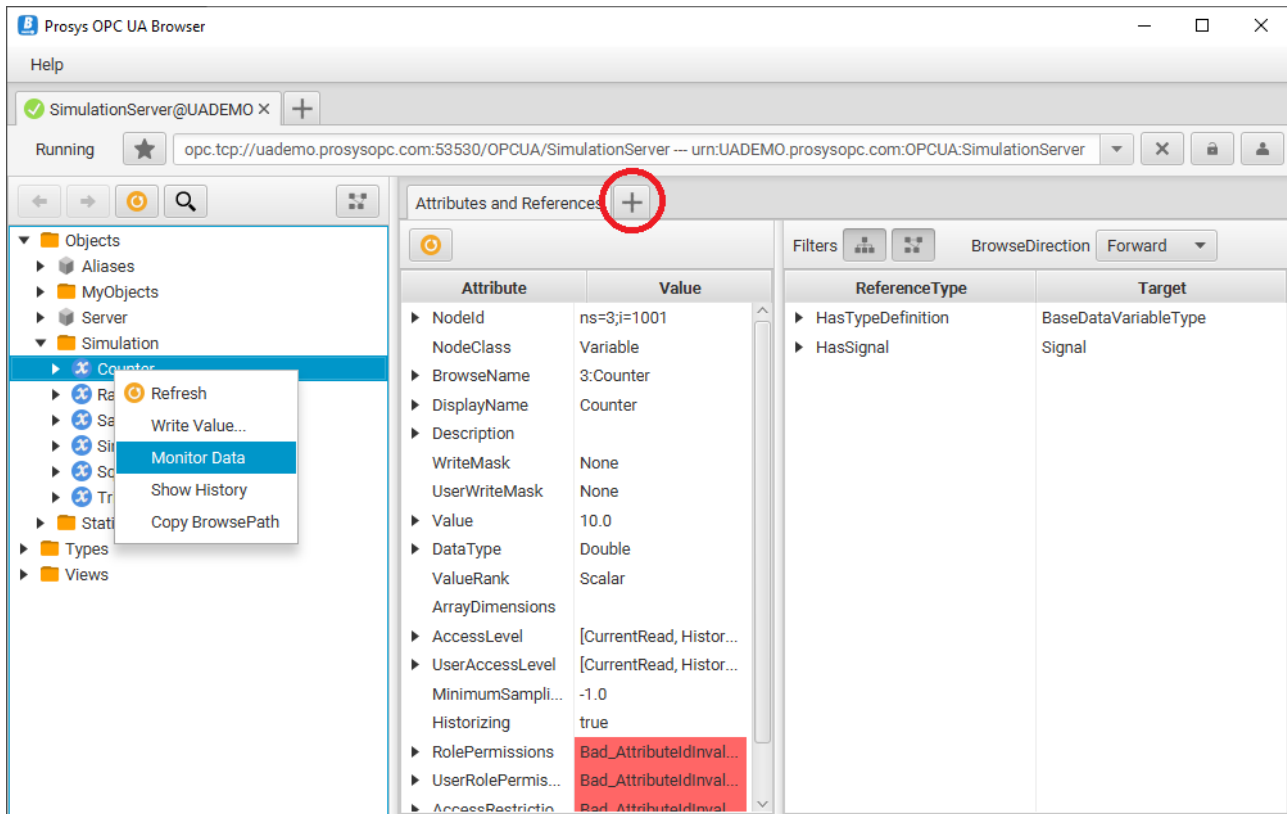


Figure 3. OPC UA Browser Overview

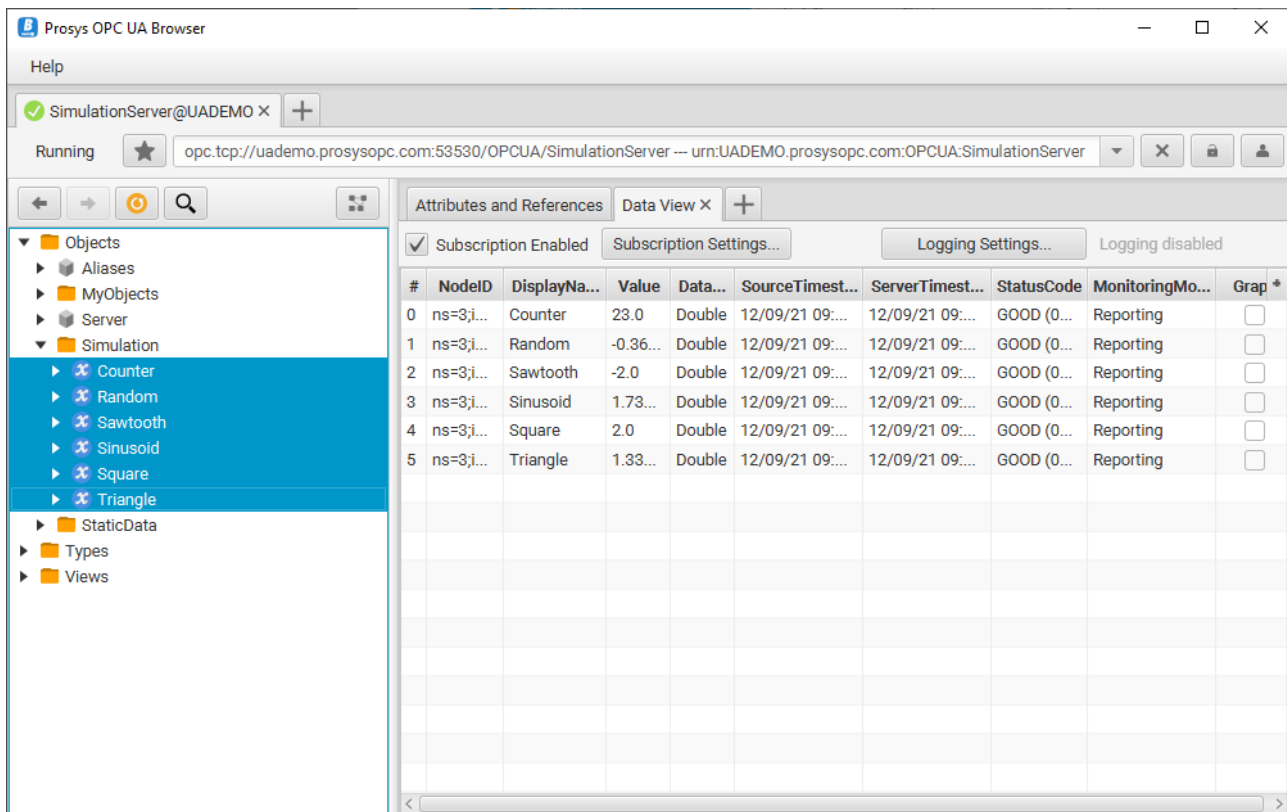


Figure 4. Data View with six Variables.

OPC UA Servers

OPC UA Servers are connected with a server address. To establish a connection, you need to find out the server's address 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 computer's hostname where the server is running (`localhost` always works locally) and the TCP port the server is listening to. OPC UA defines a standard port number as 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 ServerName part. If the ServerName is not defined, the `/` character may be omitted before it.



If you do not have any actual OPC UA servers available to connect to, you can use the Prosystech OPC UA Simulation Server to play around with the application. You can download the Prosystech OPC UA Simulation Server from <https://www.prosysopc.com> website and run it locally. You can then use the following server address:

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

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

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

Discovery Servers

OPC UA defines two types of Discovery Servers:

- Local Discovery Server (LDS) is often installed on the PC to keep a list of installed servers on 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 on 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.

Also, each OPC UA server contains an internal discovery server that can provide similar information about the server application itself. The LDS typically listens to the TCP 4840 Port (having address `opc.tcp://localhost:4840`). Several embedded devices and PLCs, which do not have any LDS, also use Port 4840 for the actual server address.

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

Client/Server Connection

Connecting to a Server

Connecting to an OPC UA server is based on the server address that can be typed in the address bar at the top of the window (see [Figure 5](#)). 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 to and disconnecting from the server. If the client cannot connect to the server, an error dialog is shown, displaying the error message that explains the failure to establish a connection.

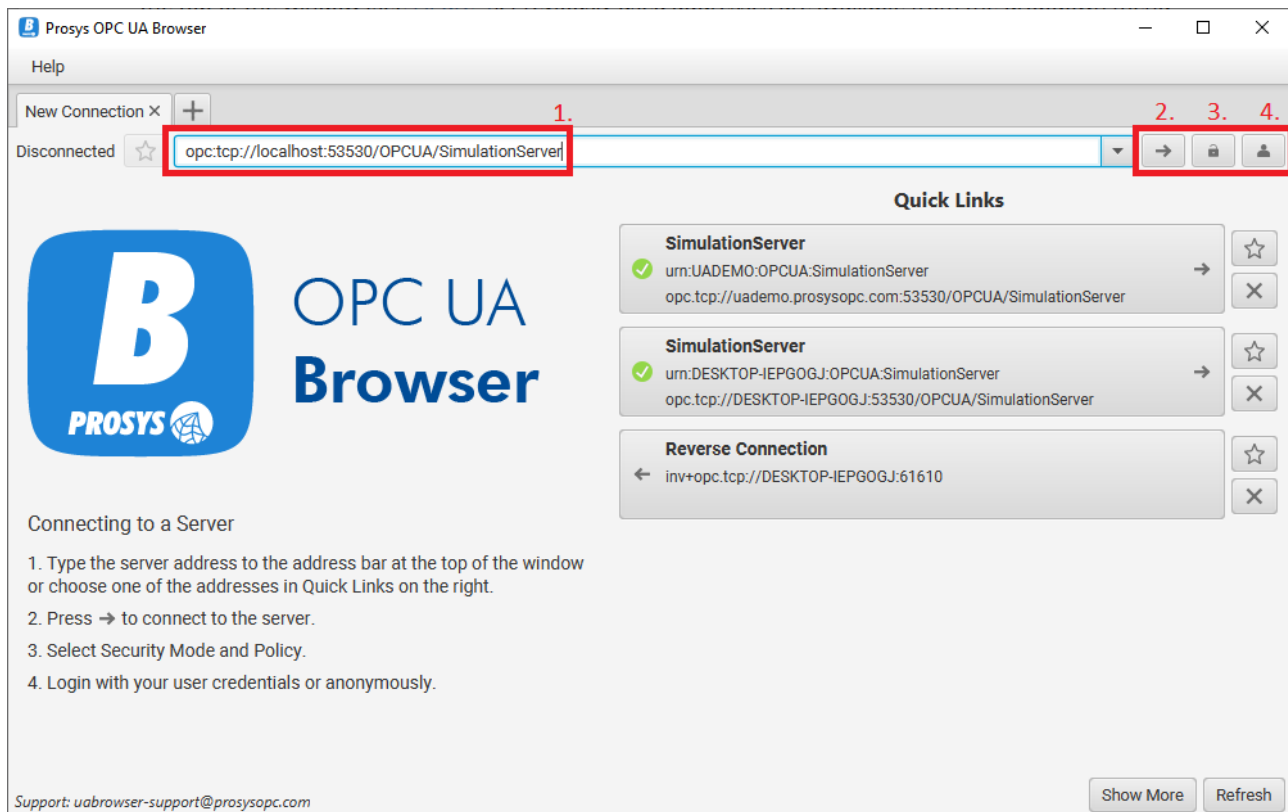


Figure 5. Server Address Bar (1.) Connect/Disconnect button (2.) Security Settings (3.) and User Authentication Settings (4.)

If the client manages to connect but determines that the target is an OPC UA Discovery Server (see [Discovery Servers](#) above), it prompts for a selection from the available server addresses provided by the discovery server. You can then choose one of the actual server addresses to make a new connection attempt.

Prosystech OPC UA Browser accepts partial addresses as well: if you only enter the hostname, **opc.tcp** is assumed to be the protocol, and the port number will be set to 4840.

Reverse Connection

Prosystech OPC UA Browser supports the connect mechanism where the OPC UA Server initiates the connection. To establish the reverse connection, you need to use the **inv+** prefix like this:

```
inv+opc.tcp://localhost:61610
```

Alternatively, you can use one of the following shorthand notations **inv**, **inverse**, **rev**, **reverse** with a port number. They will create address **inv+opc.tcp://<hostname>:port**. It is treated as an IPv6 wildcard address to determine which network interfaces are listened for reverse connections. For example, entering:

```
reverse:61610
```

on a machine with hostname **EXAMPLE** would produce the following address

```
inv+opc.tcp://EXAMPLE:61610
```

when the connection is established. Moreover, it would also listen to incoming connections by binding to all IP addresses the machine has.



The hostname part is used to determine which local address(es) are listened to for incoming connections. The machine hostname is treated as a wildcard for all the IPs the machine has. Otherwise, it must be one of your machine's IP addresses or a hostname that your DNS resolves to a local IP address.



Reverse Connections only work on **opc.tcp**.

Security Options

Whenever you make a connection to a 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 from being seen by any third parties.

The security policy determines the algorithm for signing and encrypting. As defined by the OPC UA Specification v1.04 ([Figure 6](#)), there are multiple security policies available. Each one of them defines a set of security algorithms and parameters to use for the OPC UA TCP communication:

1. Basic128Rsa15
2. Basic256
3. Basic256Sha256
4. Aes128Sha256RsaOaep
5. Aes256Sha256RsaPss



Technically, there is also a Security Policy **None**, but that is not visible in the UI, as it is only used (automatically) in combination with the Security Mode **None**.

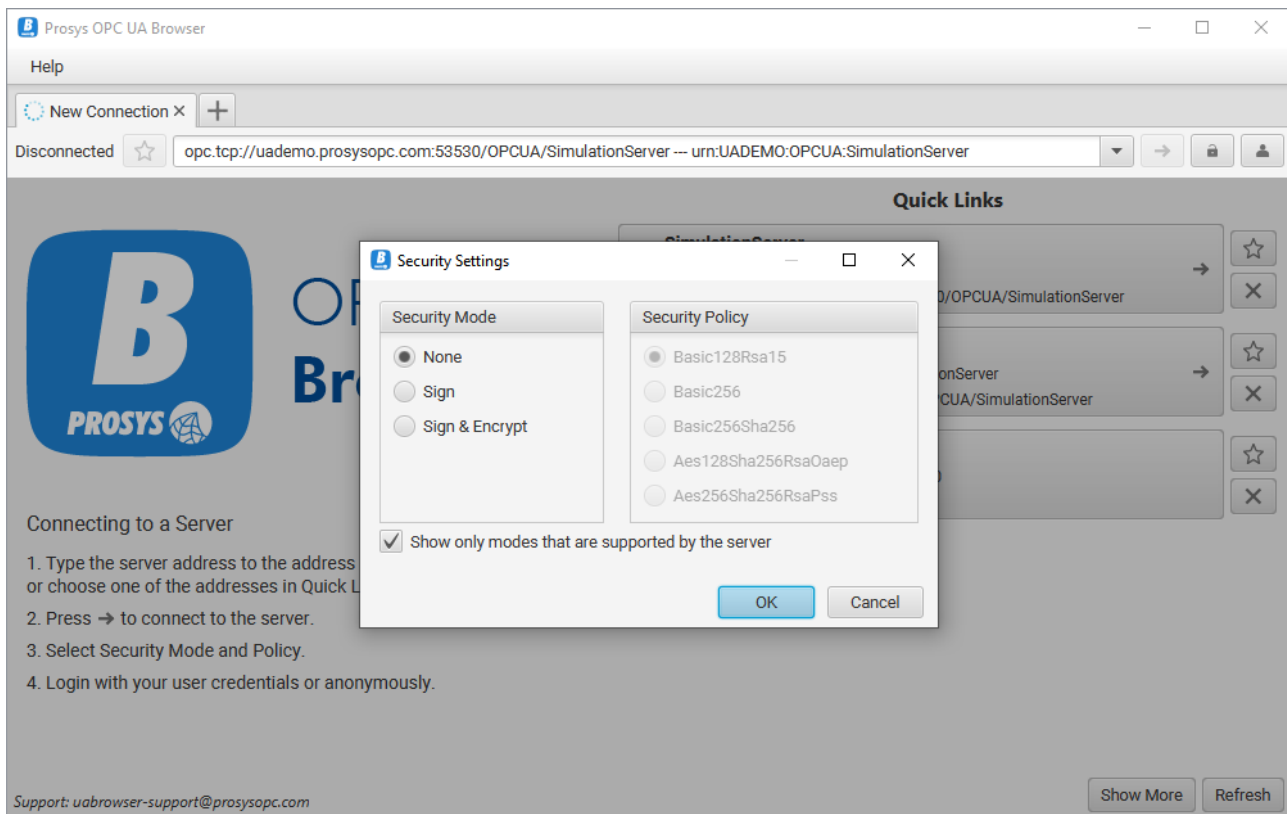


Figure 6. Security Settings

To establish a connection, the selected mode and policy must match the ones supported by the server application. To help find an applicable mode, you can use the *Show only modes that are supported by the server* option.

