LabVIEW OPC UA Toolkit API Reference

2023-08-01

Contents

OPC UA Toolkit. 3

- Related Documentation. 3
- Error Codes. 4
- Protecting OPC UA Data Items. 11
- Connections between an OPC UA Server and an OPC UA Client. 12
- Tutorial: Creating an OPC UA Server and an OPC UA Client. 13
  - Part 1: Creating an OPC UA Server. 13
  - Part 2: Using an OPC UA Server. 18
  - Part 3: Establishing Connections between an OPC UA Server and an OPC UA Client. 26
  - Part 4: Using an OPC UA Client. 29
- OPC UA VIs. 39
  - OPC UA Common Operation Level Status Codes. 40
  - Create Certificate. 54
  - OPC UA Client VIs. 55
  - OPC UA Server VIs. 315
  - OPC UA Properties. 545

OPC UA Toolkit

November 2020, 376230C-01

The LabVIEW OPC UA Toolkit includes VIs for exchanging data between OPC UA servers and clients.

OPC Unified Architecture (UA) is an OPC Foundation specification for device connectivity. Unlike the classic OPC specification, which uses Microsoft DCOM/COM-based technology, OPC UA is platform-independent and able to connect servers and clients over various types of networks that have access to a common address space. OPC UA servers and clients use unique certificates to provide authentication and encryption capabilities to protect data when servers and clients communicate with each other.

Visit `ni.com/info` and enter the Info Code exxt3g to understand the importance of OPC UA and for a comparison between OPC UA and classic OPC. Refer to the OPC Foundation website at www.opcfoundation.org for more information about OPC UA specifications.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The **LabVIEW Help** highlights this topic in the **Contents** tab so you can navigate the related topics.

Refer to the `<National Instruments>\_Legal Information` directory for information about NI copyright, patents, trademarks, warranties, product warnings, and export compliance.

Related Documentation (OPC UA Toolkit)

The following documents contain information that might be helpful as you use the LabVIEW OPC UA Toolkit.

- **LabVIEW OPC UA Toolkit Readme**—Use this file to obtain introductory information about the OPC UA Toolkit, such as product overview, system requirements, installation instructions, and known issues. Open this readme by navigating to the `labview\readme` directory and opening `readme_OPCUA.html`.
- LabVIEW OPC UA Toolkit Example VIs—Refer to the `labview\examples\Data Communication\OPCUA` directory for example VIs that demonstrate common tasks using the OPC UA Toolkit. You also can access these VIs by selecting **Help»Find Examples** from the pull-down menu and selecting **Toolkits and Modules»OPC UA** in the NI Example Finder window.
- Additional LabVIEW documentation.

The following resources were used as references to produce the OPC UA Toolkit and they offer useful background information on the general concepts discussed in this documentation. These resources are provided for general informational purposes only and are not affiliated, sponsored, or endorsed by NI. The content of these resources is not a representation of, may not correspond to, and does not imply current or future functionality in the OPC UA Toolkit or any other NI product.

- **OPC Unified Architecture Specification**
- **List of Trade Facilitation Recommendations N° 20**

Error Codes (OPC UA Toolkit)

The OPC UA VIs can return the following error codes. Refer to the KnowledgeBase for more information about correcting errors in LabVIEW.

Code	Description
−356740	The input LV variant is NULL or invalid.
−356739	An OPC UA server is already created. The LabVIEW OPC UA Toolkit only supports one server application per process.
−356738	The data type or node type of the normal state node is invalid.
−356737	The data type or node type of the input node is invalid.

−356736	The data type or node type of the setpoint node is invalid.
−356735	The data type or node type of the source node is invalid.
−356734	The node ID of the notifier does not exist.
−356733	The node name is invalid. Possible reasons: the node name is empty or there is '@' in the node name.
−356732	The data type of the input node does not match the data type of the normal state node.
−356731	The ID of the input node is invalid.
−356730	The ID of the normal state node is invalid.
−356729	The ID of the setpoint node is invalid.
−356727	The license is not activated.
−356726	The limit values do not follow: highhighLimit > highLimit > lowLimit > lowlowLimit.
−356725	The historical data queue size must be larger than 0.
−356724	The condition type is not supported.
−356723	The condition type is unknown.
−356722	The event ID is invalid.
−356721	The parent node is not a notifier node.
−356720	The parent node is not a folder node.
−356719	The unregister refnum failed.
−356718	The register refnum failed.
−356717	The property value cannot be written after the OPC UA server starts.
−356716	Internal error. The property node ID is invalid.
−356715	Internal error. The condition node refnum is invalid.
−356714	The subscription ID refers to other types of subscription.
−356706	Failed to copy the certificate file to the certificate store because access is denied.

−356705	Failed to create certificate store because access is denied.
−356704	Failed to create certificate files because access is denied. Please check whether only one of the public/private key file exists and is not writable.
−356702	The OPC UA server has reached the maximum number of subscriptions allowed.
−356701	The port you specified for creating the server is occupied.
−356700	The OPC UA server is not running.
−356699	LabVIEW cannot find the certificate or private key file. Ensure the file exists and the specified file path is valid.
−356698	Unable to locate the host. Ensure the hostname is valid or use the IP address.
−356697	The OPC UA client failed to establish a secure connection to the OPC UA server. Either the OPC UA server does not trust the certificate file that the OPC UA client uses or the system time of the server machine is out of the valid time range of the certificate file that the OPC UA client uses. You must ensure that the OPC UA server trusts the certificate file that the OPC UA client uses. If you do not specify a certificate file for the OPC UA client to use, ensure that the OPC UA server trusts the default certificate file of the OPC UA client. You must also ensure that the system time of the server machine is within the valid time range of the certificate file. You can find the valid time range by opening the certificate file.
−356696	Value in the OPC UA server contains a syntax error.
−356695	The OPC UA server does not support the specified security policy.
−356694	The OPC UA server does not support the specified message mode.
−356688	The ID of the source node is invalid.

−356683	The requested operation is not supported or one or more parameter specified in the requested operation are not supported.
−356677	The node path refers to a node that does not exist in the server address space.
−356676	The OPC UA server cannot recognize the node in the request of the OPC UA client.
−356653	The status of the OPC UA server is uncertain.
−356650	The view version is not available or is not supported.
−356648	The view timestamp is not available or is not supported.
−356647	The view ID does not refer to a valid view node.
−356646	The user does not have permission to perform the requested operation.
−356645	The OPC UA client cannot recognize the response from the OPC UA server.
−356644	An unexpected error occurred.
−356643	The request cannot be processed because the request specified too many operations.
−356642	Invalid timestamp in the OPC UA client request.
−356641	The operation timed out.
−356640	The subscription ID is not valid. Ensure the ID refers to a subscription that you have already created.
−356639	The operation was cancelled because the application is shutting down.
−356638	The session cannot be used because you have not activated this session.
−356637	The session has been closed by the OPC UA client.
−356636	The OPC UA client failed to close a session. Verify that the session is active.
−356635	The OPC UA server does not support the requested service.
−356634	The server URI is not valid.

−356633	You cannot complete the operation before you connect the OPC UA client to the OPC UA server.
−356632	The OPC UA server has stopped and cannot process any requests.
−356631	An error occurred when the OPC UA server was verifying the security message of the OPC UA client.
−356630	The secure channel you specified is no longer valid.
−356629	An operating system resource is not available.
−356628	The header for the request is missing or invalid.
−356627	The OPC UA client cannot read the response message from the OPC UA server, because the message size exceeds the specified limits.
−356626	The OPC UA server cannot read the request message from the OPC UA client, because the message size exceeds the specified limits.
−356625	The OPC UA client has cancelled the request.
−356624	Not enough memory to complete this operation.
−356623	No operation executed because the OPC UA client passed a list of operations with no elements.
−356622	The nonce does not appear to be a random value or is not correct in length.
−356621	The timestamp is out of the range the OPC UA server allows.
−356620	The operation cannot complete. Possible reason is that the OPC UA server is shut down.
−356619	The OPC UA server failed to validate one or more parameters in the OPC UA client request.
−356618	The OPC UA client cannot connect to the OPC UA server because the OPC UA server rejected the security policy, username, or password.
−356617	The OPC UA client cannot connect to the OPC UA server because the security policy, username, or password is invalid.

−356616	The message encoding and decoding limits imposed by the stack have been exceeded.
−356615	Encoding halted because of invalid data in the serialized objects.
−356614	Decoding halted because of invalid data in the stream.
−356613	The extension object cannot be serialized or deserialized because the data type ID is not recognized.
−356612	A communication problem occurred. Ensure that the OPC UA server that you specify in the server endpoint URL is running and the network between the OPC UA server and the OPC UA client is connectable.
−356611	You cannot use the certificate for the requested operation.
−356610	The receiver does not trust the certificate of the sender.
−356609	The certificate URI in the OPC UA client request does not match the certificate URI that the OPC UA server expects.
−356608	The certificate has expired or is not yet valid.
−356607	The certificate has been revoked.
−356606	Unable to determine if the certificate has been revoked.
−356605	The certificate provided by the OPC UA client is not valid. Ensure the OPC UA server trusts the OPC UA client certificate.
−356603	An issuer certificate has expired or is not yet valid.
−356602	You cannot use the issuer certificate for the requested operation.
−356601	Unable to determine if the issuer certificate has been revoked.
−356600	The host name that you use to connect to an OPC UA server does not match a host name in the certificate.

−356530	The syntax of **node path** is incorrect.
−356529	You must select at least one security policy.
−356528	You cannot remove a certificate when the OPC UA server is running.
−356527	You cannot add a certificate when the OPC UA server is running.
−356526	The initial value of a node in the OPC UA server address space is uncertain.
−356525	The sensor from which the value is derived by the device or data source failed.
−356524	The device or data source that generates the value failed.
−356523	The node you want to delete does not exist in the address space.
−356522	You cannot add a node when the OPC UA server is running.
−356521	You cannot delete a node when the OPC UA server is running.
−356520	Unable to remove one or more certificate files.
−356519	Unable to add one or more certificate files.
−356517	**OPC UA client refnum in** is not valid.
−356516	**OPC UA server refnum in** is not valid.
−356515	The data type of the node does not match the data type of the value to write.
−356514	The data type of the node does not match the expected data type.
−356512	An error occurred when writing to the node.
−356511	An error occurred when reading the node.
−356510	The OPC UA server does not support this feature.
−356509	The node you want to add already exists in the address space.
−356508	Unable to add a node to the address space.
−356507	Unable to add a property to the address space.
−356506	Unable to add an item to the address space.

−356505	Unable to add a folder to the address space.
−356504	Unable to find a node in the address space.
−356503	This node does not support read and write operations.
−356502	Not enough memory to complete this operation.
−356500	An internal error occurred as a result of a programming or configuration error.
356501	Because historical access is not enabled, setting the queue size does not activate historical access.
356502	The severity level range is from 1 to 1000. Any value less than 1 will be coerced to 1 and any value greater than 1000 will be coerced to 1000.
356503	The high high severity level should be greater than the high severity level while the low low severity level should be less than the low severity level.

Protecting OPC UA Data Items (OPC UA Toolkit)

Compared with classic OPC, OPC Unified Architecture (UA) offers expanded security across servers and clients by utilizing private keys and public keys. A certificate file contains a pair of keys: a public key and a private key.

You can use the Add Trusted Clients VI to add a public key that an OPC UA client uses to an OPC UA server. You can use the Connect VI to specify the public key that an OPC UA server uses.

By default, the OPC UA server or client trusts the certificate file it uses. Ensure that the public key and private key have the same name and reside in the same folder. You can use an existing certificate file. If you do not specify a certificate file for the OPC UA server or client to use, LabVIEW creates a certificate file, `Default OPC UA`, to use when you create an OPC UA server or client at run time. You also can use the Create Certificate VI to create a certificate file.

The following table shows the location of the certificate file that LabVIEW creates at run time or when you use the Create Certificate VI.

Operating System	File Path to the Certificate File	Note
Windows	`C:\ProgramData\National Instruments\certstore\opcua\`	You can find public keys and private keys in the file path to the certificate file.
NI Linux Real-Time	`/var/local/natinst/certstore/opcua/`	You can find public keys and private keys in the file path to the certificate file.

Note Public key commonly appears with a `.der` file extension and private key has a `.pem` file extension.

Connections between an OPC UA Server and an OPC UA Client (OPC UA Toolkit)

The LabVIEW OPC UA Toolkit supports both non-secure connections and secure connections between an OPC UA server and an OPC UA client.

In a non-secure connection, the OPC UA server and OPC UA client do not need to trust each other. When the OPC UA server supports a non-secure connection, the OPC UA client can connect to the OPC UA server without security.

In a secure connection, the OPC UA server and OPC UA client must trust each other to protect the data exchange between the OPC UA server and OPC UA client. To establish a secure connection between an OPC UA server and an OPC UA client, you must complete the following tasks:

- Ensure that the OPC UA server supports a secure connection.
- Ensure that the OPC UA server trusts the certificate file that the OPC UA client uses.
- Ensure that the OPC UA client uses the secure message modes and corresponding security policies supported by the OPC UA server.

  A message mode specifies the encryption mode that the OPC UA client uses when the OPC UA server and OPC UA client send messages to each other. The OPC UA Toolkit supports three message mode options: None, Sign, and Sign and Encrypt. A security policy specifies how the OPC UA server and OPC UA client sign and encrypt messages. The OPC UA Toolkit supports Basic128Rsa15 and Basic256 security policy options.

- Ensure that the OPC UA client trusts the certificate file that the OPC UA server uses.

Creating an OPC UA Server and an OPC UA Client (OPC UA Toolkit)

This tutorial introduces you to the basics of creating an OPC UA server and an OPC UA client by using the OPC UA VIs. This tutorial consists of the following parts:

- Part 1: Creating an OPC UA Server
- Part 2: Using an OPC UA Server
- Part 3: Establishing Connections between an OPC UA Server and an OPC UA Client
- Part 4: Using an OPC UA Client

Next ➡
Creating an OPC UA Server

Part 1: Creating an OPC UA Server (OPC UA Toolkit)

You can use the OPC UA Server VIs to create an OPC UA server application that communicates with any OPC UA client. In Part 1 of this tutorial, you complete the following tasks:

- Creating an OPC UA server that supports only non-secure connections
- Creating an OPC UA server that supports only secure connections
- Creating an OPC UA server that supports both non-secure and secure connections

Creating an OPC UA Server That Supports Only Non-Secure Connections

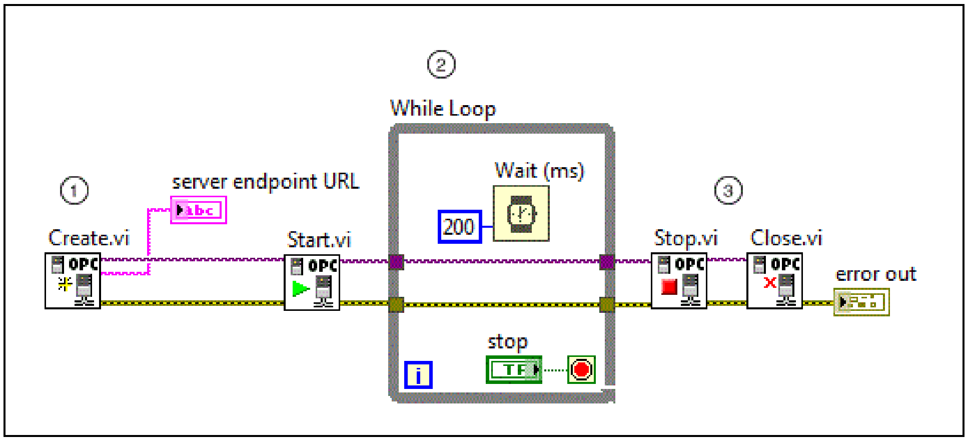
What to Use

Use the following objects to create an OPC UA server that supports only non-secure connections:

Create VI		Start VI		Stop VI		Close VI	
OPC		OPC		OPC		OPC	
+ Add		+ Add		+ Add		+ Add	

What to Do

Create the following block diagram to add an OPC UA server that supports only non-secure connections.



The following list describes important details about the previous diagram:

①	The **server endpoint URL** output of the Create VI returns the unique identifier of the OPC UA server.
②	(Recommended) The While Loop enables the VI to run continuously. Because an OPC UA server does not have any background services, LabVIEW destroys the OPC UA server when the VI stops. Therefore, you can create a structure, such as a While Loop, Event structure, or Time

	Delay, to enable the example VI to run continuously and avoid destroying the OPC UA server.
③	Use the Stop VI to stop the OPC UA server before you use the Close VI to close and destroy the OPC UA server.

Creating an OPC UA Server That Supports Only Secure Connections

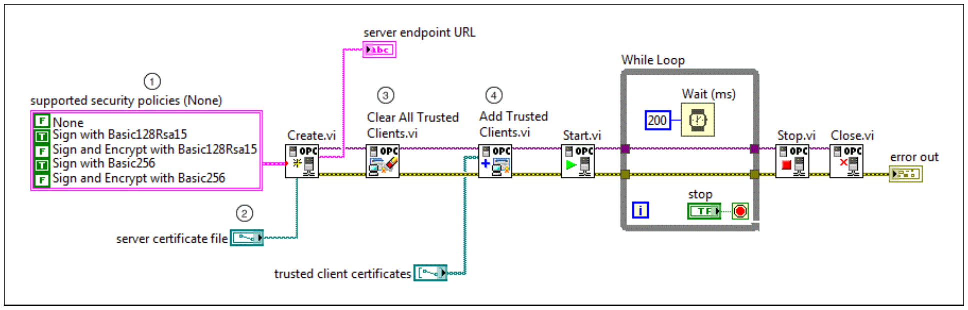
What to Use

Use the following objects to create an OPC UA server that supports only secure connections:

Create VI	Clear All Trusted Clients VI	Add Trusted Clients VI	Start VI	Stop VI	Close VI
OPC	OPC	OPC	OPC	OPC	OPC
+ Add	+ Add	+ Add	+ Add	+ Add	+ Add

What to Do

Create the following block diagram to add an OPC UA server that supports only secure connections.



The following list describes important details about the previous diagram:

①	The **supported security policies** constant of the Create VI specifies the message mode and security policies that the OPC UA server supports. To prevent the OPC UA client from establishing a non-secure connection to the OPC UA server, you must set **None** to FALSE. To establish secure connections between the OPC UA client and the OPC UA server, you must select one or multiple message modes and security policies.
②	The **server certificate file** control of the Create VI specifies the path or name of the public key. **Note** If the OPC UA server does not trust the certificate file that the OPC UA client uses, LabVIEW returns an error. Ensure that you add trusted OPC UA client certificates to an OPC UA server when you create an OPC UA server that supports only secure connections.
③	(Optional) The Clear All Trusted Clients VI clears the OPC UA client certificates that the OPC UA server trusts. After you start the OPC UA server, you cannot clear the trusted client certificates until the OPC UA server stops.
④	The Add Trusted Clients VI adds trusted OPC UA client certificates to the OPC UA server. After you start the OPC UA server, you cannot add trusted client certificates until the OPC UA server stops. The **trusted client certificates** control of the Add Trusted Clients VI specifies the file paths of the certificates that the OPC UA server trusts. You must manually copy the trusted client certificates from the OPC UA client

	machine to the OPC UA server machine, and then enter the path in **trusted client certificates**.

Creating an OPC UA Server That Supports Both Non-Secure and Secure Connections

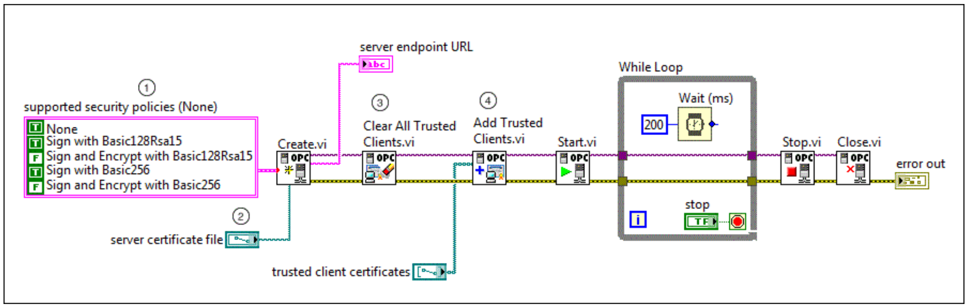
What to Use

Use the following objects to create an OPC UA server that supports both non-secure and secure connections:

Create VI	Clear All Trusted Clients VI	Add Trusted Clients VI	Start VI	Stop VI	Close VI
OPC	OPC	OPC	OPC	OPC	OPC
+ Add	+ Add	+ Add	+ Add	+ Add	+ Add

What to Do

Create the following block diagram to add an OPC UA server that supports both non-secure and secure connections.



The following list describes important details about the previous diagram:

①	The **supported security policies** constant of the Create VI specifies the message mode and

	security policies that the OPC UA server supports. To establish a non-secure connection between the OPC UA client and the OPC UA server, you must set **None** to TRUE. To establish secure connections between the OPC UA client and the OPC UA server, you must select one or multiple message modes and security policies.
②	The **server certificate file** control of the Create VI specifies the path or name of the public key. **Note** To establish a non-secure connection with a client, make sure that the domain names include the machine name of the server application.
③	(Optional) The Clear All Trusted Clients VI clears the OPC UA client certificates that the OPC UA server trusts. After you start the OPC UA server, you cannot clear the trusted client certificates until the OPC UA server stops.
④	The Add Trusted Clients VI adds trusted OPC UA client certificates to the OPC UA server. After you start the OPC UA server, you cannot add trusted client certificates until the OPC UA server stops. The **trusted client certificates** control of the Add Trusted Clients VI specifies the file paths of the certificates that the OPC UA server trusts. You must manually copy the trusted client certificates from the OPC UA client machine to the OPC UA server machine, and then enter the path in **trusted client certificates**.

⇦	**Previous** Creating an OPC UA Server and an OPC UA Client	**Next** Using an OPC UA Server	⇨

Part 2: Using an OPC UA Server (OPC UA Toolkit)

In Part 2 of this tutorial, you complete the following tasks:

- Constructing an address space
- Reading and writing the value of a node
- Registering and unregistering an OPC UA server with the UA Local Discovery Server (LDS)
- Reading, updating, and deleting the history data of a node
- Reading the history events of a condition
- Updating and deleting the history events of a condition

Constructing an Address Space

You can use the OPC UA Server VIs to construct an OPC UA server address space before you use the address space to store data. An address space consists of nodes that an OPC UA server allows an OPC UA client to browse. Nodes include folders, items, properties, notifiers, and conditions.

Constructing an address space includes the following tasks:

- Creating a folder in an address space
- Adding a notifier as a child to the folder
- Adding a source node to a notifier
- Adding a condition node to the notifier
- Adding an item as a child to the folder
- Adding a property to an item

What to Use

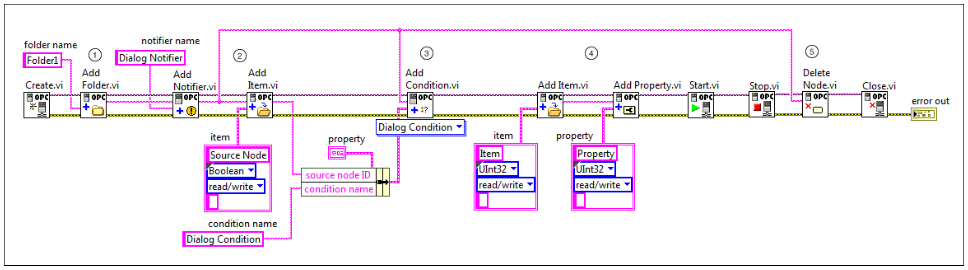
Use the following objects to create a folder in an address space:

Add Folder VI		Add Notifier VI		Add Item VI		Add Condition VI		Add Property VI	
OPC		OPC		OPC		OPC		OPC	

⊞ Add		⊞ Add		⊞ Add		⊞ Add		⊞ Add	

What to Do

Create the following block diagram to construct an address space for an OPC UA server.



The following list describes important details about the previous diagram:

①	The Add Folder VI creates a folder named `Folder1`.
②	The Add Notifier VI adds a notifier named `Dialog Notifier`. The Add Item VI adds an item named `Source Node` to `Dialog Notifier`.
③	The Add Condition VI adds a dialog condition node to the address space.
④	The Add Item VI creates an item named `Item` at the top level of the address space. The Add Property VI adds a property named `Property` to `Item`.
⑤	(Optional) The Delete Node VI deletes `Dialog Notifier` and its child source node `Source Node` and child condition node `Dialog Condition`. When you use the Delete Node VI to delete a node, you also delete all of the child nodes of this node. Before you delete the node, you can use the Stop VI to stop the OPC UA server. You can also add nodes to the OPC UA server or change the locations of nodes after the OPC UA server stops. After you delete the

	node, you can use the Start VI to restart the OPC UA server.

Reading and Writing the Value of a Node

You can use the OPC UA Server VIs to read the value of a node and write a value to a node on an OPC UA server.

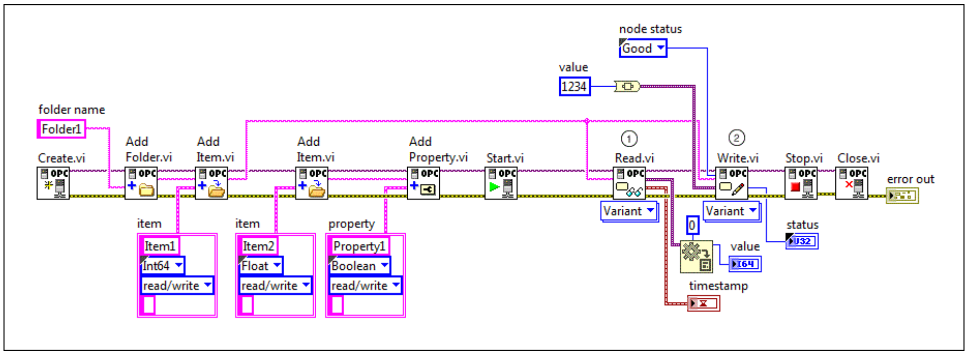
What to Use

Use the following objects to read the value of a node and write a value to a node on an OPC UA server:

Add Folder VI		Add Item VI		Add Property VI		Read VI		Write VI	
OPC		OPC		OPC		OPC		OPC	
+ Add		+ Add		+ Add		+ Add		+ Add	

What to Do

Create the following block diagram to read the value, timestamp, and status of a node, and writes a value and status to a node on an OPC UA server.



The following list describes important details about the previous diagram:

①	The Read VI reads the value, timestamp, and status of `Item1`.

②	The Write VI writes a value, `1234`, and status, `Good`, to the item that the Add Item VI created.

Registering and Unregistering an OPC UA Server with the LDS

You can use the OPC UA Server VIs to register an OPC UA server with the LDS and unregister the OPC UA server with the LDS.

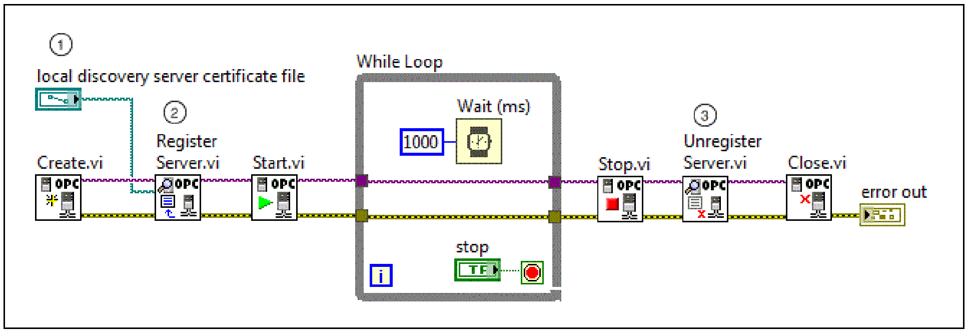
What to Use

Use the following objects to register an OPC UA server with the LDS and unregister the OPC UA server with the LDS:

Register Server VI		Unregister Server VI
+ Add		+ Add

What to Do

Create the following block diagram to register an OPC UA server with the LDS and unregisters the OPC UA server with the LDS.



The following list describes important details about the previous diagram:

①	**local discovery server certificate file** specifies the file path of the certificate file that the LDS uses. You must ensure that the OPC UA server and LDS trust each other. The server trusts a valid LDS certificate file. To ensure LDS trust the server, you can put the server

	certificate file to a folder that LDS public key infrastructure trusts.
②	The Register Server VI registers the OPC UA server with the LDS. You must use this VI before the server starts. This VI registers the server to LDS after the server starts.
③	The Unregister Server VI unregisters the OPC UA server with the LDS. This VI sends the offline registration to LDS after the server stops. To register the server with the LDS after the server starts again, you must use Register Server VI.

Reading, Updating, and Deleting History Data

You can use the Historical Access VIs of the OPC UA Server VIs to read, update, and delete history data. History data refers to the time series data stored for a node that supports historical access. Examples of history data include device data, calculated data, status information, dynamically changing system data, and diagnostic data.

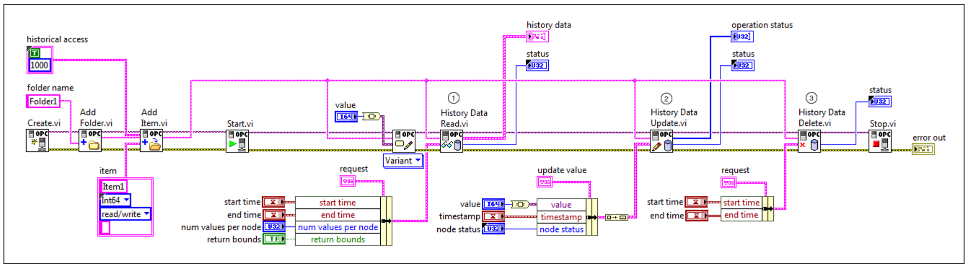
What to Use

Use the following objects to read, update, and delete history data:

Add Folder VI		Add Item VI		Write VI		History Data Read VI		History Data Update VI		History Data Delete VI	
OPC		OPC		OPC		OPC		OPC		OPC	
+ A dd		+ A dd		+ A dd		+ A dd		+ A dd		+ A dd	

What to Do

Create the following block diagram to read, update, and delete history data.



The following list describes important details about the previous diagram:

①	The History Data Read VI reads the history data of `Item1` within a time range.
②	The History Data Update VI updates the history data of `Item1` at a specific timestamp.
③	The History Data Delete VI deletes the history data of `Item1`.

Reading History Events

You can use the Historical Access VIs of the OPC UA Server VIs to read history events. History events refer to the time series events stored in some historical system. Examples of such data include notifications, system alarms, operator action events, and system triggers.

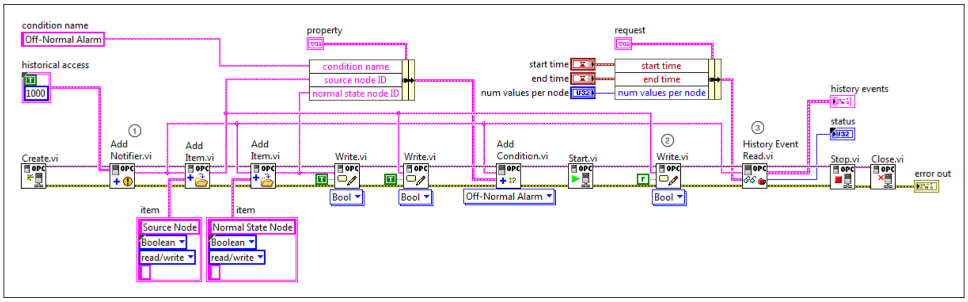
What to Use

Use the following objects to read history events:

Add Notifier VI		Add Item VI		Write VI		Add Condition VI		History Event Read VI	
OPC		OPC		OPC		OPC		OPC	
+ Ad d		+ Ad d		+ Ad d		+ Ad d		+ Ad d	

What to Do

Create the following block diagram to read history events.



The following list describes important details about the previous diagram:

①	The Add Notifier VI adds a notifier to the OPC UA server. To establish an event hierarchy, you can add a condition node and a source node as child nodes to a notifier.
②	The Write VI writes FALSE to the `Source Node`. An off-normal alarm is triggered because the value that the Write VI writes to the `Source Node` does not equal the value of the `Normal State Node`.
③	The History Event Read VI reads history events from the notifier.

Updating and Deleting History Events

You can use the Historical Access VIs of the OPC UA Server VIs to update and delete history events. History events refer to the time series events stored in some historical system. Examples of such data include notifications, system alarms, operator action events, and system triggers.

