CTCopc

2

CONTROL TECHNOLOGY CORPORATION

CTCopc 2 Reference

CTCopc 2 Reference

CONTROL TECHNOLOGY CORPORATION

CTCopc 2 Reference

---

25 South Street
Hopkinton, MA 01748 USA
Phone 508.435.9595 • Fax 508.435.2373
Doc. No. 951-530024-002

February 4, 2021

Table of Contents

CHAPTER

1

Introduction

An introduction to CTCopc 2

CTCopc 2 (CTCopc) is a 32-bit Windows OPC server application that provides real-time client access to CTC automation controllers.

In 1994, several vendors formed what is now known as the OPC Foundation. This foundation defines standards that allow multiple vendors from various segments of the industrial marketplace to interoperate and share data.

In 1996, a specification was released that defines how data-oriented, server applications should interact with client application software. This specification, known as Data Access (OPC-DA) is now in its third major revision.

CTCopc conforms to the first three revisions, namely the 1.0, 2.0 and 3.0 OPC-DA standards.

By using OPC technology, users can avoid using custom, single-vendor application programming interfaces (API's) and drivers that are traditionally required for exchanging data between factory-floor devices and the enterprise.

CHAPTER 2

Installation

Installing CTCopc 2

CTCopc can be installed on any modern Windows-based operating system. Since OPC is rooted in Microsoft's COM (Component Object Model) technology, non-Windows operating systems, such as Linux, are not supported.

CTCopc uses the latest Microsoft technology, namely Microsoft .NET. Before installing or using CTCopc, core *.NET framework* components must be first installed on the target computer.

These framework components are:

Microsoft .NET Framework Version 2.0 Runtime

Microsoft Visual J# Version 2.0 Runtime

The installers for these runtime components are available on the Internet and **should be installed in the order listed above**.

These components can be downloaded from Microsoft at:

Microsoft .NET Framework 2.0

http://www.microsoft.com/downloads/details.aspx?FamilyID=0856eacb-4362-4b0d-8edd-aab15c5e04f5&displaylang=en

Microsoft Visual J# 2.0

http://www.microsoft.com/downloads/details.aspx?familyid=f72c74b3-ed0e-4af8-ae63-2f0e42501be1&displaylang=en

Once these components are installed properly, then the CTCopc installer (CTCopc2 Setup.MSI) can be double-clicked and run. After the installation is complete, a desktop icon is created, as well as a shortcut located in the Start menu in the ***CTC*** folder (program group).

Reference the 'CTC OPC Server Windows 7 and 10 64 bit Installation' manual for details on those operating systems.

CHAPTER

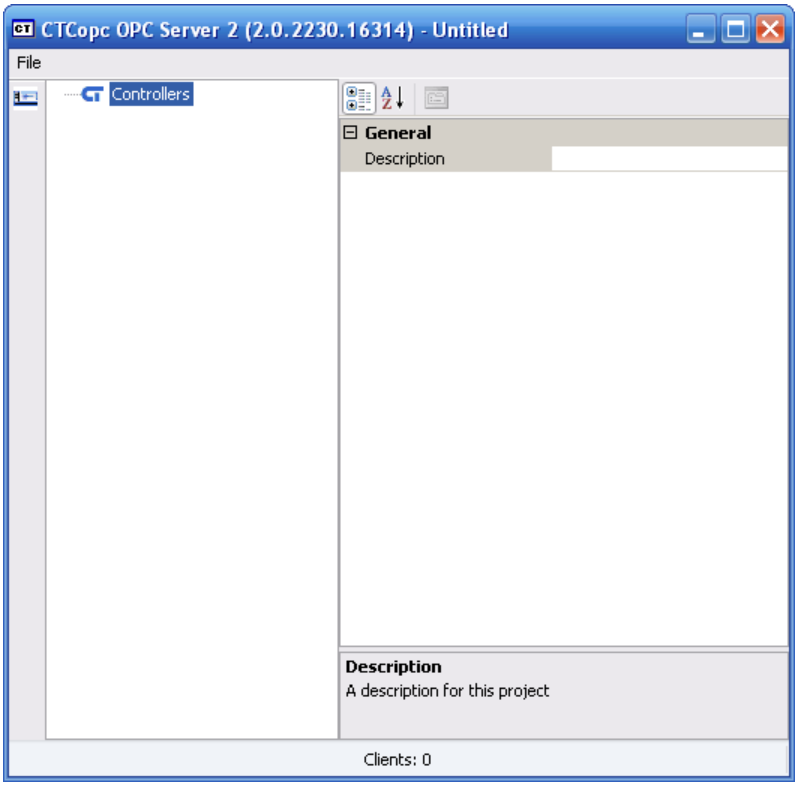
3

CTCopc 2 Projects

An introduction to creating and editing projects in CTCopc 2

Once CTCopc is properly installed, the application can be launched (from the desktop icon, or the shortcut within the START menu).

After launching, an empty project is created and the main application window is presented:



The CTCopc application window consists of two primary panes as well as a left-docked toolbar.

The left-most pane is the *tag-tree* and visually represents the hierarchical structure of the CTCopc project. A CTCopc project consists of a series of controllers and associated tags (data). Since OPC itself is also hierarchical, this model fits well to that architecture.

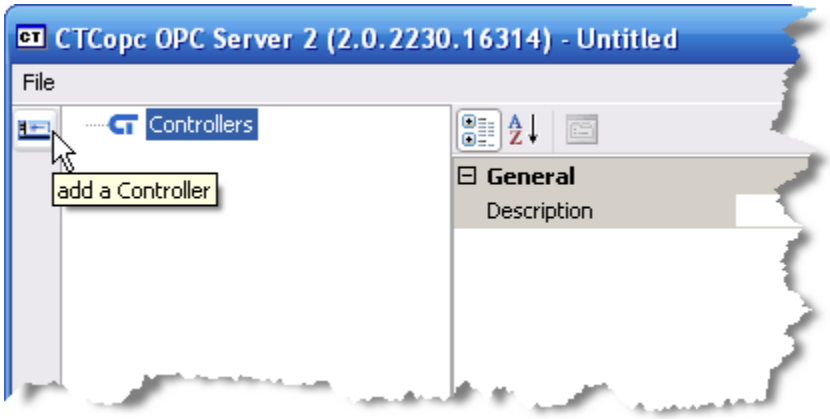
The right-most pane is called the *property editor* and allows the user to edit various parameters ("properties") for the selected node within the *tag-tree*.

The easiest way to understand the architecture of OPC and the CTCopc application is by a simple example.

CREATING YOUR FIRST PROJECT

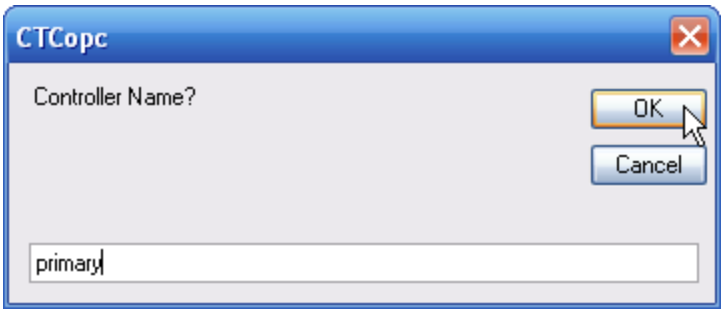
For this example, it is assumed that there are two controllers located on the network at IP addresses 192.168.44.71 and 192.168.44.73.