If you wish to use a secure connection (Sign or Sign&Encrypt), the client and server applications must trust each other. To build a trust relationship, the applications use Application Instance Certificates.

Application Instance Certificates

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

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 these cases, we talk about Self-signed certificates. In the case of self-signed certificates, each application must form trust with other applications separately. Therefore, by default, when you try to connect securely to a new server for the first time, you 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 prompts you accordingly (Figure 7).

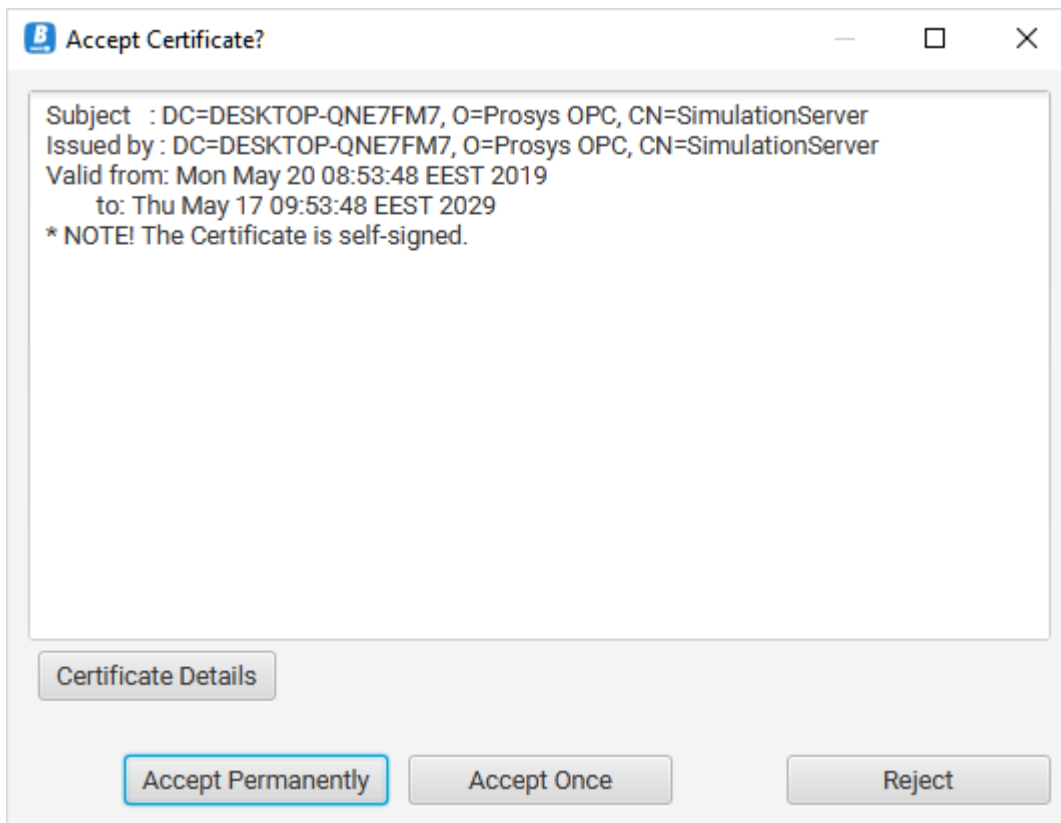


Figure 7. Certificate verification dialog in the client application

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

The Certificate contains information about the application and a cryptographic Public Key. A separate Private Key accompanies it. These two keys enable asymmetric encryption, which is the basis of the OPC UA security mechanism. 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.

You can find the certificate for the OPC UA Browser itself and all the certificates given to the application by other applications in the *Certificates* dialog that can be opened from the *Help* menu. In this dialog, you can examine, trust and reject certificates.

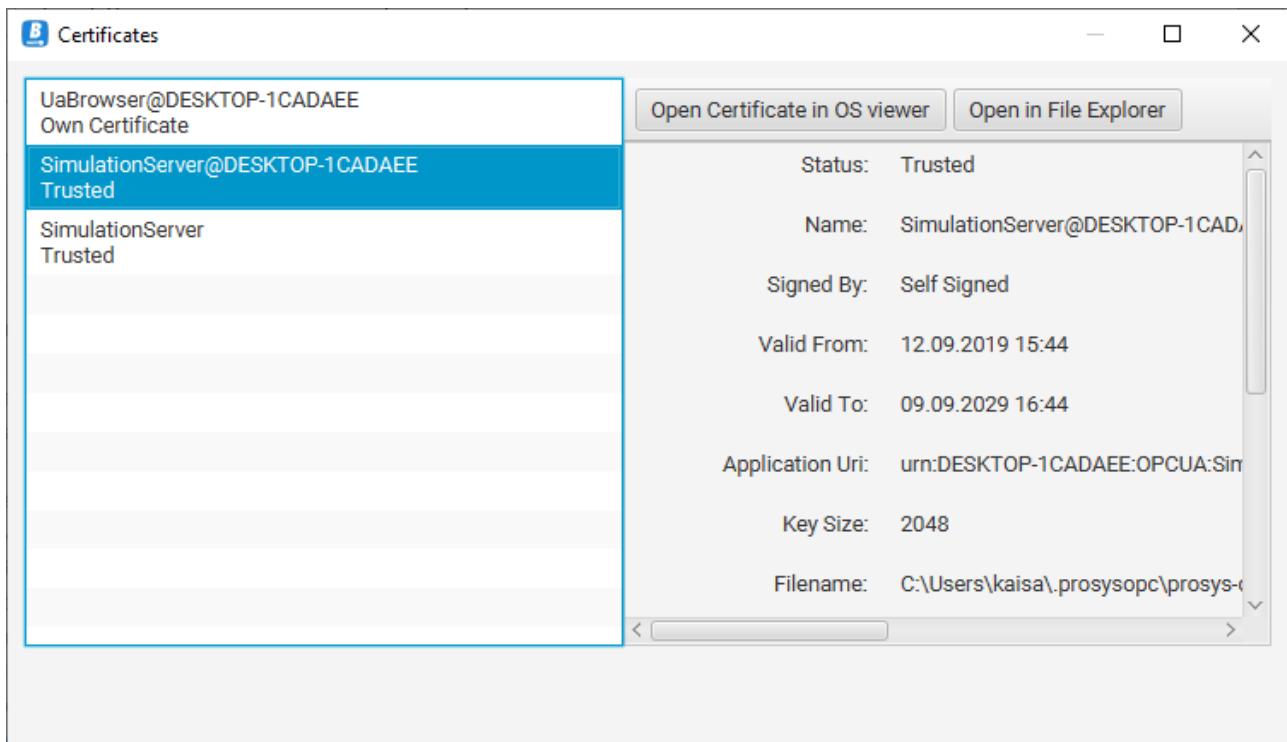


Figure 8. Certificates dialog lists all certificates that have been given to the OPC UA Browser

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. Prosystech OPC UA Browser currently supports all but external authentication systems.

The user identity may be defined before establishing the connection or while the connection is open. Select the user authentication setting by clicking the user icon in the upper right corner of the user interface (see [Figure 5](#) and [Figure 9](#)). It can also be applied when the connection is established to impersonate the session to the identified user account. Note, that the username and password must be set on the server in order for you to use them.

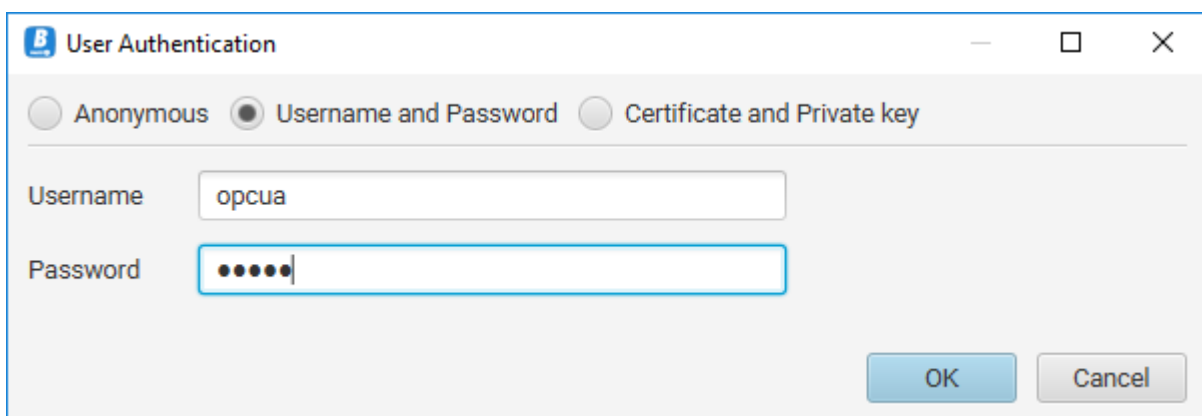


Figure 9. Defining the user authentication mode and user account. You can change the user account before or during a connection. Prosystech OPC UA Simulation Server enables you to define your usernames and passwords to test with.

Address Space Browser

Once you are connected to a server, you can browse the server's Address Space to locate the data and information you are interested in. You can do this with the Address Space Browser, which is on the left-hand side of the application window. As defined by the OPC UA Specification, address space always contains three main folders: Objects, Types, and Views.

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

The *Types* folder contains all metadata of the server: object types, variable types, data types, event types, and reference types. The metadata helps to understand the definition of the instances, 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 of the server's information model, but it is not commonly in use.

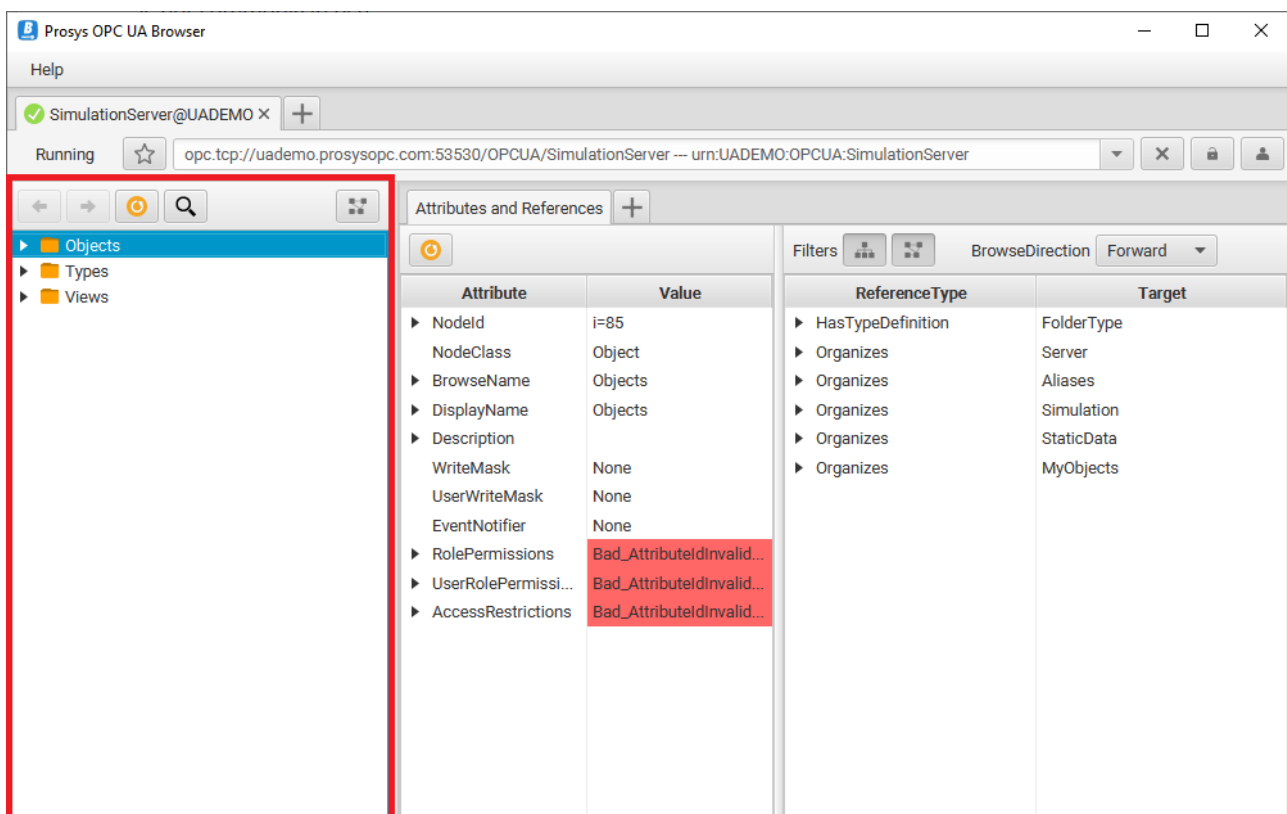


Figure 10. The Address Space Browser is used to locate instances (in the Objects folder) and types (in Types folder) from the server. Views folder is used only very seldom.

Nodes

All items in the address space are called nodes: Objects, Variables, Types, et cetera. The nodes have several Attributes that define specific details of each node. For example, Variables have a Value attribute.

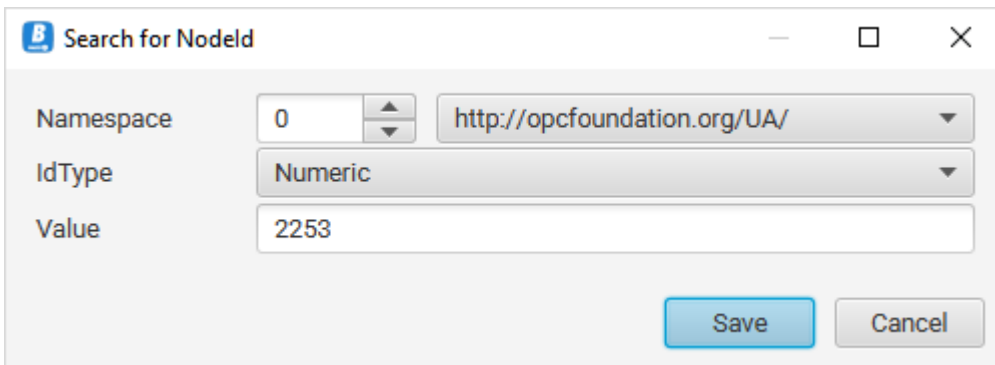
The main attribute of each node is the NodeId; a unique identifier used to identify the node in the server.

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 [Attributes and References View](#) for more information.

Searching for a Node

At the top of the Address Space Browser, you can find a Search button, which helps you locate a node with a particular NodeId. In the search dialog (see [Figure 11](#)), you can specify the Namespace, IdType, and Value of the NodeId. If the respective node exists, it is activated in the Address Space Browser.



The image shows a dialog box titled "Search for NodeId". It contains three input fields: "Namespace" with a value of "0" and a dropdown menu showing "http://opcfoundation.org/UA/"; "IdType" with a dropdown menu showing "Numeric"; and "Value" with a text input field containing "2253". At the bottom right, there are two buttons: "Save" and "Cancel".

Figure 11. The node search dialog.

Copying browse path

Every Node in the Address Space Browser has a context menu option to *Copy BrowsePath* (see [Figure 12](#)). The whole BrowsePath will be copied to clipboard, so you can paste it somewhere useful.