What to Use

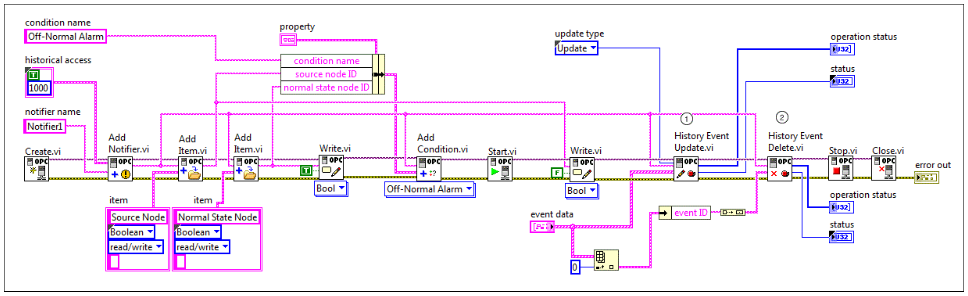
Use the following objects to update and delete history events:

Add Notifier VI		Add Item VI		Write VI		Add Condition VI		History Event Update VI		History Event Delete VI
OPC		OPC		OPC		OPC		OPC		OPC

➕ A dd	➕ A dd	➕ A dd	➕ A dd	➕ A dd	➕ A dd

What to Do

Create the following block diagram to update and delete history events.



The following list describes important details about the previous diagram:

①	The History Event Update VI updates history events from the notifier.
②	The History Event Delete VI deletes history events from the notifier.

⇦	**Previous** Creating an OPC UA Server	**Next** Establishing Connections between an OPC UA Server and an OPC UA Client	⇨

Part 3: Establishing Connections between an OPC UA Server and an OPC UA Client (OPC UA Toolkit)

You can use the OPC UA Client VIs to create an OPC UA client application that communicates with any OPC UA server. In Part 3 of this tutorial, you establish a non-secure connection and secure connection between an OPC UA server and an OPC UA client.

Establishing a Non-Secure Connection between an OPC UA Server and an OPC UA Client

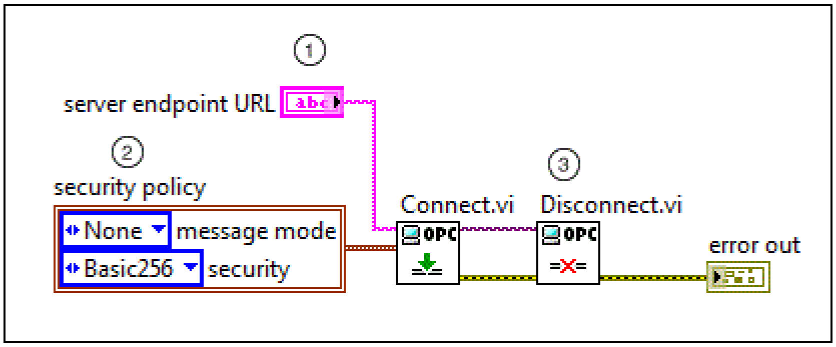
What to Use

Use the following objects to establish a non-secure connection between an OPC UA server and an OPC UA client.

Connect VI		Disconnect VI	
OPC		OPC	
＋ Add		＋ Add	

What to Do

Create the following block diagram to establish a non-secure connection between an OPC UA server and an OPC UA client.



The following list describes important details about the previous diagram:

①	The **server endpoint URL** control of the Connect VI specifies the endpoint URL to which the OPC UA client connects.
②	The **security policy** constant of the Connect VI specifies the message mode and security policy to use. To connect an OPC UA client to an OPC UA server in non-security mode, you must set **message mode** to **None**. An OPC UA client has to trust the OPC UA server certificate even for an insecure connection.

③	The Disconnect VI disconnects the OPC UA client from the OPC UA server.

Establishing a Secure Connection between an OPC UA Server and an OPC UA Client

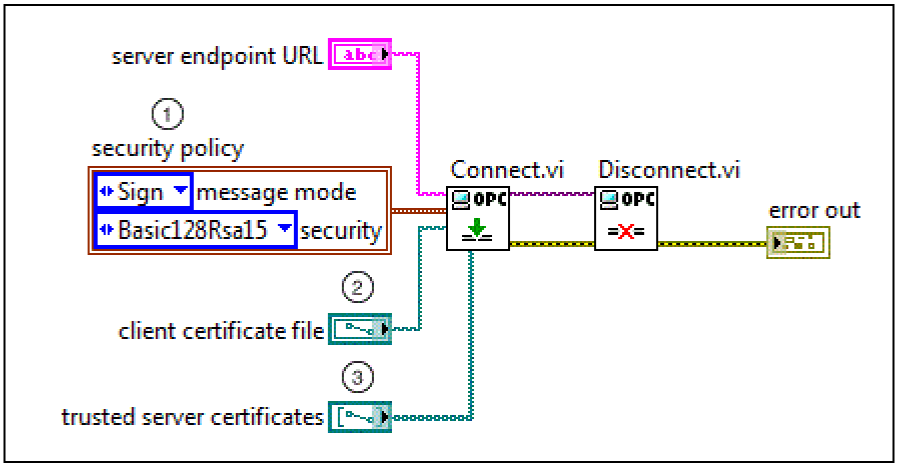
What to Use

Use the following objects to establish a secure connection between an OPC UA server and an OPC UA client.

Connect VI		Disconnect VI	
OPC		OPC	
➕ Add		➕ Add	

What to Do

Create the following block diagram to establish a secure connection between an OPC UA server and an OPC UA client.



The following list describes important details about the previous diagram:

①	The **security policy** constant of the Connect VI specifies the message mode and security policy to use. To establish secure connections between an OPC UA server and an OPC UA client, you must set **message mode** to either **Sign** or **Sign and Encrypt** depending on the message

	mode that the OPC UA server supports. You must also specify a security policy that the OPC UA server supports.
②	The **client certificate file** control of the Connect VI specifies the path or name of the public key. If you do not specify the **client certificate file**, the Connect VI generates and uses the default certificate file.
③	The **trusted server certificates** control of the Connect VI specifies the file paths of the certificates that the OPC UA client trusts. You must manually copy the trusted server certificates from the OPC UA server machine to the OPC UA client machine, and then enter the path in **trusted server certificates**.

Note If the OPC UA server does not trust the certificate file that the OPC UA client uses, LabVIEW returns an error. Ensure that you add trusted OPC UA client certificates to an OPC UA server when you create an OPC UA server that supports only secure connections.

⇦	**Previous** Using an OPC UA Server	**Next** Using an OPC UA Client	⇨

Part 4: Using an OPC UA Client (OPC UA Toolkit)

In Part 4 of this tutorial, you learn to use an OPC UA client to complete the following tasks:

- Browsing a node
- Getting the attributes of a node
- Reading and writing the value of a node
- Reading history data and events
- Creating a data subscription
- Creating an event subscription

- Responding to an acknowledgeable condition
- Shelving and unshelving an alarm

Browsing a Node

An OPC UA server uses an address space to store data. The address space consists of nodes that include folders, items, properties, notifiers, and conditions. A node ID describes the ID of a node in the address space of the OPC UA server. You can use the OPC UA Client VIs to browse a node and get the ID, name, and type of all the child nodes of this node.

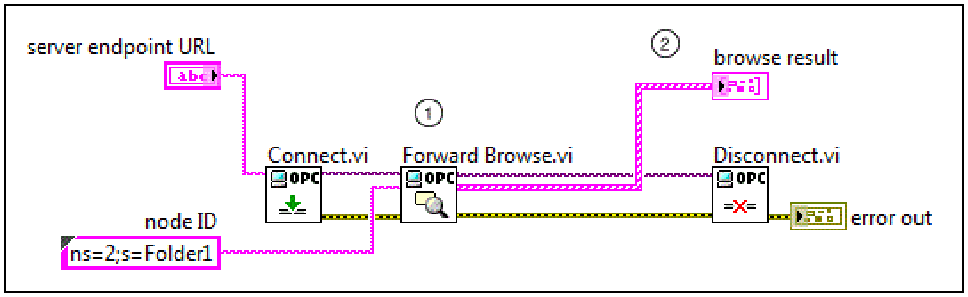
What to Use

Use the following object to browse a node:

Forward Browse VI	
OPC	
+ Add	

What to Do

Create the following block diagram to browse a node of an address space to get the reference type, ID, name, class, and type of all the child nodes of this node.



The following list describes important details about the previous diagram:

①	The Forward Browse VI browses a node whose ID is `ns=2;s=Folder1`.
②	The **browse result** output of the Forward Browse VI returns the reference type, ID, name,

	class, and type of all the child nodes of this node.

Getting the Attributes of a Node

The attributes of a node include the name, node type, data type, and access attribute of the node. You can use the OPC UA Client VIs to get the attributes of a node. The node attributes include the node name, the node class, the data type, the access level, and the description.

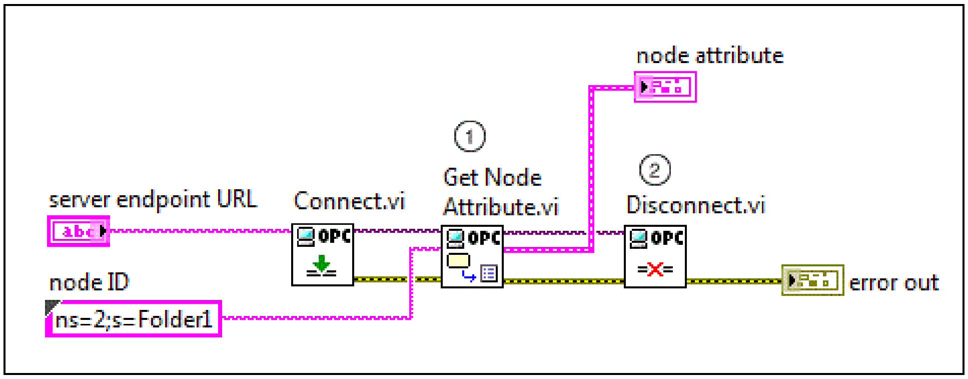
What to Use

Use the following object to get the attributes of a node:

Get Node Attribute VI	
OPC	
+ Add	

What to Do

Create the following block diagram to get the attributes of a node.



The following list describes important details about the previous diagram:

①	The Get Node Attribute VI gets the attributes of a node whose ID is `ns=2;s=Folder1`.
②	The **node attribute** output of the Get Node Attribute VI returns the attributes of the node whose ID is `ns=2;s=Folder1`. The attributes

	contain the name, class, data type, access level, and description of the node.

Reading and Writing the Value of a Node

An OPC UA server allows an OPC UA client to read and update the data in an address space of the OPC UA server. You can use the OPC UA Client VIs to read the value of a node and write a value to a node in an OPC UA server.

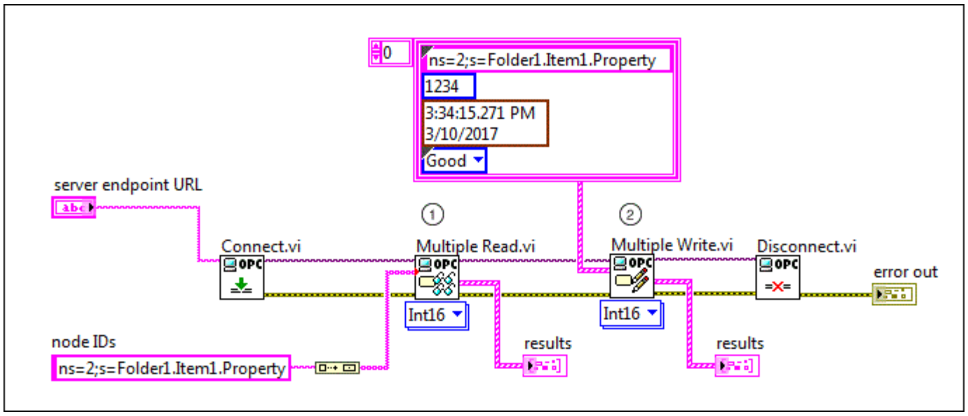
What to Use

Use the following objects to read and write the value of a node:

Multiple Read VI		Multiple Write VI
+ Add		+ Add

What to Do

Create the following block diagram to read the value, timestamp, and status of a node and write a value to the node on an OPC UA server.



The following list describes important details about the previous diagram:

①	The Multiple Read VI reads the value, timestamp, and status of the node whose ID is `ns=2;s=Folder1.Item1.Property`.

②	The Multiple Write VI writes a value, 1234, and status, Good, to the node whose ID is `ns=2;s=Folder1.Item1.Property`.

Reading History Data and Events

You can use the Historical Access VIs of the OPC UA Client VIs to read history data from nodes and history events from notifiers. History data refers to the time series data stored for a node that supports historical access. Examples of history data include device data, calculated data, status information, dynamically changing system data, and diagnostic data. History events refer to the time series events stored in some historical system. Examples of such data include notifications, system alarms, operator action events, and system triggers.

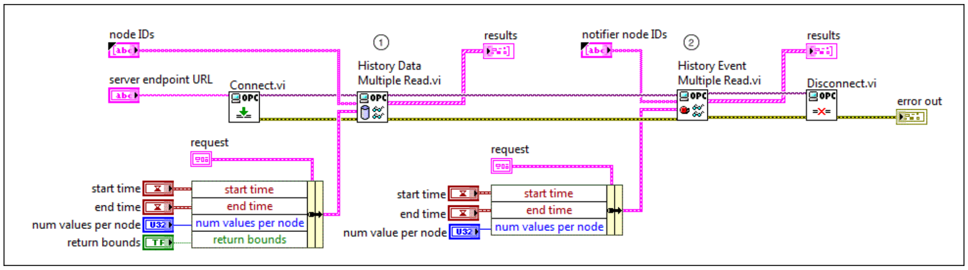
What to Use

Use the following objects to read history data and events:

History Data Multiple Read VI		History Event Multiple Read VI
OPC		OPC
+ Add		+ Add

What to Do

Create the following block diagram to read the history data and events.



The following list describes important details about the previous diagram:

①	The History Data Multiple Read VI reads the history data. **num values per nodes** specifies the maximum number of values to return over the time range.

②	The History Event Multiple Read VI reads the history events.

Creating a Data Subscription

You can use the OPC UA Client VIs to create a data subscription to the nodes of an OPC UA server. When the nodes that an OPC UA client subscribes to incur data changes, the OPC UA server collects the data change and sends a notification message to the OPC UA client.

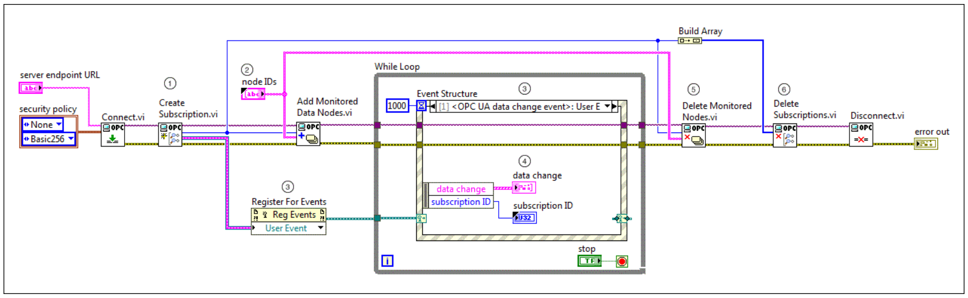
What to Use

Use the following objects to create a data subscription:

Create Subscription VI		Add Monitored Data Nodes VI		Delete Monitored Nodes VI		Delete Subscriptions VI	
OPC		OPC		OPC		OPC	
Add		Add		Add		Add	

What to Do

Create the following block diagram to create a data subscription to the nodes of an OPC UA server.



The following list describes important details about the previous diagram:

①	To create a data subscription to the nodes of the OPC UA server and add nodes to the subscription, you must first wire the

	subscription ID out output of the Create Subscription VI to the **subscription ID in** input of the Add Monitored Data Nodes VI.
②	The **node IDs** control of the Add Monitored Data Nodes VI specifies an array of node IDs that this VI adds to a subscription.
③	The Event structure handles the data change event. You can wire the **OPC UA data change event** output of the Create Subscription VI to the **event source** input of the Register For Events function and wire the **event registration refnum** output of the Register For Events function to the Event Dynamic Registration terminal of the Event structure to handle the data change event. Within the While Loop, the Event structure continuously gets the data change event.
④	The **Data Change** output returns updates of the data on the OPC UA server.
⑤	(Optional) The Delete Monitored Nodes VI deletes monitored nodes from a subscription. You can wire the **subscription ID out** output of the Create Subscription VI to the **subscription ID in** input of the Delete Monitored Nodes VI to delete monitored nodes. The **node IDs** control of the Delete Monitored Nodes VI specifies an array of node IDs that this VI deletes from a subscription.
⑥	(Optional) The Delete Subscriptions VI deletes one or more subscriptions. By deleting a subscription, you delete all the monitored nodes from the subscription.

Creating an Event Subscription

You can use the OPC UA Client VIs to create an event subscription to the notifier of an OPC UA server. When the notifier that an OPC UA client subscribes to receives event notifications, the OPC UA server collects the events and sends a notification message to the OPC UA client.

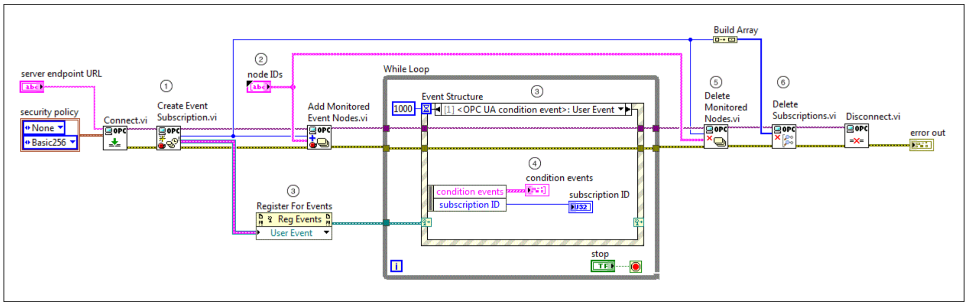
What to Use

Use the following objects to create an event subscription:

Create Event Subscription VI		Add Monitored Event Nodes VI		Delete Monitored Nodes VI		Delete Subscriptions VI	
OPC		OPC		OPC		OPC	
+ Add		+ Add		+ Add		+ Add	

What to Do

Create the following block diagram to create an event subscription to the notifier of an OPC UA server.



The following list describes important details about the previous diagram:

①	To create an event subscription to monitor the notifier of the OPC UA server, you must first wire the **subscription ID out** output of the Create Event Subscription VI to the **subscription ID in** input of the Add Monitored Event Nodes VI.
②	The **node IDs** control of the Add Monitored Events Nodes VI specifies an array of node IDs of notifiers that this VI adds to an event subscription.

③	The Event structure handles the data change event. You can wire the **OPC UA condition event** output of the Create Event Subscription VI to the **event source** input of the Register For Events function and wire the **event registration refnum** output of the Register For Events function to the Event Dynamic Registration terminal of the Event structure to handle the data change event. Within the While Loop, the Event structure continuously gets the condition event.
④	The **condition events** output returns event notifications from the OPC UA server.
⑤	(Optional) The Delete Monitored Nodes VI deletes monitored nodes from a subscription. You can wire the **subscription ID out** output of the Create Subscription VI to the **subscription ID in** input of the Delete Monitored Nodes VI to delete monitored nodes. The **node IDs** control of the Delete Monitored Nodes VI specifies an array of node IDs that this VI deletes from a subscription.
⑥	(Optional) The Delete Subscriptions VI deletes one or more subscriptions. By deleting a subscription, you delete all the monitored notifiers from the event subscription.

Responding to an Acknowledgeable Condition

You can use the Alarms and Conditions VIs of the OPC UA Client VIs to respond to an acknowledgeable condition.

What to Use

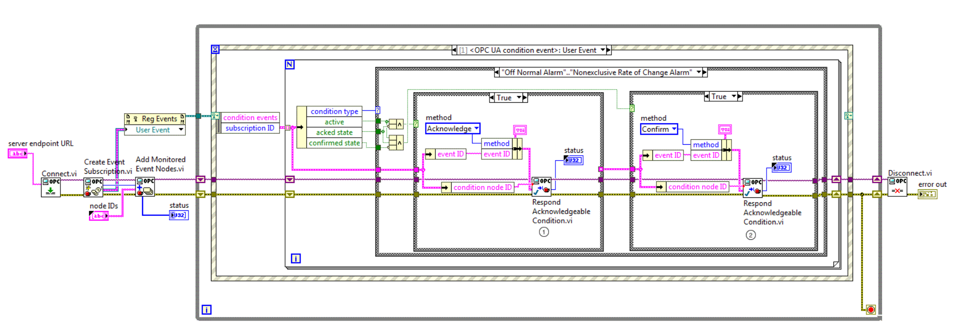
Use the following objects to respond to an acknowledgeable condition:

Create Event Subscription VI		Add Monitored Event Nodes VI		Respond Acknowledgeable Condition VI

OPC		OPC		OPC	
+ Add		+ Add		+ Add	

What to Do

Create the following block diagram to change the states of conditions and alarms.



The following list describes important details about the previous diagram:

①	The Respond Acknowledgeable Condition VI acknowledges an acknowledgeable condition.
②	The Respond Acknowledgeable Condition VI confirms an acknowledgeable condition. A condition has to be acknowledged before it is confirmed.

Shelving and Unshelving an Alarm

You can use the Alarms and Conditions VIs of the OPC UA Client VIs to shelve and unshelve an alarm.

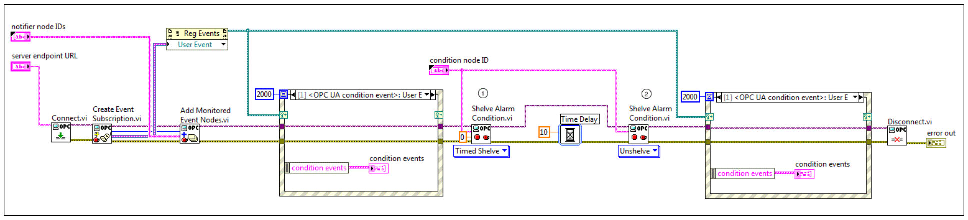
What to Use

Use the following objects to shelve and unshelve an alarm:

Create Event Subscription VI		Add Monitored Event Nodes VI		Shelve Alarm Condition VI	
OPC		OPC		OPC	
+ Add		+ Add		+ Add	

What to Do

Create the following block diagram to shelve and unshelve an alarm.



The following list describes important details about the previous diagram:

①	The Timed Shelve instance of the Shelve Alarm Condition VI shelves an alarm for a definite time period. When **shelving time** is 0, Shelve Alarm Condition VI shelves an alarm for the maximum time that an alarm condition is in shelved state. The OPC UA client cannot receive events when the alarm is in shelved state, unless the alarm is changed to unshelved or inactive state.
②	The unshelve instance of Shelve Alarm Condition VI unshelves an alarm.

⇦	**Previous** Establishing Connections between an OPC UA Server and an OPC UA Client	**Home** Creating an OPC UA Server and an OPC UA Client

OPC UA VIs

November 2020, 376802D-01

Owning Palette: Data Communication VIs and Functions

Requires: OPC UA Toolkit. This topic might not match its corresponding palette in LabVIEW depending on your operating system, licensed product(s), and target.

Use the OPC UA VIs to create a customized OPC UA server application or OPC UA client application. You can use the OPC UA Server VIs and the OPC UA Client VIs to exchange data between OPC UA client and OPC UA server applications.

You must activate the LabVIEW Real-Time Module to use the OPC UA VIs on a real-time target.

The VIs on this palette can return general LabVIEW error codes or specific OPC UA error codes.

Palette Object	Description
Create Certificate	Creates a pair of certificate files with the name you specify. You can use the certificate files to create an OPC UA server or connect to an OPC UA server.

Subpalette	Description
OPC UA Client VIs	Use the OPC UA Client VIs to create a customized OPC UA client application.
OPC UA Server VIs	Use the OPC UA Server VIs to create a customized OPC UA server application.

OPC UA Common Operation Level Status Codes (OPC UA Toolkit)

The following table contains descriptions of the codes returned by the **status** terminal.

Good	0x00000000	The operation was successful.
Uncertain	0x40000000	The value is uncertain but the reason is unknown.
Bad	0x80000000	The value is bad but the reason is unknown.
Bad_UnexpectedError	0x80010000	An unexpected error occurred.
Bad_InternalError	0x80020000	An internal error occurred as a result of a programming or configuration error.
Bad_OutOfMemory	0x80030000	Not enough memory to complete the operation.
Bad_ResourceUnavailable	0x80040000	An operating system resource is not available.

Bad_CommunicationError	0x80050000	A low level communication error occurred.
Bad_EncodingError	0x80060000	Encoding halted because of invalid data in the objects being serialized.
Bad_DecodingError	0x80070000	Decoding halted because of invalid data in the stream.
Bad_EncodingLimitsExceeded	0x80080000	The message encoding/decoding limits imposed by the stack have been exceeded.
Bad_RequestTooLarge	0x80B80000	The request message size exceeds limits set by the server.
Bad_ResponseTooLarge	0x80B90000	The response message size exceeds limits set by the client.
Bad_UnknownResponse	0x80090000	An unrecognized response was received from the server.
Bad_Timeout	0x800A0000	The operation timed out.
Bad_ServiceUnsupported	0x800B0000	The server does not support the requested service.
Bad_Shutdown	0x800C0000	The operation was cancelled because the application is shutting down.
Bad_ServerNotConnected	0x800D0000	The operation could not complete because the client is not connected to the server.
Bad_ServerHalted	0x800E0000	The server has stopped and cannot process any requests.
Bad_NothingToDo	0x800F0000	There was nothing to do because the client passed a list of operations with no elements.
Bad_TooManyOperations	0x80100000	The request could not be processed because it specified too many operations.
Bad_TooManyMonitoredItems	0x80DB0000	The request could not be processed because there are too many monitored items in the subscription.

Bad_DataTypeIdUnknown	0x80110000	The extension object cannot be (de)serialized because the data type ID is not recognized.
Bad_CertificateInvalid	0x80120000	The certificate provided as a parameter is not valid.
Bad_SecurityChecksFailed	0x80130000	An error occurred verifying security.
Bad_CertificateTimeInvalid	0x80140000	The certificate has expired or is not yet valid.
Bad_CertificateIssuerTimeInvalid	0x80150000	An Issuer certificate has expired or is not yet valid.
Bad_CertificateHostNameInvalid	0x80160000	The HostName used to connect to a serverdoes not match a HostName in the certificate.
Bad_CertificateUriInvalid	0x80170000	The URI specified in the ApplicationDescription does not match the URI in the certificate.
Bad_CertificateUseNotAllowed	0x80180000	The certificate may not be used for the requested operation.
Bad_CertificateIssuerUseNotAllowed	0x80190000	The Issuer certificate may not be used for the requested operation.
Bad_CertificateUntrusted	0x801A0000	The certificate is not trusted.
Bad_CertificateRevocationUnknown	0x801B0000	It was not possible to determine if the certificate has been revoked.
Bad_CertificateIssuerRevocationUnknown	0x801C0000	It was not possible to determine if the Issuer certificate has been revoked.
Bad_CertificateRevoked	0x801D0000	The certificate has been revoked.
Bad_CertificateIssuerRevoked	0x801E0000	The Issuer certificate has been revoked.
Bad_UserAccessDenied	0x801F0000	User does not have permission to perform the requested operation.

Bad_IdentityTokenInvalid	0x80200000	The user identity token is not valid.
Bad_IdentityTokenRejected	0x80210000	The user identity token is valid but the server has rejected it.
Bad_SecureChannelIdInvalid	0x80220000	The specified secure channel is no longer valid.
Bad_InvalidTimestamp	0x80230000	The timestamp is outside the range allowed by the server.
Bad_NonceInvalid	0x80240000	The nonce does appear to be not a random value or it is not the correct length.
Bad_SessionIdInvalid	0x80250000	The session ID is not valid.
Bad_SessionClosed	0x80260000	The session was closed by the client.
Bad_SessionNotActivated	0x80270000	The session cannot be used because ActivateSession has not been called.
Bad_SubscriptionIdInvalid	0x80280000	The subscription ID is not valid.
Bad_RequestHeaderInvalid	0x802A0000	The header for the request is missing or invalid.
Bad_TimestampsToReturnInvalid	0x802B0000	The timestamps to return parameter are invalid.
Bad_RequestCancelledByClient	0x802C0000	The request was cancelled by the client.
Good_SubscriptionTransferred	0x002D0000	The subscription was transferred to another session.
Good_CompletesAsynchronously	0x002E0000	The processing will complete asynchronously.
Good_Overload	0x002F0000	Sampling has slowed down due to resource limitations.
Good_Clamped	0x00300000	The value written was accepted but was clamped.
Bad_NoCommunication	0x80310000	Communication with the data source is defined but not established and there is no last known value available.

Bad_WaitingForInitialData	0x80320000	Waiting for the server to obtain values from the underlying data source.
Bad_NodeIdInvalid	0x80330000	The syntax of the node ID is not valid.
Bad_NodeIdUnknown	0x80340000	The node ID refers to a node that does not exist in the server address space.
Bad_AttributeIdInvalid	0x80350000	The attribute is not supported for the specified node.
Bad_IndexRangeInvalid	0x80360000	The syntax of the index range parameter is invalid.
Bad_IndexRangeNoData	0x80370000	No data exists within the range of indexes specified.
Bad_DataEncodingInvalid	0x80380000	The data encoding is invalid.
Bad_DataEncodingUnsupported	0x80390000	The server does not support the requested data encoding for the node.
Bad_NotReadable	0x803A0000	The access level does not allow reading or subscribing to the node.
Bad_NotWritable	0x803B0000	The access level does not allow writing to the node.
Bad_OutOfRange	0x803C0000	The value was out of range.
Bad_NotSupported	0x803D0000	The requested operation is not supported.
Bad_NotFound	0x803E0000	A requested item was not found or a search operation ended without success.
Bad_ObjectDeleted	0x803F0000	The object cannot be used because it has been deleted.
Bad_NotImplemented	0x80400000	Requested operation is not implemented.
Bad_MonitoringModeInvalid	0x80410000	The monitoring mode is invalid.
Bad_MonitoredItemIdInvalid	0x80420000	The monitoring item ID does not refer to a valid monitored item .

Bad_MonitoredItemFilterInvalid	0x80430000	The monitored item filter parameter is not valid.
Bad_MonitoredItemFilterUnsupported	0x80440000	The server does not support the requested monitored item filter.
Bad_FilterNotAllowed	0x80450000	A monitoring filter cannot be used in combination with the attribute specified.
Bad_StructureMissing	0x80460000	A mandatory structured parameter was missing or null.
Bad_EventFilterInvalid	0x80470000	The event filter is not valid.
Bad_ContentFilterInvalid	0x80480000	The content filter is not valid.
Bad_FilterOperatorInvalid	0x80C10000	An unregognized operator was provided in a filter.
Bad_FilterOperatorUnsupported	0x80C20000	A valid operator was provided but the server does not provide support for this filter operator.
Bad_FilterOperandCountMismatch	0x80C30000	The number of operands provided for the filter operator was less than expected for the operand provided.
Bad_FilterOperandInvalid	0x80490000	The operand used in a content filter is not valid.
Bad_FilterElementInvalid	0x80C40000	The referenced element is not a valid element in the content filter.
Bad_FilterLiteralInvalid	0x80C50000	The referenced literal is not a valid value.
Bad_ContinuationPointInvalid	0x804A0000	The continuation point is no longer valid.
Bad_NoContinuationPoints	0x804B0000	The operation could not be processed because all continuation points have been allocated.
Bad_ReferenceTypeIdInvalid	0x804C0000	The reference type ID is not valid.
Bad_BrowseDirectionInvalid	0x804D0000	The browse direction is not valid.

Bad_NodeNotInView	0x804E0000	The node is not part of the view .
Bad_ServerUriInvalid	0x804F0000	The ServerUri is not a valid URI.
Bad_ServerNameMissing	0x80500000	No ServerName was specified.
Bad_DiscoveryUrlMissing	0x80510000	No DiscoveryUrl was specified.
Bad_SempahoreFileMissing	0x80520000	The semaphore file specified by the client is not valid.
Bad_RequestTypeInvalid	0x80530000	The security token request type is not valid.
Bad_SecurityModeRejected	0x80540000	The security mode does not meet the requirements set by the server.
Bad_SecurityPolicyRejected	0x80550000	The security policy does not meet the requirements set by the server.
Bad_TooManySessions	0x80560000	The server has reached its maximum number of sessions.
Bad_UserSignatureInvalid	0x80570000	The user token signature is missing or invalid.
Bad_ApplicationSignatureInvalid	0x80580000	The signature generated with the client certificate is missing or invalid.
Bad_NoValidCertificates	0x80590000	The client did not provide at least one software certificate that is valid and meets the profile requirements for the server.
Bad_IdentityChangeNotSupported	0x80C60000	The server does not support changing the user identity assigned to the session.
Bad_RequestCancelledByRequest	0x805A0000	The request was cancelled by the client with the Cancel service.
Bad_ParentNodeIdInvalid	0x805B0000	The parent node ID does not refer to a valid node.
Bad_ReferenceNotAllowed	0x805C0000	The reference could not be created because it violates constraints imposed by the data model.