The first controller can be added to the (empty) CTCopc project by clicking the controller icon on the toolbar or by right-mouse clicking the **Controllers** node in the *tag-tree*.

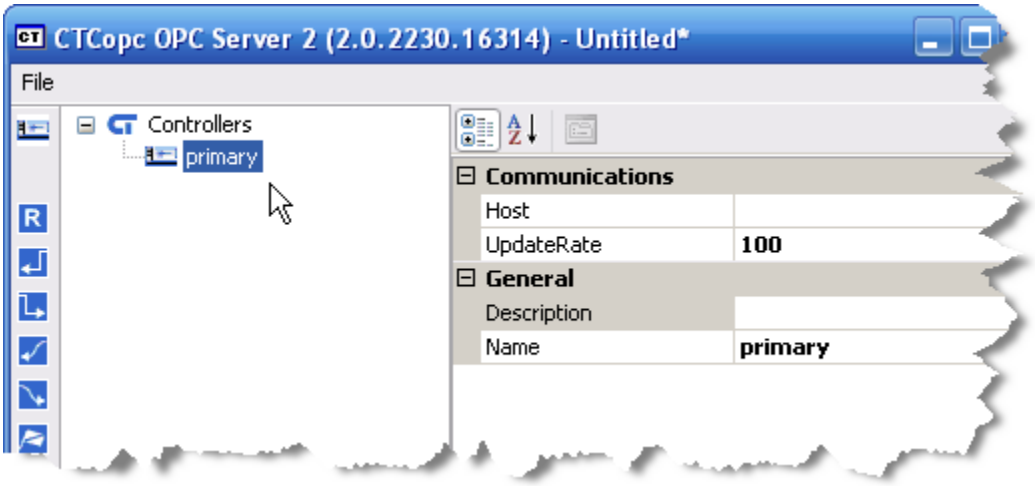


A dialog box will appear prompting for the name for this controller. This name is published out to OPC clients as well, so it should be chosen carefully (although it can be renamed).

In this example, we will call the controller at 192.168.44.71 "primary".



Once the name is entered and the **OK** button is pressed, the controller will appear in the *tag-tree*.

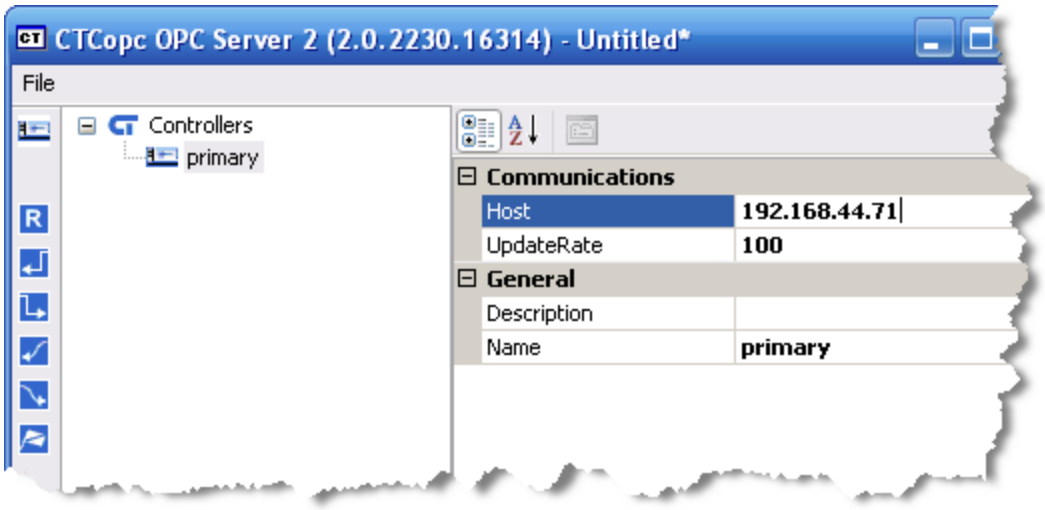


As you may have noticed, the right pane (the *property editor*) has filled in with a few default values for this node.

The *property editor* will change and update depending on which node is selected in the *tag-tree*. In this case, since a controller is selected the *property editor* shows only those properties relevant for this type of node.

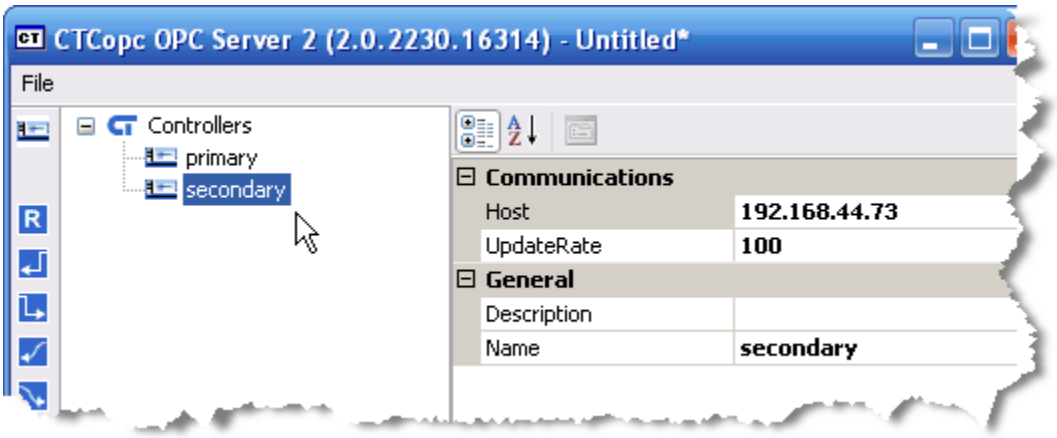
The most import property that we need to edit is **Host**. This property defines the address and method that the CTCopc application should use to communicate to the controller. In our example, the **Host** parameter should be edited to be 192.168.44.71 (an IP address that implies that the controllers are IP-networked).

This is accomplished by clicking in the blank field next to **Host** and entering the address.



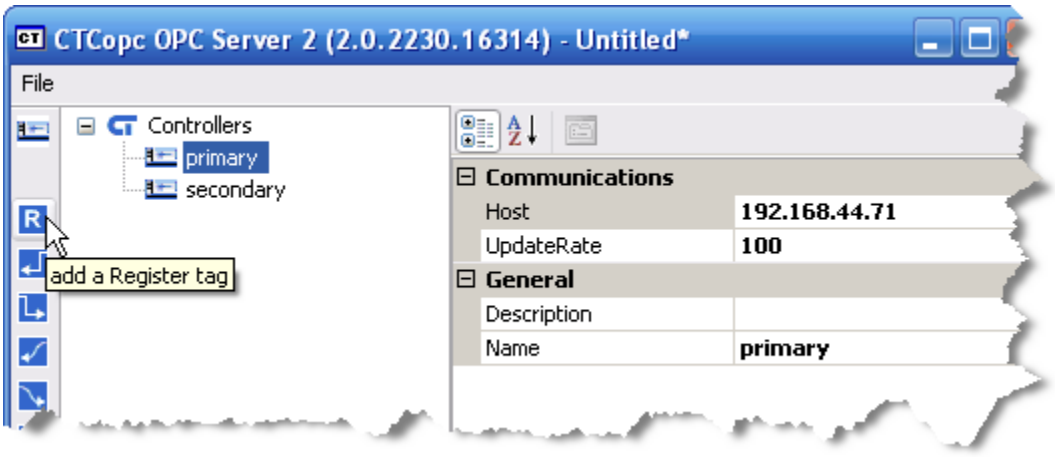
The other properties for a controller will be discussed later in this document.

