OmniServer by Software Toolbox

USER MANUAL

Welcome to OmniServer by Software Toolbox

1 Welcome to OmniServer by Software Toolbox

OmniServer by Software Toolbox: The industry leader for unsurpassed quality and performance in data acquisition since 1995.

OmniServer is an easy to use, yet extremely powerful solution designed to help you communicate with virtually any one device or multiple devices. This is done through a series of dialog boxes that guide you through building a description of the data stream used to communicate with the device. Once built, OmniServer takes care of the communications with the device and the delivery of the data to a client program.

Thank you for choosing OmniServer by Software Toolbox. This Help File has been provided to assist you in using and implementing the server over a wide range of applications.

Before you begin, here are some important links to review:

- General Installation Information
- Upgrading from Version 1.x
- Upgrading from Version 2.0
- Upgrading from Version 2.5
- Upgrading from Version 2.6 (or higher)

⊟ Click here if you are a first time user

Since this is your first time using the server, we suggest you follow this path through the help file. It will give you the quickest and best overall knowledge of how the server works.

- Getting Up and Running
- Tutorials
- Troubleshooting

⊟ Click here if you've upgraded from Version 1.x

You are familiar and/or have used previous versions of the server. Although the functionality is the same, the user interface has changed dramatically.

To guide you through what is new with the server, we suggest you take the following path:

- Server Details: User Interface
- Server Details: Configuration Windows
- Server Details: Diagnostic Windows
- Tutorials
- Trouble Shooting

⊟ Click here if you've upgraded from Version 2.0

- You will need to re-license your server. Please see the document Registration for more information.
- While the GUI interface has not changed, there are some changes internally for which you will need to be aware. Please review the Release History for a complete list of changes from V2.0 to this current version.

⊟ Click here if you've upgraded from Version 2.5

- You will need to re-license your server. Please see the document Registration for more information.
- While the GUI interface has not changed, there are some changes internally for which you will need to be aware. Please review the Release History for a complete list of changes from V2.5 to this current version.

⊟ Click here if you've upgraded from Version 2.6 (or Higher)

- You will need to re-license your server. Please see the document Registration for more information.
- While the GUI interface has not changed, there are some changes internally for which you will need to be aware. Please review the Release History for a complete list of changes from V2.6 (or higher) to this current version.

Need More Help: Customer support information.

End-User License Agreement

2 End-User License Agreement

Software Toolbox OmniServer End User License Agreement and Limited Warranty

By opening this package or clicking the OK button, you indicate that you have read this agreement and accept its terms and conditions. This is a legal agreement between you (the end user or “Licensee”) and Software Toolbox (Matthews, NC) , hereafter the “Licensor”. If you do not agree to the terms of this Agreement, promptly return the disk package and accompanying items (including written materials) to the place where you obtained them for a full refund.   The software accompanying this license agreement (the Software) is the property of Licensor or its licensors, and is protected by United States and International Copyright laws and International treaty provisions. No ownership rights are granted by this Agreement or possession of the Software. Therefore, you must treat the Software like any other copyrighted material (e.g., a book or musical recording), except that you may make a single copy for backup or archival purposes.

For purposes of this Agreement, a “Computer” is defined as a single physical computer or a single instance of a virtual PC, virtual server, whether it be implemented using Microsoft Virtual PC, Virtual Server, Hypervisor, VMWare, or other virtual computing software applications that enable multiple isolated operating system instances to be run on a single hardware platform.

Return Policy

The original licensee of the Software can return it within thirty (30) days of purchase. Please call us for a Return Material Authorization Number. Returns after 30 days of purchase may require a restocking fee. Any extensions of this return period must be pre-approved by Licensor technical support and documented in writing, where e-mail is considered to be in writing. No returns are accepted without a Return Material Authorization Number.