Bad_NodeIdRejected	0x805D0000	The requested node ID was rejected because it was invalid or server does not allow node IDs to be specified by the client.
Bad_NodeIdExists	0x805E0000	The requested node ID is already used by another node.
Bad_NodeClassInvalid	0x805F0000	The node class is not valid.
Bad_BrowseNameInvalid	0x80600000	The browse name is invalid.
Bad_BrowseNameDuplicated	0x80610000	The browse name is not unique among nodes that share the same relationship with the parent.
Bad_NodeAttributesInvalid	0x80620000	The node attributes are not valid for the node class.
Bad_TypeDefinitionInvalid	0x80630000	The type definition node ID does not reference an appropriate type node.
Bad_SourceNodeIdInvalid	0x80640000	The source node ID does not reference a valid node.
Bad_TargetNodeIdInvalid	0x80650000	The target node ID does not reference a valid node.
Bad_DuplicateReferenceNotAllowed	0x80660000	The reference type between the nodes is already defined.
Bad_InvalidSelfReference	0x80670000	The server does not allow this type of self-reference on this node.
Bad_ReferenceLocalOnly	0x80680000	The reference type is not valid for a reference to a remote server.
Bad_NoDeleteRights	0x80690000	The server will not allow the node to be deleted.
Uncertain_ReferenceNotDeleted	0x40BC0000	The server was not able to delete all target references.
Bad_ServerIndexInvalid	0x806A0000	The server index is not valid.
Bad_ViewIdUnknown	0x806B0000	The view ID does not refer to a valid view node.

Bad_ViewTimestampInvalid	0x80C90000	The view timestamp is not available or not supported.
Bad_ViewParameterMismatch	0x80CA0000	The view parameters are not consistent with each other.
Bad_ViewVersionInvalid	0x80CB0000	The view version is not available or not supported.
Uncertain_NotAllNodesAvailable	0x40C00000	The list of references may not be complete because the underlying system is not available.
Good_ResultsMayBeIncomplete	0x00BA0000	The server should have followed a reference to a node in a remote server but did not. The result set may be incomplete.
Bad_NotTypeDefinition	0x80C80000	The provided node ID was not a type definition node ID.
Uncertain_ReferenceOutOfServer	0x406C0000	One of the references to follow in the relative path references to a node in the address space in another server.
Bad_TooManyMatches	0x806D0000	The requested operation has too many matches to return.
Bad_QueryTooComplex	0x806E0000	The requested operation requires too many resources on the server.
Bad_NoMatch	0x806F0000	The requested operation has no match to return.
Bad_MaxAgeInvalid	0x80700000	The max age parameter is invalid.
Bad_HistoryOperationInvalid	0x80710000	The history details parameter is not valid.
Bad_HistoryOperationUnsupported	0x80720000	The server does not support the requested operation.
Bad_InvalidTimestampArgument	0x80BD0000	The defined timestamp to return was invalid.
Bad_WriteNotSupported	0x80730000	The server does not support writing the combination of value status and timestamps provided.

Bad_TypeMismatch	0x80740000	The value supplied for the attribute is not of the same type as the attribute's value.
Bad_MethodInvalid	0x80750000	The method ID does not refer to a method for the specified object.
Bad_ArgumentsMissing	0x80760000	The client did not specify all of the input arguments for the method.
Bad_TooManySubscriptions	0x80770000	The server has reached its maximum number of subscriptions.
Bad_TooManyPublishRequests	0x80780000	The server has reached the maximum number of queued publish requests.
Bad_NoSubscription	0x80790000	There is no subscription available for this session.
Bad_SequenceNumberUnknown	0x807A0000	The sequence number is unknown to the server.
Bad_MessageNotAvailable	0x807B0000	The requested notification message is no longer available.
Bad_InsufficientClientProfile	0x807C0000	The Client of the current Session does not support one or more Profiles that are necessary for the Subscription.
Bad_StateNotActive	0x80BF0000	The sub-state machine is not currently active.
Bad_TcpServerTooBusy	0x807D0000	The server cannot process the request because it is too busy.
Bad_TcpMessageTypeInvalid	0x807E0000	The type of the message specified in the header is invalid.
Bad_TcpSecureChannelUnknown	0x807F0000	The SecureChannelId and/or TokenId are not currently in use.
Bad_TcpMessageTooLarge	0x80800000	The size of the message specified in the header is too large.
Bad_TcpNotEnoughResources	0x80810000	There are not enough resources to process the request.
Bad_TcpInternalError	0x80820000	An internal error occurred.

Bad_TcpEndpointUrlInvalid	0x80830000	The serverdoes not recognize the QueryString specified.
Bad_RequestInterrupted	0x80840000	The request could not be sent because of a network interruption.
Bad_RequestTimeout	0x80850000	Timeout occurred while processing the request.
Bad_SecureChannelClosed	0x80860000	The secure channel has been closed.
Bad_SecureChannelTokenUnknown	0x80870000	The token has expired or is not recognized.
Bad_SequenceNumberInvalid	0x80880000	The sequence number is not valid.
Bad_ProtocolVersionUnsupported	0x80BE0000	The applications do not have compatible protocol versions.
Bad_ConfigurationError	0x80890000	There is a problem with the configuration that affects the usefulness of the value.
Bad_NotConnected	0x808A0000	The variable should receive its value from another variable but has never been configured to do so.
Bad_DeviceFailure	0x808B0000	There has been a failure in the device/data source that generates the value that has affected the value.
Bad_SensorFailure	0x808C0000	There has been a failure in the sensor from which the value is derived by the device/data source.
Bad_OutOfService	0x808D0000	The source of the data is not operational.
Bad_DeadbandFilterInvalid	0x808E0000	The deadband filter is not valid.
Uncertain_NoCommunicationLastUsableValue	0x408F0000	Communication to the data source has failed. The variable value is the last value that had a Good quality.

Uncertain_LastUsableValue	0x40900000	Whatever was updating this value has stopped doing so.
Uncertain_SubstituteValue	0x40910000	The value is an operational value that was manually overwritten.
Uncertain_InitialValue	0x40920000	The value is an initial value for a variable that normally receives its value from another variable.
Uncertain_SensorNotAccurate	0x40930000	The value is at one of the sensor limits.
Uncertain_EngineeringUnitsExceeded	0x40940000	The value is outside of the range of values defined for this parameter.
Uncertain_SubNormal	0x40950000	The value is derived from multiple sources and contains less than the required number of Good sources.
Good_LocalOverride	0x00960000	The value has been overridden.
Bad_RefreshInProgress	0x80970000	This Condition refresh failed or a Condition refresh operation is already in progress.
Bad_ConditionAlreadyDisabled	0x80980000	This condition has already been disabled.
Bad_ConditionAlreadyEnabled	0x80CC0000	This condition has already been enabled.
Bad_ConditionDisabled	0x80990000	The property is not available or this condition is disabled.
Bad_EventIdUnknown	0x809A0000	The specified event ID is not recognized.
Bad_EventNotAcknowledgeable	0x80BB0000	The event cannot be acknowledged.
Bad_DialogNotActive	0x80CD0000	The dialog condition is not active.
Bad_DialogResponseInvalid	0x80CE0000	The response is not valid for the dialog.
Bad_ConditionBranchAlreadyAcked	0x80CF0000	The condition branch has already been acknowledged.

Bad_ConditionBranchAlreadyConfirmed	0x80D00000	The condition branch has already been confirmed.
Bad_ConditionAlreadyShelved	0x80D10000	The condition has already been shelved.
Bad_ConditionNotShelved	0x80D20000	The condition is not currently shelved.
Bad_ShelvingTimeOutOfRange	0x80D30000	The shelving time is not within an acceptable range.
Bad_NoData	0x809B0000	No data exists for the requested time range or event filter.
Bad_BoundNotFound	0x80D70000	No data found to provide upper or lower bound value.
Bad_BoundNotSupported	0x80D80000	The server cannot retrieve a bound for the variable.
Bad_DataLost	0x809D0000	Data is missing due to collection started/stopped/lost.
Bad_DataUnavailable	0x809E0000	Expected data is unavailable for the requested time range due to an unmounted volume, an offline archive or tape, or similar reason for temporary unavailability.
Bad_EntryExists	0x809F0000	The data or event was not successfully inserted because a matching entry exists.
Bad_NoEntryExists	0x80A00000	The data or event was not successfully updated because no matching entry exists.
Bad_TimestampNotSupported	0x80A10000	The client requested history using a timestamp format the server does not support. In other words, the client requested ServerTimestamp when server only supports SourceTimestamp.
Good_EntryInserted	0x00A20000	The data or event was successfully inserted into the historical database.

Good_EntryReplaced	0x00A30000	The data or event was successfully replaced in the historical database.
Uncertain_DataSubNormal	0x40A40000	The value is derived from multiple values and contains less than the required number of Good values.
Good_NoData	0x00A50000	No data exists for the requested time range or event filter.
Good_MoreData	0x00A60000	More data exists for the requested time range or event filter.
Bad_AggregateListMismatch	0x80D40000	The requested number of Aggregates does not match the requested number of node IDs.
Bad_AggregateNotSupported	0x80D50000	The requested Aggregate is not support by the server.
Bad_AggregateInvalidInputs	0x80D60000	The aggregate value could not be derived due to invalid input data.
Bad_AggregateConfigurationRejected	0x80DA0000	The aggregate configuration is not valid for the specified node.
Good_DataIgnored	0x00D90000	The request specifies fields which are not valid for the EventType or cannot be saved by the historian.
Good_CommunicationEvent	0x00A70000	The communication layer has raised an event.
Good_ShutdownEvent	0x00A80000	The system is shutting down.
Good_CallAgain	0x00A90000	The operation is not finished and needs to be called again.
Good_NonCriticalTimeout	0x00AA0000	A non-critical timeout occurred.
Bad_InvalidArgument	0x80AB0000	One or more arguments are invalid.
Bad_ConnectionRejected	0x80AC0000	Could not establish a network connection to remote server.
Bad_Disconnect	0x80AD0000	The server has disconnected from the client.

Bad_ConnectionClosed	0x80AE0000	The network connection has been closed.
Bad_InvalidState	0x80AF0000	The operation cannot be completed because the object is closed, uninitialized or in some other invalid state.
Bad_EndOfStream	0x80B00000	Cannot move beyond end of the stream.
Bad_NoDataAvailable	0x80B10000	No data is currently available for reading from a non-blocking stream.
Bad_WaitingForResponse	0x80B20000	The asynchronous operation is waiting for a response.
Bad_OperationAbandoned	0x80B30000	The asynchronous operation was abandoned by the caller.
Bad_ExpectedStreamToBlock	0x80B40000	The stream did not return all data requested (possibly because it is a non-blocking stream).
Bad_WouldBlock	0x80B50000	Non-blocking behavior is required and the operation would block.
Bad_SyntaxError	0x80B60000	The value had an invalid syntax.
Bad_MaxConnectionsReached	0x80B70000	The operation could not be finished because all available connections are in use.

Create Certificate VI

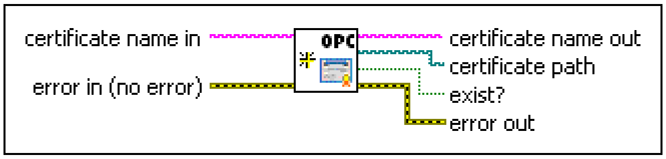
Owning Palette: OPC UA VIs

Requires: OPC UA Toolkit

Creates a pair of certificate files with the name you specify. You can use the certificate files to create an OPC UA server or connect to an OPC UA server.

The start time of the certificate file is the current system time. The lifetime is four years.

Example



certificate name in specifies the name of the certificates to create. If you do not specify this input, this VI creates a pair of certificates with the default name of `Default OPC UA`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

certificate name out returns the name of the certificates that you create.

certificate name out contains only the filename without the file extension. Both the public key and the private key use the same certificate name but with different file extensions. Public key commonly appears with a `.der` file extension and private key has a `.pem` file extension.

certificate path returns the full path of the certificate file. You can find the public key file in this path.

exist? returns whether a certificate with the same name already exists. If a certificate with the same name already exists, this VI does not create the certificate.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Create Certificate VI.

OPC UA Client VIs

Owning Palette: OPC UA VIs

Requires: OPC UA Toolkit. This topic might not match its corresponding palette in LabVIEW depending on your operating system, licensed product(s), and target.

Use the OPC UA Client VIs to create a customized OPC UA client application.

Example

Palette Object	Description
Add Monitored Data Nodes	Adds nodes to a data subscription to monitor da ta changes.
Connect	Creates a connection to an OPC UA server.
Create Subscription	Creates a subscription to the nodes of the OPC UA server. When the nodes that the OPC UA clie nt subscribes to incur data changes, the OPC UA server collects the data changes and sends a no tification message to the OPC UA client.
Delete Monitored Nodes	Deletes one or multiple monitored nodes from a subscription.
Delete Subscriptions	Deletes one or multiple subscriptions. By deleti ng a subscription, you delete all the monitored nodes from the subscription.
Disconnect	Disconnects an OPC UA client from an OPC UA s erver. The OPC UA client automatically deletes a ll the subscriptions before disconnecting from t he OPC UA server.
Forward Browse	Browses a node and returns information about i ts child nodes.
Get Node Attribute	Gets the attributes of a node.
Multiple Read	Reads the value, timestamp, and status of one o r multiple nodes. NI recommends choosing the Variant instance for all value data types. You mu st manually select the polymorphic instance to use.
Multiple Write	Writes values to one or multiple nodes. NI reco mmends choosing the Variant instance for all va

	lue data types. You must manually select the polymorphic instance to use.

Subpalette	Description
Alarms and Conditions VIs	Use the Alarms and Conditions VIs to subscribe to alarms and conditions for OPC UA client applications.
Historical Access VIs	Use the Historical Access VIs to access historical data and events for OPC UA client applications.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the OPC UA Client VIs.

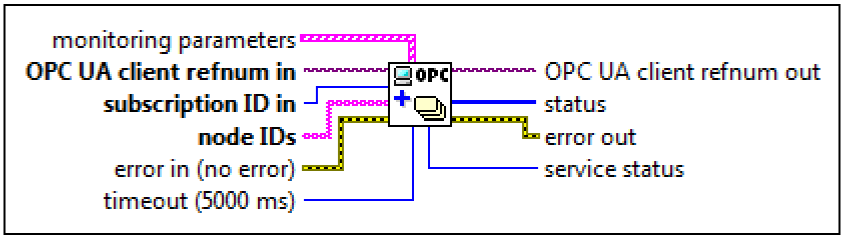
Add Monitored Data Nodes VI

Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit

Adds nodes to a data subscription to monitor data changes.

Example



monitoring parameters specifies the settings for monitoring data changes.

DBL

sampling interval specifies the maximum rate for the OPC UA server to sample its underlying source for data changes. The default is -1, which specifies that the sampling interval is the same as the publishing interval of the subscription.

U32 **queue size** specifies the size of the queue that stores the data change notification to the subscription. The default is 1.

TF **discard oldest** specifies whether to discard the oldest element in the queue. The default is TRUE, which specifies to discard the oldest element in the queue.

data change filter specifies values to filter data changes.

trigger specifies the conditions under which LabVIEW reports a data change notification.

0	**Status**—LabVIEW

	reports a notification only when the status code associated with the value changes.
1	**Status Value** (default)—LabVIEW reports a notification when the value or the

	status code associated with the value changes.
2	**StatusValueTimestamp**—LabVIEW reports a notification when the status code associated with the value, the

	value, or the source timestamp changes. If you specify the deadband filter, this option has the same behavior as **Status Value** .

deadband type

defines the type of the deadband. Deadband is the range for value changes that do not trigger a data change notification.

0	**None** (default)
1	**Absolute**
2	**Percent**

DBL **deadband value**

specifies the value of the deadband. The default is 0. For the `Absolute` deadband type, **deadband value** is the absolute change in a data value that causes LabVIEW to generate a notification. For the `Percent` deadband type, **deadband value** is the percentage of the EU range and only applies to analog items.

I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

U32 **subscription ID in** specifies the ID of the subscription.

[abc] **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

> If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identif`

`ier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32	**timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.
I/O	**OPC UA client refnum out** returns the reference for the OPC UA client.
U32	**status** returns the status code.
	error out contains error information. This output provides standard error out functionality.
U32	**service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Add Monitored Data Nodes VI.

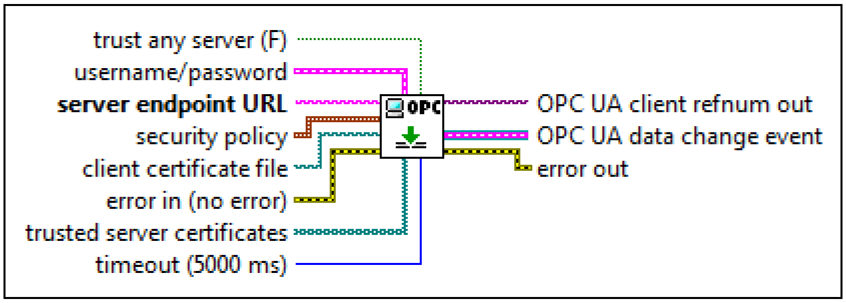
Connect VI

Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit

Creates a connection to an OPC UA server.

Example



TF	**trust any server** specifies whether the OPC UA client trusts any OPC UA server without adding

the server certificate to **trusted server certificates**. The default is FALSE, which specifies that the OPC UA client trusts only OPC UA servers that have server certificates in **trusted server certificates**.

username/password specifies the username and password to authenticate when you create a connection to an OPC UA server. You must specify **username/password** if the OPC UA server requires authentication.

username specifies the username for connecting to the OPC UA server.

password specifies the password for connecting to the OPC UA server.

server endpoint URL specifies the hostname and the port of the OPC UA server. The format of **server endpoint URL** is `opc.tcp://hostname:port`. The hostname is the IP address or computer name.

security policy specifies the message mode and the corresponding security policy.

message mode specifies the mode of the message. If the **message mode** is `None`, this VI ignores **security**.

0	**None** (default)—The OPC UA client and O

	PC UA server do not need to trust each other.
1	**Sign**—The OPC UA server supports signed messages but not encrypted messages.
2	**Sign and Encrypt**—The OPC UA server supports signed and encrypted messages.

security specifies the security policy.

0	**Basic256** (default)
1	**Basic128Rsa15**

client certificate file specifies the path or name of the public key. The file extension of the file you specify must be `.der`. The public key and private key must have the same name and reside in the same folder. If you do not specify **client certificate file**, this VI generates and uses a new certificate file. By default, an OPC UA client trusts the certificate it is using.

You can specify a relative path or filename for **client certificate file**. If you specify a relative path, the path is relative to the caller VI or to the application directory. If you specify a filename, LabVIEW searches the certificate files only in the location where the caller VI resides or in the application directory.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

trusted server certificates specifies the paths or names of the public keys that the OPC UA client trusts. By default, OPC UA client applications trust the public key that the OPC UA client uses. You can specify a relative path or filename for **trusted server certificates**. If you specify a relative path, the path is relative to the caller VI or to the application directory. If you specify a filename, LabVIEW searches the certificate files only in the location where the caller VI resides or in the application directory.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

OPC UA data change event returns notifications to the OPC UA client from all the subscriptions that the OPC UA client has created. The notifications contain updates of the data on the OPC UA server. NI does not recommend using this output.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Connect VI.

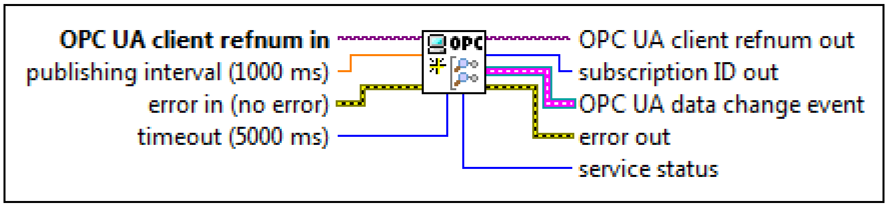
Create Subscription VI

Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit

Creates a subscription to the nodes of the OPC UA server. When the nodes that the OPC UA client subscribes to incur data changes, the OPC UA server collects the data changes and sends a notification message to the OPC UA client.

Example



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

DBL **publishing interval** defines the rate, in milliseconds, that the OPC UA server returns data change notifications to the OPC UA client. The default is 1000. **publishing interval** must be greater than 0.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

U32 **subscription ID out** returns the reference of the subscription that this VI accessed.

OPC UA data change event returns notifications to the OPC UA client if the client has created a subscription. The notifications contain updates of the data on the OPC UA server.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Create Subscription VI.

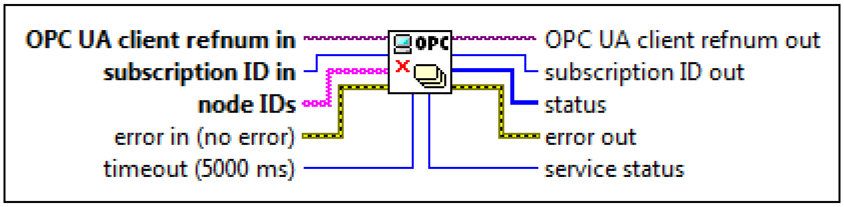
Delete Monitored Nodes VI

Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit

Deletes one or multiple monitored nodes from a subscription.

Example



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

U32 **subscription ID in** specifies the ID of the subscription.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of

elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

subscription ID out returns the reference of the subscription that this VI accessed.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Delete Monitored Nodes VI.

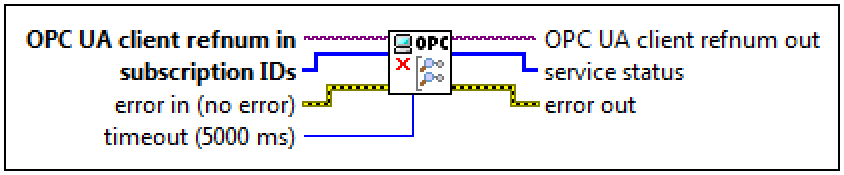
Delete Subscriptions VI

Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit

Deletes one or multiple subscriptions. By deleting a subscription, you delete all the monitored nodes from the subscription.

Example



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

[U32] **subscription IDs** specifies the subscriptions to delete.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Delete Subscriptions VI.

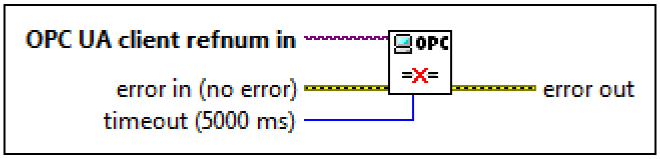
Disconnect VI

Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit

Disconnects an OPC UA client from an OPC UA server. The OPC UA client automatically deletes all the subscriptions before disconnecting from the OPC UA server.

Example



OPC UA client refnum in specifies the reference for the OPC UA client.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Disconnect VI.

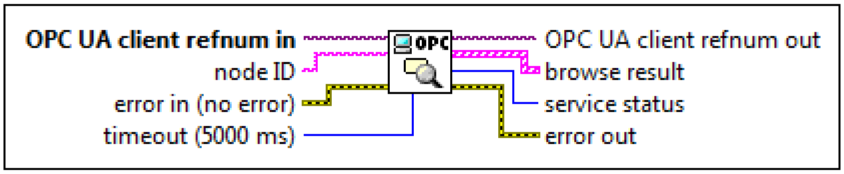
Forward Browse VI

Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit

Browses a node and returns information about its child nodes.

Example



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

browse result returns information about child nodes.

reference type returns the type of the reference.

0	**Unspecified**
35	**Organizes**

36	Has Event Source
45	Has Subtype
46	Has Property
47	Has Component
48	Has Notifier
49	Has Ordered Component

abc **node ID** returns the ID of the child node.

abc **browse name** returns the name of the child node.

U32 **node class** returns the node class.

0	Unspecified
1	Object
2	Variable
4	Method
8	Object Type
16	Variable Type
32	Reference Type

64	Data Type
128	View

U32

node type returns the type of the child node.

0	Unspecified
24	BaseDataType
58	BaseObjectType
61	FolderType
62	BaseVariableType
63	BaseDataVariableType
68	PropertyType
69	DataTypeDescriptionType
72	DataTypeDictionaryType
75	DataTypeSystemType
76	DataTypeEncodingType

77	ModellingRuleType
120	NamingRuleType
256	IdType
303	UserTokenType
307	ApplicationType
315	SecurityTokenRequestType
398	EnumeratedTestType
399	ScalarTestType
402	ArrayTestType
405	CompositeTestType
718	DeadbandType
853	RedundantServerDataType
856	SamplingIntervalDiagnosticsDataType

859	ServerDiagnosticsSummaryDataType
862	ServerStatusDataType
865	SessionDiagnosticsDataType
868	SessionSecurityDiagnosticsDataType
871	ServiceCounterDataType
874	SubscriptionDiagnosticsDataType
877	ModelChangeStructureDataType
894	ProgramDiagnosticDataType
897	SemanticChangeStructureDataType

2004	ServerType
2013	ServerCapabilitiesType
2020	ServerDiagnosticsType
2026	SessionsDiagnosticsSummaryType
2029	SessionDiagnosticsObjectType
2033	VendorServerInfoType
2034	ServerRedundancyType
2036	TransparentRedundancyType
2039	NonTransparentRedundancyType
2041	BaseEventType
2052	AuditEventType

2058	AuditSecurityEventType
2059	AuditChannelEventType
2060	AuditOpenSecureChannelEventType
2069	AuditSessionEventType
2071	AuditCreateSessionEventType
2075	AuditActivateSessionEventType
2078	AuditCancelEventType
2080	AuditCertificateEventType
2082	AuditCertificateDataMismatchEventType

2085	AuditCertificateExpiredEventType
2086	AuditCertificateInvalidEventType
2087	AuditCertificateUntrustedEventType
2088	AuditCertificateRevokedEventType
2089	AuditCertificateMismatchEventType
2090	AuditNodeManagementEventType
2091	AuditAddNodesEventType
2093	AuditDeleteNodesEventType

2095	AuditAddReferencesEventType
2097	AuditDeleteReferencesEventType
2099	AuditUpdateEventType
2100	AuditWriteUpdateEventType
2104	AuditHistoryUpdateEventType
2127	AuditUpdateMethodEventType
2130	SystemEventType
2131	DeviceFailureEventType
2132	BaseModelChangeEventType

2133	GeneralModelChangeEventType
2137	ServerVendorCapabilityType
2138	ServerStatusType
2150	ServerDiagnosticsSummaryType
2164	SamplingIntervalDiagnosticsArrayType
2165	SamplingIntervalDiagnosticsType
2171	SubscriptionDiagnosticsArrayType
2172	SubscriptionDiagnosticsType
2196	SessionDiagnosticsArrayType

2197	SessionDiagnosticsVariableType
2243	SessionSecurityDiagnosticsArrayType
2244	SessionSecurityDiagnosticsType
2299	StateMachineType
2307	StateType
2309	InitialStateType
2310	TransitionType
2311	TransitionEventType
2315	AuditUpdateStateEventType
2318	HistoricalDataConfigurationType
2330	HistoryServerCapabilitiesType

2340	AggregateFunctionType
2365	DataItemType
2368	AnalogItemType
2372	DiscreteItemType
2373	TwoStateDiscreteType
2376	MultiStateDiscreteType
2378	ProgramTransitionEventType
2380	ProgramDiagnosticType
2391	ProgramStateMachineType
2738	SemanticChangeEventType
2748	AuditUrlMismatchEventType
2755	StateVariableType

2760	FiniteStateVariableType
2762	TransitionVariableType
2767	FiniteTransitionVariableType
2771	FiniteStateMachineType
2782	ConditionType
2787	RefreshStartEventType
2788	RefreshEndEventType
2789	RefreshRequiredEventType
2790	AuditConditionEventType
2803	AuditConditionEnableEventType

2829	AuditConditionCommentEventType
2830	DialogConditionType
2881	AcknowledgeableConditionType
2915	AlarmConditionType
2929	ShelvedStateMachineType
2955	LimitAlarmType
2999	AuditHistoryEventUpdateEventType
3006	AuditHistoryValueUpdateEventType
3012	AuditHistoryDeleteEventType

3014	AuditHistoryRawModifyDeleteEventType
3019	AuditHistoryAtTimeDeleteEventType
3022	AuditHistoryEventDeleteEventType
3035	EventQueueOverflowEventType
3051	BuildInfoType
3806	ProgramTransitionAuditEventType
7594	EnumValueType
8912	TimeZoneDataType
8921	LockType

8927	AuditConditionRespondEventType
8944	AuditConditionAcknowledgeEventType
8961	AuditConditionConfirmEventType
8995	TwoStateVariableType
9002	ConditionVariableType
9318	ExclusiveLimitStateMachineType
9341	ExclusiveLimitAlarmType
9482	ExclusiveLevelAlarmType
9623	ExclusiveRateOfChangeAlarmType

9764	ExclusiveDeviationAlarmType
9906	NonExclusiveLimitAlarmType
10060	NonExclusiveLevelAlarmType
10214	NonExclusiveRateOfChangeAlarmType
10368	NonExclusiveDeviationAlarmType
10523	DiscreteAlarmType
10637	OffNormalAlarmType
10751	TripAlarmType
11093	AuditConditionShelvingEventType

11163	BaseConditionClassType
11164	ProcessConditionClassType
11165	MaintenanceConditionClassType
11166	SystemConditionClassType
11187	AggregateConfigurationType
11234	HistoryUpdateType
11238	MultiStateValueDiscreteType
11293	PerformUpdateType
11436	ProgressEventType
11446	SystemStatusChangeEventType

11487	OptionSetType
11564	OperationLimitsType
11575	FileType
11595	AddressSpaceFileType
11616	NamespaceMetadataType
11645	NamespacesType
11737	BitFieldMaskDataType
11753	SystemOffNormalAlarmType
11856	AuditProgramTransitionEventType
11943	EndpointUrlListDataType
11944	NetworkGroupDataType

11945	NonTransparentNetworkRedundancyType
12021	ArrayItemType
12029	YArrayItemType
12038	XYArrayItemType
12047	ImageItemType
12057	CubeItemType
12068	NDimensionArrayItemType
12080	XVType
12171	ComplexNumberType
12172	DoubleComplexNumberType
12522	TrustListType
12554	TrustListDataType
12555	CertificateGroupType

12556	CertificateType
12557	ApplicationCertificateType
12558	HttpsCertificateType
12559	RsaMinApplicationCertificateType
12560	RsaSha256ApplicationCertificateType
12561	TrustListUpdatedAuditEventType
12581	ServerConfigurationType
12620	CertificateUpdatedAuditEventType
13225	CertificateExpirationAlarmType
13353	FileDirectoryType

13813	CertificateGroupFolderType

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Forward Browse VI.

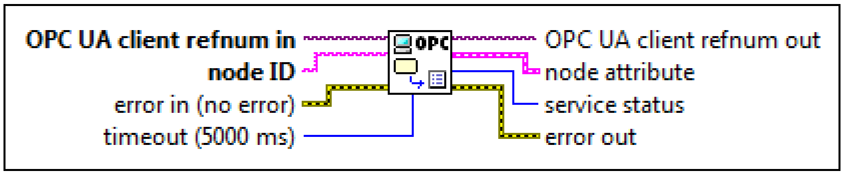
Get Node Attribute VI

Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit

Gets the attributes of a node.

Example



OPC UA client refnum in specifies the reference for the OPC UA client.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  > If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string

type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

node attribute returns the attributes of the node.

name returns the name of the node.

node class returns the node class.

0	**Unspecified**
1	**Object**
2	**Variable**
4	**Method**
8	**Object Type**
16	**Variable Type**
32	**Reference Type**
64	**Data Type**
128	**View**

U16

data type returns the data type of the node.

Value	Data Type
1	Boolean
2	SByte
3	Byte
4	Int16
5	UInt16
6	Int32
7	UInt32
8	Int64
9	UInt64
10	Float
11	Double
12	String
13	DateTime
15	ByteString
101	Array of Boolean
102	Array of SByte
103	Array of Byte
104	Array of Int16
105	Array of UInt16

106	Array of Int32
107	Array of UInt32
108	Array of Int64
109	Array of UInt64
110	Array of Float
111	Array of Double
112	Array of String
113	Array of DateTime
115	Array of ByteString
201	Matrix of Boolean
202	Matrix of SByte
203	Matrix of Byte
204	Matrix of Int16
205	Matrix of UInt16
206	Matrix of Int32

207	Matrix of UInt32
208	Matrix of Int64
209	Matrix of UInt64
210	Matrix of Float
211	Matrix of Double
212	Matrix of String
213	Matrix of DateTime
215	Matrix of ByteString

access returns the access level of the node.