Adding the second controller is as simple as repeating the process described above, except that we will use the name “secondary” and the IP address 192.168.44.73.

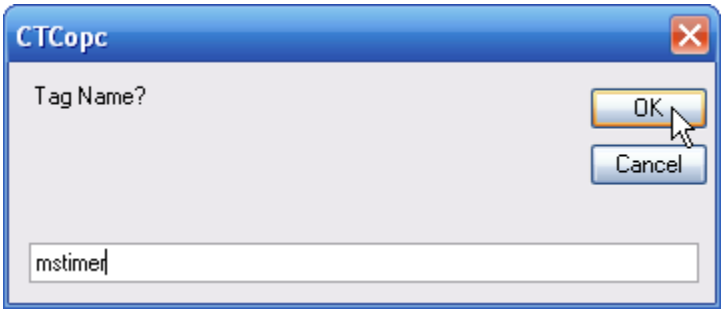


To add a tag to a controller, you click on it within the *tag-tree* and then click the appropriate icon in the toolbar for the type of tag you want to create.

For this example, we want to add the millisecond timer (register 13002) to each of the controllers, so we will be clicking on the icon that resembles the letter “R”.

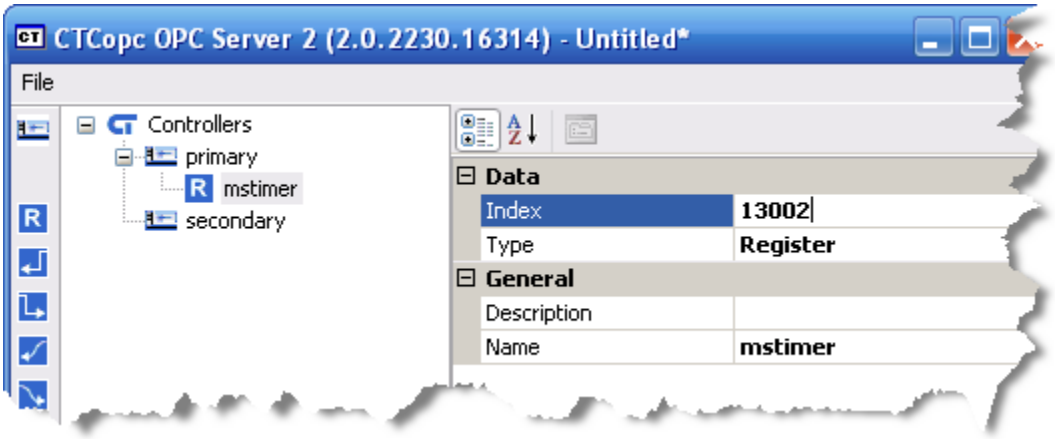


A prompt will appear for the name of the tag. In this example, we will choose the name "mstimer" and then click the OK button.

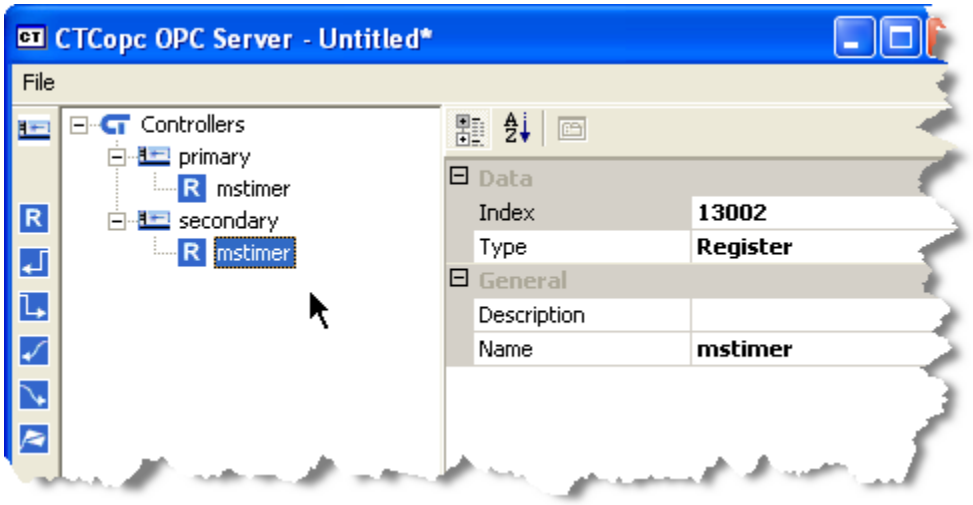


If a numeric name (such as "13002") is entered for a tag name, then **Index** property is automatically set to the value of the numeric name. If an alphanumeric name is entered (such as "mstimer") then the **Index** property must be manually set.

Therefore, before creating the "mstimer" tag for the "secondary" controller, we should edit the register number for this tag located in the **Index** property from its default value of 0 (disabled) to a value of 13002.



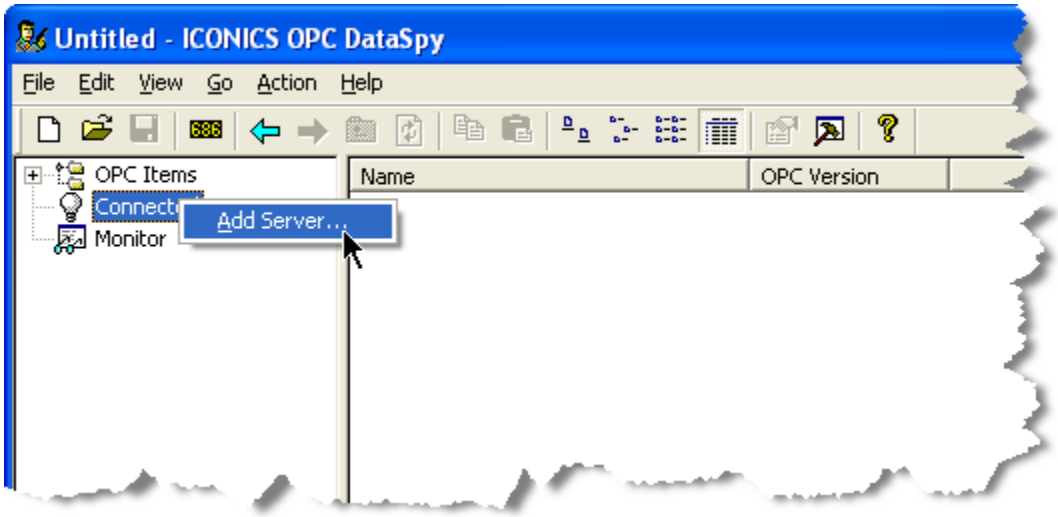
Next, we click on the “secondary” controller and repeat the process.