Demonstration and Enabled Software

The software may run in two modes, demonstration and enabled. A software license number ("License Code") is provided with each purchased copy of the Software. The Software runs in demonstration mode without the License Code. Once the Licence Code is entered into the software during the registration process, a code to unlock the software (“Unlock

Code") will be provided by the Licensor. The software will operate in enabled mode only with both the License Code and the Unlock Code.

Grant Of License

1. Licensor grants to you the right to use the Software in demonstration on any computer.

2. Licensor grants to you the right to use the License Code and Unlock Code on a single computer. You may load one copy into permanent memory of one computer and may use it only on that same computer. Hence, the License Code and Unlock Code may enable the Software only on one computer.

Restrictions on Use and Transfer

1. If you have not purchased the license, then you may only execute the Software in a demonstration mode for evaluation purposes only.

2. This Software package contains only a CD-ROM disk. You may not make copies of the CD-ROM disk itself.

3. You may not transfer or communicate the License Code or Unlock Code to another user except as part of a permanent transfer as detailed below.

4. You may not copy the written materials, the License Code or the Unlock Code.

5. You may permanently transfer the Software, License Code, Unlock Code, and accompanying written materials (including the most recent update and all prior versions) if you retain no copies and the transferee agrees to be bound by the terms of this Agreement. Such a transfer terminates your license. You may not rent or lease the Software or otherwise transfer or assign the right to use the Software, except as stated in this paragraph.

6. You may not reverse engineer, decompile, or disassemble the Software.

Limited Warranty

1. Licensor warrants that the Software will perform substantially in accordance with the accompanying written materials for a period of 90 days from the date of your receipt of the Software. Any implied warranties on the Software are limited to 90 days. Some states do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you.

2. LICENSOR DISCLAIMS ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, WITH RESPECT TO THE SOFTWARE AND THE ACCOMPANYING WRITTEN MATERIALS. This limited warranty gives you specific legal rights. You may have others, which vary from state to state.

3. LICENSOR'S ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE, AT LICENSOR'S CHOICE, EITHER (A) RETURN OF THE PRICE PAID OR (B) REPLACEMENT OF THE SOFTWARE THAT DOES NOT MEET LICENSOR'S LIMITED WARRANTY AND WHICH IS RETURNED TO LICENSOR WITH A COPY OF YOUR RECEIPT. Any replacement Software will be warranted for the remainder of the original warranty period or 30 days, whichever is longer. These remedies are not available outside the United States of America.

4. This Limited Warranty is void if failure of the Software has resulted from modification, accident, abuse, or misapplication.

5. IN NO EVENT WILL LICENSOR BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY LOSS OF PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OR INABILITY TO USE THE SOFTWARE. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

Dual-Media Software.

You may receive the Software in more than one medium. Regardless of the type or size of medium you receive, you may use only one medium that is appropriate for the Computer. You may not use or install the other medium on another computer. You may not loan, rent, lease, or otherwise transfer the other medium to another user, except as part of the permanent transfer (as provided above) of the Software.

Severability

Each section and subsection of this Agreement shall be considered severable, and if any provision of this Agreement shall be held illegal, invalid, or otherwise unenforceable under controlling law, the remaining provisions of this Agreement shall not be affected thereby but shall continue in effect.

Export Restrictions

You acknowledge that the Software is subject to U.S. export jurisdiction. You agree to comply with all applicable international and national laws that apply to the Software, including the U.S. Export Administration Regulations, as well as end-user, end- use and destination restrictions issued by U.S. and other governments. Prior to delivery of product, you may be required to provide to Licensor the necessary documentation as required by the U.S. government and it’s agencies to certify the final end user destination of the Software. If such documentation is requested and you do not or are not able to provide it, the Software will not be delivered to you in any form, and any pre-paid funds for license fees will be refunded to you.

Trademarks

Software Toolbox and OmniServer are registered trademarks of Software Toolbox, Inc.

Other

1. This Agreement is governed by the laws of the State of North Carolina, United States of America.

2. Support is limited to phone, or email for 30 days after purchase unless the Licensor has a valid maintenance contract which would extend support.