Digital Display	Access
0	Null
1	Read
2	Write
3	Read&Write

abc

description returns the description of the node.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Get Node Attribute VI.

Multiple Read VI

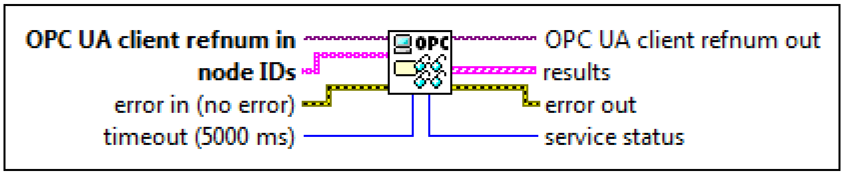
Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit

Reads the value, timestamp, and status of one or multiple nodes. NI recommends choosing the Variant instance for all value data types. You must manually select the polymorphic instance to use.

Example

Multiple Read (Variant)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string

type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

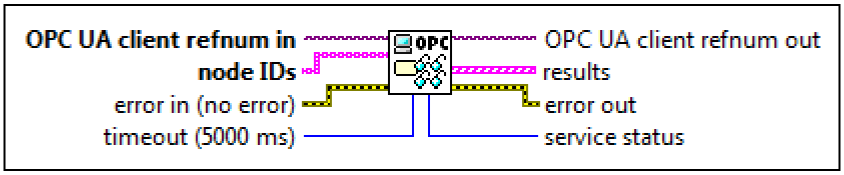
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **value** returns the value that this VI reads from the node. **value** can also return the Matrix data type.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Bool)



I/O

OPC UA client refnum in specifies the reference for the OPC UA client.

abc

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

value returns the value that this VI reads from the node.

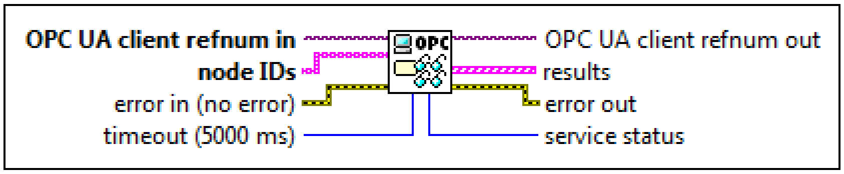
timestamp returns the date and time associated with **value**.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (SByte)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

abc **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

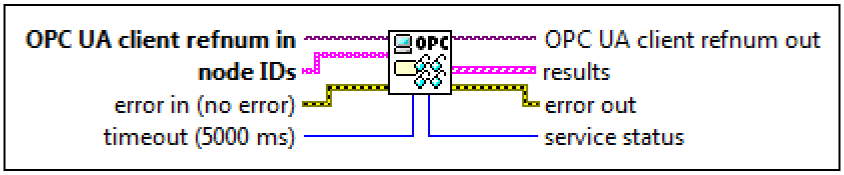
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **value** returns the value that this VI reads from the node.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Byte)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string

type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

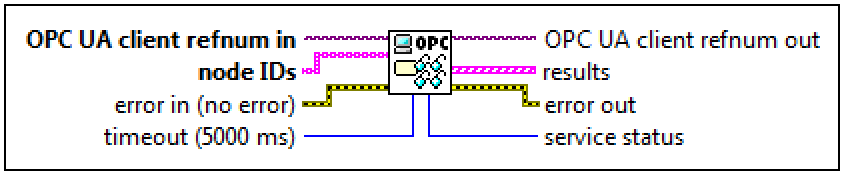
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **value** returns the value that this VI reads from the node.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Int16)



I/O

OPC UA client refnum in specifies the reference for the OPC UA client.

abc

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

value is the number read from shared memory.

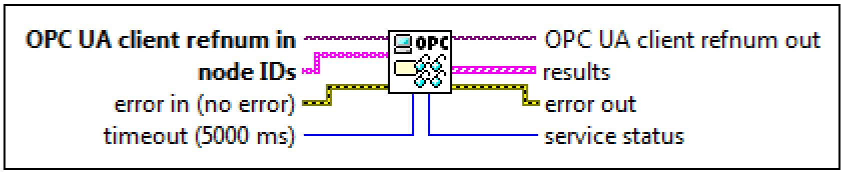
timestamp returns the date and time associated with **value**.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (UInt16)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

value returns the value that this VI reads from the node.

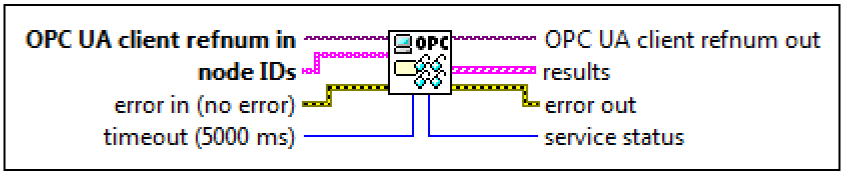
timestamp returns the date and time associated with **value**.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Int32)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string

type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

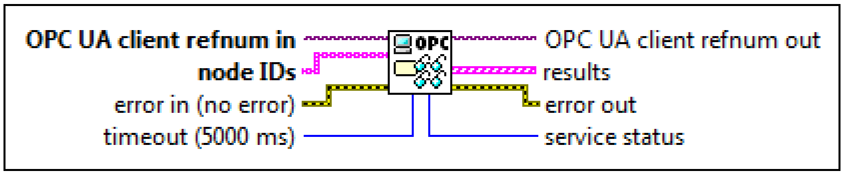
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **value** returns the value of the node.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (UInt32)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

value returns the value that this VI reads from the node.

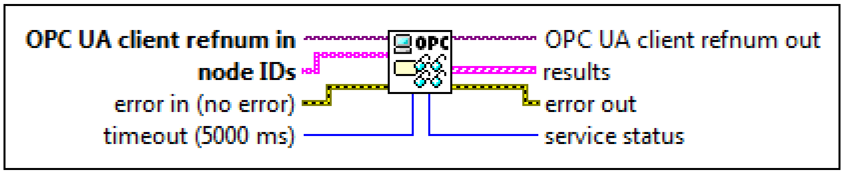
timestamp returns the date and time associated with **value**.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Int64)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

abc **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc **node ID** returns the ID of the node.

I64 **value** returns the value that this VI reads from the node.

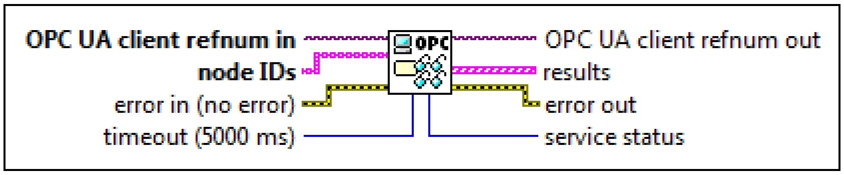
timestamp returns the date and time associated with **value**.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (UInt64)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

[abc] **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string

type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

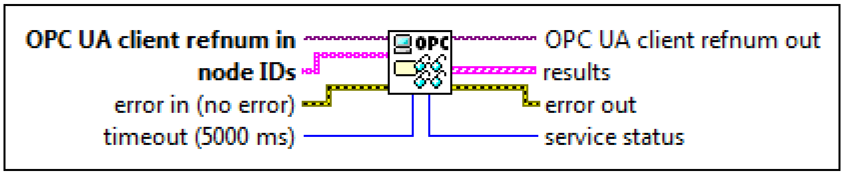
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **value** returns the value that this VI reads from the node.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Float)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

value returns the value that this VI reads from the node.

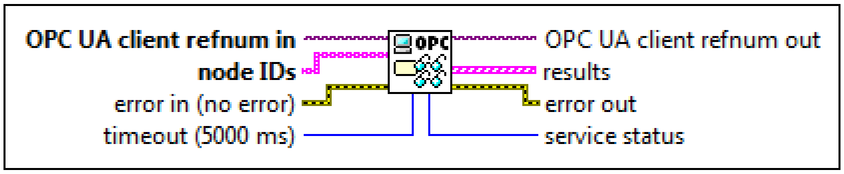
timestamp returns the date and time associated with **value**.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Double)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

abc **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

value returns the value that this VI reads from the node.

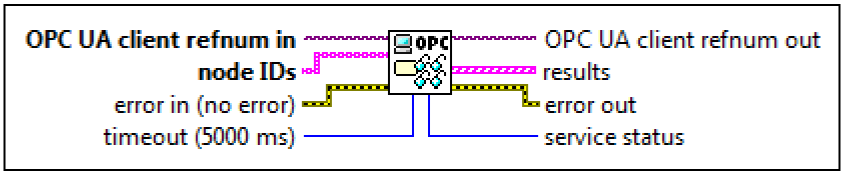
timestamp returns the date and time associated with **value**.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (String)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string

type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

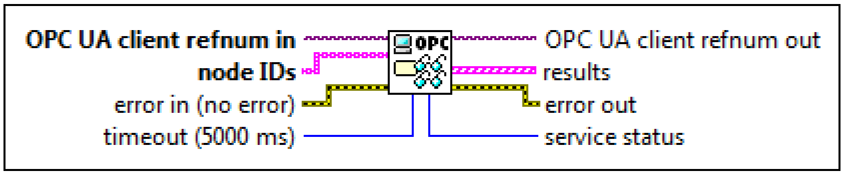
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **value** returns the value that this VI reads from the node.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (DateTime)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  > If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

value returns the value that this VI reads from the node.

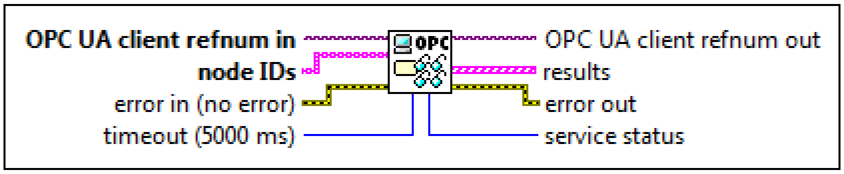
timestamp returns the date and time associated with **value**.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (ByteString)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

[abc] **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

value returns the value that this VI reads from the node.

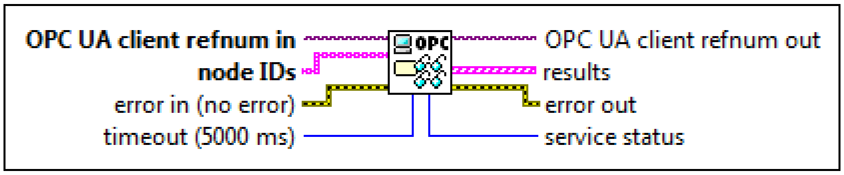
timestamp returns the date and time associated with **value**.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Bool Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string

type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

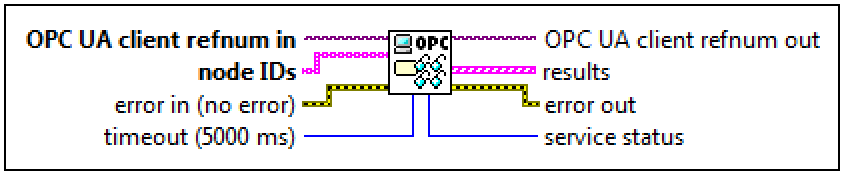
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **values** returns the values that this VI reads from the nodes.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (SByte Array)



I/0

OPC UA client refnum in specifies the reference for the OPC UA client.

abc

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

values returns the values that this VI reads from the nodes.

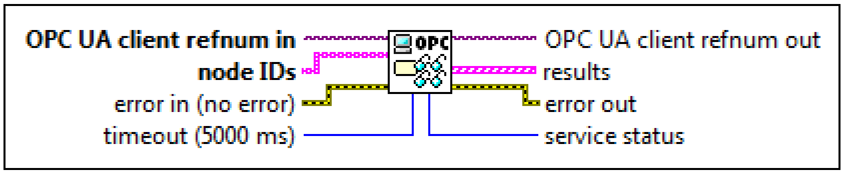
timestamp returns the date and time associated with **value**.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Byte Array)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

abc **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  > If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

values returns the values that this VI reads from the nodes.

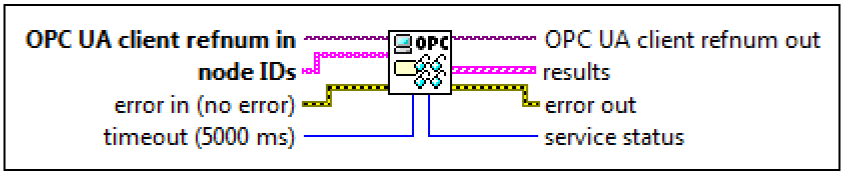
timestamp returns the date and time associated with **value**.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Int16 Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string

type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

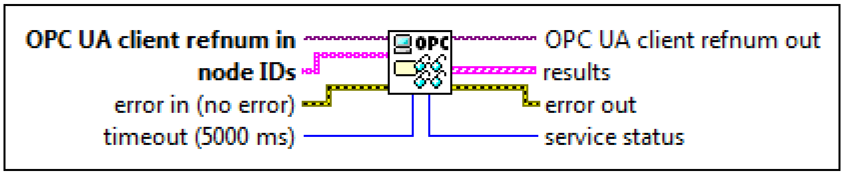
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **values** returns the values that this VI reads from the nodes.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (UInt16 Array)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

abc **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

values returns the values that this VI reads from the nodes.

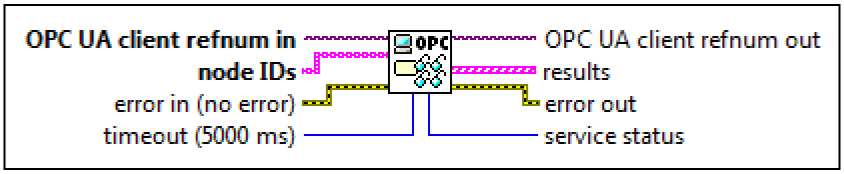
timestamp returns the date and time associated with **value**.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Int32 Array)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

abc **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

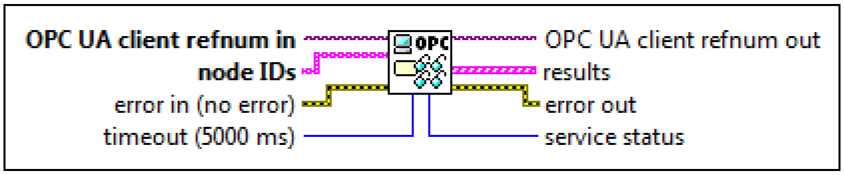
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **values** returns the values that this VI reads from the nodes.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (UInt32 Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string

type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

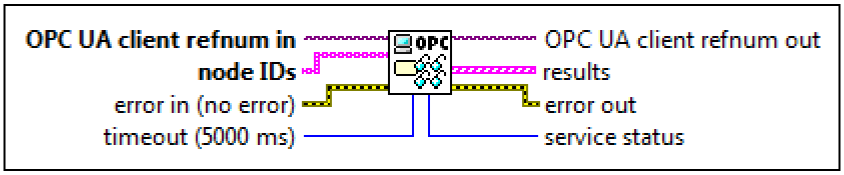
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **values** returns the values that this VI reads from the nodes.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Int64 Array)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

abc **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

values returns the values that this VI reads from the nodes.

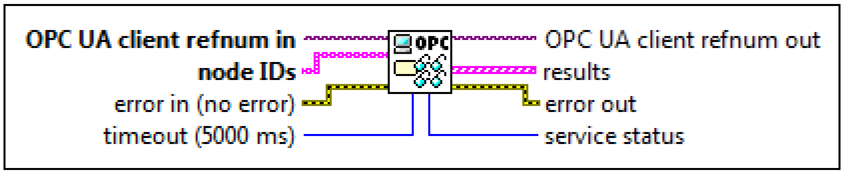
timestamp returns the date and time associated with **value**.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (UInt64 Array)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

abc **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

values returns the values that this VI reads from the nodes.

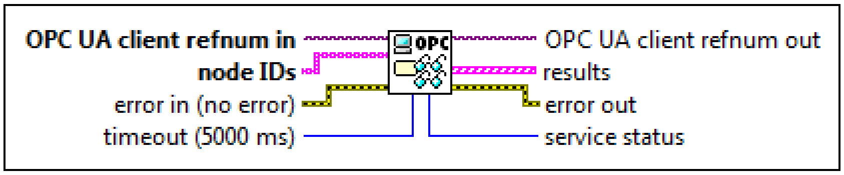
timestamp returns the date and time associated with **value**.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Float Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

> If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string

type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

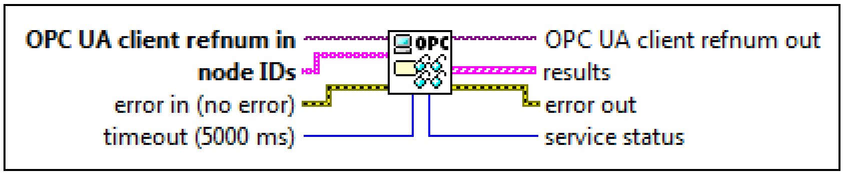
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **values** returns the values that this VI reads from the nodes.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (Double Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

values returns the values that this VI reads from the nodes.

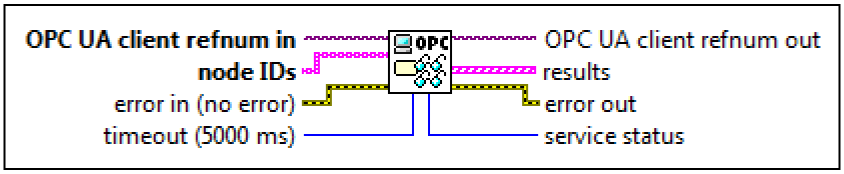
timestamp returns the date and time associated with **value**.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (String Array)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

abc **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

values returns the values that this VI reads from the nodes.

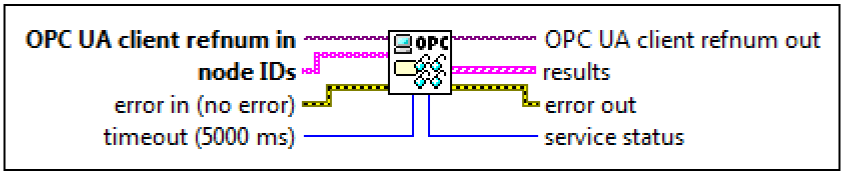
timestamp returns the date and time associated with **value**.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (DateTime Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

node IDs specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string

type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

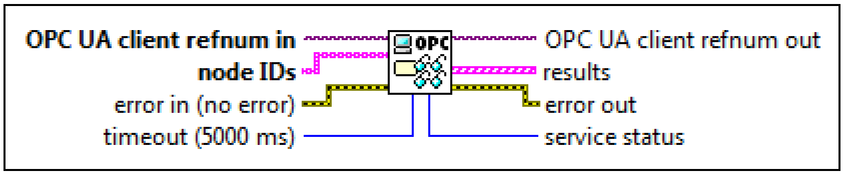
results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **values** returns the values that this VI reads from the nodes.
- **timestamp** returns the date and time associated with **value**.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Read (ByteString Array)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

abc **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the values of the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID returns the ID of the node.

values returns the values that this VI reads from the nodes.

timestamp returns the date and time associated with **value**.

U32	**status** returns the status code.
	error out contains error information. This output provides standard error out functionality.
U32	**service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Multiple Read VI.

Multiple Write VI

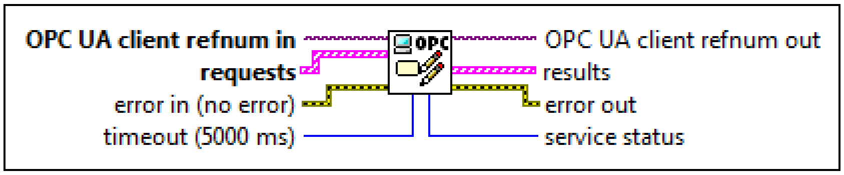
Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit

Writes values to one or multiple nodes. NI recommends choosing the Variant instance for all value data types. You must manually select the polymorphic instance to use.

Example

Multiple Write (Variant)



I/O	**OPC UA client refnum in** specifies the reference for the OPC UA client.
	requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps

associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of

elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node. **value** also supports the Matrix data type.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

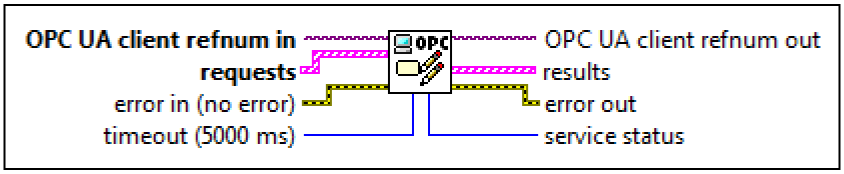
node ID returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Bool)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

TF **value** specifies the value of the node.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

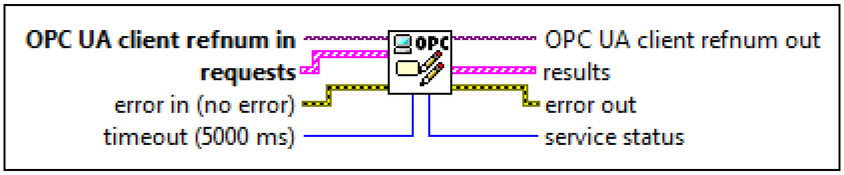
node ID returns the ID of the node.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (SByte)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

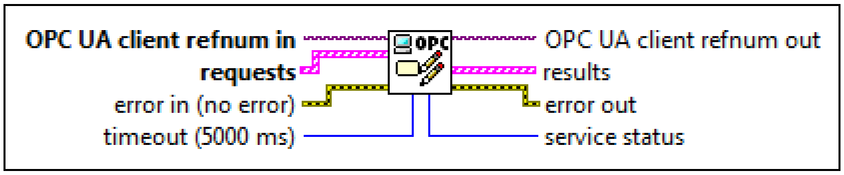
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Byte)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

U8 **value** specifies the value of the node.

timestamp specifies the date and time associated with **value**.

U32 **node status** specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

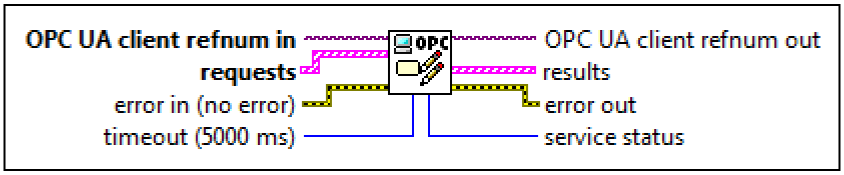
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Int16)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

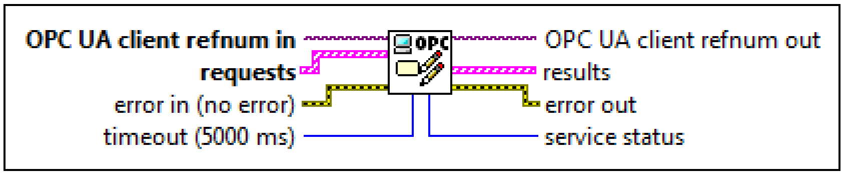
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (UInt16)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

If namespace index is 0, the format of the node ID can be <identifier type>=<identifier>. The namespace index for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

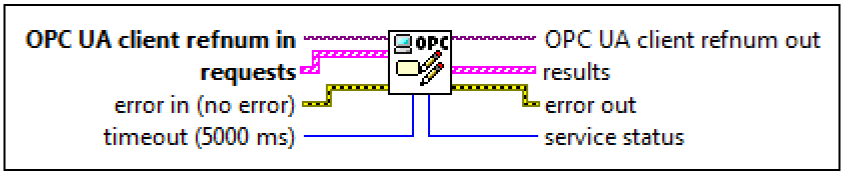
node ID returns the ID of the node.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Int32)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

I32 **value** specifies the value of the node.

timestamp specifies the date and time associated with **value**.

U32 **node status** specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

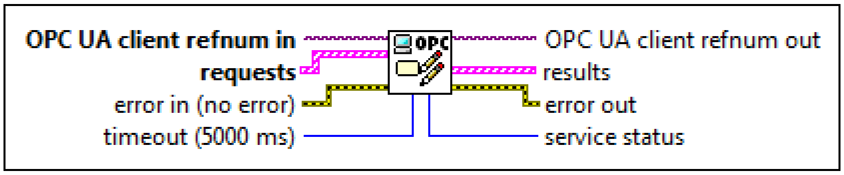
node ID returns the ID of the node.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (UInt32)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

If `name space index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

U32 **value** specifies the value of the node.

timestamp specifies the date and time associated with **value**.

U32 **node status** specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

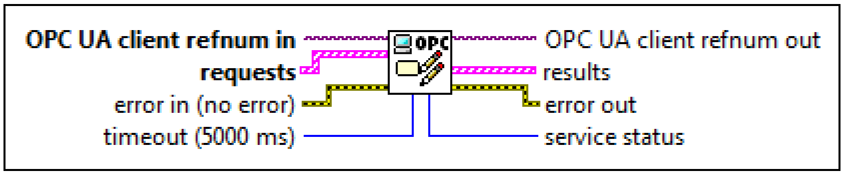
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Int64)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

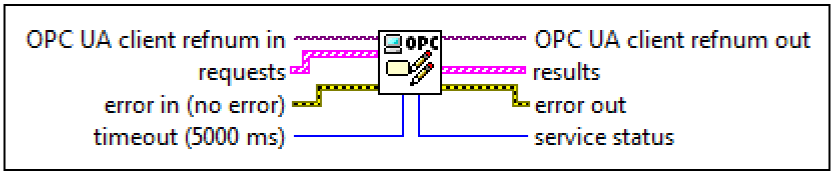
node ID returns the ID of the node.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (UInt64)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

If namespace index is 0, the format of the node ID can be <identifier type>=<identifier>. The namespace index for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

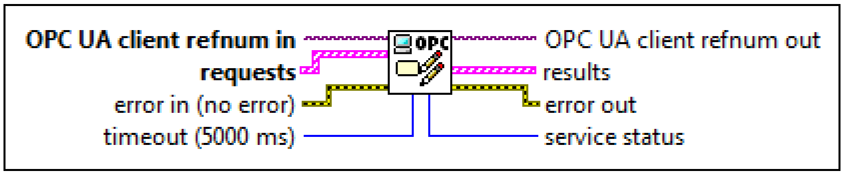
node ID returns the ID of the node.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Float)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

abc

node ID returns the ID of the node.

U32

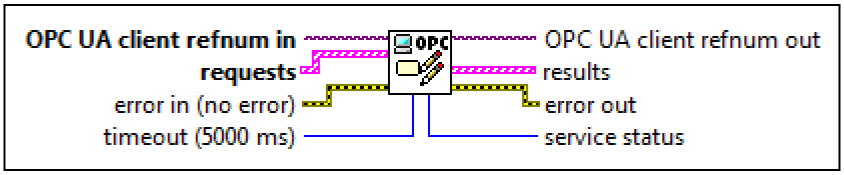
status returns the status code.

error out contains error information. This output provides standard error out functionality.

U32

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Double)



I/O

OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

DBL **value** specifies the value of the node.

timestamp specifies the date and time associated with **value**.

U32 **node status** specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

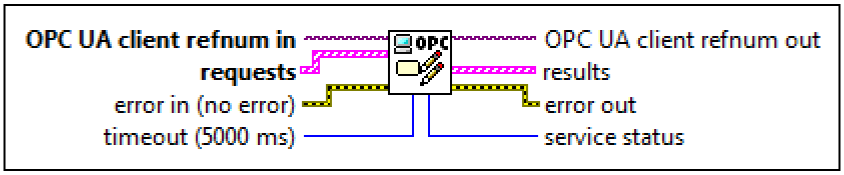
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (String)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

		This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name. For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.
	abc	**value** specifies the value of the node.
		timestamp specifies the date and time associated with **value**.
	U32	**node status** specifies the status of the node.
		error in describes error conditions that occur before this node runs. This input provides standard error in functionality.
U32		**timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.
I/O		**OPC UA client refnum out** returns the reference for the OPC UA client.
		results returns the IDs and the statuses of the nodes.

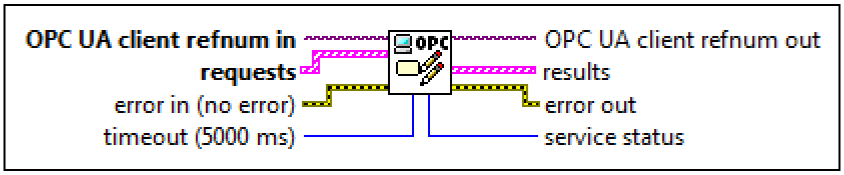
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (DateTime)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

If namespace index is 0, the format of the node ID can be <identifier type>=<identifier>. The namespace index for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

timestamp specifies the date and time associated with **value**.

U32 **node status** specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

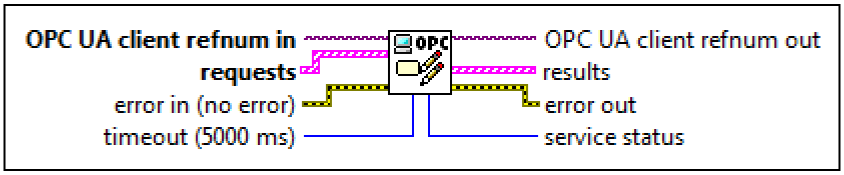
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (ByteString)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

[U8] **value** specifies the value of the node.

timestamp specifies the date and time associated with **value**.

U32 **node status** specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

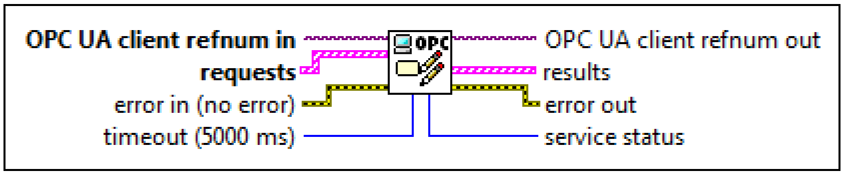
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (BoolArray)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

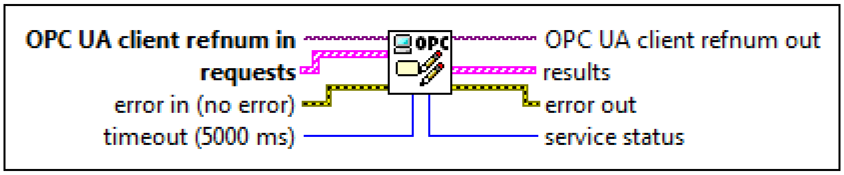
node ID returns the ID of the node.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (SByteArray)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

> If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

[I8] **values** specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

U32 **node status** specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

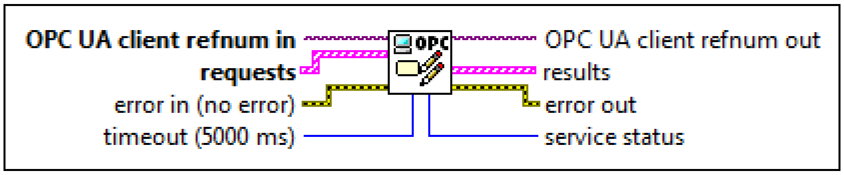
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (ByteArray)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

> If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

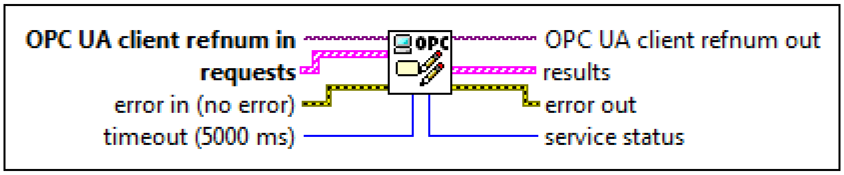
node ID returns the ID of the node.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Int16 Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

> If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

	This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name. For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.
I16	**values** specifies the values of the nodes.
	timestamp specifies the date and time associated with **value**.
U32	**node status** specifies the status of the node.

	error in describes error conditions that occur before this node runs. This input provides standard error in functionality.
U32	**timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.
I/O	**OPC UA client refnum out** returns the reference for the OPC UA client.
	results returns the IDs and the statuses of the nodes.

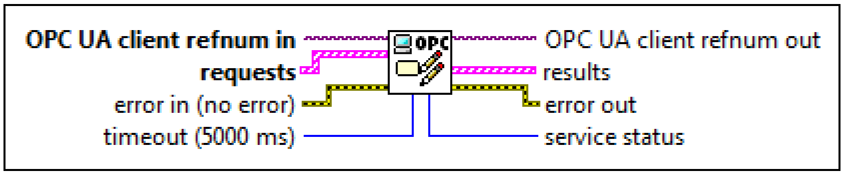
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (UInt16 Array)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>. For example, ns=2;s=Folder.Array@1:2. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be Device.folder.item.