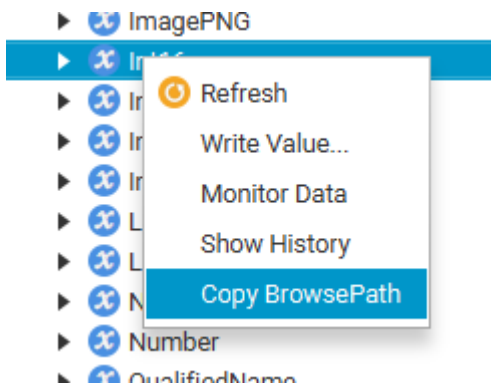


Figure 12. Option to *Copy BrowsePath* in the context menu.

When you paste the BrowsePath somewhere, it will be in a format like this:

```
/Root/Types/DataTypes/OPC Binary/Opc.Ua
```


Viewing DataTypeDictionary data

For some Nodes, Address Space Browser offers an option to view their DataTypeDictionary. This option can be found in the context menu of the Address Space Browser. A separate dialog will appear, representing the data in a format easily copied and pasted somewhere else.

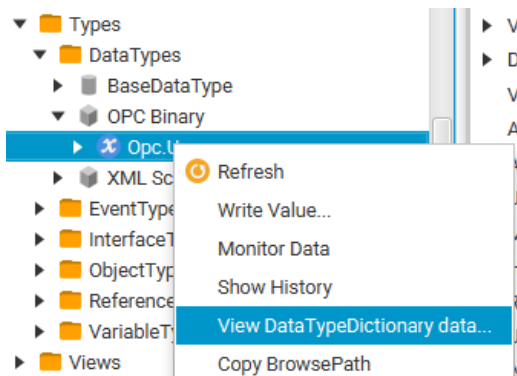


Figure 13. You can view a Node's DataTypeDictionary data from the Address Space Browser context menu.

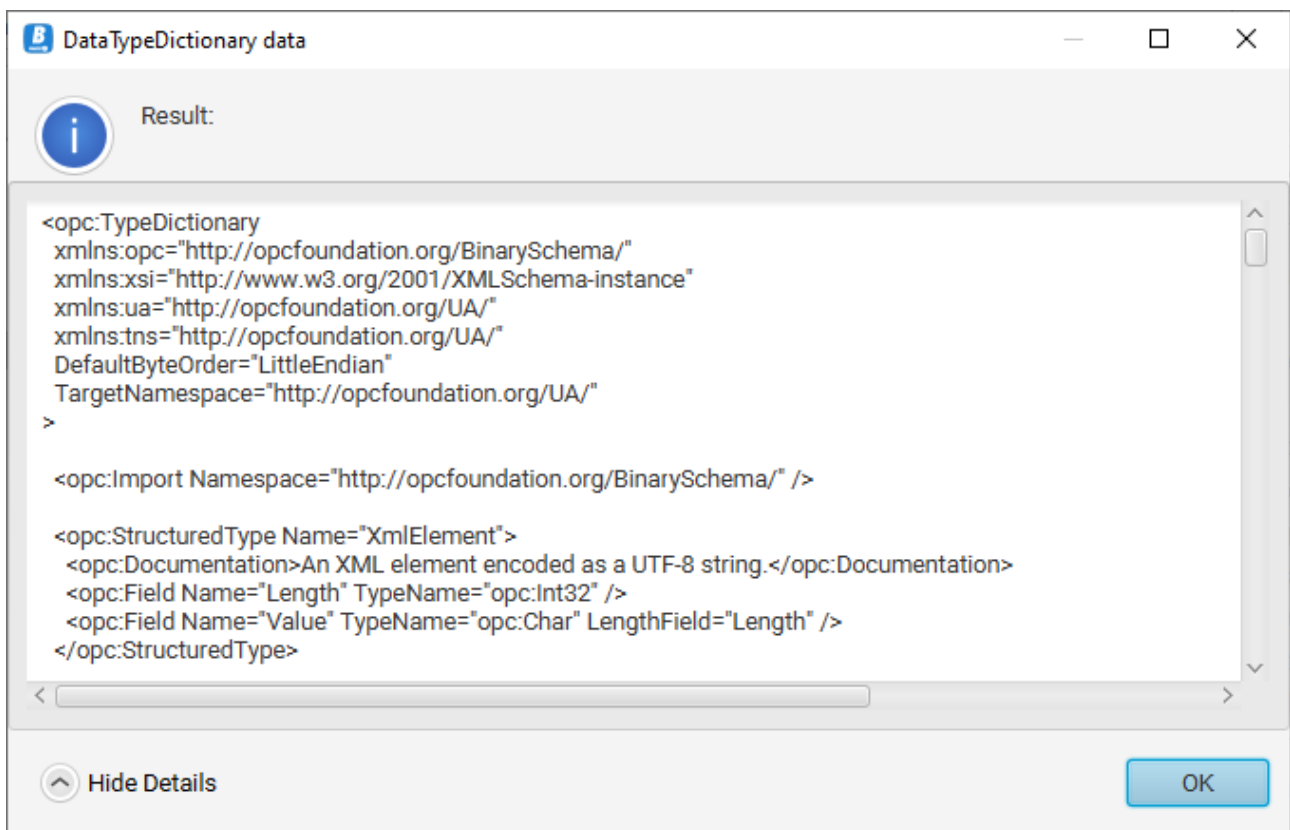


Figure 14. A dialog showing DataTypeDictionary data of a Node.

Writing to a Variable

A context menu option *Write Value...* is available for every variable. The latest version of the OPC UA Browser supports writing to more complex variables, such as Structures, Arrays and abstract DataTypes.

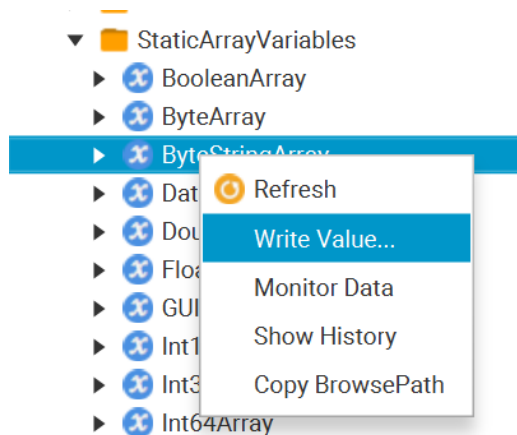


Figure 15. You can Write the Value of a Variable from the Address Space Browser context menu.

The dialog for writing to variables looks like [Figure 16](#). In the first cell you have the name of the variable, the second cell defines the DataType of the value and the cell for *Value* is where you will write or select the new value. If the DataType of a variable is abstract, the dialog will give you a drop-down box of options to define a non-abstract DataType.

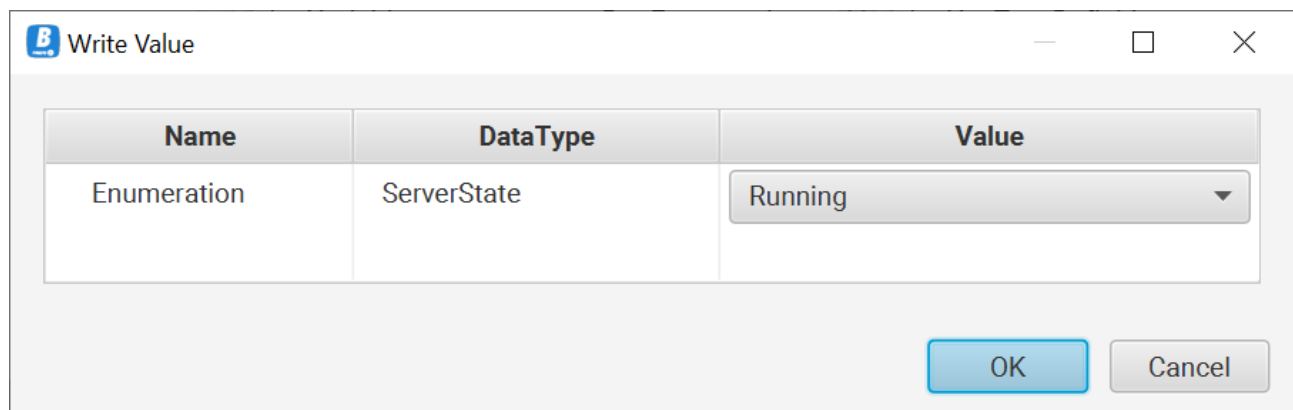


Figure 16. Write dialog

For array variables, you can define values individually for every cell in the array. In [Figure 17](#) the dialog lets the user select the DataType to be defined for every array cell. Next, the user can proceed to define the information for the cells. The required fields vary according to the DataType.

Write Value

Name	Data Type	Value
▼ VariantArray	StructureDefinition	StructureDefinition[6]
▼ [0]	StructureDefinition	StructureDefinition
DefaultEncodi...	NodeId	i=0
BaseDataType	NodeId	i=0
StructureType	StructureType	<Choose a Value>
Fields	StructureField	StructureField[0]
▶ [1]	StructureDefinition	StructureDefinition
▶ [2]	StructureDefinition	StructureDefinition
▶ [3]	StructureDefinition	StructureDefinition
▶ [4]	StructureDefinition	StructureDefinition
▶ [5]	StructureDefinition	StructureDefinition

OK
Cancel

Figure 17. Write dialog for an array

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. The latest version of the OPC UA Browser supports a variety of DataTypes and ValueRanks to use as input arguments.

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 on the server. If the call succeeds, you might see the return values in the lower table. Possible errors in parameters, for example, are shown in the Status field.

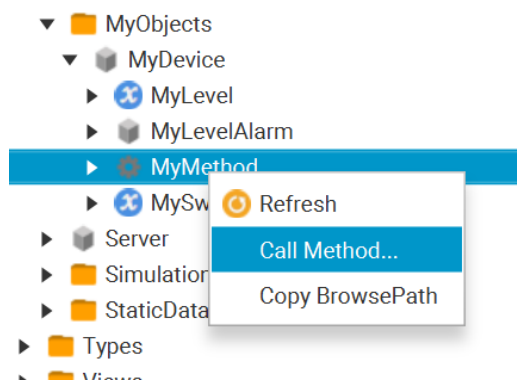


Figure 18. You can Call a Method from the Address Space Browser context menu.

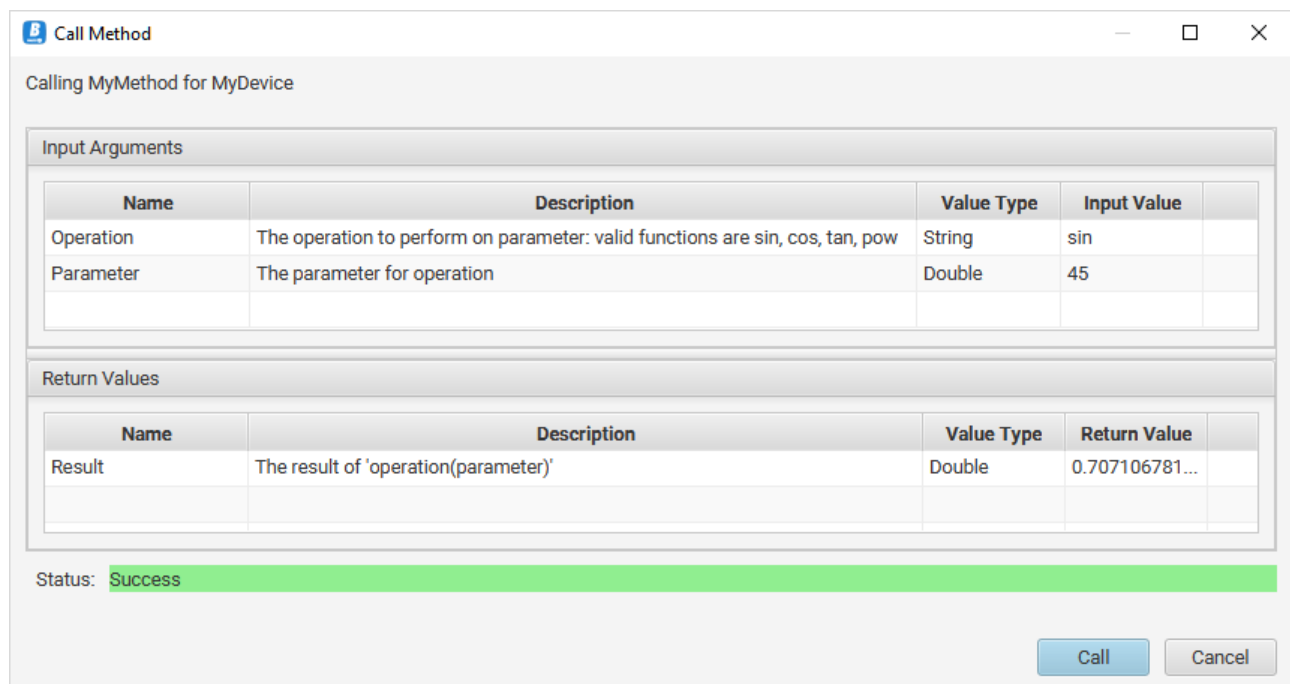


Figure 19. Calling a Method. MyMethod of the Prosystech OPC UA Simulation Server enables you to define an operation (sin/cos/tan/pow) and a parameter (in degrees for the trigonometric functions).

Attributes and References View

The *Attributes and References View* is the default view displayed on the right-hand side of the main window. It consists of two tables: one representing node attributes and another representing 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.

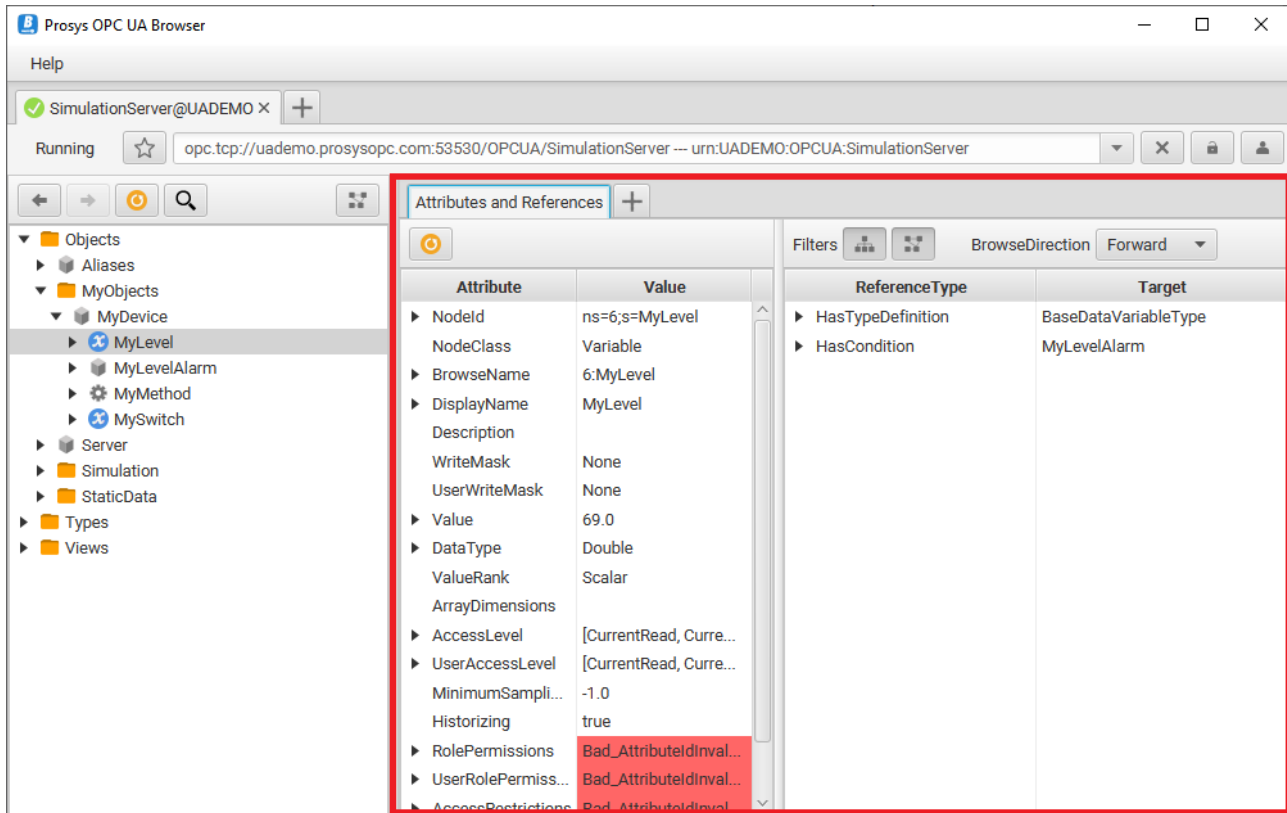


Figure 20. Attributes and References View showing the attributes and references of a Variable node.