3. Re-registration, which is the issuance of a new Unlock Code for the same License Code for use of the software on the same or a different computer, often the result of loss of license because of hardware failure or re-installation of software is limited to 1 events without a maintenance contract. LICENSOR reserves the right to deny issuance of a new Unlock Code if it has reason to believe that the reason for the request of a new Unlock Code is to circumvent the terms of this Agreement granting license to use the Software on a single computer.

4. If you have any questions concerning this Agreement or wish to contact Licensor for any reason, please write: Software Toolbox, Inc. 148A East Charles Street, Matthews, North Carolina 28105 USA or call +1-704-849-2773.

5. Government Restricted Rights. The Software and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at

DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software—Restricted Rights at 48 CFR 52.227-19, as applicable.

Third Party Licenses and Open Source License Acknowledgements

The following license terms apply to OPC UA server interface portions of the OmniServer product.

1 OpenSSL

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

1.1 Copyright

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)

This Windows version of this product includes software written by Tim Hudson (tjh@cryptsoft.com)

1.2 License Issues

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact openssl-core@openssl.org.

OpenSSL License:

Copyright © 1998-2007 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Original SSLeay License:

2 LibXML2

This product includes code that was developed for the XML toolkit from the GNOME project (http://xmlsoft.org/).

2.1 Copyright

Copyright (C) 1998-2003 Daniel Veillard. All Rights Reserved.

2.2 License Issues

The libxml2 library is released under the MIT Licence and includes following copyright notice:

3 OPC UA ANSI C Stack

This product includes software that was developed by the OPC Foundation (http://www.opcfoundation.org/). The included software, the OPC UA ANSI C Stack, contains of two modules which are different in licensing:

- the Ansi C Stack Portability Layer (containing the platform specific abstraction)
- the Ansi C Stack Core Module (containing the general functionality)

3.1 Ansi C Stack Portability Layer

This module is released under the MIT License and includes following copyright notice, except where otherwise noted in the source code all the files are:

3.1.1 Copyright

3.1.2 License Issues

sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The complete license agreement can be found here:

http://opcfoundation.org/License/MIT/1.00/

3.2 Ansi C Stack Core Module

This module is released under the RCL Licence and includes following copyright notice, except where otherwise noted in the source code all the files are:

3.2.1 Copyright

Copyright (c) 2005-2009 The OPC Foundation, Inc. All rights reserved.

3.2.2 License Issues

Unless explicitly acquired and licensed from Licensor under another license, the contents of this file are subject to the Reciprocal Community License ("RCL") Version 1.00, or subsequent versions as allowed by the RCL, and You may not copy or use this file in either source code or executable form, except in compliance with the terms and conditions of the RCL.

All software distributed under the RCL is provided strictly on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, AND LICENSOR HEREBY DISCLAIMS ALL SUCH WARRANTIES, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT, OR NON-INFRINGEMENT. See the RCL for specific language governing rights and limitations under the RCL.

3.2.3 License in Short

The Ansi C Stack Core Module stays under the Reciprocal Community License (RCL) which is based on the concept of reciprocity or, if you prefer, fairness.

The RCL is adapted from the Open Source Reciprocal Public License (RPL) where the â€œPublicâ€ in the Open Source RPL license is replaced by the â€œCommunityâ€ in the RCL License. In short, the RPL license grew out of a desire to close loopholes in previous open source licenses, loopholes that allowed parties to acquire open source software and derive financial benefit from it without having to release their improvements or derivatives to the community which enabled them. This occurred any time an entity did not release their application to a "third party". While there is a certain freedom in this model of licensing, it struck the authors of the RPL as being unfair to the open source community at large and to the original authors of the works in particular. After all, bug fixes, extensions, and meaningful and valuable derivatives were not consistently faster, growth and expansion of the overall open source software base.

While you should clearly read and understand the entire license, the essence of the RCL is found in two definitions: "Deploy" and "Required Components".