[U16] **values** specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

U32 **node status** specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

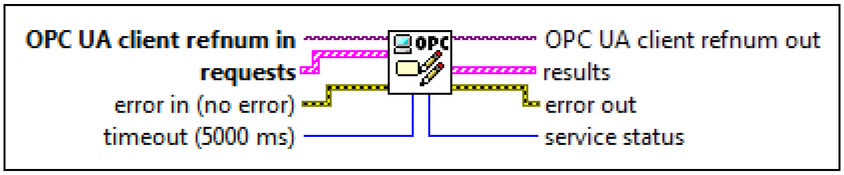
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Int32 Array)



I/O

OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

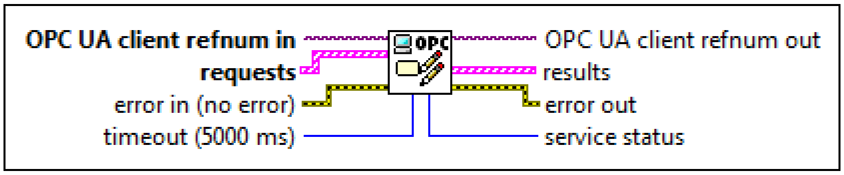
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (UInt32 Array)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

If `name space index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

[U32] **values** specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

U32 **node status** specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

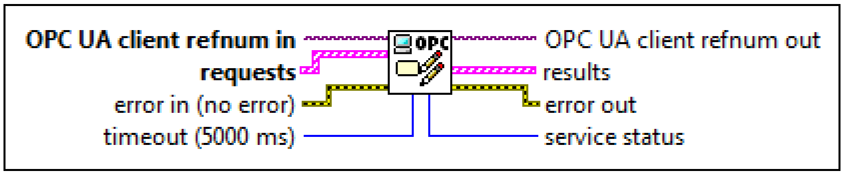
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Int64 Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

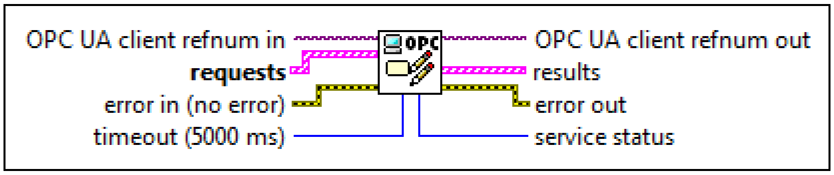
node ID returns the ID of the node.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (UInt64 Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

- **values** specifies the values of the nodes.
- **timestamp** specifies the date and time associated with **value**.
- **node status** specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

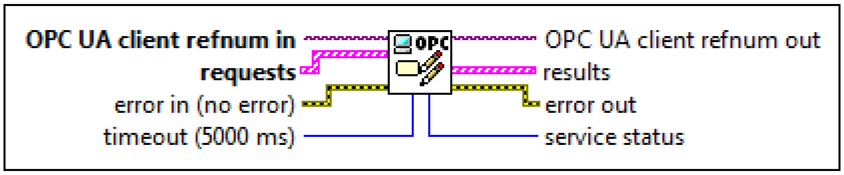
results returns the IDs and the statuses of the nodes.

- **node ID** returns the ID of the node.
- **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Float Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

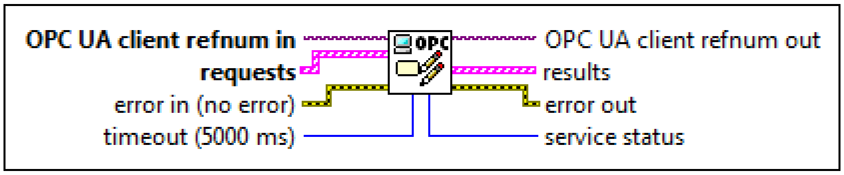
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (Double Array)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>. For example, ns=2;s=Folder.Array@1:2. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be Device.folder.item.

DBL

values specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

U32

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

abc

node ID returns the ID of the node.

U32

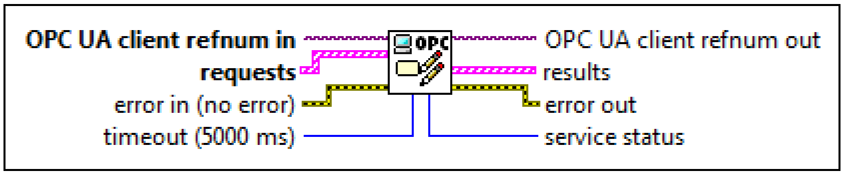
status returns the status code.

error out contains error information. This output provides standard error out functionality.

U32

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (String Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

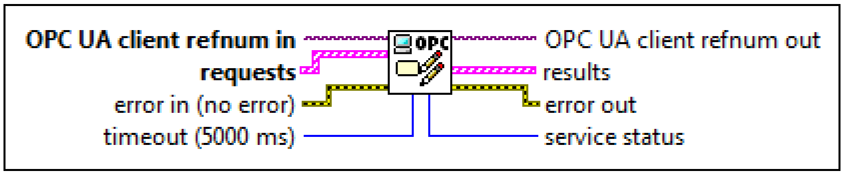
abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (DateTime Array)



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the

namespace of the node ID.

> If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric

s	String
g	GUID
b	Opaqu e

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

node status specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

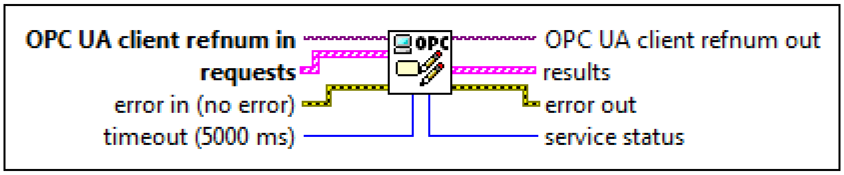
node ID returns the ID of the node.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Multiple Write (ByteString Array)



OPC UA client refnum in specifies the reference for the OPC UA client.

requests specifies the IDs of the nodes, the values to write to the nodes, the timestamps associated with the values, and the statuses of the nodes.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for

a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`.

This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

[U8] **values** specifies the values of the nodes.

timestamp specifies the date and time associated with **value**.

U32 **node status** specifies the status of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

results returns the IDs and the statuses of the nodes.

abc **node ID** returns the ID of the node.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Multiple Write VI.

Alarms and Conditions VIs

Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit. This topic might not match its corresponding palette in LabVIEW depending on your operating system, licensed product(s), and target.

Use the Alarms and Conditions VIs to subscribe to alarms and conditions for OPC UA client applications.

Example

Palette Object	Description
Add Monitored Event Nodes	Adds notifiers to an event subscription to monitor condition event notifications.
Create Event Subscription	Creates a subscription to the notifiers of an OPC UA server.
Disable Condition	Moves a condition to disabled state.
Enable Condition	Moves a condition to enabled state.
Respond Acknowledgeable Condition	Responds to an acknowledgeable condition by confirming, acknowledging, or adding comments

	to the condition. Acknowledgeable conditions expose states to indicate whether a condition must be acknowledged or confirmed.
Respond Dialog Condition	Returns a response option and ends a dialog.
Shelve Alarm Condition	Transits among the one-shot shelved state, the timed shelved state, and the unshelved state for an alarm. The one-shot shelved state prevents an alarm from displaying for its current active state. The timed shelved state prevents an alarm from displaying for a definite time period. The unshelved state allows an alarm to display. You must manually select the polymorphic instance to use.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Alarms and Conditions VIs.

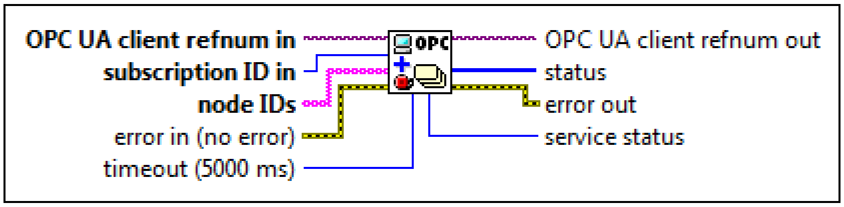
Add Monitored Event Nodes VI

Owning Palette: Alarms and Conditions VIs

Requires: OPC UA Toolkit

Adds notifiers to an event subscription to monitor condition event notifications.

Example



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

U32

subscription ID in specifies the ID of the subscription.

abc

node IDs specifies the IDs of notifier nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Add Monitored Event Nodes VI.

Create Event Subscription VI

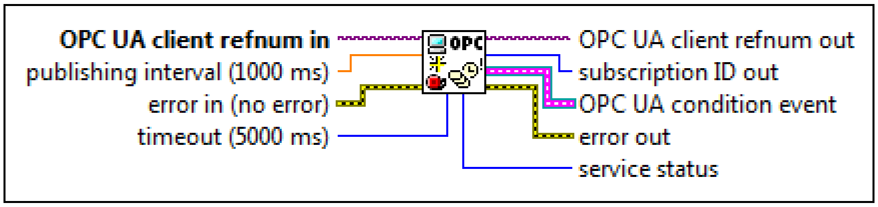
Owning Palette: Alarms and Conditions VIs

Requires: OPC UA Toolkit

Creates a subscription to the notifiers of an OPC UA server.

A notifier is an object that you can subscribe to get events from the associated condition nodes.

Example



I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

DBL **publishing interval** defines the rate, in milliseconds, that the OPC UA server returns event notifications to the OPC UA client. The default is 1000. **publishing interval** must be greater than 0.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

U32 **subscription ID out** returns the reference of the subscription that this VI accessed.

OPC UA condition event returns notifications to the OPC UA client if the client has created an event subscription. The notifications contain updates of the events on the OPC UA server. The **publishing interval** of Create Event Subscription VI defines the frequency of sending notifications.

error out contains error information. This output provides standard error out functionality.

U32

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Create Event Subscription VI.

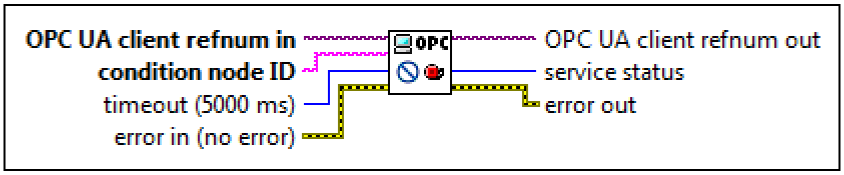
Disable Condition VI

Owning Palette: Alarms and Conditions VIs

Requires: OPC UA Toolkit

Moves a condition to disabled state.

Example



I/O

OPC UA client refnum in specifies the reference for the OPC UA client.

abc

condition node ID specifies the ID of the condition node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If namespace index is 0, the format of the node ID can be <identifier type>=<identifier>. The namespace index for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>. For example, ns=2;s=Folder.Array@1:2. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **condition node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be Device.folder.item.

U32	**timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.
	error in describes error conditions that occur before this node runs. This input provides standard error in functionality.
I/O	**OPC UA client refnum out** returns the reference for the OPC UA client.
U32	**service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.
	error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Disable Condition VI.

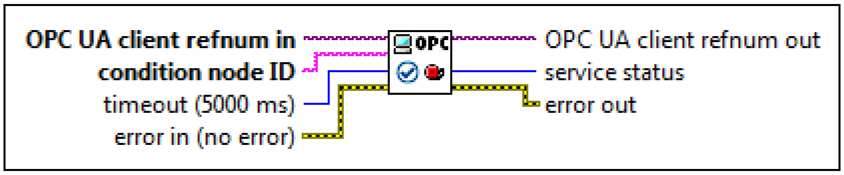
Enable Condition VI

Owning Palette: Alarms and Conditions VIs

Requires: OPC UA Toolkit

Moves a condition to enabled state.

Example



I/O	**OPC UA client refnum in** specifies the reference for the OPC UA client.

condition node ID specifies the ID of the condition node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of

elements of an array. You cannot use @ in a node name.

For backwards compatibility, **condition node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Enable Condition VI.

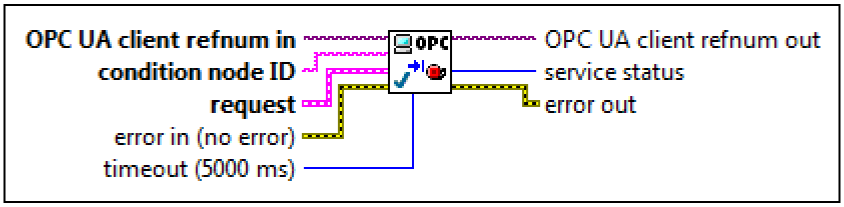
Respond Acknowledgeable Condition VI

Owning Palette: Alarms and Conditions VIs

Requires: OPC UA Toolkit

Responds to an acknowledgeable condition by confirming, acknowledging, or adding comments to the condition. Acknowledgeable conditions expose states to indicate whether a condition must be acknowledged or confirmed.

Example



I/O	**OPC UA client refnum in** specifies the reference for the OPC UA client.
abc	**condition node ID** specifies the ID of an acknowledgeable condition node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  > If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String

g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **condition node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

request specifies information that this VI uses to respond to an acknowledgeable condition.

method specifies the operation this VI executes.

0	**Add Comment**
1	**Acknowledge**
2	**Confirm**

event ID specifies the event ID.

comment specifies the string to associate with a certain state of condition.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Respond Acknowledgeable Condition VI.

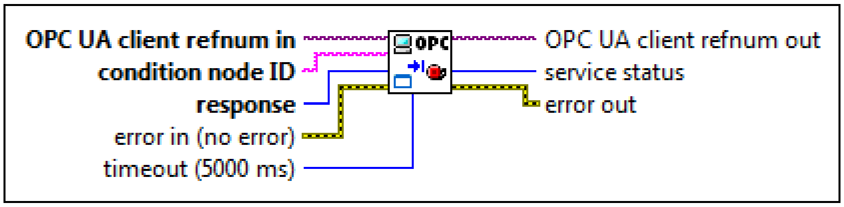
Respond Dialog Condition VI

Owning Palette: Alarms and Conditions VIs

Requires: OPC UA Toolkit

Returns a response option and ends a dialog.

Example



I/O

OPC UA client refnum in specifies the reference for the OPC UA client.

abc

condition node ID specifies the ID of the dialog condition node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **condition node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

response specifies a response option.

1	**OK**
2	**Cancel**

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Respond Dialog Condition VI.

Shelve Alarm Condition VI

Owning Palette: Alarms and Conditions VIs

Requires: OPC UA Toolkit

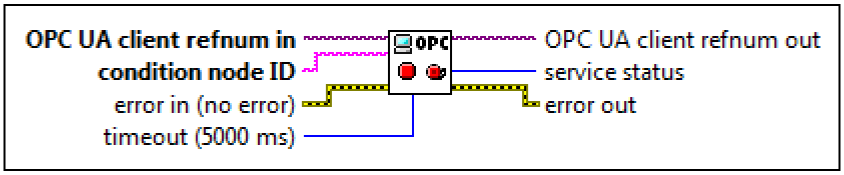
Transits among the one-shot shelved state, the timed shelved state, and the unshelved state for an alarm.

The one-shot shelved state prevents an alarm from displaying for its current active state. The timed shelved state prevents an alarm from displaying for a definite time period. The unshelved state allows an alarm to display.

You must manually select the polymorphic instance to use.

Example

One-Shot Shelve Alarm Condition



I/0

OPC UA client refnum in specifies the reference for the OPC UA client.

abc

condition node ID specifies the ID of the condition node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **condition node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

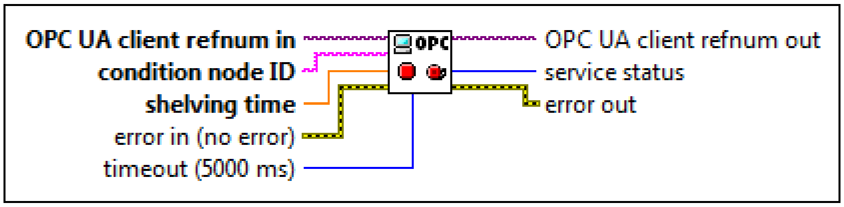
timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

error out contains error information. This output provides standard error out functionality.

Timed Shelve Alarm Condition



OPC UA client refnum in specifies the reference for the OPC UA client.

condition node ID specifies the ID of the condition node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier`

`type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **condition node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

DBL

shelving time specifies the time period, in milliseconds, during which an alarm is in a shelved state. **shelving time** should be less than the maximum time that an alarm condition is in shelved state. You can set or get the maximum time by the Max Time Shelved property.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

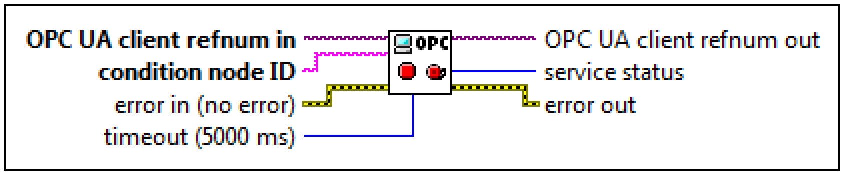
timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

service status returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

error out contains error information. This output provides standard error out functionality.

Unshelve Alarm Condition



OPC UA client refnum in specifies the reference for the OPC UA client.

condition node ID specifies the ID of the condition node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format

of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **condition node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

U32 **timeout** specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

I/O **OPC UA client refnum out** returns the reference for the OPC UA client.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Shelve Alarm Condition VI.

Historical Access VIs

Owning Palette: OPC UA Client VIs

Requires: OPC UA Toolkit. This topic might not match its corresponding palette in LabVIEW depending on your operating system, licensed product(s), and target.

Use the Historical Access VIs to access historical data and events for OPC UA client applications.

Example

Palette Object	Description
History Data Multiple Read	Reads history data for one or multiple nodes.
History Event Multiple Read	Reads one or multiple history events.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Historical Access VIs.

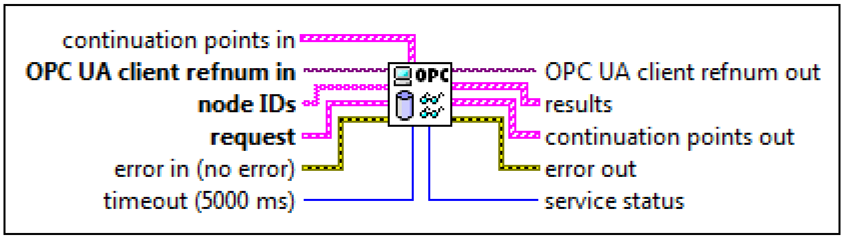
History Data Multiple Read VI

Owning Palette: Historical Access VIs

Requires: OPC UA Toolkit

Reads history data for one or multiple nodes.

Example



continuation points in specifies the continuation point and whether the history data has a continuation point.

continuation point specifies the point from which this VI continues to read if the OPC UA server cannot return all values in one response.

has continuation point? specifies whether the history data has a continuation point. The default is TRUE, which specifies that the history data has a continuation point.

I/O **OPC UA client refnum in** specifies the reference for the OPC UA client.

[abc] **node IDs** specifies the IDs of the nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<nam`

`espace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

request specifies the start time, the end time, the maximum number of values to return over the time range, and whether to return bounding values.

start time specifies the timestamp at which this VI reads the first history data.

end time specifies the timestamp at which this VI reads the last history data.

num values per node specifies the maximum number of values to return over the time range. The default is 0, which specifies that this VI returns all values over the time range.

return bounds specifies whether to return bounding values. The default is

FALSE, which specifies that this VI does not return bounding values. Bounding values are values associated with the starting time and the ending time. An OPC UA client can require bounding values to determine the starting and ending values when requesting data over a time range.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time that this VI waits to get a response from the OPC UA Server. The default value is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns the IDs of the nodes, the history data, and the statuses of the nodes.

node ID returns the ID of the node.

history data returns history data that this VI reads.

value returns the history data.

timestamp returns the

timestamp at which data is generated.

U32 **node status** returns the status of the node.

U32 **status** returns the status code.

continuation points out returns the continuation point and whether the history data has a continuation point.

continuation point returns the point from which this VI continues reading if the OPC UA server cannot return all values in one response.

TF **has continuation point?** returns whether the history data has a continuation point.

error out contains error information. This output provides standard error out functionality.

U32 **service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the History Data Multiple Read VI.

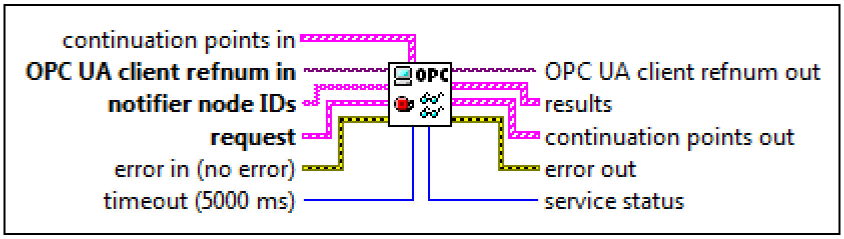
History Event Multiple Read VI

Owning Palette: Historical Access VIs

Requires: OPC UA Toolkit

Reads one or multiple history events.

Example



continuation points in specifies the continuation point and whether the history event has a continuation point.

continuation point specifies the point from which this VI continues to read if the OPC UA server cannot return all values in one response.

has continuation point? specifies whether the history event has a continuation point. The default is TRUE, which specifies that the history event has a continuation point.

OPC UA client refnum in specifies the reference for the OPC UA client.

notifier node IDs specifies the IDs of one or multiple notifier nodes. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **notifier node IDs** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

request specifies the start and end time to read history events and the maximum number of values to return for all event notifiers.

start time specifies the timestamp at which this VI reads the first history event.

end time specifies the timestamp at which this VI reads the last history event.

num values per node specifies the maximum number of events to return for any event notifier over the time range. The default is 0, which specifies that this VI returns all events over the time range.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

OPC UA client refnum out returns the reference for the OPC UA client.

results returns detailed information about the history events, the statuses of the notifier nodes, and the IDs of the notifier nodes.

notifier node ID returns the ID of the notifier.

history events returns detailed information about the history events that this VI reads.

condition type affects the validity of the output data in **history events**.

- For the dialog condition type, the validity of the output data is shown in the following table:

Output	Validity
condition node ID	valid
source node ID	valid

/input node ID	
event ID	valid
active	valid
time	valid
severity	valid
quality	valid
comment	valid
limit state	invalid
acked state	invalid
confirmed state	invalid
prompt	valid

- For the off-normal alarm condition type, the validity of the output data is shown in the following table:

Output	Validity
condition node ID	valid
source node ID /input node ID	valid
event ID	valid

active	valid
time	valid
severity	valid
quality	valid
comment	valid
limit state	invalid
acked state	valid
confirmed state	valid
prompt	invalid

- For the limit alarm condition type, the validity of the output data is shown in the following table:

Output	Validity
condition node ID	valid
source node ID/input node ID	valid
event ID	valid
active	valid
time	valid
severity	valid

quality	valid
comment	valid
limit state	valid
acked state	valid
confirmed state	valid
prompt	invalid

condition node ID

returns the ID of the condition node.

condition type

returns the type of the condition node.

0	**Dialog Condition**
1	**Off-Normal Alarm**

2	Exclusive Level Alarm
3	Exclusive Deviation Alarm
4	Exclusive Rate of Change Alarm
5	Non Exclusive Level Alarm

6	Non Exclusive Deviation Alarm
7	Non Exclusive Rate of Change Alarm
8	Refresh Start Event
9	Refresh End Event

abc **source node ID/input node ID**

returns

the ID of the source node or the input node. For nodes of the dialog condition type, **source node ID/ input node ID** returns the ID of the source node. For nodes of the off-normal alarm type and the limit alarm type, **source node ID/ input node ID** returns the ID of the input node. You can use the Add Condition VI to add dialog condition nodes, off-normal alarm nodes, or limit alarm nodes.

abc **event ID**

returns the event ID.

TF **active**

returns whether the situation the alarm is representing exists.

time

returns the timestamp at which the event occurs.

U16 **severity**

returns a value that indicates the urgency of the event. The value ranges from 1 to 1000, with 1 being the lowest severity and 1000 being the highest severity.

I32 **quality**

returns the status code to display the quality of the values that the condition depends on.

abc **comment**

returns the string to associate with a certain state of condition.

limit state

returns the state of the alarm.

TF **high high state**

returns whether

the condition value exceeds the high high limit.

TF **high state**

returns whether the condition value exceeds the high limit.

TF **low state**

returns whether the

condition value exceeds the low limit.

TF **low low state**

returns whether the condition value exceeds the low low limit.

TF **acked state**

returns whether the condition is acknowledged.

confirmed state returns whether the condition is confirmed.

prompt returns the textual content of a dialog prompt only when the condition type is dialog condition.

status returns the status code.

continuation points out returns the continuation point and whether the history event has a continuation point.

continuation point returns the point from which this VI continues reading if the OPC UA server cannot return all values in one response.

has continuation point? returns whether the history event has a continuation point.

	error out contains error information. This output provides standard error out functionality.
U32	**service status** returns the status of an OPC UA service call. OPC UA services contain parameters that are conveyed between an OPC UA client and an OPC UA server.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the History Event Multiple Read VI.

OPC UA Server VIs

Owning Palette: OPC UA VIs

Requires: OPC UA Toolkit. This topic might not match its corresponding palette in LabVIEW depending on your operating system, licensed product(s), and target.

Use the OPC UA Server VIs to create a customized OPC UA server application.

Example

Palette Object	Description
Add Analog Item	Adds an analog item to the address space and configures the analog item.
Add Folder	Creates a folder under the parent folder in the address space.
Add Item	Adds an item as a child to a folder you specify.
Add Property	Adds a property to an item.
Add Trusted Clients	Adds trusted OPC UA client certificates to an OPC UA server.
Clear All Trusted Clients	Clears the OPC UA client certificates that an OPC UA server trusts.
Close	Closes and destroys an OPC UA server. After you close an OPC UA server, you lose all the data that belongs to the OPC UA server.

Create	Creates and initializes an OPC UA server.
Delete Node	Deletes a node, such as a folder, item, notifier, or condition. When you delete a folder, an item, or a notifier, you delete all the child nodes.
Read	Reads the value, timestamp, and status of a node. You must manually select the polymorphic instance to use.
Register Server	Registers an OPC UA server with the UA Local Discovery Server (LDS). This VI routinely registers the OPC UA server based on **register rate**.
Start	Starts an OPC UA server. After you start an OPC UA server, you cannot add or delete folders, items, notifiers, or conditions until the OPC UA server stops. Use the Stop VI to stop an OPC UA server.
Stop	Stops an OPC UA server and disconnects all OPC UA clients. When you stop an OPC UA server, you still keep all the folders, items, and properties. Use the Start VI to restart the OPC UA server.
Unregister Server	Unregisters a registered OPC UA server with the UA Local Discovery Server (LDS).
Write	Writes the value and status to a node from an OPC UA server. You must manually select the polymorphic instance to use. You can write to a node before the OPC UA server starts.

Subpalette	Description
Alarms and Conditions VIs	Use the Alarms and Conditions VIs to subscribe to alarms and conditions for OPC UA server applications.
Historical Access VIs	Use the Historical Access VIs to access historical data and events for OPC UA server applications.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the OPC UA Server VIs.

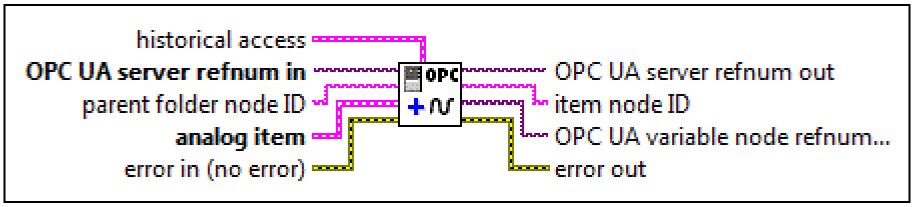
Add Analog Item VI

Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Adds an analog item to the address space and configures the analog item.

Example



historical access specifies whether the analog item supports historical access and the size of the history data queue.

TF **enable** specifies whether the analog item supports historical access. The default is FALSE, which specifies that the analog item does not support historical access.

U32 **queue size** specifies the size of the queue of history data. The default is 1000.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **parent folder node ID** specifies the ID of the parent folder. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. If the parent folder does not exist, this VI creates a folder at the root level. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility,

parent folder node ID also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

analog item specifies configuration information about the analog item.

abc **name** specifies the name of the analog item.

U16 **data type** specifies the data type of the analog item.

2	SByte
3	Byte
4	Int16
5	UInt16
6	Int32
7	UInt32
8	Int64
9	UInt64
10	Float
11	Double
102	Array of SByte
103	Array of Byte
104	Array of Int16
105	Array of UInt16

106	Array of Int32
107	Array of UInt32
108	Array of Int64
109	Array of UInt64
110	Array of Float
111	Array of Double
202	Matrix of SByte
203	Matrix of Byte
204	Matrix of Int16
205	Matrix of UInt16
206	Matrix of Int32
207	Matrix of UInt32
208	Matrix of Int64
209	Matrix of UInt64
210	Matrix of Float
211	Matrix of Double

U16 **access level** specifies the access method for the analog item.

1	read-only
2	write-only
3	read/write

instrument range specifies the range of the values that the analog item returns.

- DBL **low** specifies the low limit of the value range that the analog item returns.
- DBL **high** specifies the high limit of the value range that the analog item returns.

EU range specifies the range of the value that you can obtain in normal operation.

DBL **low**

specifies the low limit of the value range that you can obtain in normal operation.

DBL **high**

specifies the high limit of the value range that you can obtain in normal operation.

I32 **engineering units**

specifies the units of the value of the analog item. The default is **mutually defined**.

Select **mutually defined** to customize a unit that acquires mutual acknowledgement.

The OPC UA Foundation recommends using the Codes for Units of Measurement. Refer to **List of Trade Facilitation**

Recommendations N° 20 for details.

abc **description** specifies additional information of the analog item.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

I/O **OPC UA server refnum out** returns the reference data value of the OPC UA server.

abc **item node ID** returns the ID of the analog item.

I/O **OPC UA variable node refnum out** returns the reference to the variable node.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Add Analog Item VI.

Add Folder VI

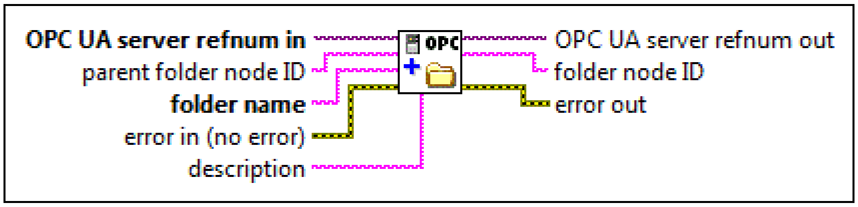
Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Creates a folder under the parent folder in the address space.

After you start an OPC UA server, you cannot add folders until the OPC UA server stops. Use the Stop VI to stop an OPC UA server.

Example



I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

parent folder node ID specifies the ID of the parent folder. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. If the parent folder does not exist, this VI creates a folder at the root level. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **parent folder node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

folder name specifies the name of the folder to create. If a folder with the same name already exists in the parent folder, this VI returns an error.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

description specifies additional information about the folder.

OPC UA server refnum out returns the reference data value of the OPC UA server.

folder node ID returns the ID of the new folder that this VI creates.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Add Folder VI.

Add Item VI

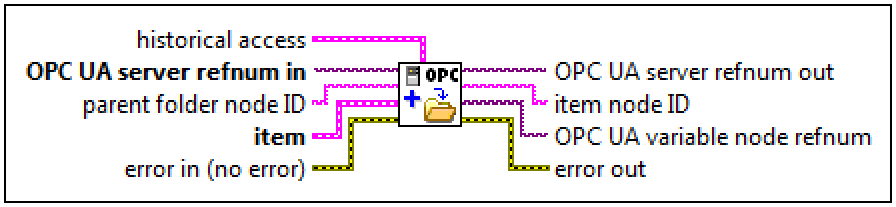
Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Adds an item as a child to a folder you specify.

After you start an OPC UA server, you cannot add items until the OPC UA server stops. Use the Stop VI to stop an OPC UA server.

Example



historical access specifies whether the item supports historical access and the size of the history data queue.

TF **enable** specifies whether the item supports historical access. The default is FALSE, which specifies that this item does not support historical access.

U32 **queue size** specifies the size of the queue of history data. The default is 1000.

I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

parent folder node ID specifies the ID of the parent folder. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. If the parent folder does not exist, this VI creates a folder at the root level. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<`

`identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **parent folder node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

item specifies the item attributes.

name specifies the name of the item. If an item or folder with the same name already exists in the parent folder, this VI returns an error.

data type specifies the data type of the item.