The *Reference Table* shows the references of the node: *ReferenceType* represents the type of the reference, while *Target* represents the node 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 complete reference information of each reference if you hover the mouse over the table or expand the individual rows.

Data View

The Data View corresponds to an OPC UA subscription, which can monitor data changes in the OPC UA Server. You can add variables in a few different ways:

1. dragging variables from the Address Space Browser to the *Monitored Items Table* of the Data View
2. using the *Add Custom Node* action (chapter [Searching for a Node](#)) in the context menu of the *Monitored Items Table*
3. using the *Monitor* action in the context menu of the Address Space Browser to add variables to the Data View
4. dragging an Object node to add Variables below it, with optionally searching for Variables recursively for all Object nodes below the dragged Object node

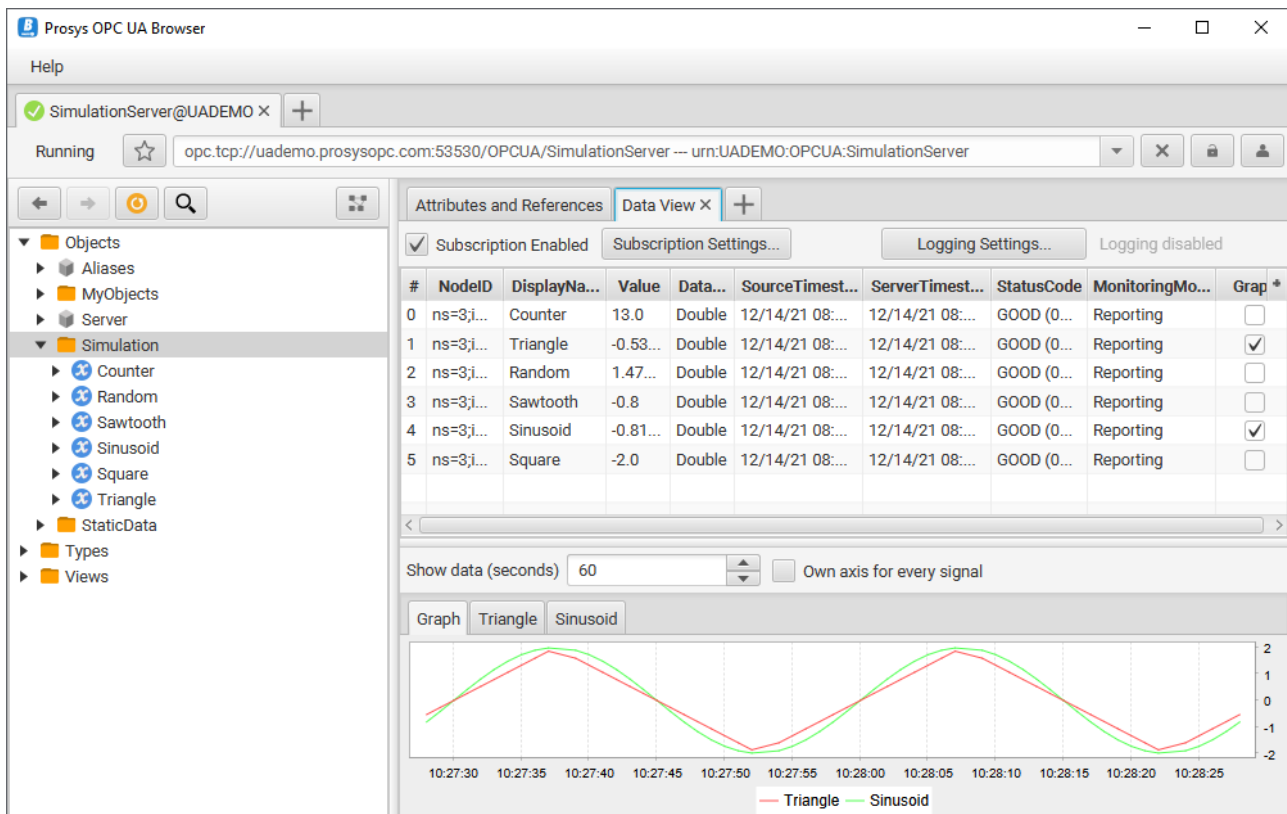


Figure 21. The Data View corresponds to an OPC UA subscription. It contains a *Monitored Items Table* and *Trend Graph*.

You can modify the *Subscription Enabled* parameter of the subscription at the top of the screen. You can also open the *Subscription Settings* dialog (see [Figure 22](#)), which enables a more advanced configuration of standard OPC UA parameters.

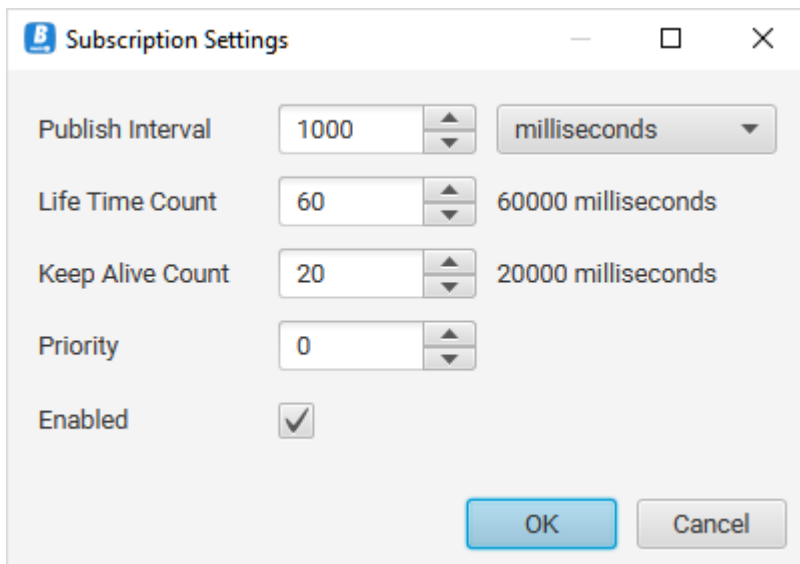


Figure 22. Subscription Settings dialog in Data View.

Publishing Interval defines how often the server should send notification messages. This is the maximum rate that the server may send notifications at. If there are no changes to send, the server sends an empty *Keep Alive Message* after several intervals defined by the *Keep Alive Count*. If the client does not receive anything from the server for a long time, it may determine that the subscription has timed out. Likewise, the server is monitoring the lifetime of the client application. If the client does not send any acknowledgements back to the server for the time corresponding to the *Life Time Count*, the server may stop the subscription and remove all the resources related to it. *Priority* may be used to define the relative importance of each subscription: higher numbers correspond to a higher priority. Finally, each subscription may be enabled or disabled without removing it from the server.

OPC UA Browser creates a Monitored Item for each variable that is added to the subscription. The following information is displayed for each Monitored Item (see [Figure 21](#)):

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 applied to a variable value by the data source. It should indicate the last change of the value or status code and be generated by the same physical clock.

ServerTimestamp is used to reflect when 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 the status is Bad, then Value may 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 displayed in a trend graph. The length of the time axis is 60 seconds by default, and it can be changed between 1 and 9999 seconds. By default, the variables share 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, remove the selected items from the subscription, or add custom nodes.

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

You can configure individual settings for each item from *Monitored Item Settings* (see [Figure 23](#)).

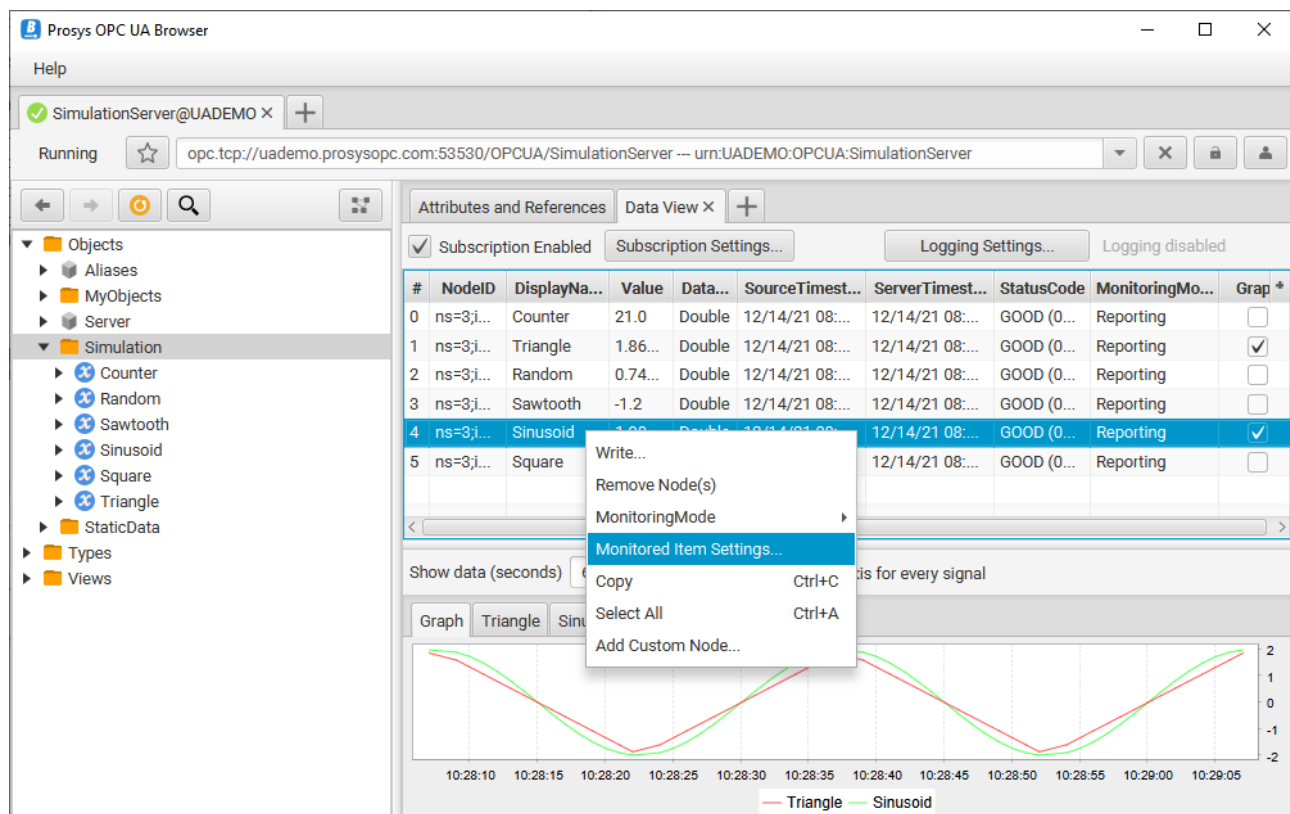
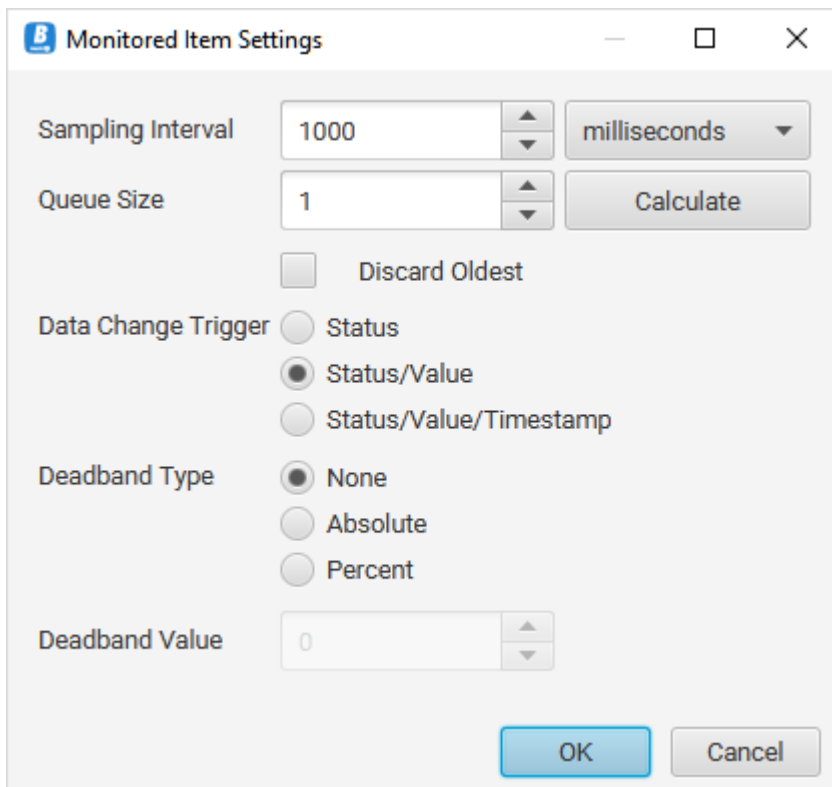


Figure 23. Context menu of the Monitored Items Table.

The settings for Monitored Items include the following (see [Figure 24](#)):



The dialog box titled "Monitored Item Settings" contains the following controls:

- Sampling Interval:** A text box with the value "1000" and a unit dropdown menu set to "milliseconds".
- Queue Size:** A text box with the value "1" and a "Calculate" button.
- Discard Oldest:** An unchecked checkbox.
- Data Change Trigger:** Three radio buttons: "Status", "Status/Value" (which is selected), and "Status/Value/Timestamp".
- Deadband Type:** Three radio buttons: "None" (which is selected), "Absolute", and "Percent".
- Deadband Value:** A text box with the value "0".
- Buttons:** "OK" and "Cancel" buttons at the bottom right.

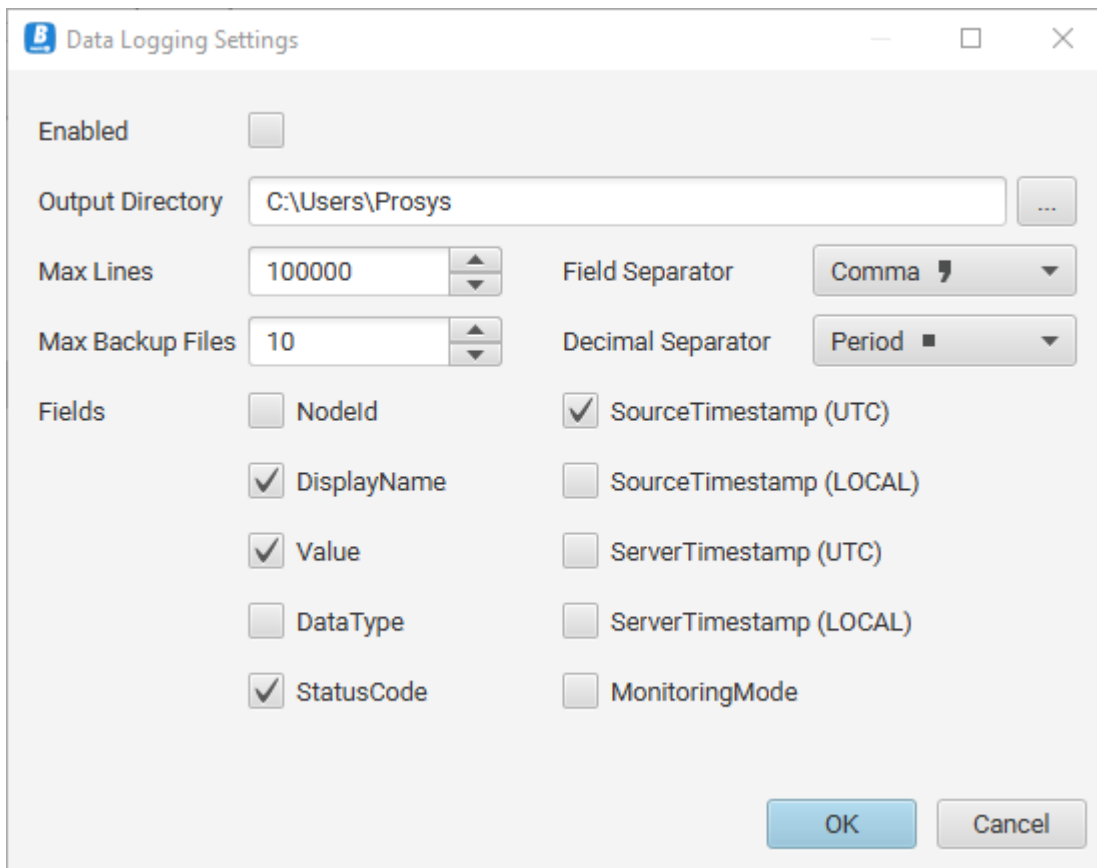
Figure 24. Monitored Item Settings dialog in Data View.