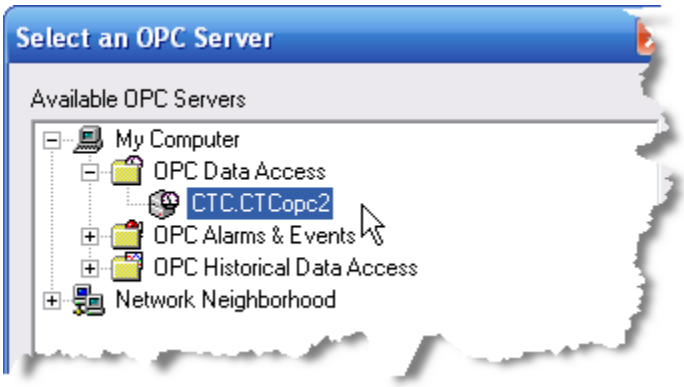
Next, we can launch an OPC client application to view these tags.

For this example, we will use a freely available test client from ICONICS called OPC DataSpy.

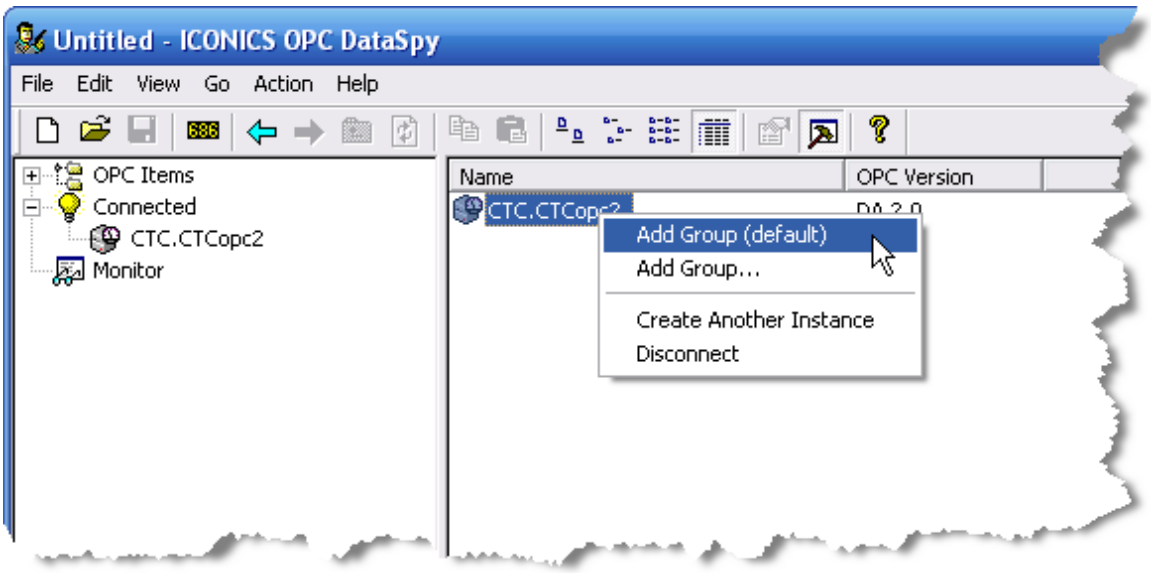
After launching DataSpy, we first need to create a connection (by right-mouse clicking on **Connected**) to the CTCopc OPC Server.



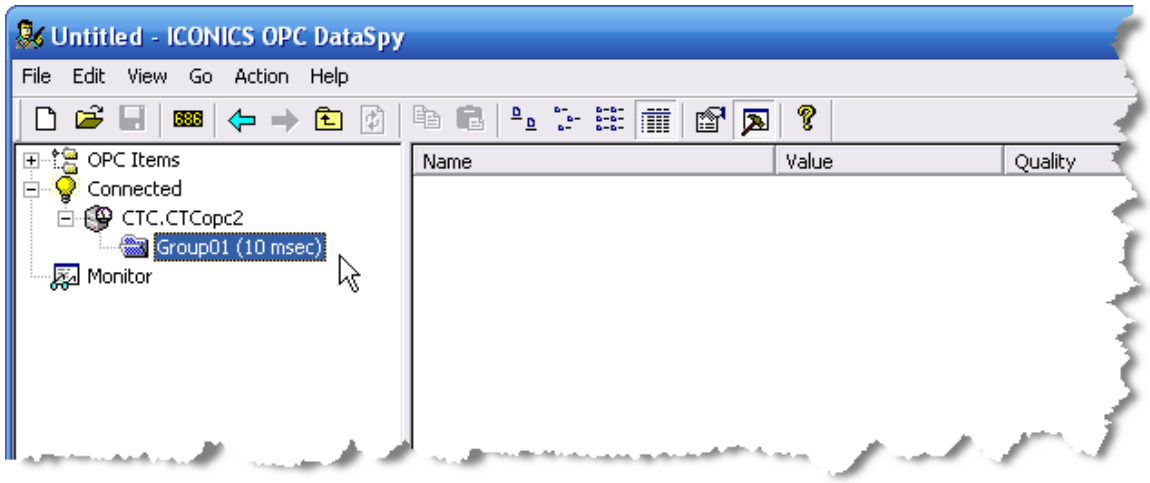
The name of the server as published to OPC clients is **CTC.CTCopc2**. Choose this server from the dialog box and press OK.



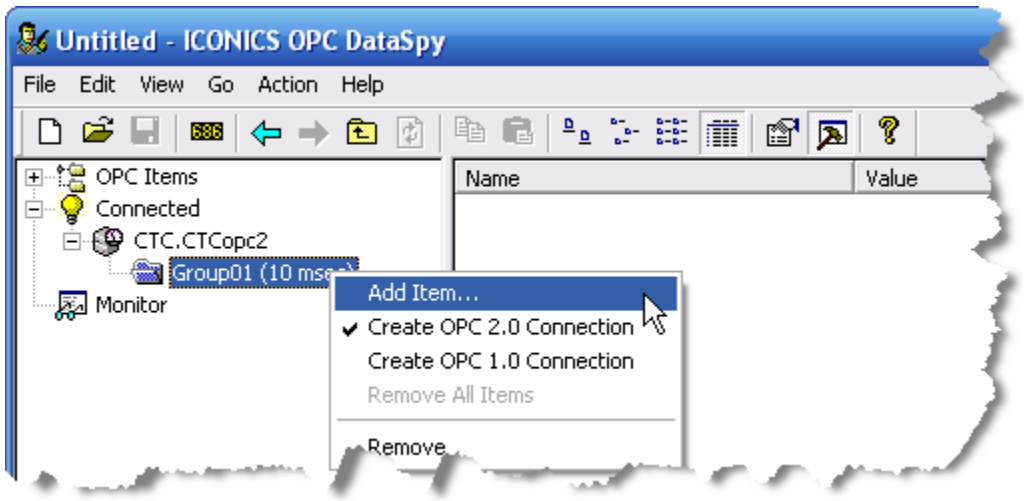
Next, add a default group to the server. OPC Groups are a part of the OPC specification and their purpose is as a container for published OPC tags.