1	Boolean
2	SByte
3	Byte
4	Int16
5	UInt16
6	Int32
7	UInt32
8	Int64
9	UInt64

10	Float
11	Double
12	String
13	DateTime
15	ByteString
101	Array of Boolean
102	Array of SByte
103	Array of Byte
104	Array of Int16
105	Array of UInt16
106	Array of Int32
107	Array of UInt32
108	Array of Int64
109	Array of UInt64
110	Array of Float
111	Array of Double
112	Array of String

113	Array of DateTime
115	Array of ByteString
201	Matrix of Boolean
202	Matrix of SByte
203	Matrix of Byte
204	Matrix of Int16
205	Matrix of UInt16
206	Matrix of Int32
207	Matrix of UInt32
208	Matrix of Int64
209	Matrix of UInt64
210	Matrix of Float
211	Matrix of Double
212	Matrix of String
213	Matrix of DateTime

215	Matrix of ByteString

access level specifies the access method for the item.

1	read-only
2	write-only
3	read/write

description specifies a free-form text string that you enter to describe this item.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

item node ID returns the ID of the item node that this VI adds.

OPC UA variable node refnum out returns the reference to the variable node.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Add Item VI.

Add Property VI

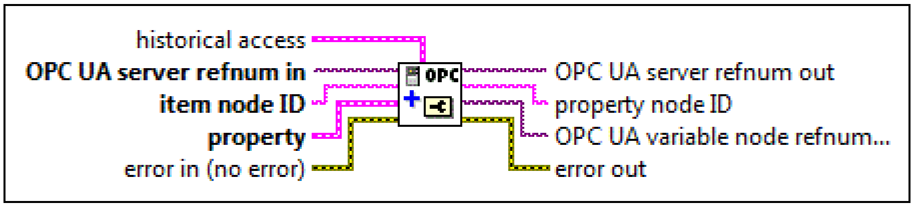
Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Adds a property to an item.

After you start an OPC UA server, you cannot add properties until the OPC UA server stops. Use the Stop VI to stop an OPC UA server.

Example



historical access specifies whether the item supports historical access and the size of the history data queue.

TF **enable** specifies whether the item supports historical access. The default is FALSE, which specifies that this item does not support historical access.

U32 **queue size** specifies the size of the queue of history data. The default is 1000.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **item node ID** specifies the ID of the parent node. The parent node must be an item.

property specifies the information about the property.

abc

name specifies the name of the property. If a property with the same name already exists in the parent item, this VI returns an error.

U16

data type specifies the data type of the property.

1	Boolean
2	SByte
3	Byte
4	Int16
5	UInt16
6	Int32
7	UInt32
8	Int64
9	UInt64
10	Float
11	Double
12	String
13	DateTime
15	ByteString
101	Array of Boolean
102	Array of SByte

103	Array of Byte
104	Array of Int16
105	Array of UInt16
106	Array of Int32
107	Array of UInt32
108	Array of Int64
109	Array of UInt64
110	Array of Float
111	Array of Double
112	Array of String
113	Array of DateTime
115	Array of ByteString
201	Matrix of Boolean
202	Matrix of SByte
203	Matrix of Byte

204	Matrix of Int16
205	Matrix of UInt16
206	Matrix of Int32
207	Matrix of UInt32
208	Matrix of Int64
209	Matrix of UInt64
210	Matrix of Float
211	Matrix of Double
212	Matrix of String
213	Matrix of DateTime
215	Matrix of ByteString

U16

access level specifies whether you can read or write to the property.

Value	Access Level
1	read-only

2	write-only
3	read/write

description specifies a free-form text string that you enter to describe this property.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

property node ID returns the ID of the property node that this VI adds.

OPC UA variable node refnum out returns the reference to the variable node.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Add Property VI.

Add Trusted Clients VI

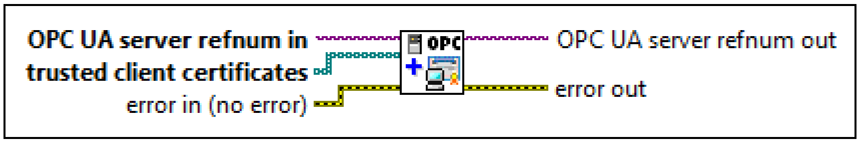
Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Adds trusted OPC UA client certificates to an OPC UA server.

The OPC UA client uses the security settings in communication only when the OPC UA server trusts the OPC UA client. Otherwise, the OPC UA client can only communicate with an unsecured server. By default, the OPC UA server application that you create trusts the default OPC UA client certificate.

After you start an OPC UA server, you cannot add trusted client certificates until the OPC UA server stops. Use the Stop VI to stop an OPC UA server.



OPC UA server refnum in specifies the reference data value of the OPC UA server.

trusted client certificates specifies the file paths of the public keys that the OPC UA server trusts. You can specify a relative path or filename for **trusted client certificates**. If you specify a relative path, the path is relative to the caller VI or to the application directory. If you specify a filename, LabVIEW searches the certificate files only in the location where the caller VI resides or in the application directory.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

error out contains error information. This output provides standard error out functionality.

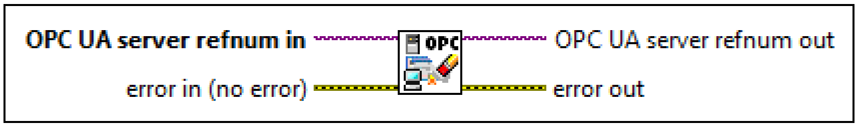
Clear All Trusted Clients VI

Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Clears the OPC UA client certificates that an OPC UA server trusts.

After you start an OPC UA server, you cannot clear the trusted client certificates until the OPC UA server stops. Use the Stop VI to stop an OPC UA server.



OPC UA server refnum in specifies the reference data value of the OPC UA server.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

error out contains error information. This output provides standard error out functionality.

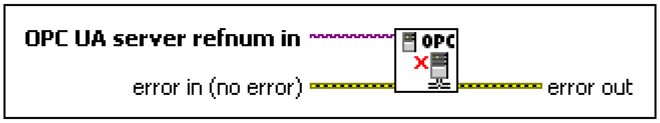
Close VI

Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Closes and destroys an OPC UA server. After you close an OPC UA server, you lose all the data that belongs to the OPC UA server.

Example



OPC UA server refnum in specifies the reference data value of the OPC UA server.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Close VI.

Create VI

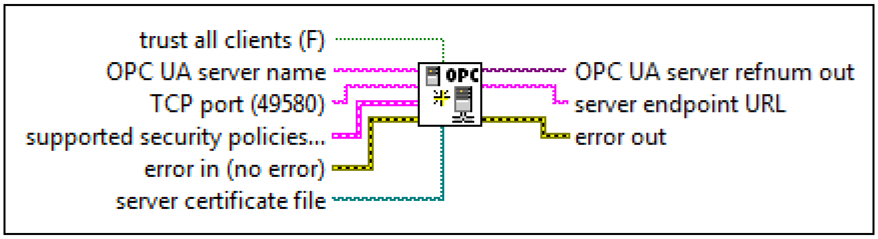
Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Creates and initializes an OPC UA server.

When you use the Connect VI to connect an OPC UA client to an OPC UA server and set the **message mode** as `None`, the OPC UA client and OPC UA server do not need to trust each other.

Example



TF

trust all clients specifies whether the OPC UA server trusts all OPC UA clients without adding the trusted client certificates to the OPC UA server. The default is FALSE, which specifies that the OPC UA server trusts only OPC UA clients that have trusted OPC UA client certificates added to the OPC UA server. You can use the Add Trusted Clients VI to add trusted OPC UA client certificates to an OPC UA server.

abc

OPC UA server name specifies the name of the OPC UA server.

abc

TCP port specifies the TCP port that the OPC UA server takes. The default is 49580. If the port that you specify is occupied, this VI is unable to create an OPC UA server. You can create multiple OPC UA servers on the same target with different ports.

supported security policies specifies the supported message modes and corresponding security policies. The default is `None`, which means the OPC UA client can connect without security. Use the Connect VI to determine which supported security policy to use when connecting the OPC UA client to the OPC UA server.

None specifies that the OPC UA server supports no security. The OPC UA client can connect without security.

Sign with Basic128Rsa15 specifies that the OPC UA server supports signing messages using the Basic128Rsa15 security policy.

Sign and Encrypt with Basic128Rsa15 specifies that the OPC UA server supports signing and encrypting messages using the Basic128Rsa15 security policy.

Sign with Basic256 specifies that the OPC UA server supports signing messages using the Basic256 security policy.

Sign and Encrypt with Basic256 specifies that the OPC UA server supports signing and encrypting messages using the Basic256 security policy.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

server certificate file specifies the path or name of the public key. You must name the public key and private key as the same and keep them in the same folder. If you do not specify the certificate file path, this VI generates and uses the default certificate. If you specify your own certificate, this VI uses and trusts your certificate. The OPC UA server always trusts the certificate that the OPC UA server is using.

You can specify a relative path or filename for **server certificate file**. If you specify a relative path, the path is relative to the caller VI or to the application directory. If you specify a filename, LabVIEW searches the certificate file only in the location where the caller VI resides or in the application directory.

OPC UA server refnum out returns the reference data value of the OPC UA server.

server endpoint URL returns the unique identifier of the OPC UA server.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Create VI.

Delete Node VI

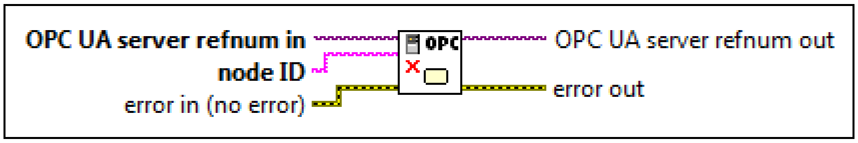
Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Deletes a node, such as a folder, item, notifier, or condition. When you delete a folder, an item, or a notifier, you delete all the child nodes.

After you start an OPC UA server, you cannot delete nodes until the OPC UA server stops. Use the Stop VI to stop an OPC UA server.

Example



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Delete Node VI.

Read VI

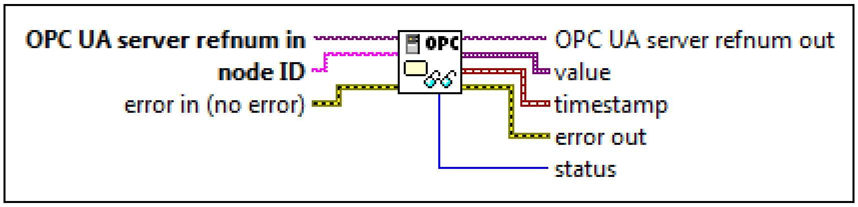
Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Reads the value, timestamp, and status of a node. You must manually select the polymorphic instance to use.

Example

Read (Variant)



OPC UA server refnum in specifies the reference data value of the OPC UA server.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format

of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

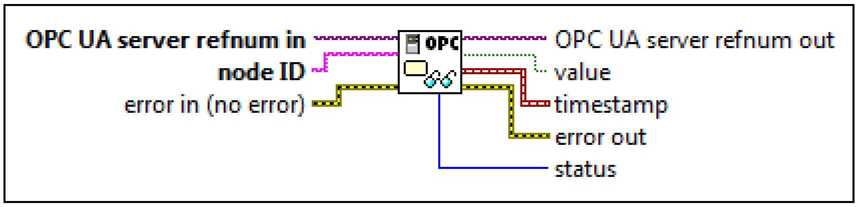
value returns the value of the node. **value** also supports the Matrix data type.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Bool)



OPC UA server refnum in specifies the reference data value of the OPC UA server.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you

created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

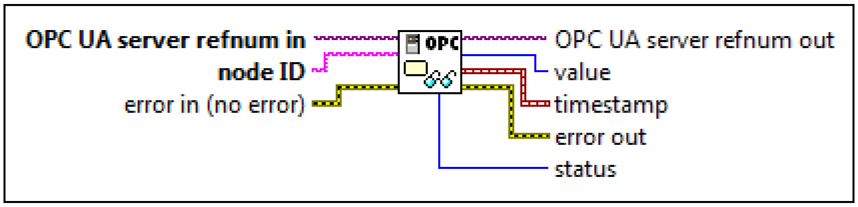
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (SByte)



OPC UA server refnum in specifies the reference data value of the OPC UA server.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  > If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

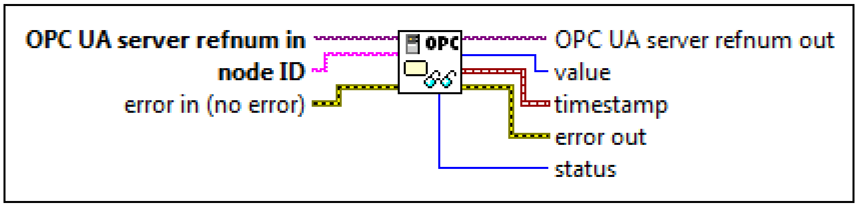
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

U32 **status** returns the status code.

Read (Byte)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

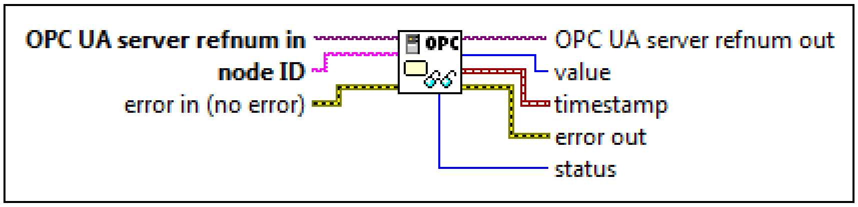
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Int16)



I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

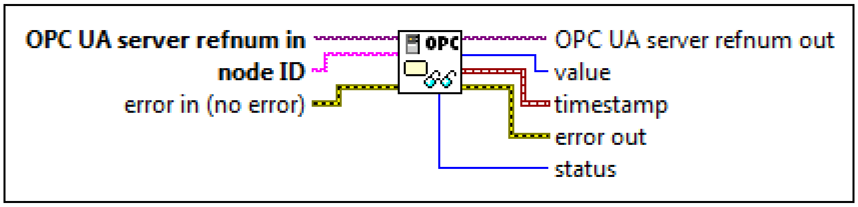
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (UInt16)



I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

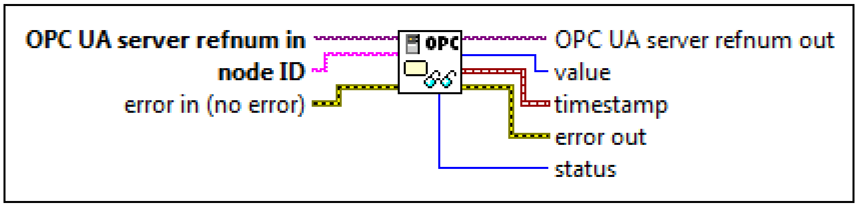
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Int32)



OPC UA server refnum in specifies the reference data value of the OPC UA server.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

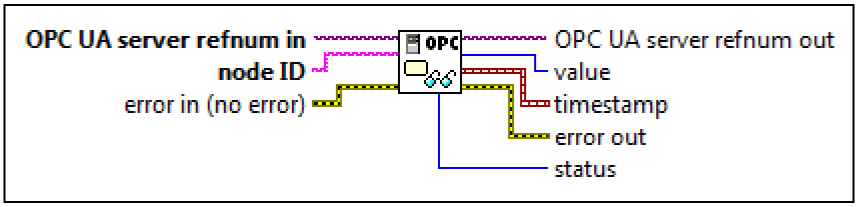
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (UInt32)



I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

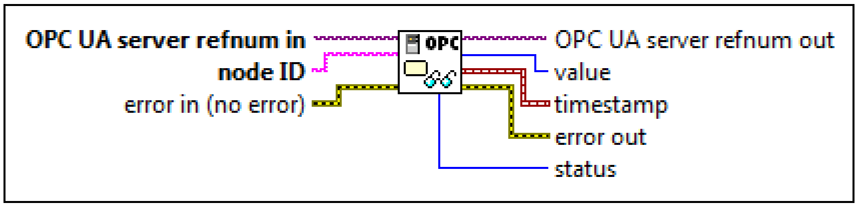
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Int64)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

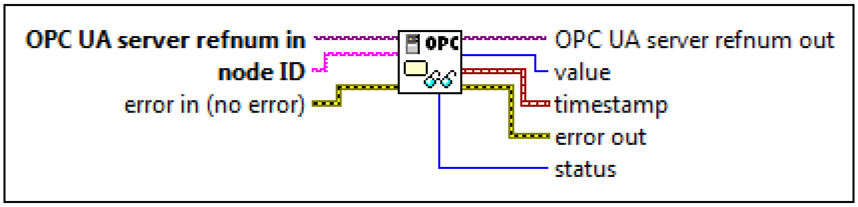
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (UInt64)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

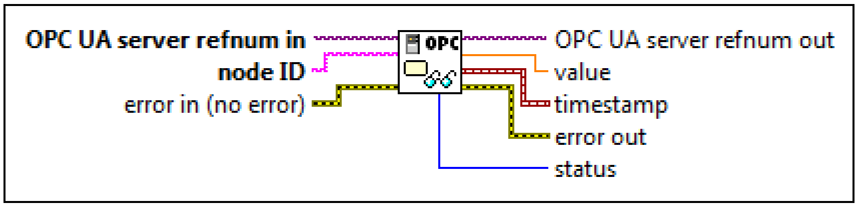
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Float)



OPC UA server refnum in specifies the reference data value of the OPC UA server.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

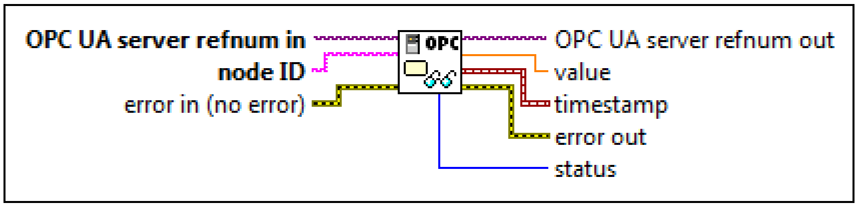
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Double)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

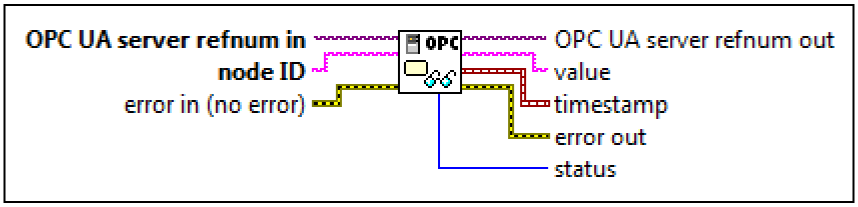
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (String)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

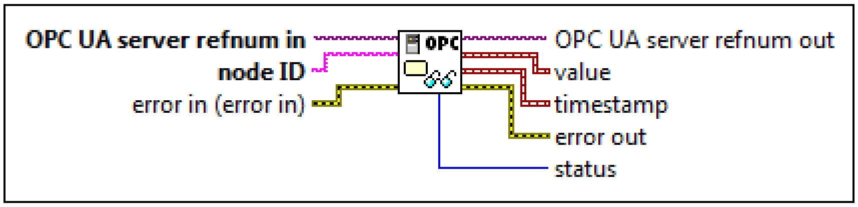
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (DateTime)



I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

  - `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

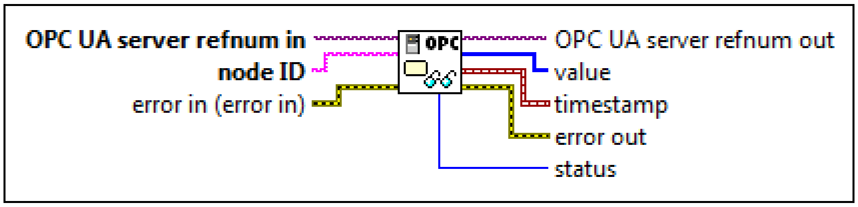
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (ByteString)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

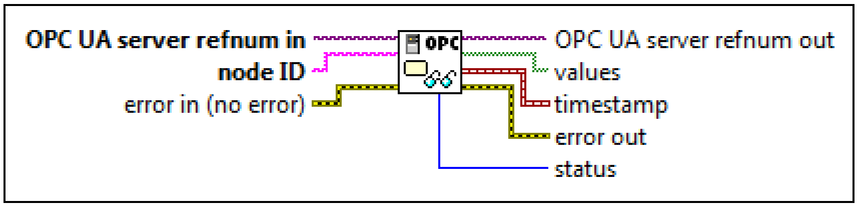
value returns the value of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Bool Array)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

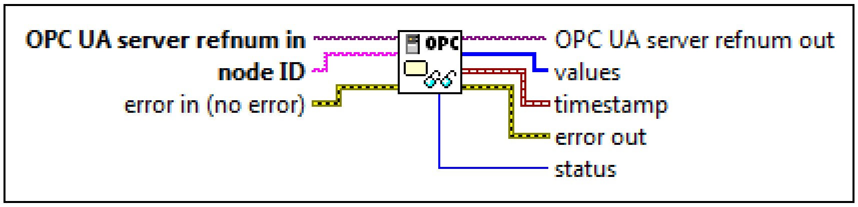
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (SByte Array)



OPC UA server refnum in specifies the reference data value of the OPC UA server.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

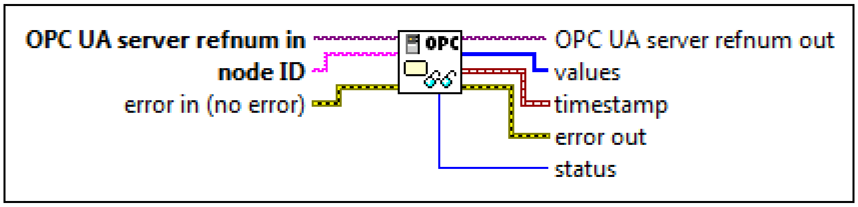
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Byte Array)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

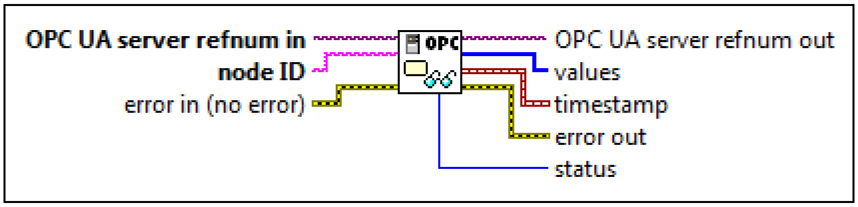
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Int16 Array)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  > If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

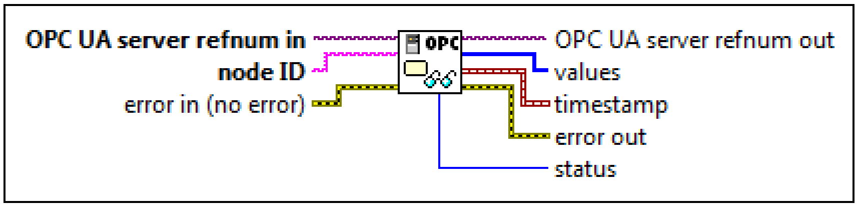
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (UInt16 Array)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

I/O **OPC UA server refnum out** returns the reference data value of the OPC UA server.

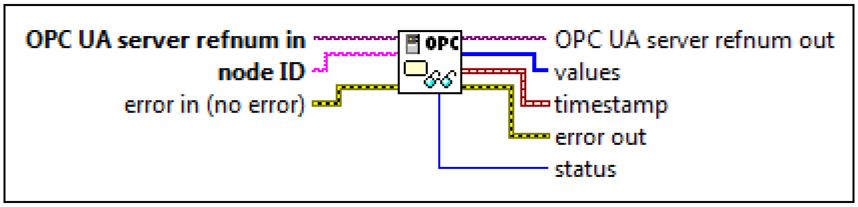
[U16] **values** returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

U32 **status** returns the status code.

Read (Int32 Array)



I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

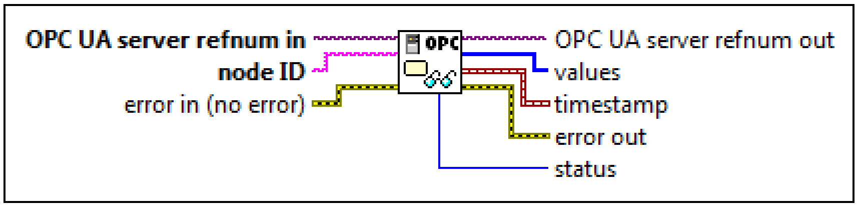
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (UInt32 Array)



I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

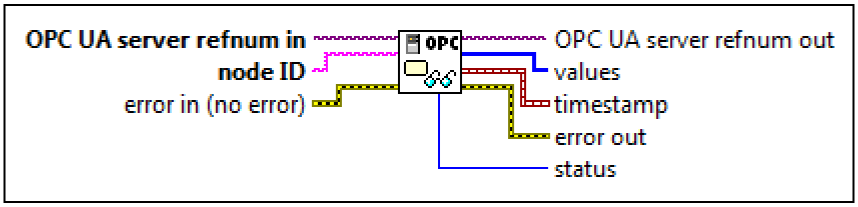
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Int64 Array)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

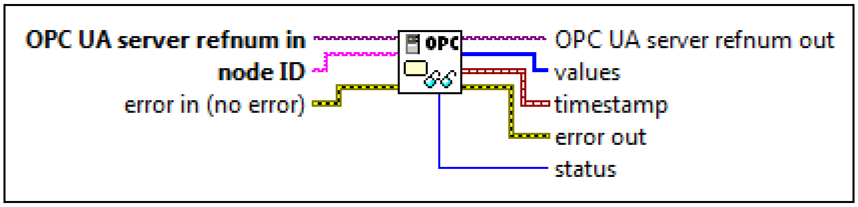
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (UInt64 Array)



I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

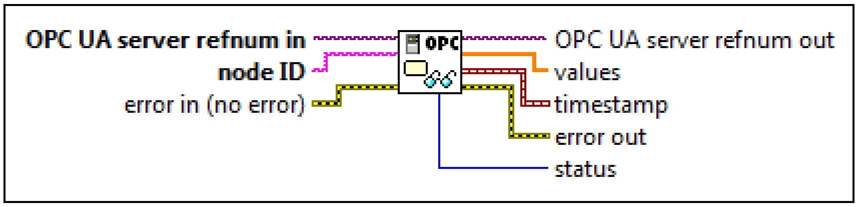
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Float Array)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

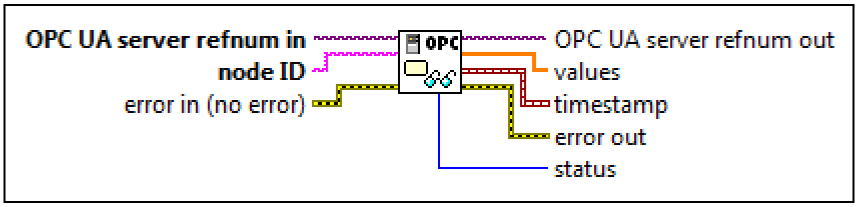
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (Double Array)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

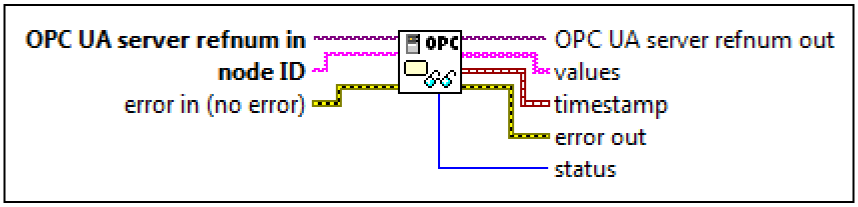
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (String Array)



I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  > If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

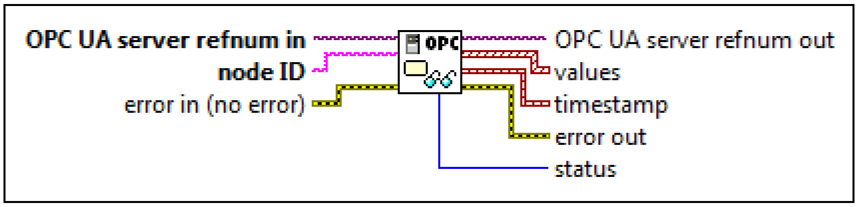
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (DateTime Array)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  > If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

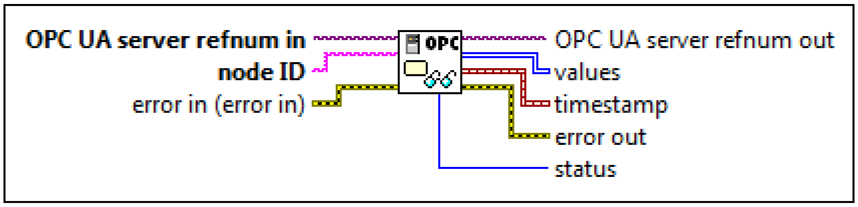
values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Read (ByteString Array)



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

values returns the values of the node.

timestamp returns the date and time associated with **value**.

error out contains error information. This output provides standard error out functionality.

status returns the status code.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Read VI.

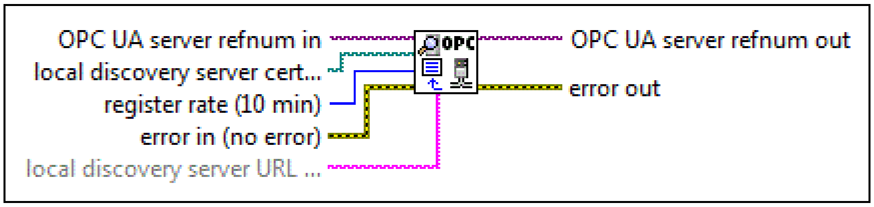
Register Server VI

Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Registers an OPC UA server with the UA Local Discovery Server (LDS). This VI routinely registers the OPC UA server based on **register rate**.

Details



I/O	**OPC UA server refnum in** specifies the reference data value of the OPC UA server.
	local discovery server certificate file specifies the path of the certificate file that the UA Local Discovery Server (LDS) uses.
U32	**register rate** specifies the rate, in minutes, at which this VI registers the OPC UA server. The default is 10.
	error in describes error conditions that occur before this node runs. This input provides standard error in functionality.
abc	**local discovery server URL** specifies the URL of the UA Local Discovery Server (LDS) on the local MulticastSubnet. The default is opc.tcp://localhost:4840. Refer to the OPC UA specification for more information about the LDS.
I/O	**OPC UA server refnum out** returns the reference data value of the OPC UA server.
	error out contains error information. This output provides standard error out functionality.

Register Server Details

Before you use the Register Server VI, you must install the UA Local Discovery Server (LDS) and make the OPC UA server and the LDS trust each other. Refer to the OPC Foundation website at www.opcfoundation.org to download the LDS. Visit `ni.com/info` and enter the Info Code ex9fm6 to access the **How Do I Make an OPC UA Server and the UA Local Discovery Server Trust Each Other?** NI support document, for information about making OPC UA servers and the LDS trust each other.

When you use the Register Server VI to register an OPC UA server, the status of the registered OPC UA server in the LDS is either online or offline. Whether the OPC UA client can connect to the registered OPC UA server depends on the status of the OPC UA server. You can use the Start VI and Stop VI to change the status of the registered OPC UA server.

The following table lists the status of the registered OPC UA server in the LDS.

Registered OPC UA Server Status	Description	Effects on the OPC UA Client
Online	The registered OPC UA server is running.	The OPC UA client can find the name and endpoint URL of the registered OPC UA server in the LDS and connect to the OPC UA server.
Offline	The registered OPC UA server is not running.	The OPC UA client can find the name of the registered OPC UA server in the LDS. However, the OPC UA client cannot connect t o the OPC UA server.

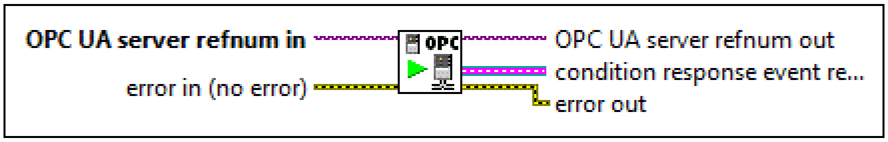
Start VI

Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Starts an OPC UA server. After you start an OPC UA server, you cannot add or delete folders, items, notifiers, or conditions until the OPC UA server stops. Use the Stop VI to stop an OPC UA server.

Example



OPC UA server refnum in specifies the reference data value of the OPC UA server.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

condition response event refnum returns a reference to the condition response event.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Start VI.