Regarding deployment, under the RCL your changes, bug fixes, extensions, etc. must be made available to the community when you Deploy in any form -- either internally or to an outside party. Once you start running the software you have to start sharing the software.

Further, under the RCL all components you author including schemas, scripts, source code, etc. -- regardless of whether they're compiled into a single binary or used as two halves of client/server application -- must be shared. You have to share the whole pie, not an isolated slice of it.

The complete license agreement can be found here:

http://www.opcfoundation.org/License/Redistributables/1.00/

http://opcfoundation.org/License/RCL/1.00/

The following license(s) apply to the MQTT Client (Wedge) functionality

1. The Eclipse Pasho.Mqtt.C library (https://github.com/eclipse/paho.mqtt.c) is used for this functionality under the following license:

General Installation Information

3 General Installation Information

Obtaining the latest OmniServer Version:

You can download the latest version of OmniServer from the product website page. Details for existing licensed users are found there.

Install Location:

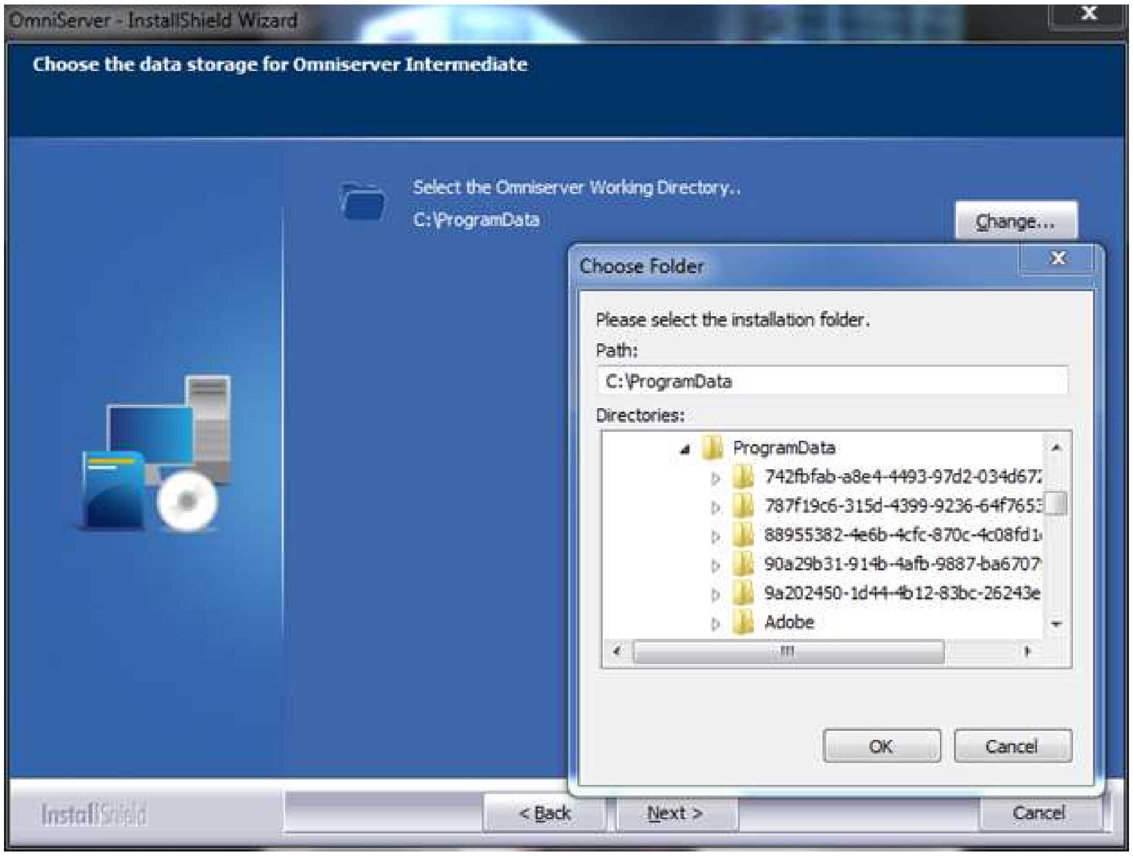
The server must be installed on a local drive the computer on which it is to run. Running the server from a network drive is not supported.