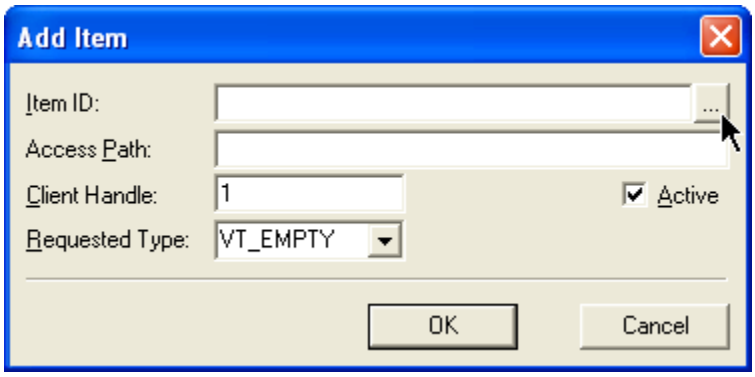
After adding a default group to the CTCopc OPC server, the DataSpy application window will show the new group (named **Group01**). After the group is created, select it as shown.



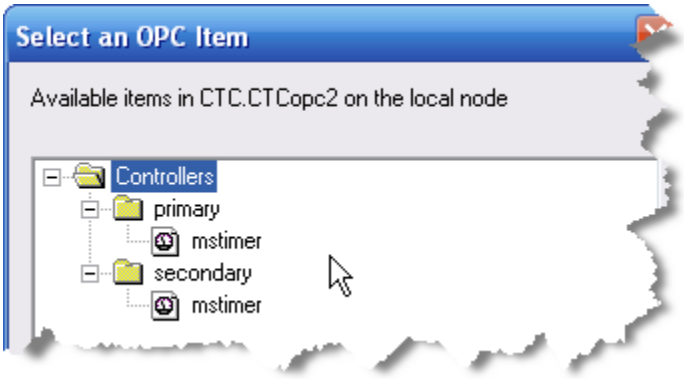
Next, we need to add a tag to this group that the CTCopc server is publishing. Right-mouse clicking on the group (**Group01**) and click **Add Item**.



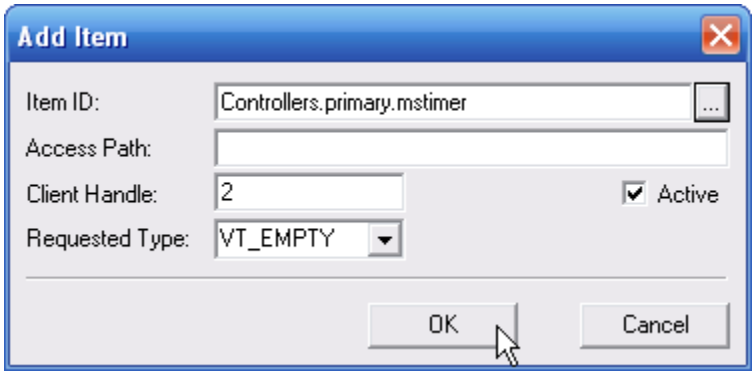
A dialog box will appear to select the item to add. By clicking on the ellipsis icon (located in the upper right), we can select a published tag to add.



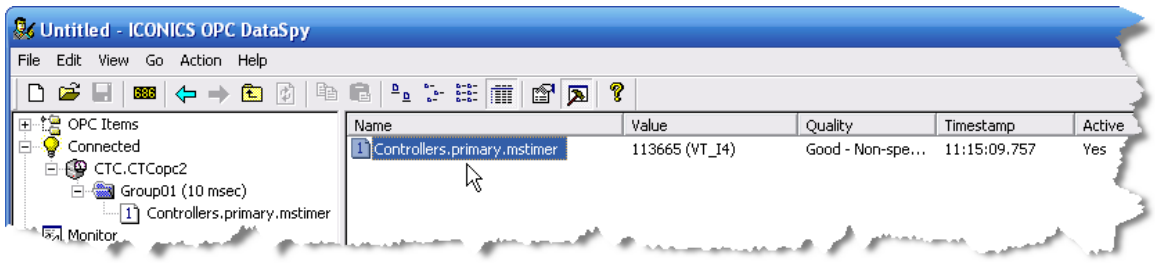
As you can see, the two defined controllers along with their respective tags are available.



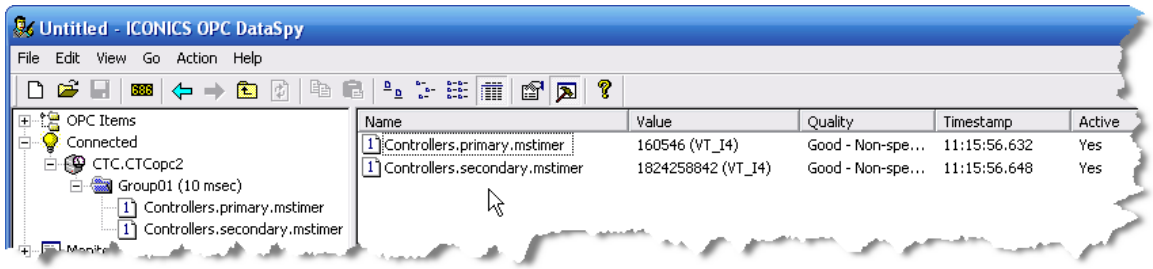
For this example, we'll select (by double-clicking) the **mstimer** tag from the first ("primary") controller and then adding it to DataSpy by clicking OK.



The tag will appear as part of the **Group01** group, and the tag will update in the right pane.



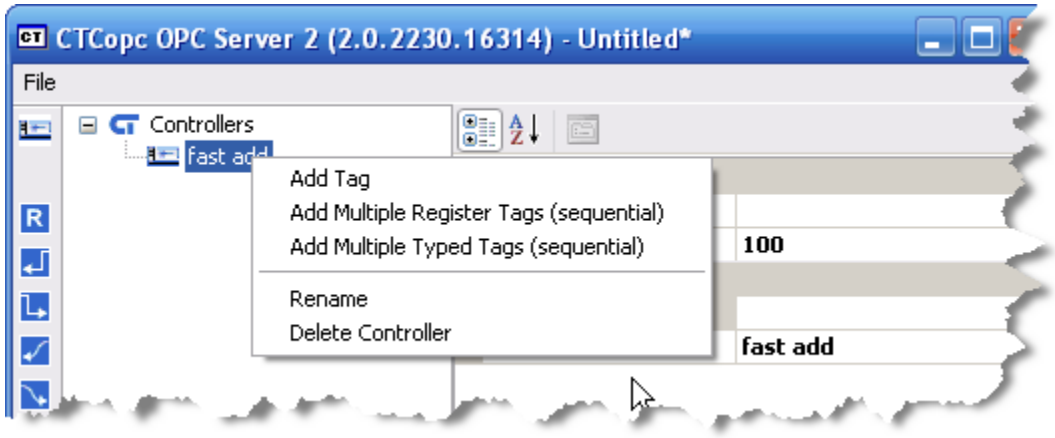
This step can be repeated for the second published tag (from the "secondary" controller) as well.



Both tags will continue to update as long as the CTCopc server application is open.