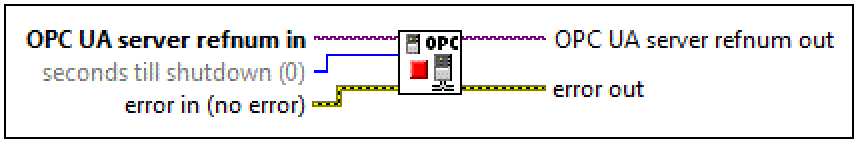
Stop VI

Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Stops an OPC UA server and disconnects all OPC UA clients. When you stop an OPC UA server, you still keep all the folders, items, and properties. Use the Start VI to restart the OPC UA server.

Example



OPC UA server refnum in specifies the reference data value of the OPC UA server.

seconds till shutdown specifies the time the OPC UA server waits, in seconds, before it shuts down. The default is 0, which means the OPC UA server shuts down immediately. During the wait time, the OPC UA server does not accept any new connections but the connected OPC UA client can perform tasks, such as disconnecting from the OPC UA server, releasing resources, and so on.

The OPC UA server shuts down immediately if it does not have a connected OPC UA client, regardless of the value of **seconds till shutdown**.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Stop VI.

Unregister Server VI

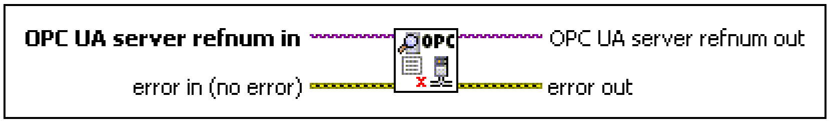
Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Unregisters a registered OPC UA server with the UA Local Discovery Server (LDS).

After you unregister an OPC UA server, an OPC UA client cannot find the unregistered OPC UA server in the LDS.

Details



OPC UA server refnum in specifies the reference data value of the OPC UA server.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

error out contains error information. This output provides standard error out functionality.

Unregister Server Details

You can use the Unregister Server VI to unregister a closed OPC UA server. If you use the Close VI to close and destroy a registered OPC UA server, the Close VI does not unregister the OPC UA server. Complete the following steps to unregister a closed OPC UA server:

1. Use the Create VI to create an OPC UA server. You must ensure that the OPC UA server that you create and the closed OPC UA server reside on the same computer.
2. Specify a TCP port for **TCP port** of the Create VI. You must ensure that the TCP port that you specify is the same as the TCP port that the closed OPC UA server used.
3. Use the Register Server VI to register the OPC UA server that you created.
4. Use the Unregister Server VI to unregister the closed OPC UA server.

Write VI

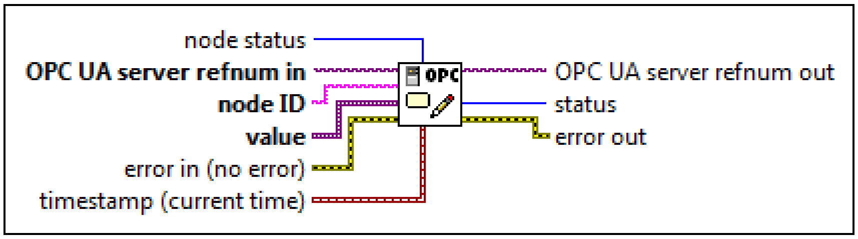
Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit

Writes the value and status to a node from an OPC UA server. You must manually select the polymorphic instance to use. You can write to a node before the OPC UA server starts.

Example

Write (Variant)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you

created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node. **value** also supports the Matrix data type.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

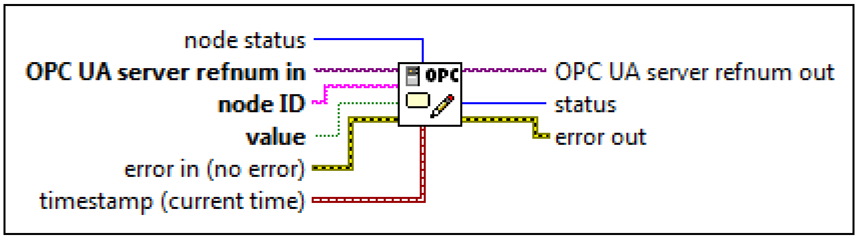
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Bool)



node status specifies the status of the node.

OPC UA server refnum in specifies the reference data value of the OPC UA server.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

> If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

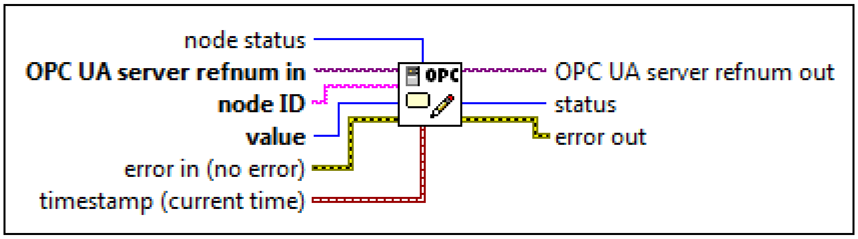
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (SByte)



node status specifies the status of the node.

OPC UA server refnum in specifies the reference data value of the OPC UA server.

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  > If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

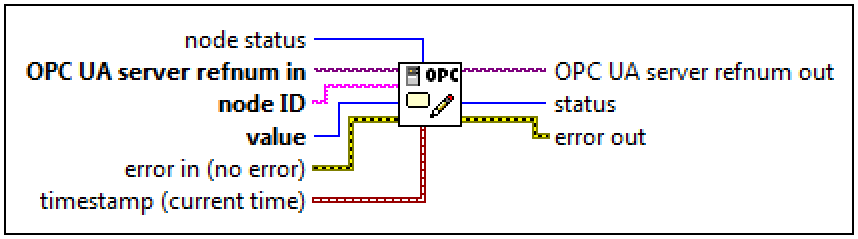
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Byte)



U32

node status specifies the status of the node.

I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

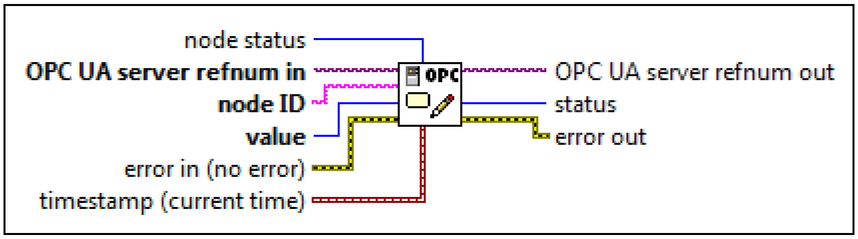
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Int16)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

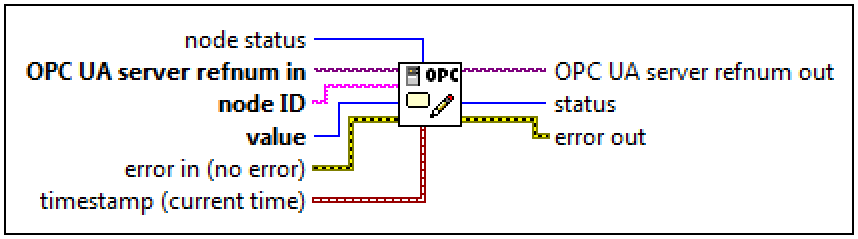
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (UInt16)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

U16 **value** specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

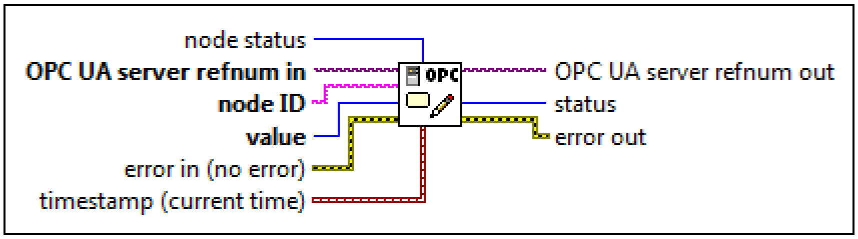
timestamp specifies the date and time associated with **value**.

I/O **OPC UA server refnum out** returns the reference data value of the OPC UA server.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Int32)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>. For example, ns=2;s=Folder.Array@1:2. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be Device.folder.item.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

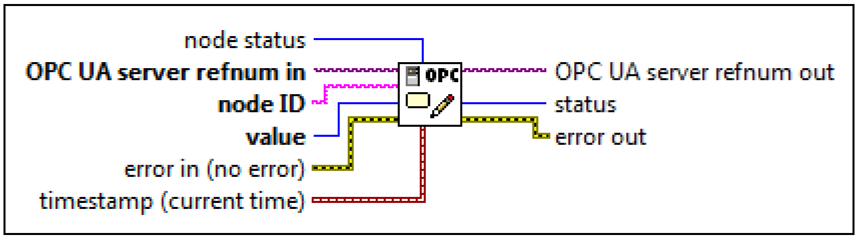
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (UInt32)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

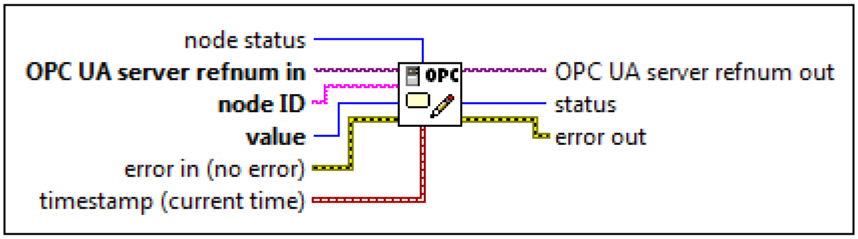
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Int64)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

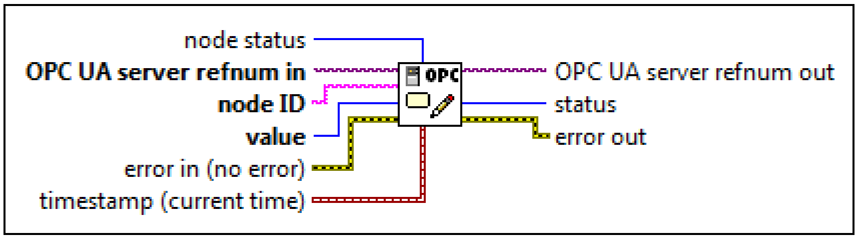
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (UInt64)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

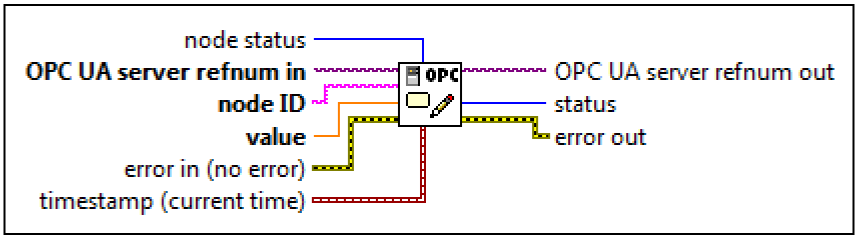
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Float)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

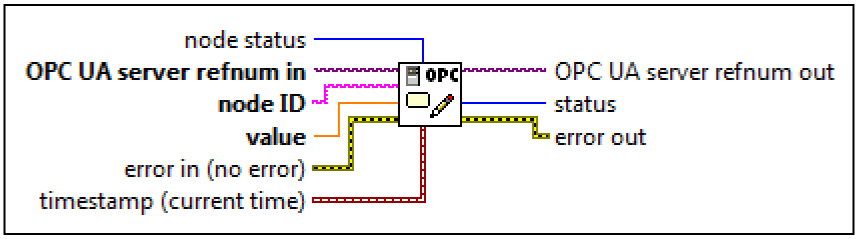
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Double)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

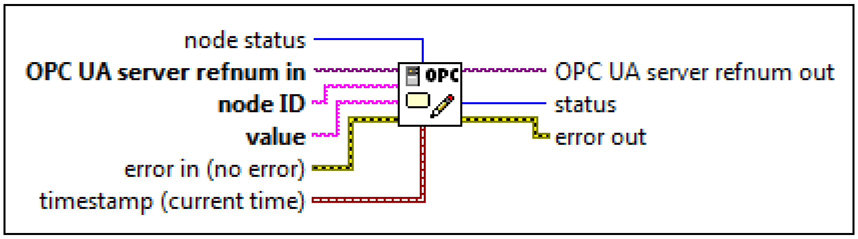
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (String)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>. For example, ns=2;s=Folder.Array@1:2. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be Device.folder.item.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

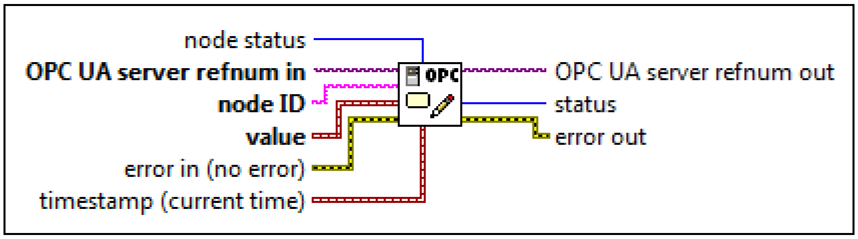
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (DateTime)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

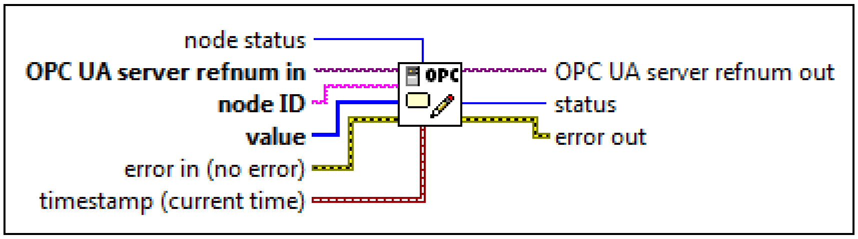
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (ByteString)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

value specifies the value of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

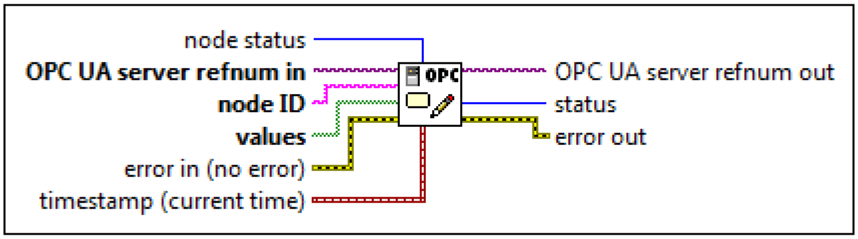
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Bool Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>. For example, ns=2;s=Folder.Array@1:2. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be Device.folder.item.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

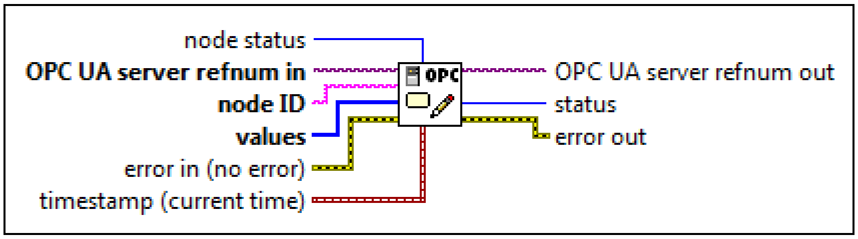
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (SByte Array)



U32 **node status** specifies the status of the node.

I/0 **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

[I8] **values** specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

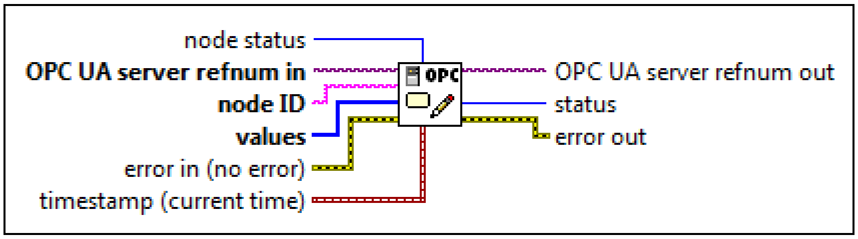
timestamp specifies the date and time associated with **value**.

I/O **OPC UA server refnum out** returns the reference data value of the OPC UA server.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Byte Array)



U32 **node status** specifies the status of the node.

I/0 **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

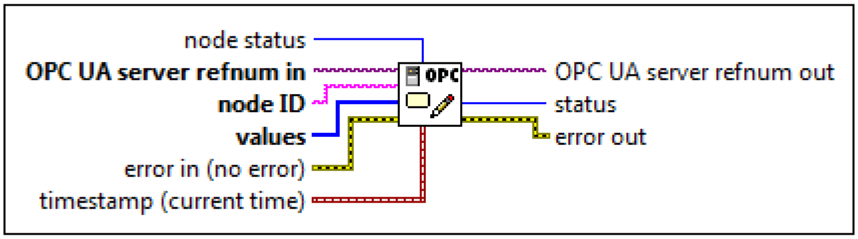
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Int16 Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

[I16] **values** specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

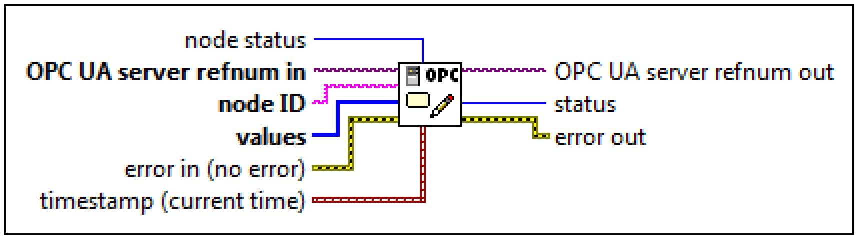
timestamp specifies the date and time associated with **value**.

I/O **OPC UA server refnum out** returns the reference data value of the OPC UA server.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (UInt16 Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

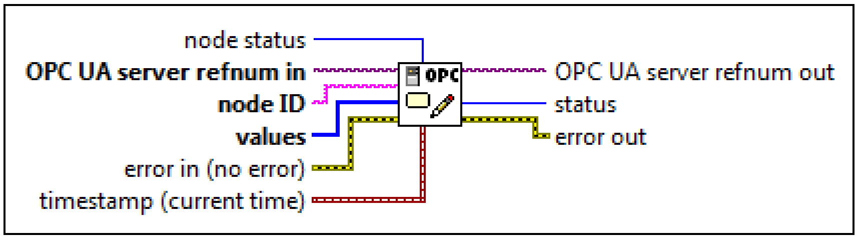
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Int32 Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

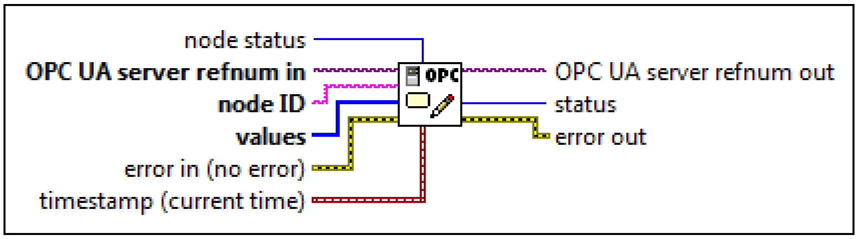
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (UInt32 Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

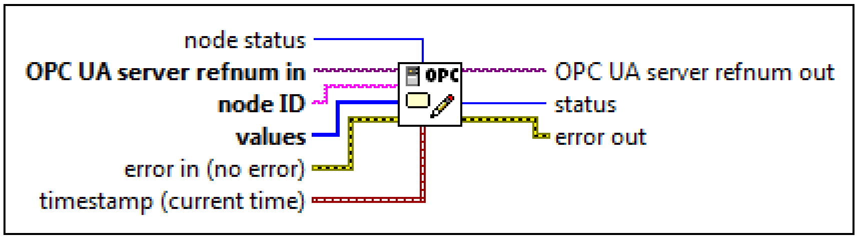
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Int64 Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

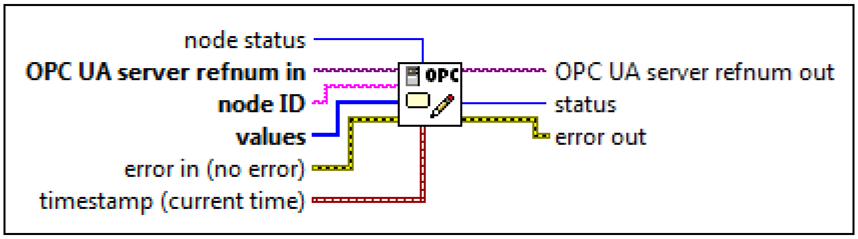
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (UInt64 Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

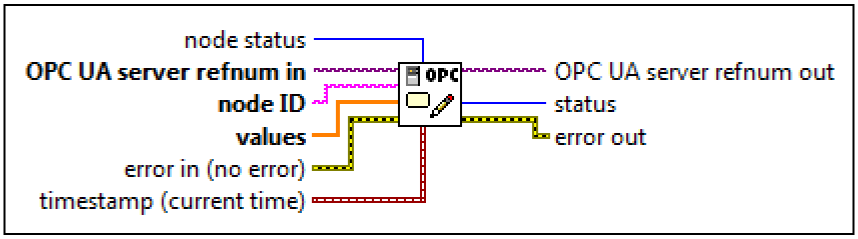
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Float Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

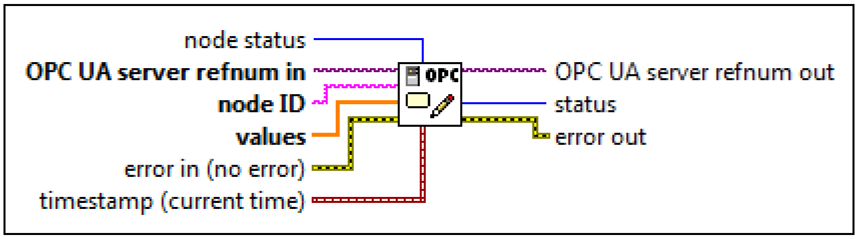
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (Double Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

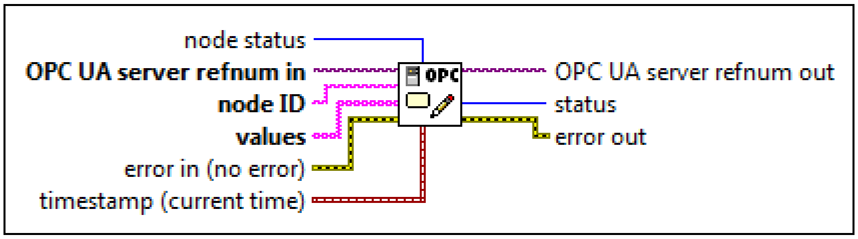
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (String Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

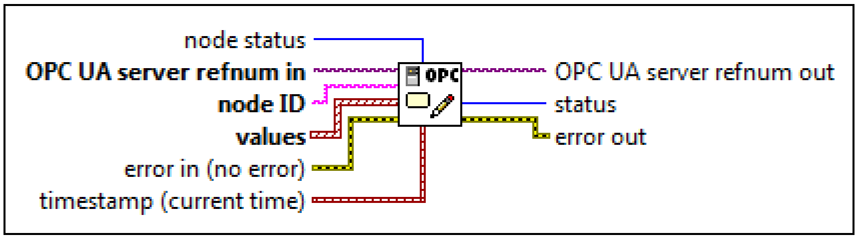
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (DateTime Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

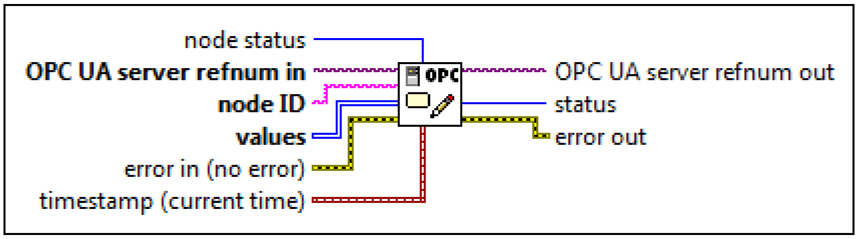
timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Write (ByteString Array)



U32 **node status** specifies the status of the node.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

values specifies the values of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timestamp specifies the date and time associated with **value**.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Write VI.

Alarms and Conditions VIs

Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit. This topic might not match its corresponding palette in LabVIEW depending on your operating system, licensed product(s), and target.

Use the Alarms and Conditions VIs to subscribe to alarms and conditions for OPC UA server applications.

Example

Palette Object	Description
Add Condition	Adds a condition node to the notifier. You must manually select the polymorphic insta nce to use.
Add Notifier	Adds a notifier to a parent folder. A notifier is an object that you can subscribe to get events from the associated condition nodes. To establish an event hierarchy, you can add a condition node a nd a source node as child nodes to a notifier.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Alarms and Conditions VIs.

Add Condition VI

Owning Palette: Alarms and Conditions VIs

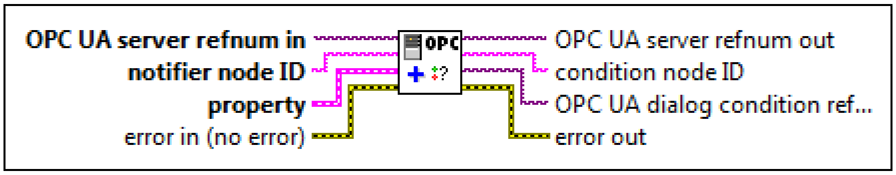
Requires: OPC UA Toolkit

Adds a condition node to the notifier.

You must manually select the polymorphic instance to use.

Details Example

Add Dialog Condition



I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

notifier node ID specifies the ID of the notifier. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **notifier node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

property specifies the properties of the condition node.

condition name specifies the name of the condition instance.

source node ID specifies the node ID of the condition source. A condition source is an element that a specific condition is based upon or related to. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **source node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

prompt specifies the textual content of a dialog prompt.

description specifies additional information of the condition.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

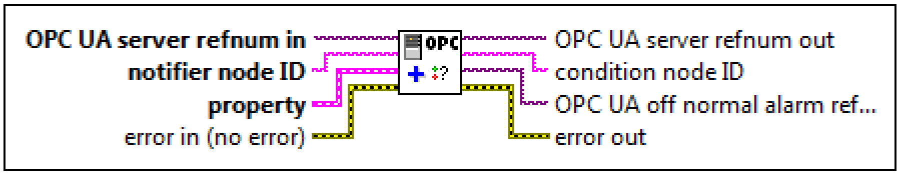
OPC UA server refnum out returns the reference data value of the OPC UA server.

condition node ID returns the ID of the condition node.

OPC UA dialog condition refnum out returns the reference to the dialog condition. The OPC UA server uses the dialog condition to request user input.

error out contains error information. This output provides standard error out functionality.

Add Off-Normal Alarm



OPC UA server refnum in specifies the reference data value of the OPC UA server.

notifier node ID specifies the ID of the notifier. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **notifier node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

property specifies the properties of the condition node.

condition name specifies the name of the condition instance.

source node ID specifies the node ID of the condition source. A condition source is an element that a specific condition is based upon or related to. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be <identifier type>=<identifier>. The namespace index for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **source node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

normal state node ID

specifies the ID of the node that has a value corresponding to one of the possible values of the source node.

description specifies additional information of the condition.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

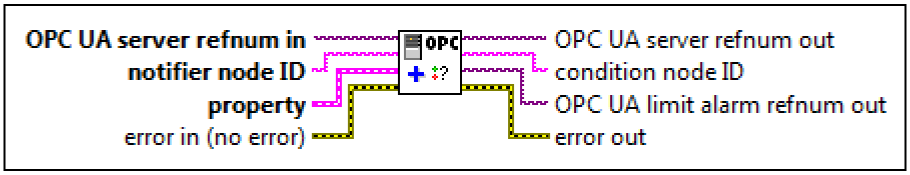
OPC UA server refnum out returns the reference data value of the OPC UA server.

condition node ID returns the ID of the condition node.

OPC UA off normal alarm refnum out returns the reference to an off-normal alarm condition node. An off-normal alarm condition represents an abnormal condition.

error out contains error information. This output provides standard error out functionality.

Add Exclusive Level Alarm



OPC UA server refnum in specifies the reference data value of the OPC UA server.

notifier node ID specifies the ID of the notifier. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **notifier node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

property specifies the property of the condition instance.

condition name specifies the name of the condition instance.

source node ID specifies the node ID of the condition source. A condition source is an element that a specific condition is based upon or related to. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`.

The namespace index for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be ns=<namespace index>;<identifier type>

`=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **source node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

limit specifies the values that causes LabVIEW to generate an alarm.

DBL **high high limit** specifies the value above which LabVIEW generates a high high limit alarm.

DBL **high limit** specifies the value above which LabVIEW generates a high limit alarm.

DBL **low limit** specifies the value below which LabVIEW generates a low limit alarm.

DBL **low low limit** specifies the value below which LabVIEW generates a low low limit alarm.

abc **description** specifies additional information of the condition.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

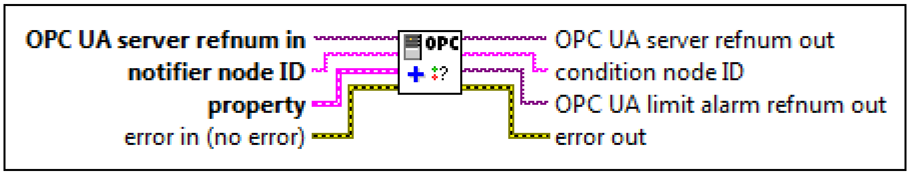
OPC UA server refnum out returns the reference data value of the OPC UA server.

condition node ID returns the ID of the condition node.

OPC UA limit alarm refnum out returns the reference to the OPC UA limit alarm condition.

error out contains error information. This output provides standard error out functionality.

Add Exclusive Deviation Alarm



OPC UA server refnum in specifies the reference data value of the OPC UA server.

notifier node ID specifies the ID of the notifier. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you

created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **notifier node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

property specifies the property of the condition instance.

condition name specifies the name of the condition instance.

source node ID specifies the node ID of the condition source. A condition source is an element that a specific condition is based upon or related to. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>. For example, ns=2;s=Folder.Array@1:2. This format only applies to the array data type and allows you to read a single element or a range of

elements of an array. You cannot use @ in a node name.

For backwards compatibility, **source node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

abc **setpoint node ID** specifies the ID of the setpoint node in the deviation calculation.

limit specifies the values that causes LabVIEW to generate an alarm.

DBL **high high limit** specifies the value above which LabVIEW generates a high high limit alarm.

DBL **high limit** specifies the value above which LabVIEW generates a high limit alarm.

low limit specifies the value below which LabVIEW generates a low limit alarm.

low low limit specifies the value below which LabVIEW generates a low low limit alarm.

description specifies additional information of the condition.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

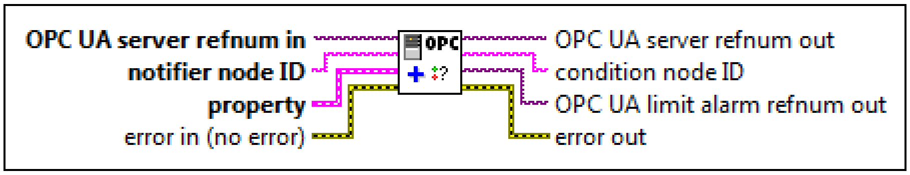
OPC UA server refnum out returns the reference data value of the OPC UA server.

condition node ID returns the ID of the condition node.

OPC UA limit alarm refnum out returns the reference to the OPC UA limit alarm condition.

error out contains error information. This output provides standard error out functionality.

Add Exclusive Rate of Change Alarm



OPC UA server refnum in specifies the reference data value of the OPC UA server.

notifier node ID specifies the ID of the notifier. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **notifier node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

property specifies the property of the condition instance.

condition name specifies the name of the condition instance.

source node ID specifies the node ID of the condition source. A condition source is an element that a specific condition is based upon or related to. The format of the node ID is `ns=<namespace`

index>;<identifier type>=<identifier>. A node ID contains the following components:

- namespace index is a base 10 number that indicates the namespace of the node ID.

  > If namespace index is 0, the format of the node ID can be <identifier type>=<identifier>. The namespace index for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	IdentifierType
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **source node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

limit specifies the values that causes LabVIEW to generate an alarm.

- DBL **high high limit** specifies the value above which LabVIEW generates a high high limit alarm.
- DBL **high limit** specifies the value above which LabVIEW generates a high limit alarm.
- DBL **low limit** specifies the value below which

LabVIEW generates a low limit alarm.