File Locations (OmniServer Configuration and Protocols):

Default file location is dependent on the operating system where OmniServer is installed (see below). This is a hidden directory on the operating system, so if your folder options do not expose hidden files/folders, you may need to edit the folder options before the referenced directory will be visible in Windows Explorer.

The folder structure created within the specified location will be \Software Toolbox\OmniServer

- Default - C:\ProgramData
- Non-default location may also be specified during the OmniServer installation process


OmniServer - InstallShield Wizard

Choose the data storage for Omniserver Intermediate

Select the Omniserver Working Directory..
C:\ProgramData

Choose Folder

Please select the installation folder.

Path:
C:\ProgramData

Directories:

ProgramData
742fbfab-a8e4-4493-97d2-034d67
787f19c6-315d-4399-9236-64f765
88955382-4e6b-4cfc-870c-4c08fd1
90a29b31-914b-4afb-9887-ba6707
9a202450-1d44-4b12-83bc-26243e
Adobe

Security:

This installation procedure makes no assumptions as to the security needs of your installation. For remote OPC DA connectivity, it might be required that DCOMCNFG be executed and the server's security settings be adjusted to meet your specific needs. Please contact your system's administrator for further details. For more information about DCOM, please see our section on DCOM.

For details on using the OPC UA interface, (no DCOM required),

please see our section on OPC UA.

[This section applies only to the OmniServer Server Edition and the OmniServer Professional Edition.]

Upgrading from Versions 1.x or Version 2.0 will require some changes in your client configuration.

For more information on upgrading from Version 1.x, please see our document on Upgrading from V1.X.

For more information on upgrading from Version 2.0, please see our document on Upgrading from V2.0.

Finally, if you are upgrading from Version 2.5 or higher, no changes on the client side are needed. However, there are some things of which you need to be aware. Please see our document on Upgrading from V2.5 or Upgrading from V2.6 (or Higher) for more information.

Re-licensing Your OmniServer:

No matter from which version of OmniServer you've upgraded, you will need to re-license or re-register your server. Please see our Registration page for more information.

Using Sample Protocols:

If you can use a sample protocol in your application, it is strongly recommended that you save the protocol under a different name. This is to prevent your changes from being overwritten by an upgrade of the product.

To make a copy of a protocol, locate the protocol file on your computer. It will be the same name as the protocol with *.DPD as the file extension. Copy the file as your normally would in Windows and rename the copy. Then restart the server in order for the change to take effect.

Using the Demonstration Version

4 Using the Demonstration Version

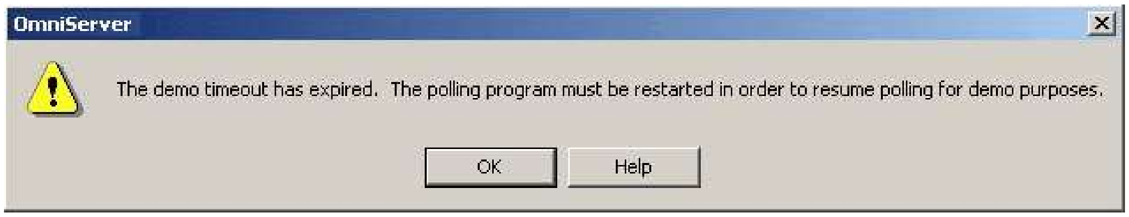
The server's Demonstration Version is a fully-functional product, except that some time limit can be imposed upon it. These time limits are:

- **Hourly:** The demonstration version will stop execution after a number of hours. The default is two hours. This is also the default time limit for a standard demonstration version.
- **Date:** The demonstration version will stop execution after a certain date. This can only be configured by the manufacturer of the server.