ADDING MULTIPLE TAGS

There are a couple shortcut methods to add multiple tags at once. If you right-mouse click on a controller, a menu will appear:

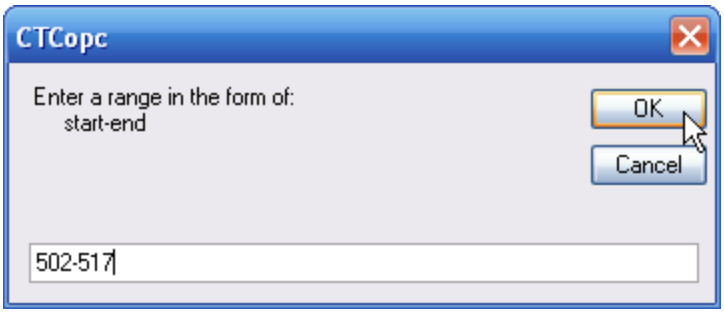


The first menu item ("Add Tag") is a simple shortcut to adding a register tag – it is the same as clicking the "R" icon along the left.

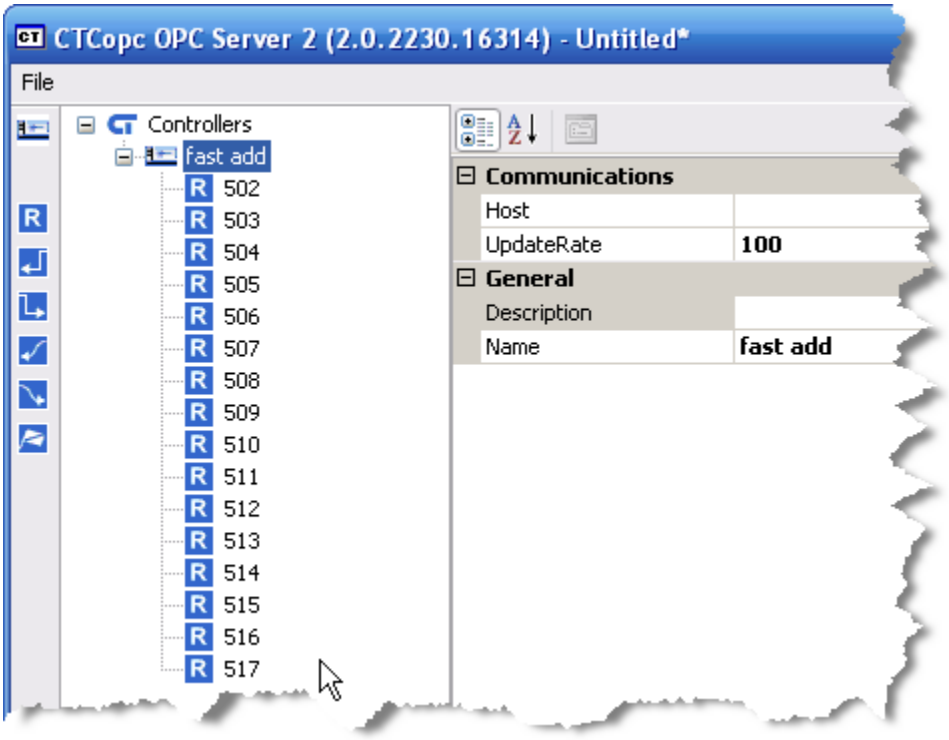
The second menu item ("Add Multiple Register Tags (sequential)") adds a series of numerically-named tags in the specified range.

As a safety precaution, only 100 tags at a time can be created using this feature – to add more than 100 tags at a time, break the add operation up into several add operations of no more than 100 tags each.

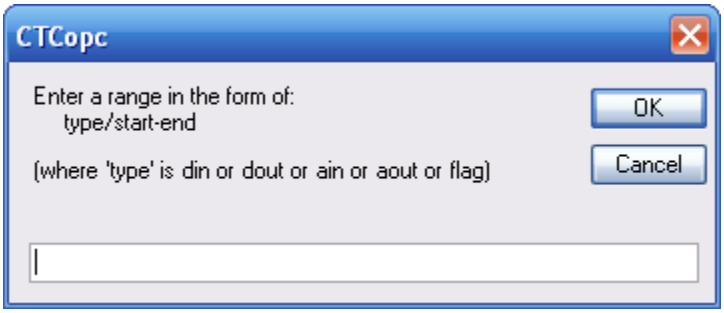
If we wanted to create a series of tags for register 502 through 517 (inclusive), we could enter 502-517 in the dialog box which is shown:



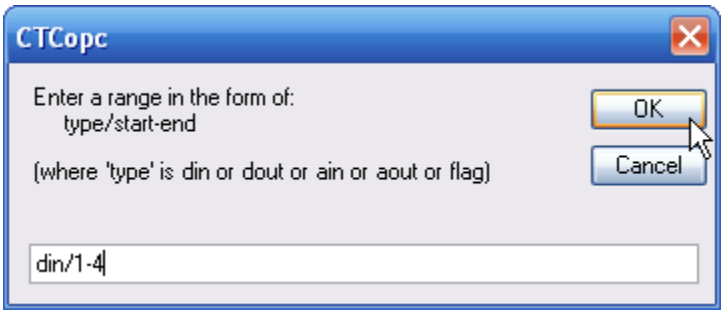
A series of tags is created and added (with their **Index** properties set appropriately) in the controller:



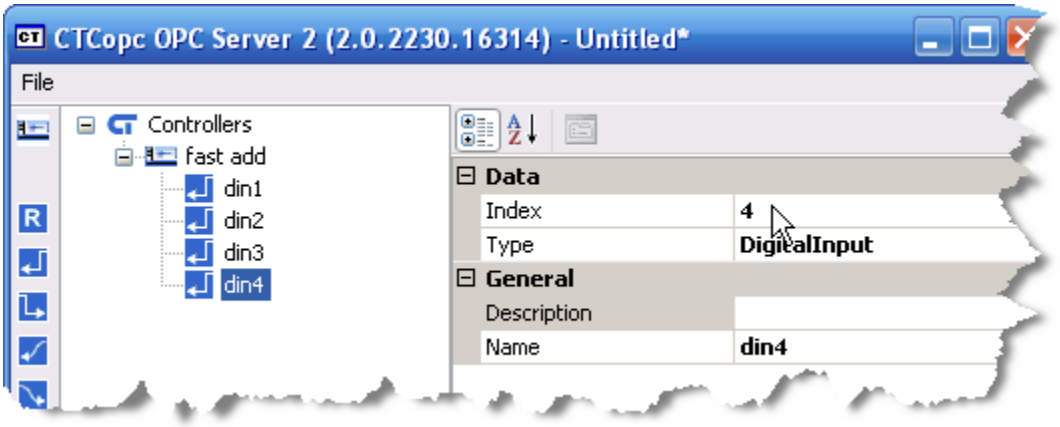
The third menu item ("Add Multiple Typed Tags (sequential)") adds a series of non-register tags. Selecting this item brings up a dialog box:



At this point the type of the tag must be entered followed by a slash ("/") and then a range to add. If, for example we wanted to create tags for the first 4 digital inputs ("din"), we would enter "din/1-4":



After entering this and clicking OK, the four tags would be automatically created along with the proper **Index** property set:



The five allowed types are their meaning are:

din	Digital input
fout	Digital output
ain	Analog input
aout	Analog output
flag	Flag register

Notes:

1. You cannot exit the OPC Server while there is an active client connected, this is by design. The Client must disconnect first. If you wish to force a disconnect you can shut the GUI down by selecting the CTCOpc application within Task Manager, right click and select 'End Task'.