Sampling Interval is typically equal to the *Publishing Interval* of the subscription. However, you may also use lower values to request faster sampling in the server. In this case, you 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 valid number). Regardless, it may be better to configure more space in the queue. In case of communication problems, the server can record more samples, even if it cannot send the notification back to the client. If the sampling queue in the server runs out, it throws old samples out of the queue. If *Discard Oldest* is set, the oldest values are removed; otherwise, the new values are discarded, except the latest (i.e., current) value, which is always 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, the default sampling might be more suitable for you.

Data Change Trigger defines when the server should record and send new samples. By default, any change in *Status* or *Value* triggers 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 requests 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 a *EURange* property. *EURange* is defined as part of *AnalogItemType* variables).

Finally, you can enable data logging to CSV files from *Logging Settings* (see [Figure 25](#)).



The image shows a 'Data Logging Settings' dialog box. It has a title bar with a 'B' icon and the text 'Data Logging Settings'. The dialog contains several settings:

- Enabled:** A checkbox that is currently unchecked.
- Output Directory:** A text field containing 'C:\Users\Prosystech' and a browse button ('...').
- Max Lines:** A spin box set to '100000'.
- Max Backup Files:** A spin box set to '10'.
- Field Separator:** A dropdown menu set to 'Comma'.
- Decimal Separator:** A dropdown menu set to 'Period'.
- Fields:** A group box containing several checkboxes:
 - ☐ NodeId
 - ☒ DisplayName
 - ☒ Value
 - ☐ DataType
 - ☒ StatusCode
 - ☒ SourceTimestamp (UTC)
 - ☐ SourceTimestamp (LOCAL)
 - ☐ ServerTimestamp (UTC)
 - ☐ ServerTimestamp (LOCAL)
 - ☐ MonitoringMode

At the bottom right, there are 'OK' and 'Cancel' buttons.

Figure 25. Data Logging Settings Dialog in Data View.

Data Logger subscribes to value changes of Nodes that are in *Monitored Items Table*. You can log files to a specific folder by setting up the *Output Directory* path. Also, you can define *Max Lines* for a single file and *Max Backup Files* to control the number of files.

Fields are used for logging wanted data from Node. By default, it logs *Display Name*, *Value*, *SourceTimestamp (UTC)*, and *StatusCode*.

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 to request the trends from the server. The layout is illustrated in [Figure 26](#).

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* or added using the *Add Custom Node* button (chapter [Searching for a Node](#)) in the History View. You can also use the *Show History* action in the context menu of the Address Space Browser to add variables to the History View.

You can define the history interval in 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 26](#) and [Figure 27](#) respectively, for example.

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.

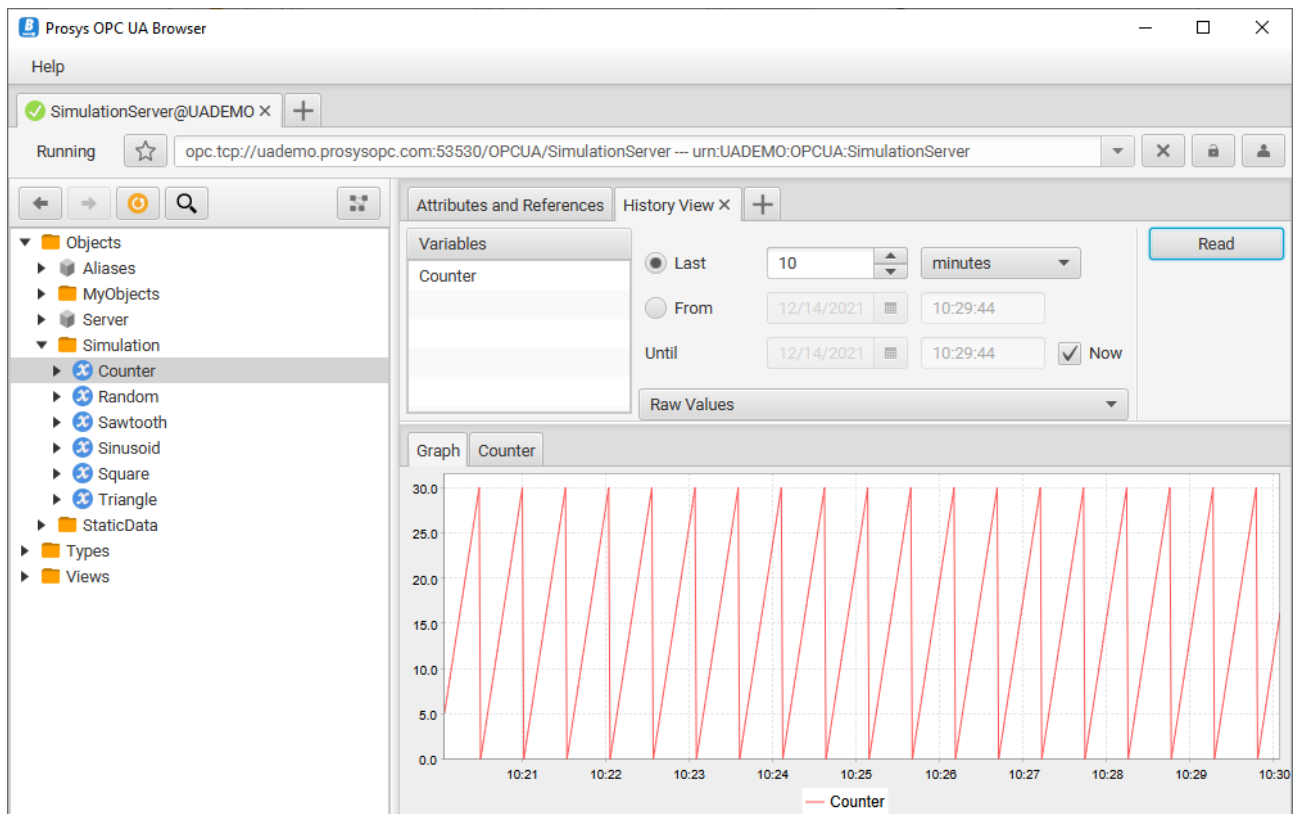


Figure 26. History View of the *MyLevel* variable for the Last 1 hour until Now.

By default, the client reads raw values from the server. Some servers also support reading processed history in the form of Aggregates. If you select an aggregate in a place of the **Raw Values** selection, an extra control for specifying the processing interval of the Aggregate appears. Additionally, some servers do not correctly display their supported Aggregates, or in some cases, servers acting as gateways do not have the information available. For this reason, there are two additional options as the last elements for the selection: **Standard Aggregate...** and **Custom Aggregate....** These can be used to open a separate dialog to select any Standard Aggregate defined in the OPC UA Specification or any NodeId via the Custom Aggregate.

The processing interval can also be configured from a dialog to accompany the aggregate function. Figure 27 shows an example of the minimum value request for every 5 seconds.

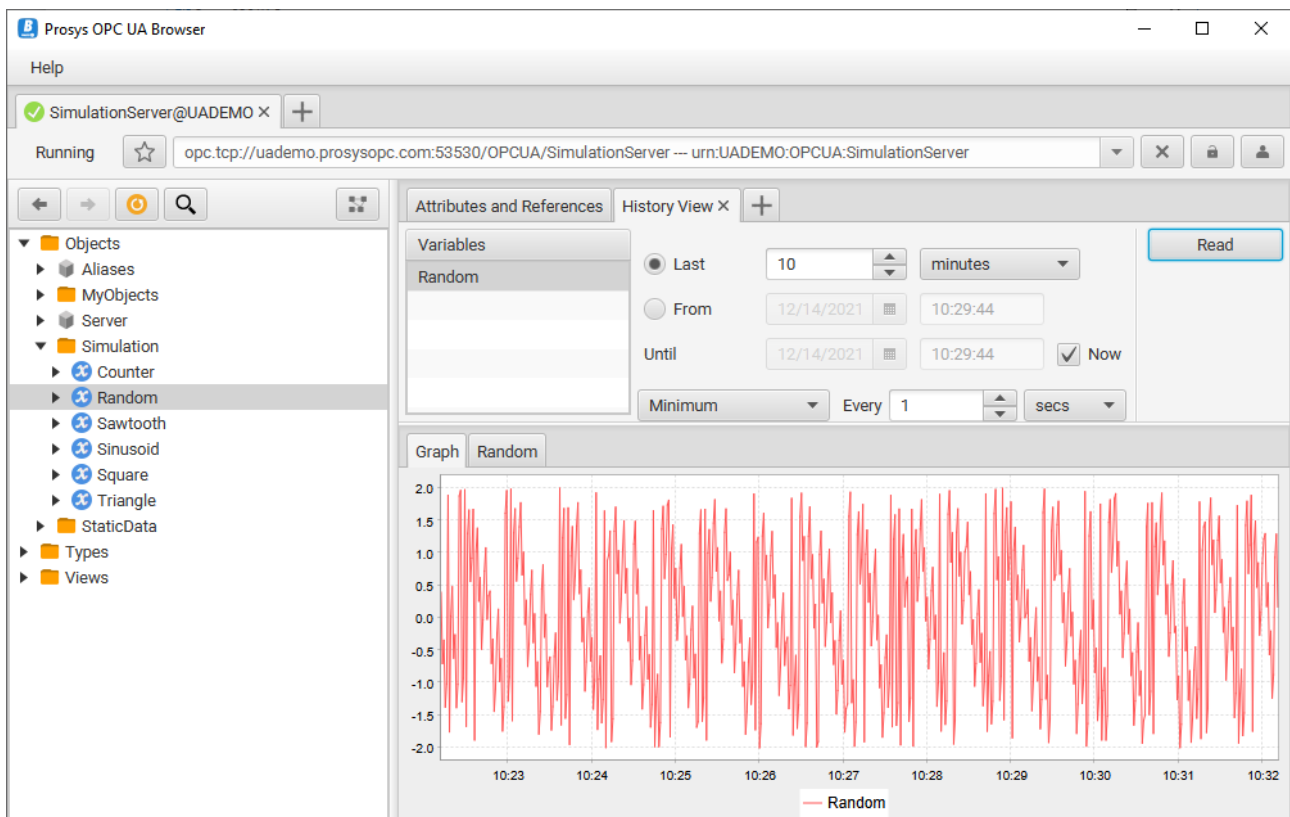


Figure 27. History View with a fixed period and data processing.

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. It is possible to monitor only one object at a time. To change *Monitored Object*, remove an active object from *Monitored Object*, then add a wanted object by dragging it from the Address Space Browser to *Monitored Object* or use the *Add Custom Node* action (chapter [Searching for a Node](#)) in the context menu of the Event Table.

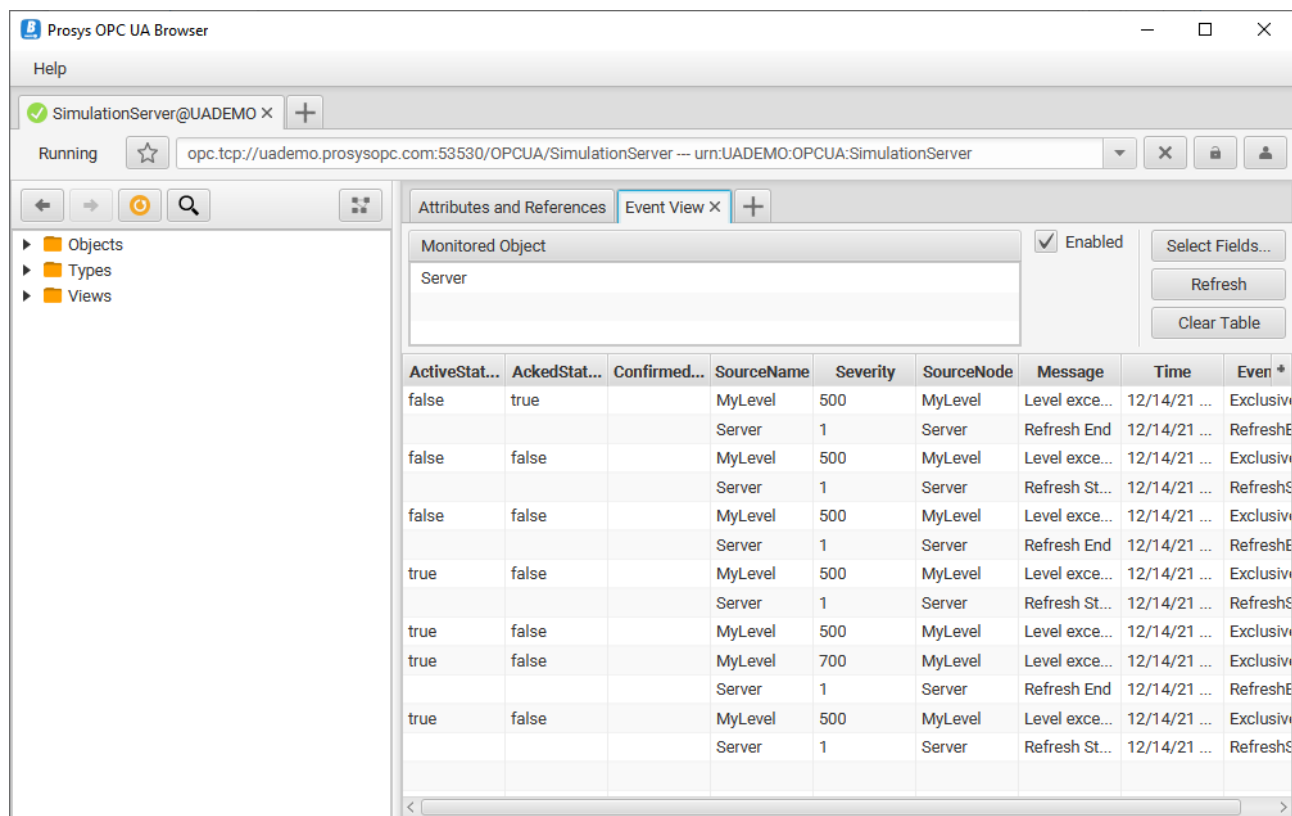


Figure 28. Event View with Server node

By 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.

Table 1. Five severity levels

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

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.

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 much information, and the important fields depend on your application and the events you expect to receive. The default fields are available for most event types.

Note that the fields are only applied to new events: the events already received from the server do not get any new data. Instead, the [Event History View](#) may be able to provide that for you.