To determine which type of demonstration version you have, select **Help | About** from the server's menu. The splash screen that appears will tell you how long the demonstration version will run.

What to do when the Demonstration Version Ends

When you have reached the end of the demonstration version, you will get the following dialog box:


OmniServer

The demo timeout has expired. The polling program must be restarted in order to resume polling for demo purposes.

The Demonstration Version Time-Out Box

When this occurs, exit all opened dialog boxes and the Configuration Program (if currently opened).

If the runtime program is not running as a service, then you will need to exit the runtime program by right-clicking on the server's icon in the icon tray and selecting **Exit** from the pop-up menu. If the runtime program is running as a service, you will need to stop and restart the server's runtime service in the control panel.

You can then restart the server by launching it as normal.

Registering Your Server

5 Registering Your Server

Once OmniServer has been installed, it will be in Demonstration mode until the software has been registered. (See the help section on Demonstration Version.)

How do I get it out of Demonstration Mode?

OmniServer supports either Software Licensing or Hardware Key Licensing. The following sections detail how to license your OmniServer depending on which licensing option applies for your project:

OmniServer Software Licensing

OmniServer Hardware Licensing

Converting an OmniServer Software License to Hardware

5.1 Software Licensing

When you purchase an OmniServer software license, you receive a Serial Number that contains a sequence of nine numbers. For example:

An OmniServer Professional License Number looks like the following:

11009999

An OmniServer Server Edition License Number looks like the following:

10909999

An OmniWedge License Number looks like the following:

11109999

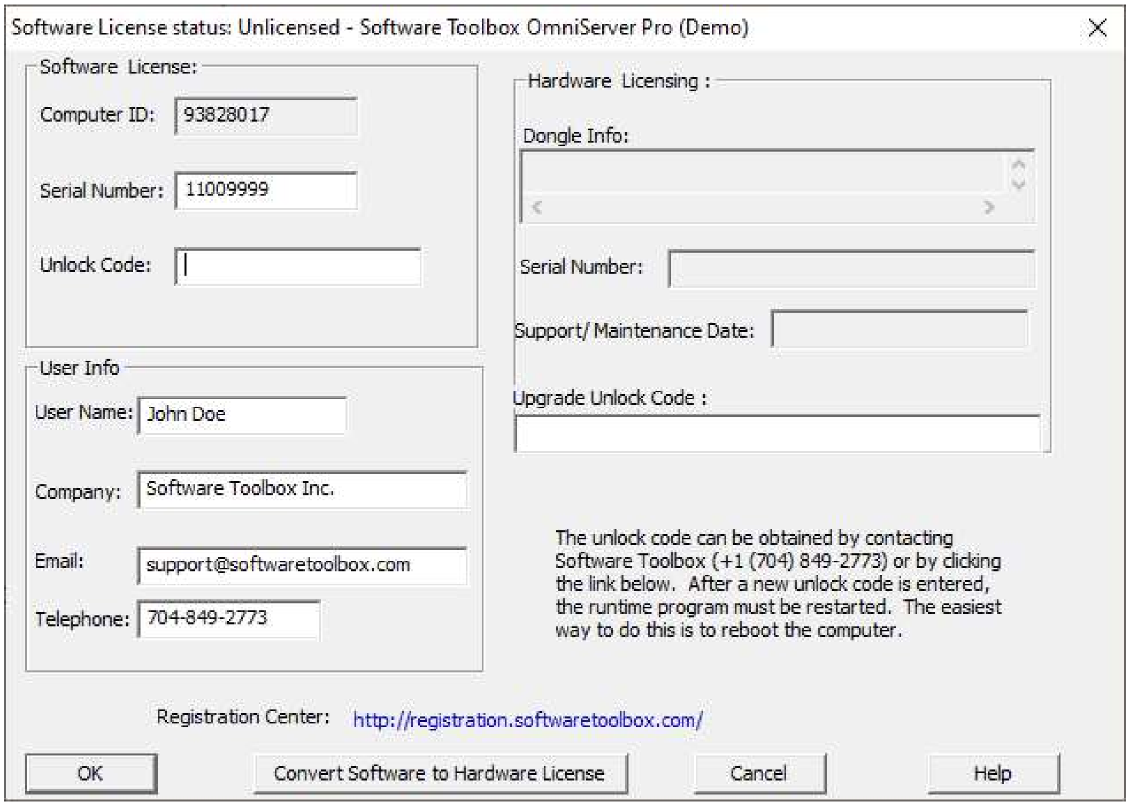
For new users or those who have purchased an upgrade, if you have received that registration code, you're all set. For existing users of an older version (2.10 or older) that are eligible for a free upgrade to the current version, please visit the Registration website to submit a re-registration request or contact us for a new Unlock Code.