2. If adding or changing a tag you must first disconnect the client, add/change the tag, then Save and Exit. Then reconnect the client before it gets published.

CHAPTER

4

Properties

Each tree-tag node type has a specific set of properties that governs the tags behavior.

There are three fundamental *tag-tree* node types used in the CTCopc application. Each type of node has a specific set of properties associated with it.

This chapter describes each type and the properties available for the node.

ROOT NODE

The first type of *tag-tree* node is called the *root node* and is the uppermost node in the entire *tag-tree* hierarchy. It contains all other nodes defined in the project. It appears in the *tag-tree* as the CTC logo with the text, *Controllers*.

DESCRIPTION

The ***Description*** property for the *root node* is simply used to hold a summary description for the entire CTCopc project. It may be left blank since it is not exposed to OPC clients in any way.

CONTROLLER NODE

The second type of *tag-tree* node, the *controller node*, represents a physical CTC automation controller. The *controller node* contains all of the tags exposed to client OPC applications.

DESCRIPTION

The ***Description*** property for the *controller node* is used to hold a summary description for this controller. It may be left blank as it is not exposed to OPC clients.

NAME

The ***Name*** property holds the prefix used when *tag nodes* are published to OPC clients. All published tags within the CTCopc project hierarchy are published to clients as:

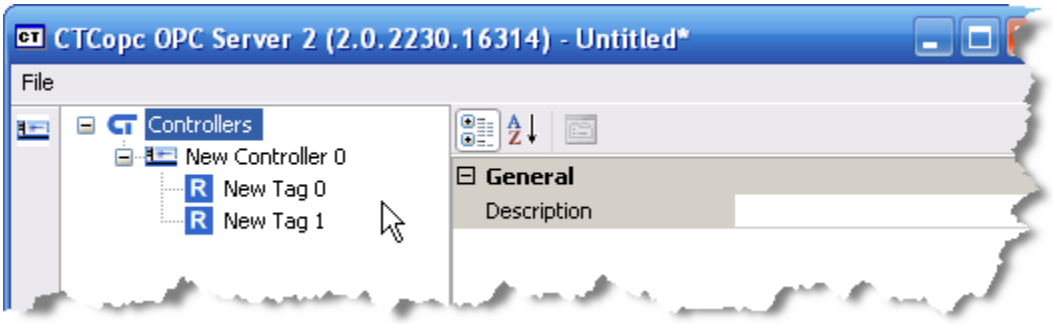
Controllers.prefix.suffix

The *prefix* is the defined name of *parent controller node* and the *suffix* is the defined name of the *child tag node*.

In the following example, there would be two published OPC tags, namely:

Controllers.New Controller 0.New Tag 0

Controllers.New Controller 0.New Tag 1



HOST

The ***Host*** property defines the address and method that the CTCopc application uses to communicate with the physical CTC automation controller.

This field is typically entered as an IP address in the form of a dotted-quad (XX.XX.XX.XX) – however, there are several other options available for this property covered in **Appendix A**.

UPDATERATE

The ***UpdateRate*** property defines how often each of the tags defined beneath this controller are updated and published to OPC clients. It is a numeric value defined in milliseconds.

The minimum allowed value is 10 milliseconds.

TAG NODE

The last type of *tag-tree* node, the *tag node*, represents a single entity residing in a physical controller.

DESCRIPTION

The ***Description*** property for the *tag node* is used to hold a summary description for this tag. It may be left blank as it is not exposed to OPC clients.

NAME

The ***Name*** property holds the suffix used this *tag node* is published to OPC clients. The *prefix* is the defined name of *parent controller node* and the *suffix* is the defined name of this *child tag node*.

INDEX

The ***Index*** property is used in conjunction with the ***Type*** property described below.

It is very important that the ***Index*** property is edited from its default value of 0 (disabled).

TYPE

The ***Type*** property defines what entity within the controller should be exposed to OPC clients for this *tag node*. The ***Index*** property is used in conjunction with the ***Type*** property as follows.

Type	Index
Register	The register number to read (or write to) within the controller. For example, a value of 13002 represents the millisecond timer.
DigitalInput	The 1-based digital input number.

	For example, a value of 3 represents the third digital input connected to the controller.
DigitalOutput	The 1-based digital output number. For example, a value of 2 represents the second digital output connected to the controller.
AnalogInput	The 1-based analog input number. For example, a value of 4 represents the fourth analog input connected to the controller.
AnalogOutput	The 1-based analog output number. For example, a value of 1 represents the first analog output connected to the controller.
Flag	The 1-based flag number. For example, a value of 6 represents the sixth flag defined in the controller.

Reference the 'Property Enhancements' Appendix for additional *tag node* properties available.

CHAPTER 5

Specifications

Specifications for CTCopc 2

Specifications are accurate at time of printing and are subject to change without notice.

OPC DA compatibility	Compatible to the OPC DA 1.0, 2.0 and 3.0 specifications.
Total number of allowed controllers defined within a project	Up to 500, but dependent on the version purchased and authorized.
Total number of allowed tags defined per controller	Unlimited, but logically limited to about 5000.
Total number of allowed tags defined within a single project	Unlimited, but logically limited to about 250000.
OPC Variant type	All published tags are published as 4-byte (32 bit) integers using the OPC Variant type VT_I4.

OPC Variant type – Updated to support VT_I4, VT_R4, VT_R8, and VT_BSTR, reference Appendix B for additional information. Only 5300 Variant registers support VT_BSTR.

APPENDIX

A

Host Property

*Available **Host** communication methods and settings for CTCopc 2 controllers*

As mentioned in an earlier chapter, the **Host** property for a controller defines host the CTCopc application communicates with a controller.

There are two primary methods that CTCopc can communicate with a controller:

through simple IP communications

through the CTC communications DLL "CTCCOM32V2.DLL"

Depending on how you format the **Host** property determines the method used.

The installation of CTCCOM32V2.DLL is not covered in this document.

The following table summarizes the format of the **Host** property.