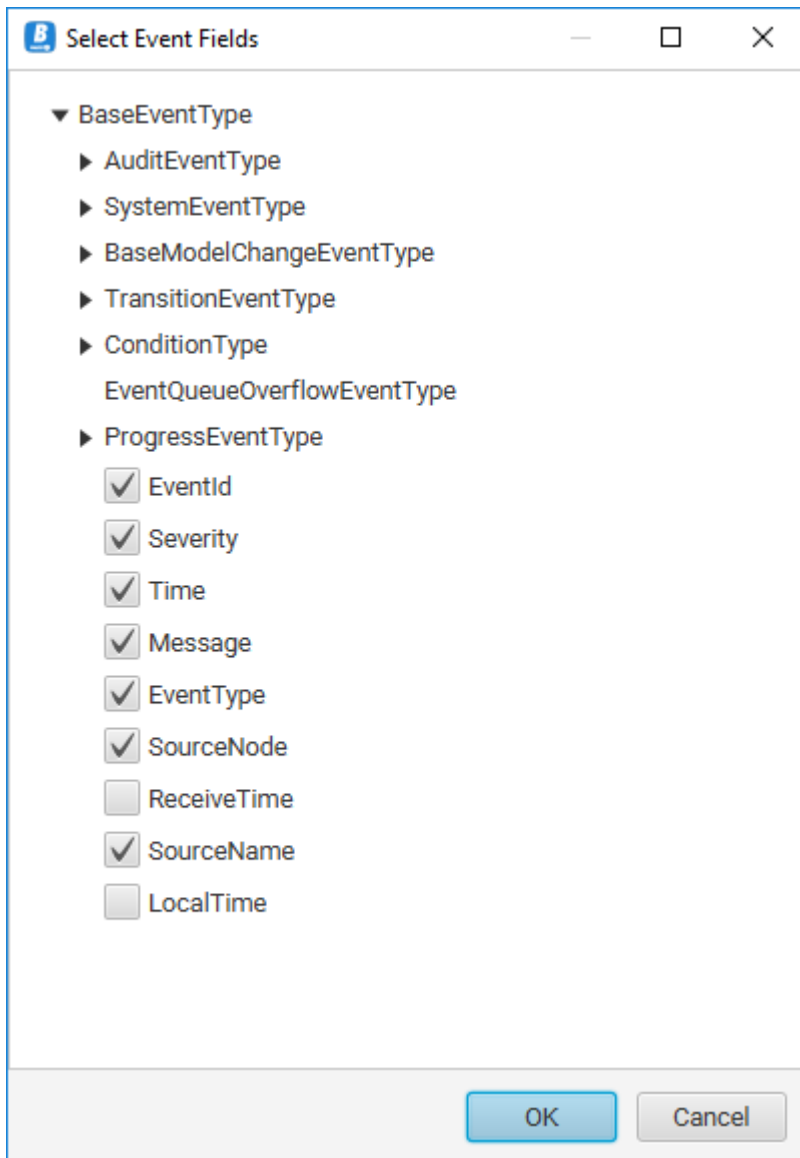


Figure 29. Selecting Event Fields.

Basic Events and Conditions

There are two main varieties of events: basic events and conditions (all descendants of *ConditionType*). The main difference is that conditions have a state: for example, alarms are active or inactive, unacknowledged or acknowledged, et cetera. The condition types are often visible in the address space.

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 sends an event, which can be considered a state change notification.

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

Condition Refresh

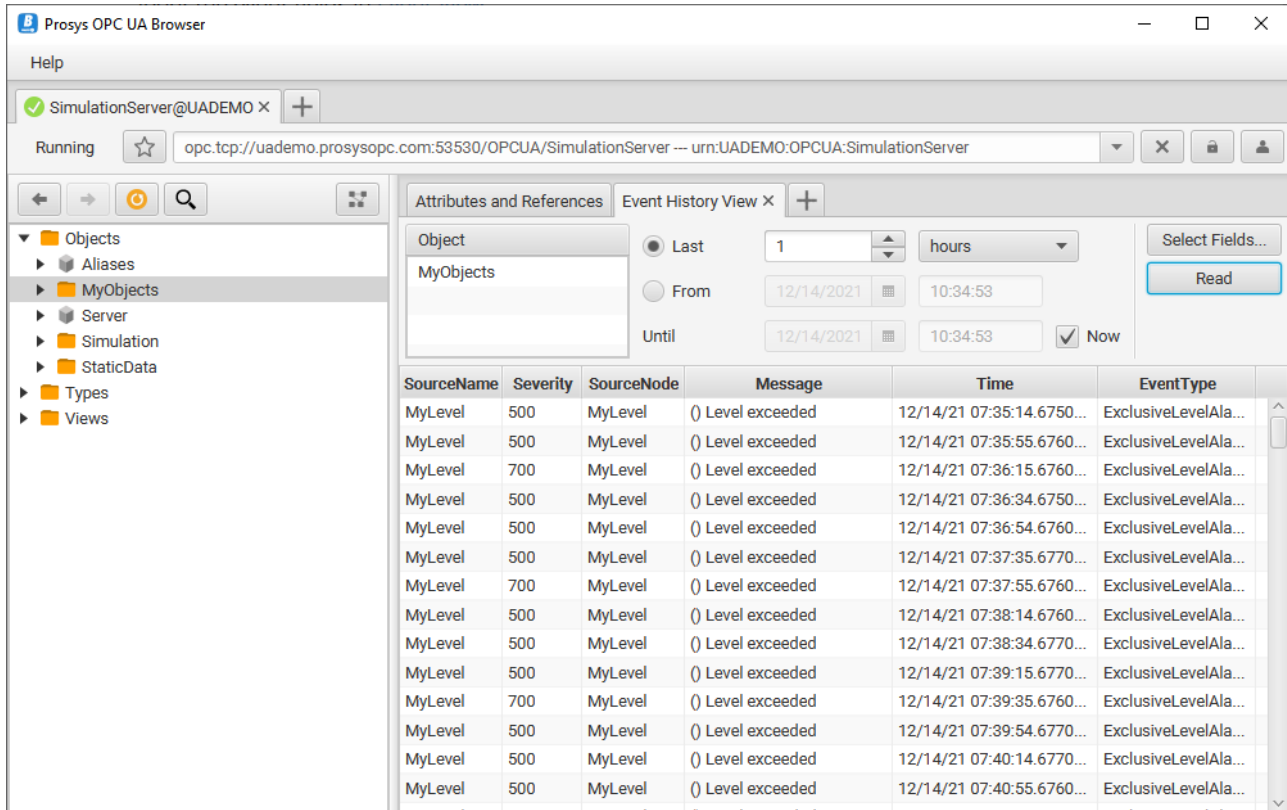
The Refresh button in the Event View makes a `ConditionRefresh` service call to the server. It requests the server to resend the current state of all active alarms to the client. Prosystech OPC UA Browser makes an automatic `ConditionRefresh` to the server when the Event View is initialized.

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 or use the *Add Custom Node* action (chapter [Searching for a Node](#)) in the context menu of the Event History Table. Then select a suitable time interval (see chapter 7. History View for details) and *Read* the history. The results are displayed in a table.

You may also use the *Select Fields...* option to define which information you want to read. See more about the event fields in [Event View](#).



The screenshot shows the Prosystech OPC UA Browser window. The top bar displays the connection URL: `opc.tcp://uademo.prosysopc.com:53530/OPCUA/SimulationServer`. The left pane shows a tree view with the following structure:

- Objects
 - Aliases
 - MyObjects (selected)
 - Server
 - Simulation
 - StaticData
 - Types
 - Views

The main pane is titled "Event History View" and contains a table with the following columns: SourceName, Severity, SourceNode, Message, Time, and EventType. The table displays a list of events for the "MyLevel" object, all with the message "Level exceeded".

SourceName	Severity	SourceNode	Message	Time	EventType
MyLevel	500	MyLevel	() Level exceeded	12/14/21 07:35:14.6750...	ExclusiveLevelAla...
MyLevel	500	MyLevel	() Level exceeded	12/14/21 07:35:55.6760...	ExclusiveLevelAla...
MyLevel	700	MyLevel	() Level exceeded	12/14/21 07:36:15.6760...	ExclusiveLevelAla...
MyLevel	500	MyLevel	() Level exceeded	12/14/21 07:36:34.6750...	ExclusiveLevelAla...
MyLevel	500	MyLevel	() Level exceeded	12/14/21 07:36:54.6760...	ExclusiveLevelAla...
MyLevel	500	MyLevel	() Level exceeded	12/14/21 07:37:35.6770...	ExclusiveLevelAla...
MyLevel	700	MyLevel	() Level exceeded	12/14/21 07:37:55.6760...	ExclusiveLevelAla...
MyLevel	500	MyLevel	() Level exceeded	12/14/21 07:38:14.6760...	ExclusiveLevelAla...
MyLevel	500	MyLevel	() Level exceeded	12/14/21 07:38:34.6770...	ExclusiveLevelAla...
MyLevel	500	MyLevel	() Level exceeded	12/14/21 07:39:15.6770...	ExclusiveLevelAla...
MyLevel	700	MyLevel	() Level exceeded	12/14/21 07:39:35.6760...	ExclusiveLevelAla...
MyLevel	500	MyLevel	() Level exceeded	12/14/21 07:39:54.6770...	ExclusiveLevelAla...
MyLevel	500	MyLevel	() Level exceeded	12/14/21 07:40:14.6770...	ExclusiveLevelAla...
MyLevel	500	MyLevel	() Level exceeded	12/14/21 07:40:55.6760...	ExclusiveLevelAla...

Figure 30. Requesting Event History for MyObjects of Prosystech OPC UA Simulation Server.

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 variable's value is displayed as an image. If the image changes (i.e., snapshots from a camera), you may see the changes as they appear. The Image View also makes a Subscription to the variable, similar to the Data View subscriptions.

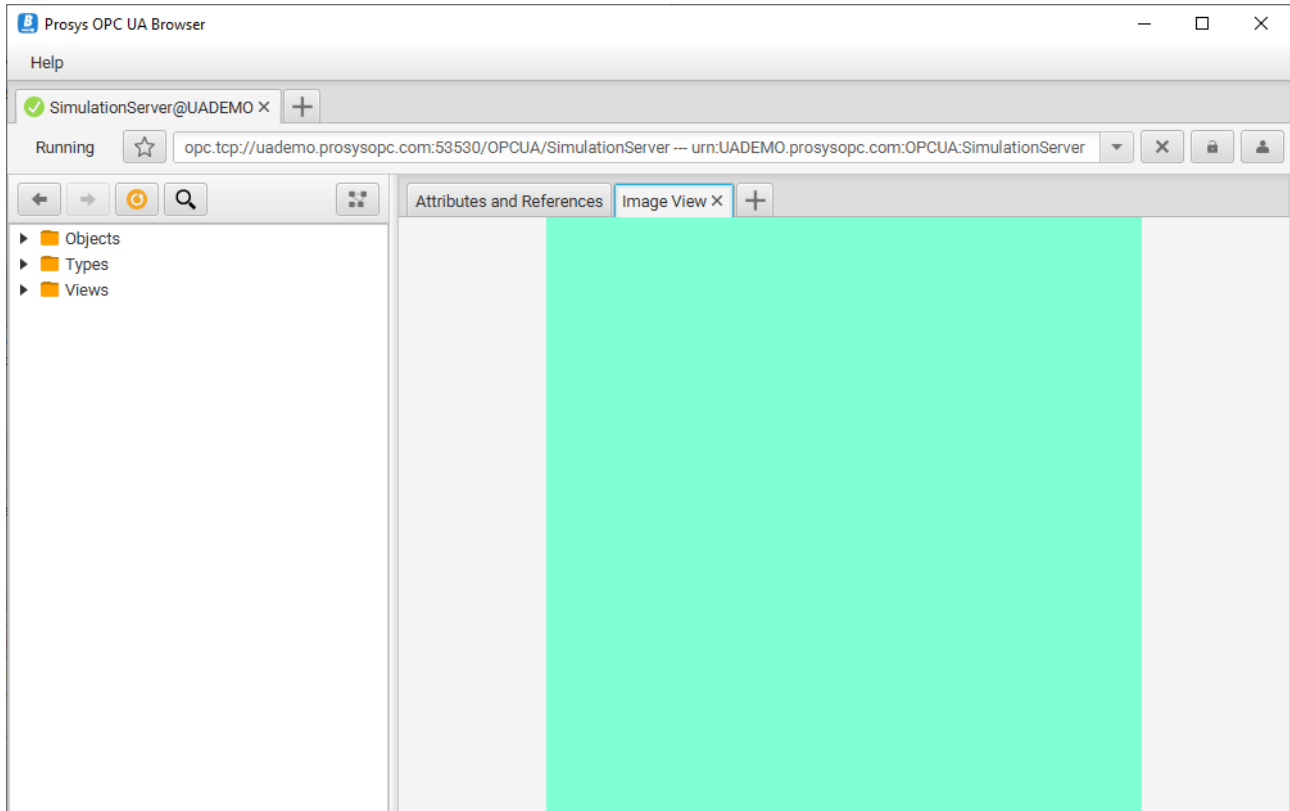


Figure 31. Image View.

PubSub Connection

In addition to an OPC UA Client, you can use Browser as an OPC UA PubSub Subscriber and connect to a PubSub Network. The starting screen gives you instructions on how to do that, as seen in [Figure 32](#).

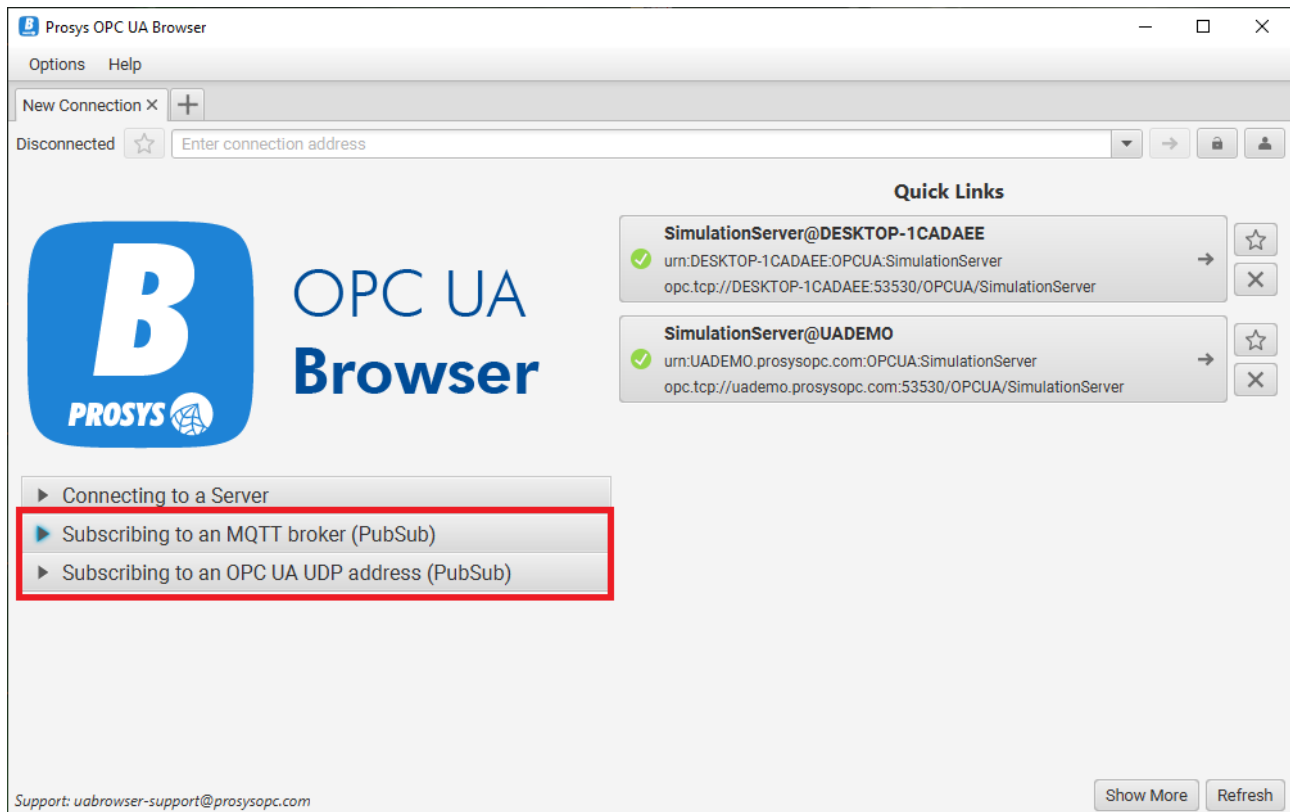


Figure 32. Instructions for the new PubSub Connection feature.

Connecting to a PubSub Network

To connect to a PubSub Network, you need to know whether the PubSub Network uses MQTT or UDP. If you are using Simulation Server as Publisher, you can check the configuration in its PubSub View. You can then connect to the Network accordingly.

When a successful connection has been established, the Browser will save your new connection settings to the *Quick Links* for easier access in future.

MQTT Network

For an MQTT Network, you will need to connect to the MQTT Broker, whose address is defined in the format

```
mqtt://<hostname>:<port>
```

or

```
mqtt://<hostname>:<port>
```

This should match the Connection Address used in your Publisher configuration (in Simulation Server, for example). Type it to the address bar at the top and press Enter or click the Connect button (→).