DBL **low low limit** specifies the value below which LabVIEW generates a low low limit alarm.

abc **description** specifies additional information of the condition.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

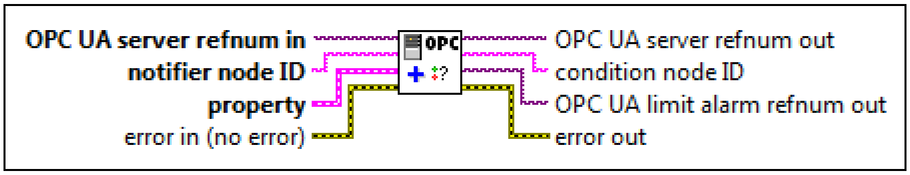
I/O **OPC UA server refnum out** returns the reference data value of the OPC UA server.

abc **condition node ID** returns the ID of the condition node.

I/O **OPC UA limit alarm refnum out** returns the reference to the OPC UA limit alarm condition.

error out contains error information. This output provides standard error out functionality.

Add Nonexclusive Level Alarm



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **notifier node ID** specifies the ID of the notifier. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  > If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<`

identifier>@<index>:<index>. For example, ns=2;s=Folder.Array@1:2. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **notifier node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be Device.folder.item.

property specifies the property of the condition instance.

condition name specifies the name of the condition instance.

source node ID specifies the node ID of the condition source. A condition source is an element that a specific condition is based upon or related to. The format of the node ID is ns=<namespace index>;<identifier type>=<identifier>. A node ID contains the following components:

- namespace index is a base 10 number that indicates the namespace of the node ID.

If namespace index is 0, the format of the node ID can be <identifier type>=<identifier>. The namespace index for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID

b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **source node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

limit specifies the values that causes

LabVIEW to generate an alarm.

DBL **high high limit**

specifies the value above which LabVIEW generates a high high limit alarm.

DBL **high limit**

specifies the value above which LabVIEW generates a high limit alarm.

DBL **low limit**

specifies the value below which LabVIEW generates a low limit alarm.

DBL **low low limit**

specifies the value below which LabVIEW

generates a low low limit alarm.

description specifies additional information of the condition.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

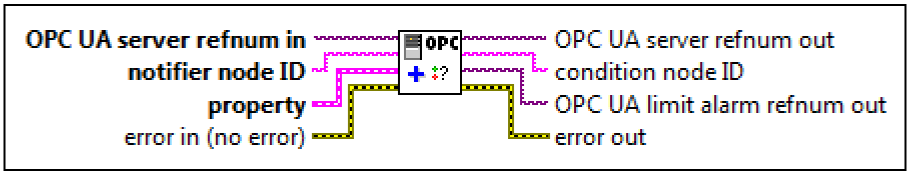
OPC UA server refnum out returns the reference data value of the OPC UA server.

condition node ID returns the ID of the condition node.

OPC UA limit alarm refnum out returns the reference to the OPC UA limit alarm condition.

error out contains error information. This output provides standard error out functionality.

Add Nonexclusive Deviation Alarm



OPC UA server refnum in specifies the reference data value of the OPC UA server.

notifier node ID specifies the ID of the notifier. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- namespace index is a base 10 number that indicates the namespace of the node ID.

  If namespace index is 0, the format of the node ID can be <identifier type>=<identifier>. The namespace index for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>. For example, ns=2;s=Folder.Array@1:2. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **notifier node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

property specifies the property of the condition instance.

condition name specifies the name of the condition instance.

source node ID specifies the node ID of the condition source. A condition source is an element that a specific condition is based upon or related to. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier typ`

e>=<identifier>. The namespace index for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node

ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **source node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

setpoint node ID specifies the ID of the setpoint node in the deviation calculation.

limit specifies the values that causes LabVIEW to generate an alarm.

high high limit specifies the value

above which LabVIEW generates a high high limit alarm.

DBL **high limit**

specifies the value above which LabVIEW generates a high limit alarm.

DBL **low limit**

specifies the value below which LabVIEW generates a low limit alarm.

DBL **low low limit**

specifies the value below which LabVIEW generates a low low limit alarm.

description specifies additional information of the condition.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

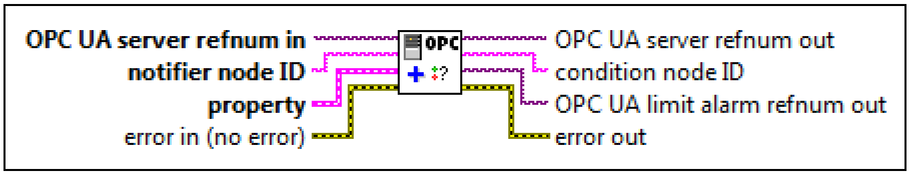
OPC UA server refnum out returns the reference data value of the OPC UA server.

condition node ID returns the ID of the condition node.

OPC UA limit alarm refnum out returns the reference to the OPC UA limit alarm condition.

error out contains error information. This output provides standard error out functionality.

Add Nonexclusive Rate of Change Alarm



OPC UA server refnum in specifies the reference data value of the OPC UA server.

notifier node ID specifies the ID of the notifier. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format

of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **notifier node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

property specifies the property of the condition instance.

condition name

specifies the name of the condition instance.

source node ID

specifies the node ID of the condition source. A condition source is an element that a specific condition is based upon or related to. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace in dex for a node that you created with the OPC UA Toolkit is 2.

- identifier type represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- identifier is a string value that represents the name of the identifier.

The format of the node ID can also be ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>. For example, ns=2;s=Fo `lder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **source node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

limit specifies the values that causes LabVIEW to generate an alarm.

DBL **high high limit** specifies the value above which LabVIEW generates a high high limit alarm.

DBL **high limit** specifies

the value above which LabVIEW generates a high limit alarm.

DBL **low limit** specifies the value below which LabVIEW generates a low limit alarm.

DBL **low low limit** specifies the value below which LabVIEW generates a low low limit alarm.

abc **description** specifies additional information of the condition.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

I/O **OPC UA server refnum out** returns the reference data value of the OPC UA server.

abc **condition node ID** returns the ID of the condition node.

OPC UA limit alarm refnum out returns the reference to the OPC UA limit alarm condition.

error out contains error information. This output provides standard error out functionality.

Add Condition Details

The dialog is a condition that an OPC UA server uses to request user input.

Alarms are specializations of acknowledgeable conditions that add the concepts of an active state, a shelving state, and a suppressed state to a condition. This VI supports the following alarm types:

- **Off-normal alarm**—Specifies a condition that is not normal.
- **Limit alarm**—Specifies limits that can generate an alarm when a value equals or exceeds the limit. There are four alarm limits: high high limit, high limit, low limit, and low low limit. This VI supports the following limit alarm types:
  - **Exclusive alarm**—Contains multiple mutually exclusive limits. For an exclusive alarm, multiple states can be active at the same time.
  - **Nonexclusive alarm**—Contains multiple nonexclusive limits. For a nonexclusive alarm, only one state can be active at a specific time.
  - **Level alarm**—Reports when a limit is exceeded.
  - **Deviation alarm**—Reports an excess deviation between a desired setpoint level of a value and an actual measurement of that value.
  - **Rate of change alarm**—Reports an unusual change or lack of change in a measured value related to the speed at which the value has changed.

The condition type affects the available data types of the source node and the normal state node:

- For the dialog condition type, the data type of the source node must be Boolean.

- For the off-normal alarm condition type, the data type of the source node and the normal state node must be scalar and they must have the same data type.
- For the deviation alarm condition type and level alarm condition type, the data type of the source node must be Double.
- For the rate of change alarm condition type, the source node must be analog data and the data type must be scalar.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Add Condition VI.

Add Notifier VI

Owning Palette: Alarms and Conditions VIs

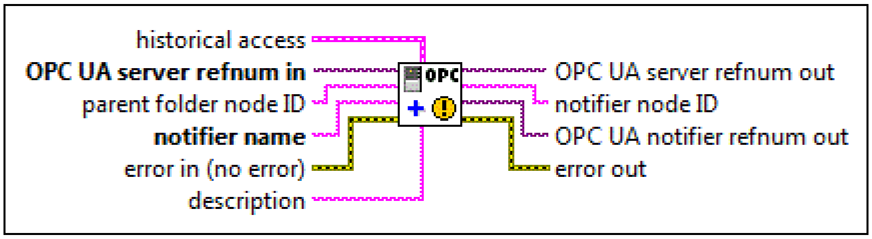
Requires: OPC UA Toolkit

Adds a notifier to a parent folder. A notifier is an object that you can subscribe to get events from the associated condition nodes. To establish an event hierarchy, you can add a condition node and a source node as child nodes to a notifier.

For some condition node types, you need to add an extra node:

- If the condition node is a deviation alarm, you need to add a setpoint node.
- If the condition node is an off-normal alarm, you need to add a normal state node.

Example



historical access specifies whether the parent folder supports historical access and the size of the history event queue.

- TF **enable** specifies whether the analog item supports historical access. The default is FALSE, which specifies that this item does not support historical access.
- U32 **queue size** specifies the size of the queue of history events. The default is 1000. A history event queue can store one history event only. You can add a history event queue only to a notifier. NI recommends adding the condition node and the source node to the notifier so that they share one history event.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **parent folder node ID** specifies the ID of the parent folder. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. If the parent folder does not exist, this VI creates a folder at the root level. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **parent folder node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

notifier name specifies the name of the notifier.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

description specifies additional information of the notifier.

OPC UA server refnum out returns the reference data value of the OPC UA server.

notifier node ID returns the ID of the notifier.

OPC UA notifier refnum out returns the reference data value of the OPC UA notifier.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Add Notifier VI.

Historical Access VIs

Owning Palette: OPC UA Server VIs

Requires: OPC UA Toolkit. This topic might not match its corresponding palette in LabVIEW depending on your operating system, licensed product(s), and target.

Use the Historical Access VIs to access historical data and events for OPC UA server applications.

Example

Palette Object	Description
History Data Delete	Deletes history data within a time range.

History Data Delete at Time	Deletes history data at specific timestamps.
History Data Read	Reads history data within a time range.
History Data Update	Updates history data at a specific timestamp.
History Event Delete	Deletes a history event.
History Event Read	Reads a notifier. A notifier node enables an OPC UA client, which subscribes to the node, to receive event notifications.
History Event Update	Updates a history event.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the Historical Access VIs.

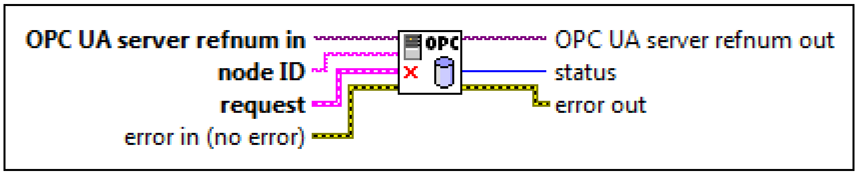
History Data Delete VI

Owning Palette: Historical Access VIs

Requires: OPC UA Toolkit

Deletes history data within a time range.

Example



I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

node ID specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

request specifies the start time and the end time that this VI deletes history data.

- **start time** specifies the timestamp at which this VI deletes the first history data.
- **end time** specifies the timestamp at which this VI deletes the last history data.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the History Data Delete VI.

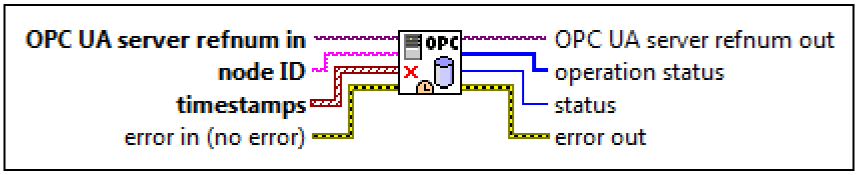
History Data Delete at Time VI

Owning Palette: Historical Access VIs

Requires: OPC UA Toolkit

Deletes history data at specific timestamps.

Example



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>. For example, ns=2;s=Folder.Array@1:2. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be Device.folder.item.

timestamps specifies the timestamps at which this VI deletes history data.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout specifies the maximum time, in milliseconds, that this VI waits to get a response from the OPC UA server. The default is 5000.

operation status returns the status of the operation.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the OPC UA Demo.lvproj in the labview\examples\Data Communication\OPCUA directory for an example of using the History Data Delete at Time VI.

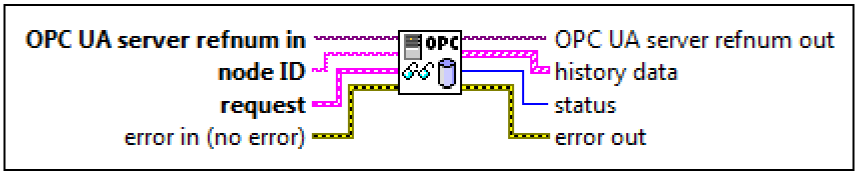
History Data Read VI

Owning Palette: Historical Access VIs

Requires: OPC UA Toolkit

Reads history data within a time range.

Example



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

request specifies the start time, the end time, the maximum number of values to return over the time range, and whether to return bounding values.

start time specifies the timestamp at which this VI reads the first history data.

end time specifies the timestamp at which this VI reads the last history data.

U32

num values per node

specifies the maximum number of values to return over the time range. The default is 0, which specifies that this VI returns all values over the time range.

TF

return bounds

specifies whether to return bounding values. The default is FALSE, which specifies that this VI does not return bounding values. Bounding values are values associated with the starting time and the ending time. An OPC UA client can require bounding values to determine the starting and ending values when requesting data over a time range.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

I/O

OPC UA server refnum out returns the reference data value of the OPC UA server.

history data returns data from the history database.

value returns the history data.

timestamp returns the timestamp at which data is generated.

node status specifies the status code of the node.

status returns the status code.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the History Data Read VI.

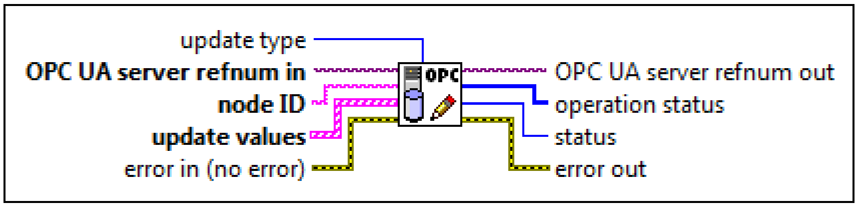
History Data Update VI

Owning Palette: Historical Access VIs

Requires: OPC UA Toolkit

Updates history data at a specific timestamp.

Example



update type specifies the type of update operation this VI performs.

1	**Insert**—Inserts new entries into the history

	database at the specified timestamps.
2	**Replace**—Replaces existing entries in the history database at the specified timestamps.
3	**Update** (default)—Inserts new entries or replaces existing entries in the history database at the specified timestamps.

I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **node ID** specifies the ID of the node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type

i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **node ID** also accepts node paths as input for OPC UA servers. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

update values specifies the value to update with, the timestamp of the data to replace, and the status code.

value specifies the value that this VI uses to update with.

timestamp specifies the timestamp at which this VI updates history data.

U32 **node status** specifies the status code of the node.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

I/O **OPC UA server refnum out** returns the reference data value of the OPC UA server.

U32 **operation status** returns the status of the operation.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the History Data Update VI.

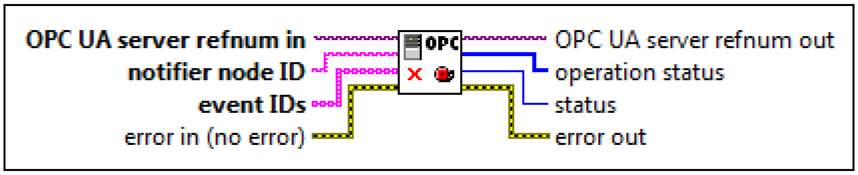
History Event Delete VI

Owning Palette: Historical Access VIs

Requires: OPC UA Toolkit

Deletes a history event.

Example



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **notifier node ID** specifies the ID of the notifier. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identif`

`ier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  > If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

	For backwards compatibility, **notifier node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.
[abc]	**event IDs** specifies the event IDs to delete.
	error in describes error conditions that occur before this node runs. This input provides standard error in functionality.
I/O	**OPC UA server refnum out** returns the reference data value of the OPC UA server.
[U32]	**operation status** returns the status of the operation.
U32	**status** returns the status code.
	error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the History Event Delete VI.

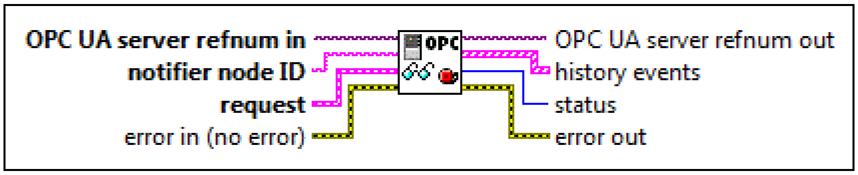
History Event Read VI

Owning Palette: Historical Access VIs

Requires: OPC UA Toolkit

Reads a notifier. A notifier node enables an OPC UA client, which subscribes to the node, to receive event notifications.

Example



I/O **OPC UA server refnum in** specifies the reference data value of the OPC UA server.

abc **notifier node ID** specifies the ID of the notifier. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **notifier node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

request specifies the start time and the end time for the VI to read the notifier.

start time specifies the timestamp at which this VI reads the first history event.

end time specifies the timestamp at which this VI reads the last history event.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

OPC UA server refnum out returns the reference data value of the OPC UA server.

history events returns the detailed information about the event that this VI reads from the history database. **condition type** affects the validity of the output data in **history event**.

- For the dialog condition type, the validity of the output data is shown in the following table:

Output	Validity
condition node ID	valid
source node ID/input node ID	valid
event ID	valid
active	valid
time	valid
severity	valid
quality	valid
comment	valid
limit state	invalid
acked state	invalid
confirmed state	invalid
prompt	valid

- For the off-normal alarm condition type, the validity of the output data is shown in the following table:

Output	Validity
condition node ID	valid
source node ID/input node ID	valid
event ID	valid
active	valid
time	valid
severity	valid
quality	valid

comment	valid
limit state	invalid
acked state	valid
confirmed state	valid
prompt	invalid

- For the limit alarm condition type, the validity of the output data is shown in the following table:

Output	Validity
condition node ID	valid
source node ID/input node ID	valid
event ID	valid
active	valid
time	valid
severity	valid
quality	valid
comment	valid
limit state	valid
acked state	valid
confirmed state	valid
prompt	invalid

condition node ID

returns the ID of the condition node.

condition type

returns the type of the condition node.

0	Dialog Condition

1	Off-Normal Alarm
2	Exclusive Level Alarm
3	Exclusive Deviation Alarm
4	Exclusive Rate of Change Alarm
5	Non Exclusive Level Alarm
6	Non Exclusive Deviation Alarm
7	Non Exclusive Rate of Change Alarm
8	Refresh Start Event
9	Refresh End Event

source node ID/ input node ID

returns the ID of the source node or the input node. For nodes of the dialog condition type, **source node ID/ input node ID** returns the ID of the source node. For nodes of the off-normal alarm type and the limit alarm type, **source node ID/ input node ID** returns the ID of the input node. You can use the Add Condition VI to add dialog condition nodes, off-normal alarm nodes, or limit alarm nodes.

abc **event ID** returns the event ID.

TF **active** returns whether the situation the alarm is representing exists.

⌛ **time** returns the timestamp at which the event occurs.

U16 **severity** returns a value that indicates the urgency of the event. The value ranges from 1 to 1000, with 1 being the lowest severity and 1000 being the highest severity.

I32 **quality** returns the status code to display the quality of the values that the condition depends on.

comment returns the string to associate with a certain state of condition.

limit state returns the state of the alarm.

- **high high state** returns whether the condition value exceeds the high high limit.
- **high state** returns whether the condition value exceeds the high limit.
- **low state** returns whether the condition value exceeds the low limit.

TF **low low state** returns whether the condition value exceeds the low low limit.

TF **acked state** returns whether the condition is acknowledged.

TF **confirmed state** returns whether the condition is confirmed.

abc **prompt** returns the textual content of a dialog prompt only when the condition type is dialog condition.

U32 **status** returns the status code.

error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the History Event Read VI.

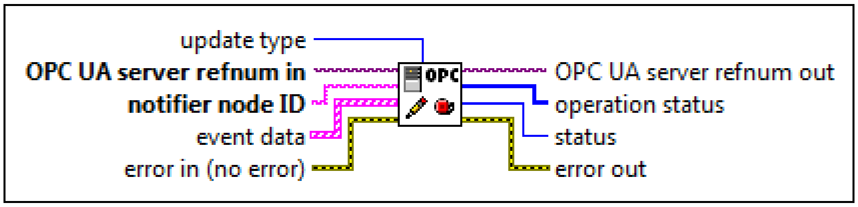
History Event Update VI

Owning Palette: Historical Access VIs

Requires: OPC UA Toolkit

Updates a history event.

Example



I32

update type specifies the type of update operation this VI performs.

1	**Insert**—Inserts new entries into the history database at the specified timestamps.
2	**Replace**—Replaces existing entries in the history database at the specified timestamps.
3	**Update** (default)—Inserts new entries or replaces existing entries in the history database at the specified timestamps.

I/O

OPC UA server refnum in specifies the reference data value of the OPC UA server.

abc

notifier node ID specifies the ID of the notifier. A notifier is an object that you can subscribe to get events from the associated condition nodes. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	Identifier Type
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **notifier node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

event data specifies information about the events to update.

condition node ID specifies the ID of the condition node. The format of the node ID is `ns=<namespace index>;<identifier type>=<identifier>`. A node ID contains the following components:

- `namespace index` is a base 10 number that indicates the namespace of the node ID.

  If `namespace index` is 0, the format of the node ID can be `<identifier type>=<identifier>`. The `namespace index` for a node that you created with the OPC UA Toolkit is 2.

- `identifier type` represents the type of the identifier and has the following values:

Value	IdentifierType
i	Numeric
s	String
g	GUID
b	Opaque

- `identifier` is a string value that represents the name of the identifier.

The format of the node ID can also be `ns=<namespace index>;<identifier type>=<identifier>@<index>:<index>`. For example, `ns=2;s=Folder.Array@1:2`. This format only applies to the array data type and allows you to read a single element or a range of elements of an array. You cannot use @ in a node name.

For backwards compatibility, **condition node ID** also accepts node paths as input for OPC UA servers only. You can regard the node path as the string type identifier of the node ID. For example, a node path can be `Device.folder.item`.

abc **event ID** specifies the event ID.

TF **active** specifies whether an alarm is in active state. The default is FALSE, which specifies that the alarm is not in active state.

⌛ **time** specifies the time at which the event occurred.

U16 **severity** specifies a value that indicates the urgency of the event. The value ranges from 1 to 1000, with 1 being the lowest severity and 1000 being the highest severity. The default is 100.

I32 **quality** specifies the quality of the values that the condition depends on. Refer to the status code for details.

abc **comment** specifies the string to associate with a certain state of the condition.

limit state specifies the state of the alarm.

TF **high high state** specifies whether the condition value exceeds the high high limit. The default is FALSE, which specifies that the condition does not exceed the high high limit.

TF **high state** specifies whether

the condition value exceeds the high limit. The default is FALSE, which specifies that the condition does not exceed the high limit.

TF

low state

specifies whether the condition value exceeds the low limit. The default is FALSE, which specifies that the condition does not exceed the low limit.

TF

low low state

specifies whether the condition value exceeds the low low limit. The default is FALSE, which specifies that the condition does not exceed the low low limit.

TF **acked state** specifies whether the condition is acknowledged. The default is FALSE, which specifies that the condition is not acknowledged.

TF **confirmed state** specifies whether the condition is confirmed. The default is FALSE, which specifies that the condition is not confirmed.

abc **prompt** specifies the textual content of a dialog prompt.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

I/O **OPC UA server refnum out** returns the reference data value of the OPC UA server.

U32 **operation status** returns the status of the operation.

U32	**status** returns the status code.
	error out contains error information. This output provides standard error out functionality.

Example

Refer to the `OPC UA Demo.lvproj` in the `labview\examples\Data Communication\OPCUA` directory for an example of using the History Event Update VI.

OPC UA Properties

Use the following properties with the Property Node to access OPC UA data.

Dialog Condition Properties

Limit Alarm Properties

Notifier Properties

Off-Normal Alarm Properties

Variable Node Properties

Dialog Condition Properties

The dialog is a condition that a server uses to request user input. Wire the **OPC UA dialog condition refnum out** output of the Dialog Condition instance of the Add Condition VI to the **reference** input of a standard Property Node to get a Dialog Condition Property Node. This Property Node has the following properties:

Property	Description
Dialog State	Whether a dialog is active and waiting for a response. Details
Enabled State	Whether a condition is in the enabled state. Details
Prompt	A dialog prompt to display to the end user. Details
Retain	Whether an OPC UA client synchronizes the condition state with that of the server. Details

Dialog State Property

Requires: OPC UA Toolkit

Class: Dialog Condition Properties

Whether a dialog is active and waiting for a response.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	TF

Enabled State Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

Whether a condition is in the enabled state.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	TF

Prompt Property

Requires: OPC UA Toolkit

Class: Dialog Condition Properties

A dialog prompt to display to the end user.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read-only
Data Type	abc

Retain Property

Requires: OPC UA Toolkit

Class: Dialog Condition Properties, Limit Alarm Properties, and Off-Normal Alarm Properties

Whether the OPC UA client synchronizes the condition state with that of the server.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	TF

Limit Alarm Properties

The limit alarm specifies limits that can generate an alarm when a value equals or exceeds the limit. There are four alarm limits: high high limit, high limit, low limit, and low low limit. Wire the **OPC UA limit alarm refnum out** output of the Exclusive Limit Alarm and Nonexclusive Limit Alarm instances of Add Condition VI to the **reference** input of a standard Property Node to get the Limit Alarm Property Node. This Property Node has the following properties:

Property	Description

Active State	Whether a condition is in the active state. Details
Enabled State	Whether a condition is in the enabled state. Details
Input Node Id	The ID of an input node. Details
Limit	The values of the high high limit, high limit, low limit, and low low limit. Details
Max Time Shelved	The maximum time that an alarm condition is in shelved state. Details
Retain	Whether an OPC UA client synchronizes the condition state with that of the server. Details
Severity Level	The values of the high high severity, high severity, low severity, and low low severity. Details
Shelving State	Whether an alarm condition is in shelved state. In shelved state, an alarm cannot display to the end user. Details
Shelving Timer Resolution in mSecs	The smallest time unit, in milliseconds, for the timer to calculate shelving time. Details
Suppressed or Shelved	Whether an alarm condition is in suppressed or shelved state. Details
Suppressed State	Whether an alarm condition is in suppressed state. In suppressed state, an alarm does not occur at all circumstances. Details

Active State Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

Whether a condition is in active state.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only

Data Type	TF

Enabled State Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

Whether a condition is in the enabled state.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	TF

Retain Property

Requires: OPC UA Toolkit

Class: Dialog Condition Properties, Limit Alarm Properties, and Off-Normal Alarm Properties

Whether the OPC UA client synchronizes the condition state with that of the server.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	TF

Max Time Shelved Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

The maximum time that an alarm condition is in shelved state.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read-only
Data Type	DBL
Default Value	5,000,000 ms

Suppressed State Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

Whether an alarm is in suppressed state.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read/Write
Data Type	TF

Input Node Id Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

The ID of the input node.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read-only
Data Type	abc

Severity Level Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties

The values of the high high severity level, high severity level, low severity level, and low low severity level.

Name	Description	Default Value
High High Severity	The value of severity level in high high limit state.	800
High Severity	The value of severity level in high limit state.	400
Low Severity	The value of severity level in low limit state.	400
Low Low Severity	The value of severity level in low low limit state.	800

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read-only
Data Type	▭

The severity level ranges from 1 to 1000. If the input value is less than 1 or greater than 1000, LabVIEW sets the severity level to 1 or 1000 and displays a warning message.

Limit Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties

The values of the high high limit, high limit, low limit, and low low limit.

Name	Description	Default Value
High High Limit	The value above which LabVIEW generates a high high limit alarm.	80
High Limit	The value above which LabVIEW generates a high limit alarm.	60
Low Limit	The value above which LabVIEW generates a low limit alarm.	40
Low Low Limit	The value above which LabVIEW generates a low low limit alarm.	20

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read-only
Data Type	▣

You must provide values for all elements. Otherwise, LabVIEW initializes the elements that do not have input values to 0. The values of the elements must conform with the following requirement:

	High High Limit > **High Limit** > **Low Limit** > **Low Low Limit** Otherwise, LabVIEW reports an error.

Shelving State Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

The shelving state of an alarm.

Values

0	OneShotShelved
1	TimedShelved
2	Unshelved

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	U32
Default Value	Unshelved

Suppressed or Shelved Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

Whether an alarm condition is in suppressed or shelved state.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	TF

Shelving Timer Resolution in mSecs Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

The smallest time unit, in milliseconds, for the timer to calculate shelving time.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read-only
Data Type	U32
Default Value	1000 ms

Off-Normal Alarm Properties

The off-normal alarm specifies a condition that is not normal. Wire the **OPC UA off normal alarm refnum out** output of the Off-Normal Alarm instance of Add Condition VI to the **reference** input of a standard Property Node to get the Off Normal Alarm Property Node. This Property Node has the following properties:

Property	Description
Active State	Whether a condition is in the active state. Details
Enabled State	Whether a condition is in the enabled state. Details
Retain	Whether the OPC UA client synchronizes the condition state with that of the server. Details
Input Node Id	The ID of an input node. Details

Max Time Shelved	The maximum time that an alarm condition is in shelved state. Details
Shelving State	Whether an alarm condition is in shelved state. In shelved state, an alarm cannot display to the end user. Details
Shelving Timer Resolution in mSecs	The smallest time unit, in milliseconds, for a timer to calculate shelving time. Details
Suppressed or Shelved	Whether an alarm condition is in suppressed or shelved state. Details
Suppressed State	Whether an alarm condition is in suppressed state. In suppressed state, an alarm does not occur at all circumstances. Details

Active State Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

Whether a condition is in active state.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	TF

Enabled State Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

Whether a condition is in the enabled state.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	TF

Input Node Id Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

The ID of the input node.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read-only
Data Type	abc

Retain Property

Requires: OPC UA Toolkit

Class: Dialog Condition Properties, Limit Alarm Properties, and Off-Normal Alarm Properties

Whether the OPC UA client synchronizes the condition state with that of the server.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only

Data Type	TF

Max Time Shelved Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

The maximum time that an alarm condition is in shelved state.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read-only
Data Type	DBL
Default Value	5,000,000 ms

Shelving Timer Resolution in mSecs Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

The smallest time unit, in milliseconds, for the timer to calculate shelving time.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read-only
Data Type	U32
Default Value	1000 ms

Shelving State Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

The shelving state of an alarm.

Values

0	OneShotShelved
1	TimedShelved
2	Unshelved

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	U32
Default Value	Unshelved

Suppressed State Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

Whether an alarm is in suppressed state.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read/Write
Data Type	TF

Suppressed or Shelved Property

Requires: OPC UA Toolkit

Class: Limit Alarm Properties and Off-Normal Alarm Properties

Whether an alarm condition is in suppressed or shelved state.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	TF

Variable Node Properties

Wire the **OPC UA variable node refnum out** output of the Add Property VI or the Add Item VI to the **reference** input of a standard Property Node to get the Variable Node Property Node. This Property Node has the following properties:

Property	Description
Historical Data Access	Whether historical access is available. Details
Historical Data Queue Size	The size of a historical data queue. You can add a historical data queue to an item, property, or condition. A history data queue can store 1000 units of history data. Details

Historical Data Access Property

Requires: OPC UA Toolkit

Class: Variable Node Properties

Whether historical data access is available.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	TF

Historical Data Queue Size Property

Requires: OPC UA Toolkit

Class: Variable Node Properties

The size of a historical data queue.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read-only
Data Type	U32
Default Value	1000

The default value is 1000 only if Historical Data Access is True.

Notifier Properties

A notifier node enables an OPC UA client, which subscribes to the node, to receive event notifications. Wire the **notifier node ID** output of the Add Notifier VI to the **reference** input of a standard Property Node to get the Variable Node Property Node. This Property Node has the following properties:

Property	Description
Historical Event Access	Whether historical event access is available. Details

Historical Event Queue Size	The size of a historical event queue. You can add a history event queue only to a notifier. A history event queue can store one history event only. Details

Historical Event Access Property

Requires: OPC UA Toolkit

Class: Notifier Properties

Whether historical event access is available.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read-only
Permissions After Server Starts	Read-only
Data Type	TF

Historical Event Queue Size Property

Requires: OPC UA Toolkit

Class: Notifier Properties

The size of a historical event queue.

Remarks

The following table lists the characteristics of this property.

Permissions Before Server Starts	Read/Write
Permissions After Server Starts	Read-only
Data Type	U32
Default Value	1000

The default value is 1000 only if Historical Event Access is True.