FORMAT	EXAMPLE	MEANING
IPAddress	192.168.1.73	Use UDP communications to the stated IP address without caching or block-reads
IPAddress-	192.168.1.73-	Use UDP communications to the stated IP address with read-caching but without block-reads (old controller software support)
IPAddress+	192.168.1.73+	Use UDP communications to the stated IP address with read-caching and block-reads
ctudp=IPAddress	ctudp=192.168.1.73	Use UDP communications to the stated IP address without caching or block-reads
ctudp=IPAddress-	ctudp=192.168.1.73-	Use UDP communications to the stated IP address with read-caching but without block-reads (old controller software support)
ctudp=IPAddress+	ctudp=192.168.1.73+	Use UDP communications to the stated IP address with read-caching and block-reads
cttcp=IPAddress	cttcp=192.168.1.73	Use TCP communications to the stated IP address without caching or block-reads
cttcp=IPAddress-	cttcp=192.168.1.73-	Use TCP communications to the stated IP address with read-caching but without block-reads (old controller software support)
cttcp=IPAddress+	cttcp=192.168.1.73+	Use TCP communications to the stated IP address with read-caching and block-reads
ctdll=*n*#	ctdll=n2	Use the CTCCOM32v2 DLL for CTNET communications to a specified destination node (see note)
ctdll=*n*#+	ctdll=n2+	Use the CTCCOM32v2 DLL for CTNET communications to a specified destination node using read-caching and block-reads (see note)
ctdll=*s*#	ctdll=s1	Use the CTCCOM32v2 DLL for Serial communications using the specified "COM" port
ctdll=uIPAddress	ctdll=u192.168.1.73	Use the CTCCOM32v2 DLL for UDP communications to the controller at the specified IP
ctdll=uIPAddress+	ctdll=u192.168.1.73+	Use the CTCCOM32v2 DLL for UDP communications to the controller at the specified IP using read-caching and block-reads
ctdll=tIPAddress	ctdll=t192.168.1.73	Use the CTCCOM32v2 DLL for TCP communications to the controller at the specified IP
ctdll=tIPAddress+	ctdll=t192.168.1.73+	Use the CTCCOM32v2 DLL for TCP communications to the controller at the specified IP using read-caching and block-reads

When CTCopc 2 utilizes the CTCCOM32v2 DLL for CTNET communications it always uses a host node number (for itself) of 32555. At present, there is no way to alter this value.

APPENDIX

B

Property Enhancements

Description Property Enhancements made since the manual was originally written.

Periodic updates and improvements have been added to CTCopc with the release of V2.21B. The basic operation remains the same as previously described but additional property fields have been added to allow such features as scaling, floating point, string, and QuickBuilder Variant data table support. Project files will automatically be converted, upon loading, adding the additional information. By default this information is compatible with that previously presented to the OPC Client.

Once an old project is loaded and then saved it is not compatible with prior releases of CTCopc. It is recommended that a backup of the .cop project file be made prior to using the new release.

Tags previously could only present their information in 'integer' format. With the new format any tag may be integer, float32 (32 bit floating point), or float64 (64 bit double, IEEE 754). 5300 Controller Variant registers also allows for the 'string' format (36000 register block).

Tag Node Properties have been updated:

TAG NODE PROPERTY UPDATES

The *tag node* represents a single entity residing in a physical controller.

DESCRIPTION

The ***Description*** property for the *tag node* is used to hold a summary description for this tag. It may be left blank as it is not exposed to OPC clients.

NAME

The ***Name*** property holds the suffix used this *tag node* is published to OPC clients. The *prefix* is the defined name of *parent controller node* and the *suffix* is the defined name of this *child tag node*.

INDEX

The ***Index*** property is used in conjunction with the ***Type*** property described below. It represents the resource index within the controller, such as a register or IO number.

It is very important that the ***Index*** property is edited from its default value of 0 (disabled).

TYPE

The ***Type*** property defines what entity within the controller should be exposed to OPC clients for this *tag node*. The ***Index*** property is used in conjunction with the ***Type*** property as follows.

Type	Index
Register	The register number to read (or write to) within the controller. For example, a value of 13002 represents the millisecond timer.
DigitalInput	The 1-based digital input number. For example, a value of 3 represents the third digital input connected to the controller.
DigitalOutput	The 1-based digital output number. For example, a value of 2 represents the second digital output connected to the controller.
AnalogInput	The 1-based analog input number. For example, a value of 4 represents the fourth analog input connected to the controller.
AnalogOutput	The 1-based analog output number.

	For example, a value of 1 represents the first analog output connected to the controller.
Flag	The 1-based flag number. For example, a value of 6 represents the sixth flag defined in the controller.

DATA

The ***Data*** property for the *tag node* is used to tell the server what data type to use when it exposes the resource to the OPC clients. A pull down is available listing 'integer' (VT_I4), 'float32' (VT_R4), and 'float64' (VT_R8). Variant registers (5300 controller only) may also expose the resource as a 'string' (VT_BSTR). For integer only resources, such as IO, a conversion to the desired format will be done.

VARIANT ROW

The ***Variant Row*** property applies only the Variant registers, where a Variant may also be a two dimensional table or one dimensional vector. Each cell in a table can be represented independently to the OPC Client. This property selects the row, which on non-Variant registers is always 0.

VARIANT COL

The ***Variant Col*** property applies only the Variant registers, where a Variant may also be a two dimensional table or one dimensional vector. Each cell in a table can be represented independently to the OPC Client. This property selects the column, which on non-Variant registers is always 0.

OFFSET

The ***Offset*** property is the value which will be added to the scaled value of a resource prior to presentation to the OPC client. Logic is "(Resource Value X ***Scale***) + ***Offset***". By default this is 0. The offset property value is applied as a double, float64.

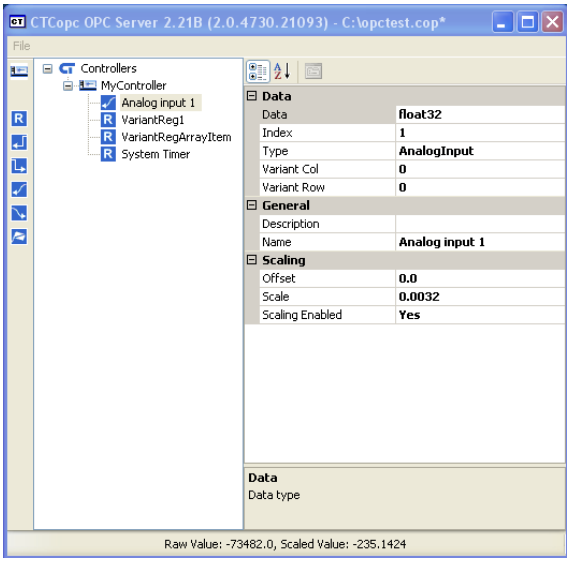
SCALE

The ***Scale*** property is the value which will be multiplied by the Resource, result being added to the ***Offset***, prior to presentation to the OPC client. Logic is "(Resource Value X ***Scale***) + ***Offset***". By default this is 1.0, representing no scaling. The scaled property value is applied as a double, float64.

SCALING ENABLED

The ***Scaling Enabled*** property is a pull down with the selection of *Yes* or *No*. *Yes* enables scaling, *No* disables it. When scaling is not used it is best to set this selection to *No* both for accuracy (doubles can have a small amount of error in representation) and performance.

Example:



Note above where scaling is enabled on an AnalogInput. The value read from the controller of -73482 (Raw Value) is multiplied by .0032 yielding a result of -235.1424 (Scaled Value). Although the controller represents the AnalogInput as an integer, the OPC Client will see a VT_R4, floating point, representation of the data with a value of -235.1424.

Note that the OPC Server and OPC Client must be shutdown and restarted for the OPC Client to properly reflect any data type changes that are made after the initial connection. If this is not done the OPC Client will continue to read the data in the previous selected data type. The OPC Server screen will immediately reflect the changes, regardless of whether it is restarted. This only effects the OPC Client connection.