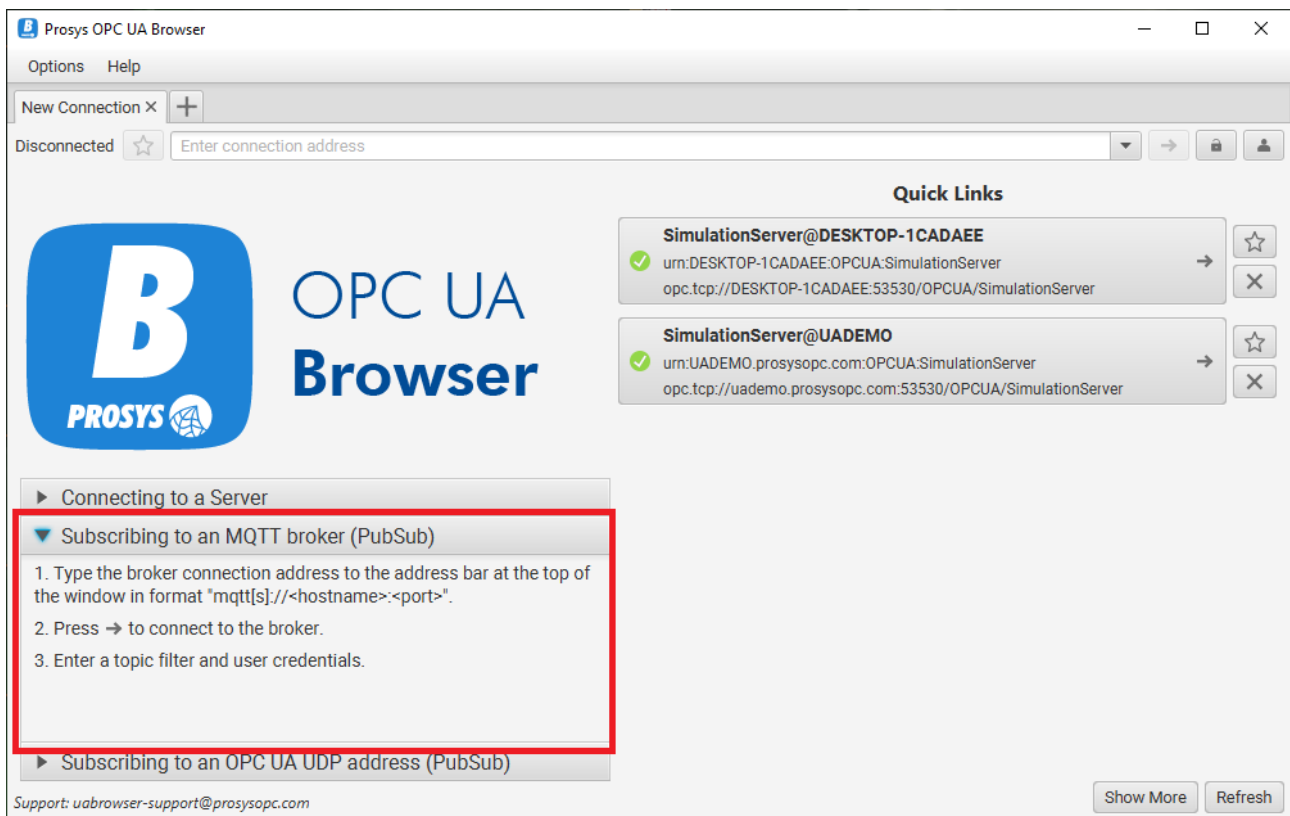


Figure 33. Step-by-step instructions for connecting to an MQTT PubSub Network.

If the Broker is available, you can then define the MQTT Topic Filter and the user credentials that you wish to use for the connection (see Figure 34). You can use the wildcard character "#" as the Topic Filter to subscribe to everything. Note however, that some brokers may not accept a subscription to everything, in which case you will need to know the topic tree that you need to subscribe to.

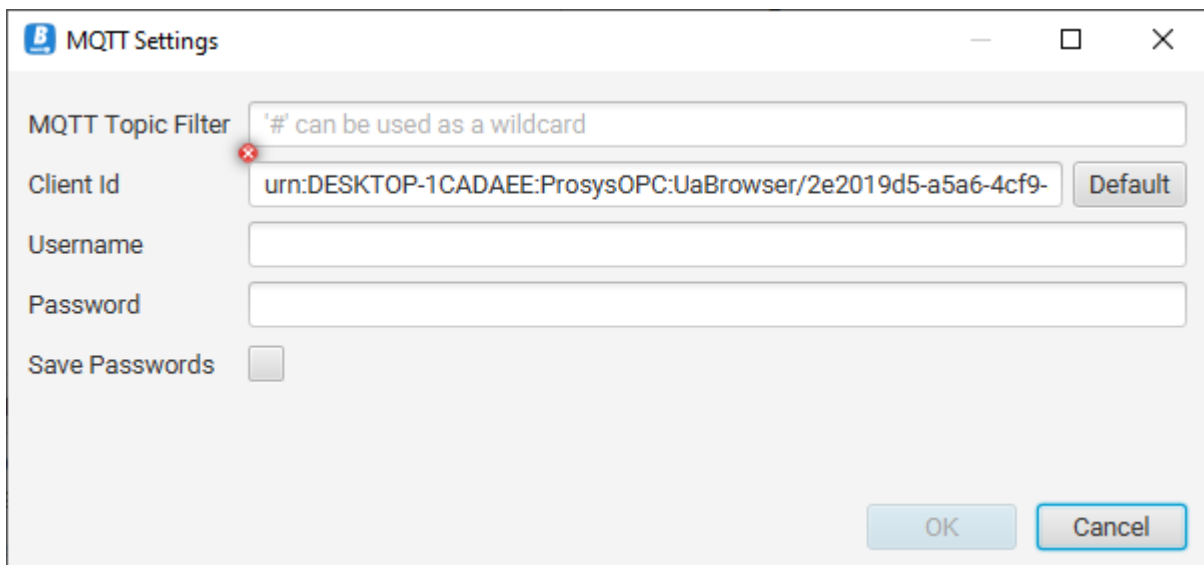


Figure 34. Dialog for the MQTT connection settings.

MQTT Topic Filter can follow the exact path of the topic that you wish to subscribe to. For example:

```
prosysopc/simserver/data
```

Or you can use wildcard characters to filter complete branches of the topic tree. For example:

```
prosysopc/#  
prosysopc+/data
```

'#' matches everything under the tree branch, whereas '+' matches with anything on that exact level in the tree.

Once you are ready, click **OK** to open the connection. If the MQTT Broker is available and it accepts your credentials, you will be connected. Otherwise, you should see an error that indicates why the connection failed.



If you choose **Save Passwords**, the password used for the connection is stored in the *conf.xml* in the application settings folder in encrypted format. However, you should be careful with the file and in case of doubt, you can choose not to save the password in the configuration.



MQTT is unencrypted communication and the password is also sent clear-text, so do not use sensitive passwords in your MQTT Broker configuration. Whenever possible, enable MQTTS in the broker instead. MQTTS uses an encrypted connection over TLS/SSL and the passwords are not visible to third parties. MQTTS also supports client authentication with X.509 certificates and there will be support for those in the Browser in a future version, too.

UDP Network

The Connection Address for OPC UA UDP can be either a multicast or a unicast address. In any case, it should match the address defined in the respective Publisher configuration (in Prosystech OPC UA Simulation Server, for example), in order to receive messages from them.

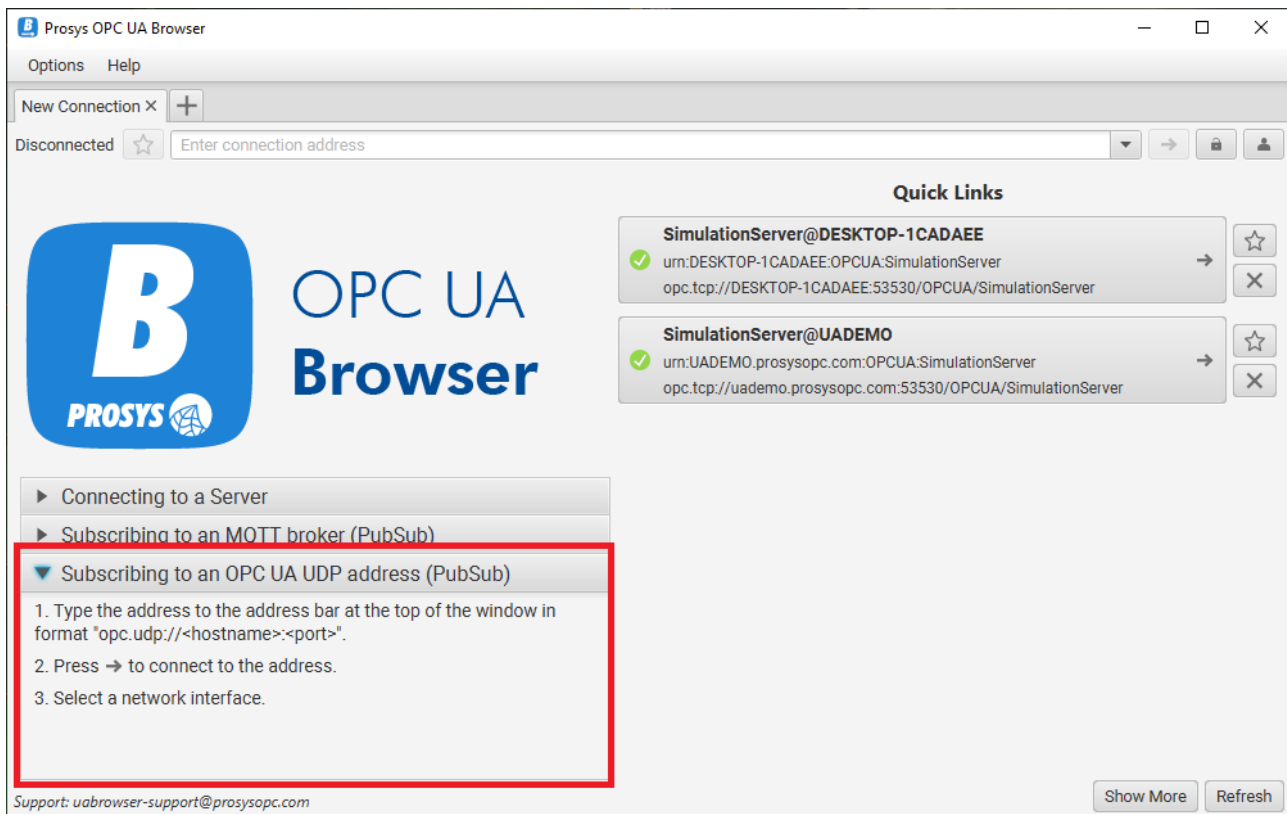


Figure 35. Step-by-step instructions for connecting to a UDP PubSub Network.

UDP Multicast

Multicast addresses are used to publish the messages to several subscribers in the network and you would typically want to use those. The Connection Address for multicast should be in the format

```
opc.udp://<address>[:<port>]
```

where address is one of the standard multicast addresses in the range 224.0.0.0 ~ 239.255.255.255 (224.0.0.0/4) as defined in [RFC 3171](#).

The [OPC UA specification](#) defines a special address, `opc.udp://224.0.2.14` for *OPC UA discovery purposes*, so avoid that for normal connections.

UDP Unicast

UDP unicast defines a point-to-point connection between two computers. In this case the connection address should be

```
opc.udp://localhost[:<port>]
```

The default port number for both multicast and unicast communication with OPC UA UDP is 4840.

Network Interface

You must also define the Network Interface that you want to listen to (see [Figure 36](#)).

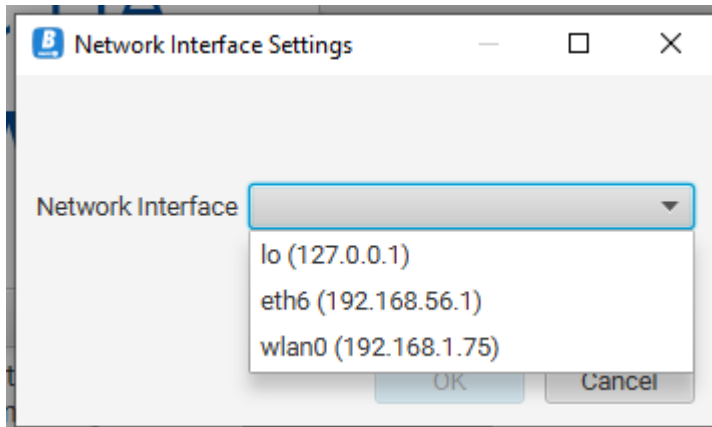


Figure 36. Dialog for setting the Network Interface for the UDP connection.



The Network Interface is currently not used for unicast addresses. Instead, Browser is always listening to all interfaces.

PubSub Connection View

The Browser will show your subscription data in a PubSub Connection View. From [Figure 37](#) you can see that the title of the tab is named after the Network type and broker (MQTT) or host (UDP). The address bar shows the Connection Address as well as the Topic you have subscribed to.



Figure 37. The title and address bar are named to describe the PubSub connection and subscription.

PubSub View

The new PubSub Connection View itself contains two tabs: [PubSub](#) and [Log](#). The *PubSub* View is divided into two parts: a list of DataSetWriters received on the left-hand side and two tabs for the data to be presented on the right-hand side.

The Browser contains a logic to determine if the messages contain Variable Data or Events, so it can separate them into the two Views on the right: Data View and Event View. [Figure 38](#) shows how the DataSetWriters are separated in both MQTT and UDP Networks. The different icons are used to identify them. If, however, the Browser guessed wrong, you can right-click the DataSetWriter on the list and select **Monitor as events** or **Monitor as data** to have it moved to the correct group.

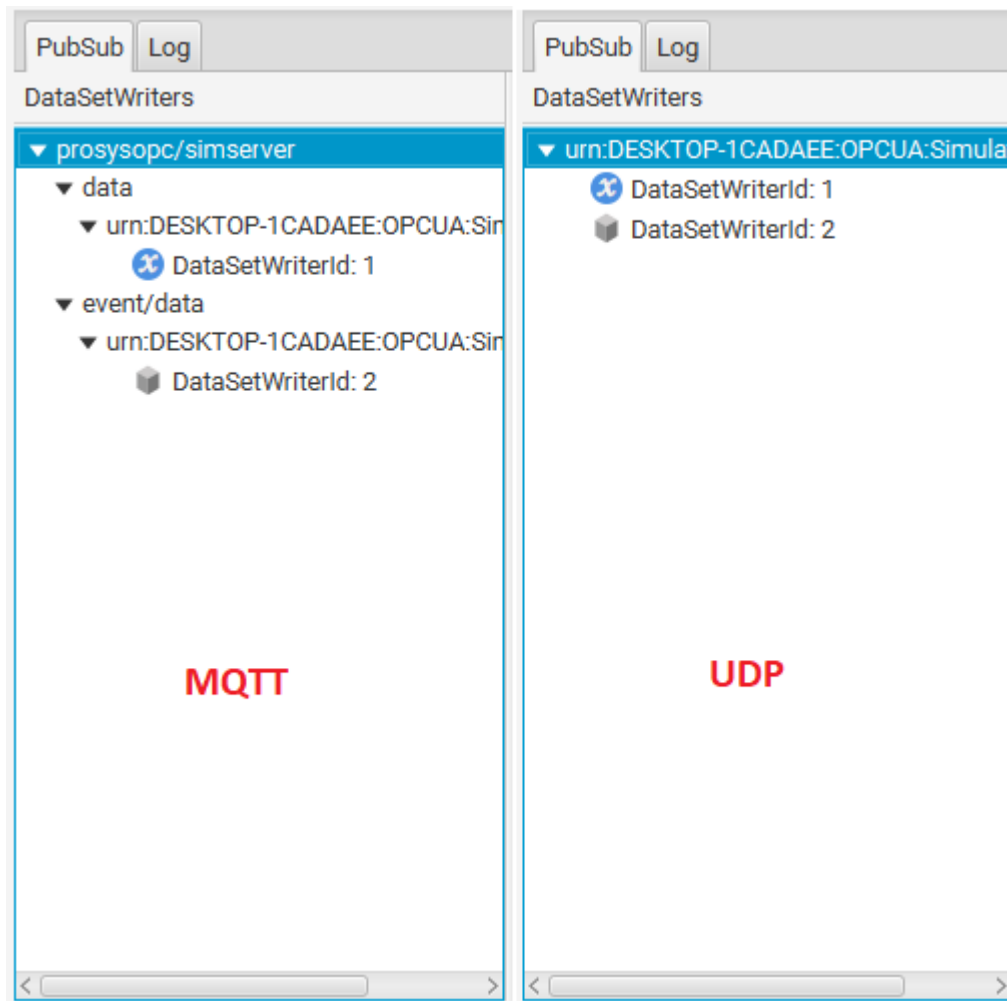


Figure 38. The list of DataSetWriters (MQTT on the left, UDP on the right).

In the MQTT case, the DataSetWriters are also classified into a hierarchy corresponding to the MQTT Topics that they are publishing to.

You can also use the DataSetWriter list to filter the content of the Data View and Event View. If you select only one DataSetWriter, it determines the view on the right, based on the writer type. If you use **Clear Selection** (on the right click menu) to select all or use the MQTT Topic Tree to filter a part of the tree, you can switch between the views and see both data and events that are received.

PubSub Data View

The *PubSub Data View* (Figure 39) represents variable data received by the subscriber in a similar way as *Data View* shows the values received by the Client/Server subscriptions. There is a separate row for each DataSet Field with their latest values and timestamps. If you wish to see how the values are changing over time, you can select the *Graph* option for them, in which case you will see a trend chart and a table of value changes for each.

Note that the actual data that can be shown depends on the configuration of the Publisher. In case you don't see any timestamps, for example, check that the Publisher is configured to send them. Also note that data is identified by Field Names, which are also configured in the Publisher.

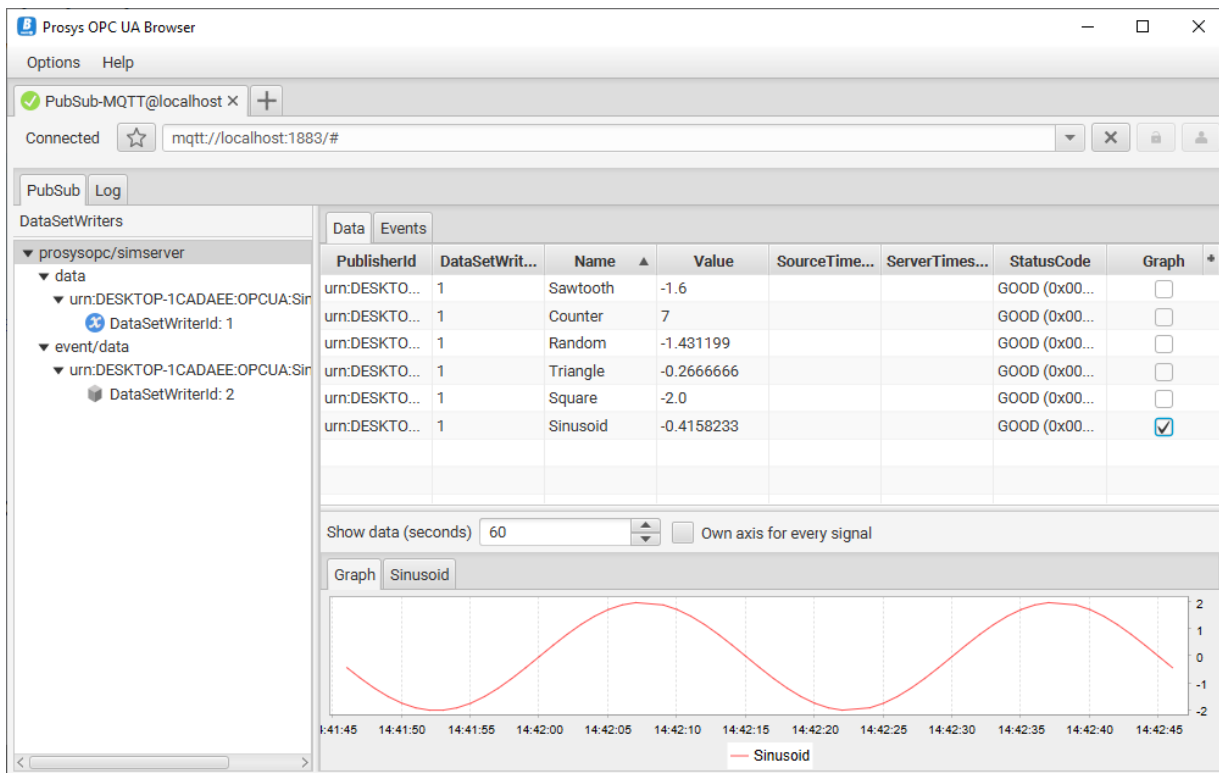


Figure 39. PubSub Data View inside PubSub View.

PubSub Event View

The *PubSub Event View* (Figure 40) represents the events received by the subscriber in the same way as events are represented in the Client/Server *Event View*. It logs all EventMessages including the Event Fields that are received from the selected DataSetWriters.

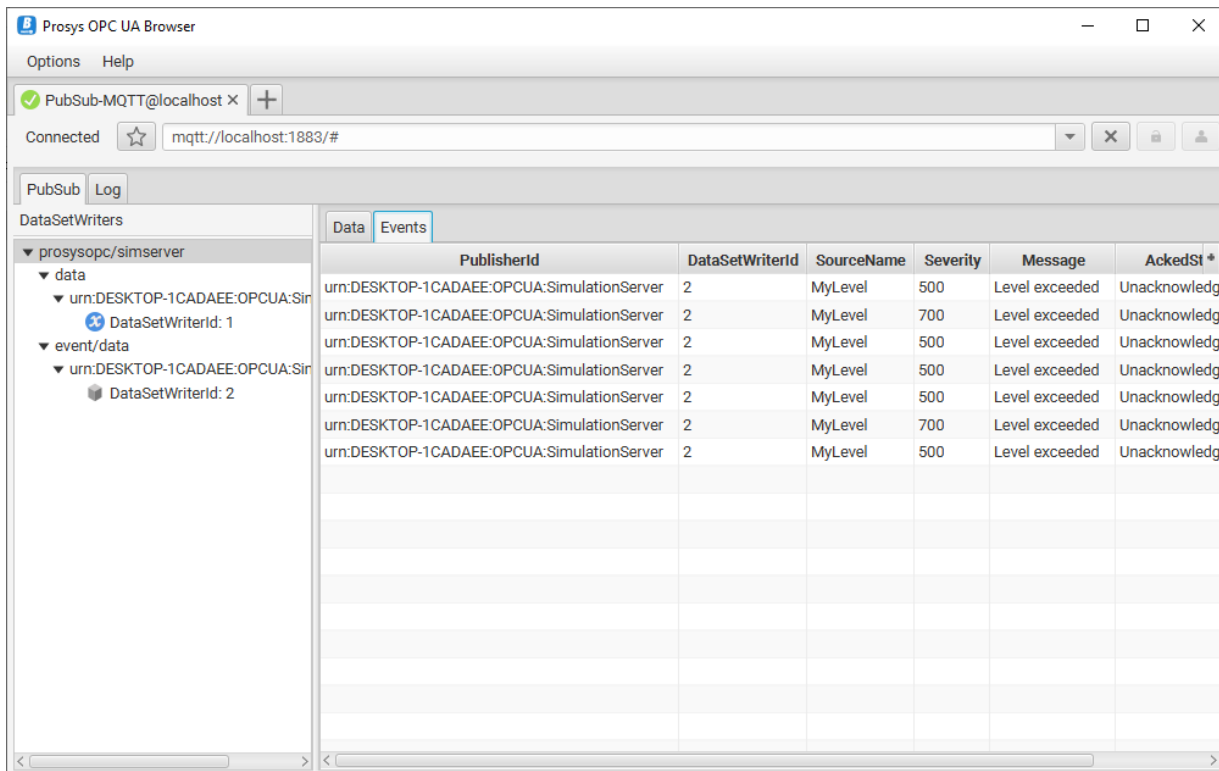


Figure 40. PubSub Event View listing Event Messages.

PubSub Log View

The *PubSub Log View* logs every message that is received from the PubSub Network. You can browse the messages and select those that you want to observe in more detail by clicking them. The whole message will be shown on the right side of the view. This view has a toolbar with two checkboxes and a button:

Active

This selection defines if the updating of the Log is active.

Keep selection in view

This selection prevents the selected message from scrolling from the view when new messages are logged.

Clear

This button clears the log.

Even though the log does not try to interpret the contents of the messages, we can show the topic tree for MQTT messages on the left. You can also use the topic tree to filter the messages that are displayed.

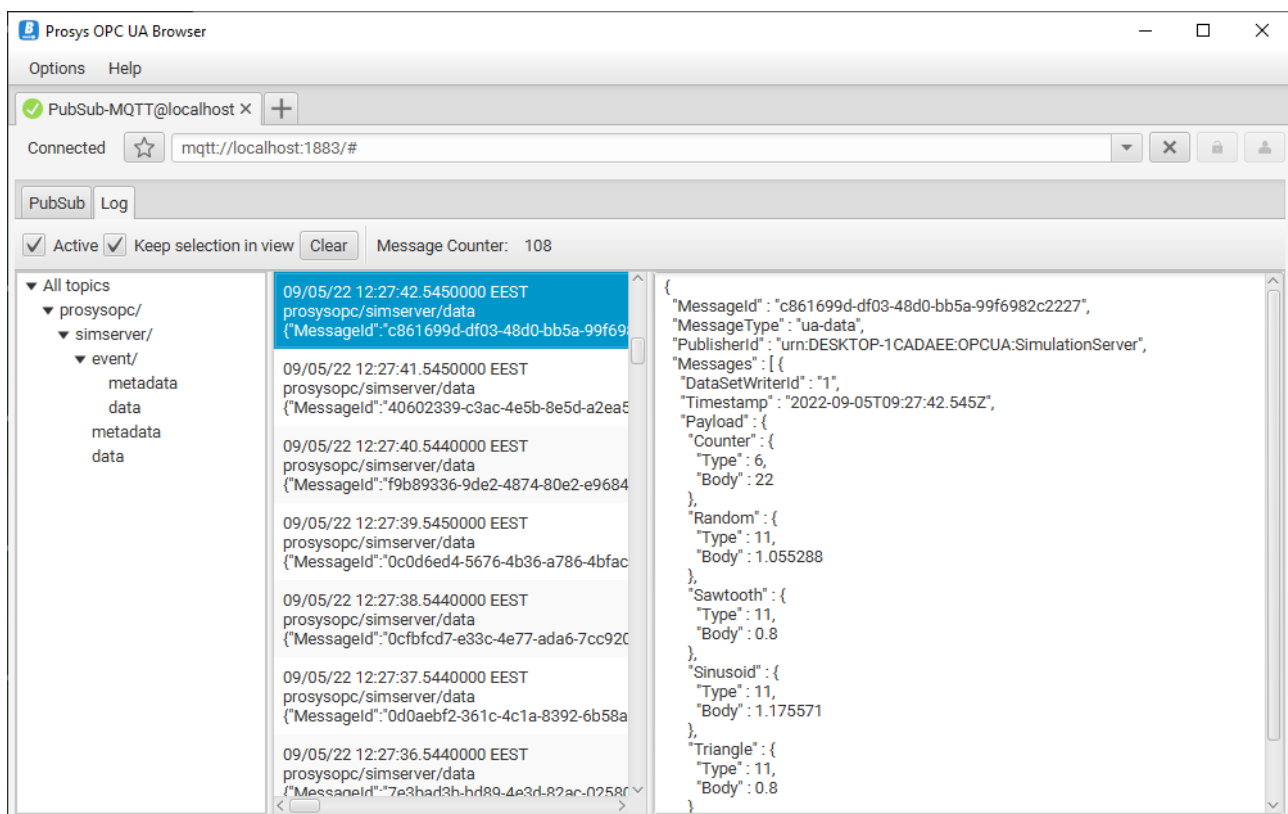


Figure 41. The PubSub Log View that lists all messages from the PubSub Network. MQTT lets you filter messages by Topics in this view.

If the messages are using JSON encoding, you can read the contents of them in the log. UADP messages are binary encoded, and we can only show them in their raw format.

MQTT messages may be either JSON or UADP, whereas UDP uses only UADP messages.

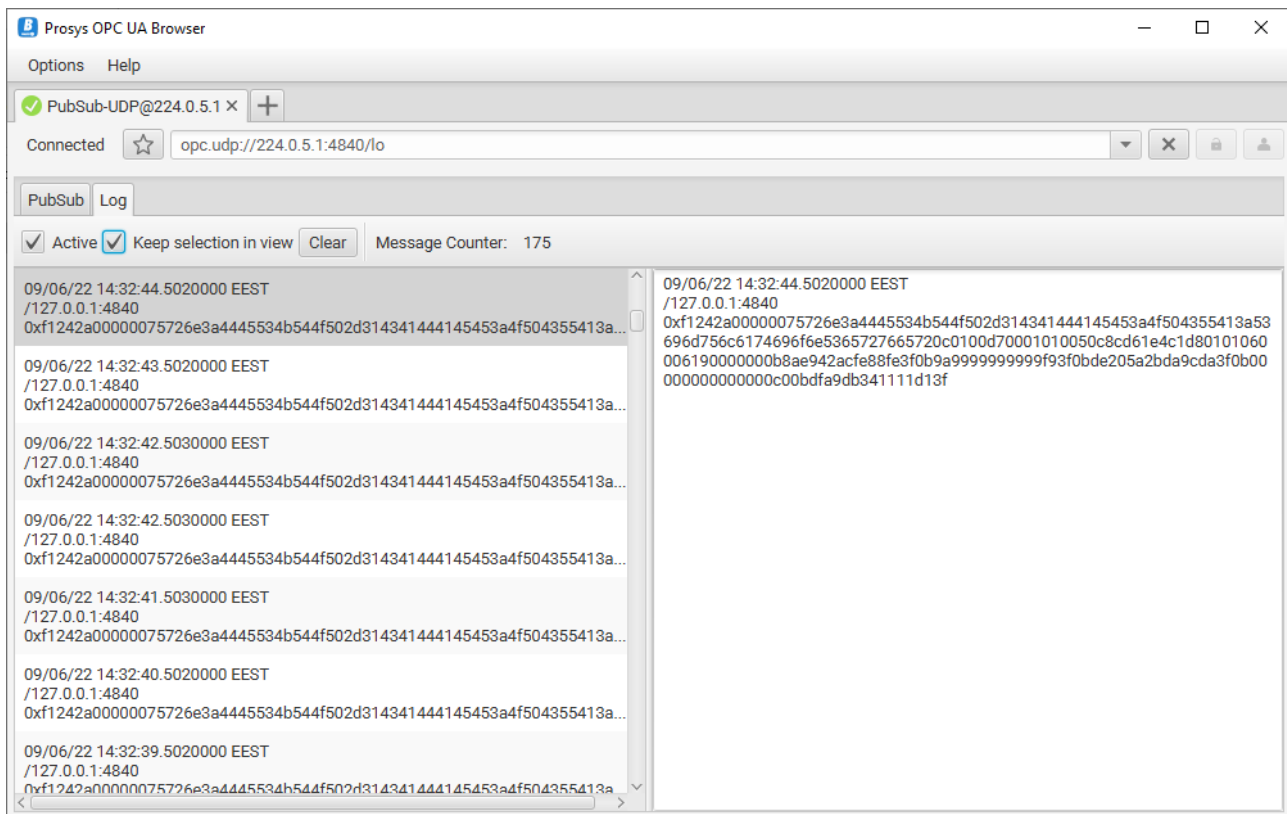


Figure 42. Log for messages from a UDP Network.

If the messages are decoded as OPC UA PubSub messages, you will find their respective data from the [PubSub View](#).

In addition to troubleshooting the details of OPC UA PubSub messages, the Log can be useful for monitoring other messages as well, especially with MQTT/JSON communication.