For V3.0 or newer, OmniServer supports maintenance/support tracking built in to the licensing. This makes it possible to upgrade to new versions without requiring a re-registration request if your maintenance/support contract is current.

Performing the Registration

To register your server, here is what you need to do:

- Select **Help | Registration** from the server's menu.
- You will now get the following dialog box:


Software License status: Unlicensed - Software Toolbox OmniServer Pro (Demo)

Software License:
Computer ID: 93828017
Serial Number: 11009999
Unlock Code:

User Info
User Name: John Doe
Company: Software Toolbox Inc.
Email: support@softwaretoolbox.com
Telephone: 704-849-2773

Hardware Licensing :
Dongle Info:
Serial Number:
Support/ Maintenance Date:
Upgrade Unlock Code :

The unlock code can be obtained by contacting Software Toolbox (+1 (704) 849-2773) or by clicking the link below. After a new unlock code is entered, the runtime program must be restarted. The easiest way to do this is to reboot the computer.

Registration Center: http://registration.softwaretoolbox.com/

- Enter your contact information in the "User Info" section on the bottom left, and your serial number in the "Serial Number" field in the **Software License** section on the top left.
- Make note of the "Computer ID" above the serial number, then click on the Registration Center link, email support at support@softwaretoolbox.com, or call +1-704-849-2773 (normal business hours are 8am - 5pm Eastern, Mon-Fri).

- Either through email, your browser window, or by phone, you will receive an unlock code. Enter in that code in the "Unlock Code" field of the **Software License** section. (Format: xxxxxxxxx-xxxxxxxxxx) Make sure to enter both sections of the unlock code separated by a dash (-).
- Click OK.

Please note: You will have to restart the server runtime process/service in order to have the registration take effect. If you are running the server interactively, locate the server's icon in the icon tray, right-click on it, and select **Exit**. If you are running the server as a service, go to your **Services** administrative tool and restart the server from there.

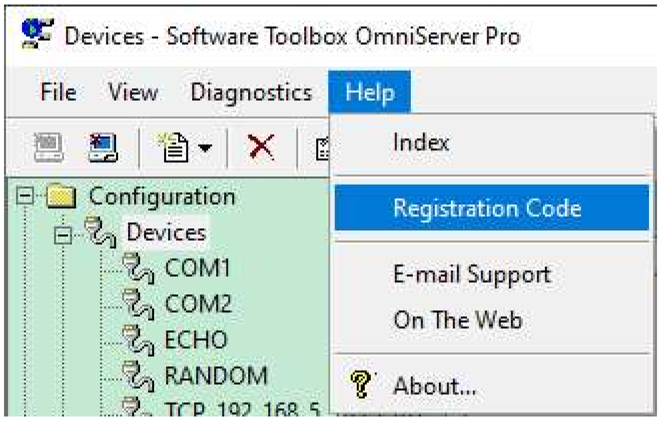
5.2 Hardware Licensing

If you purchased your new OmniServer license with a hardware key option, you will receive the hardware key dongle via shipping pre-loaded with the license. To license your OmniServer, simply insert the hardware key dongle into the desired available USB port on the OmniServer machine.

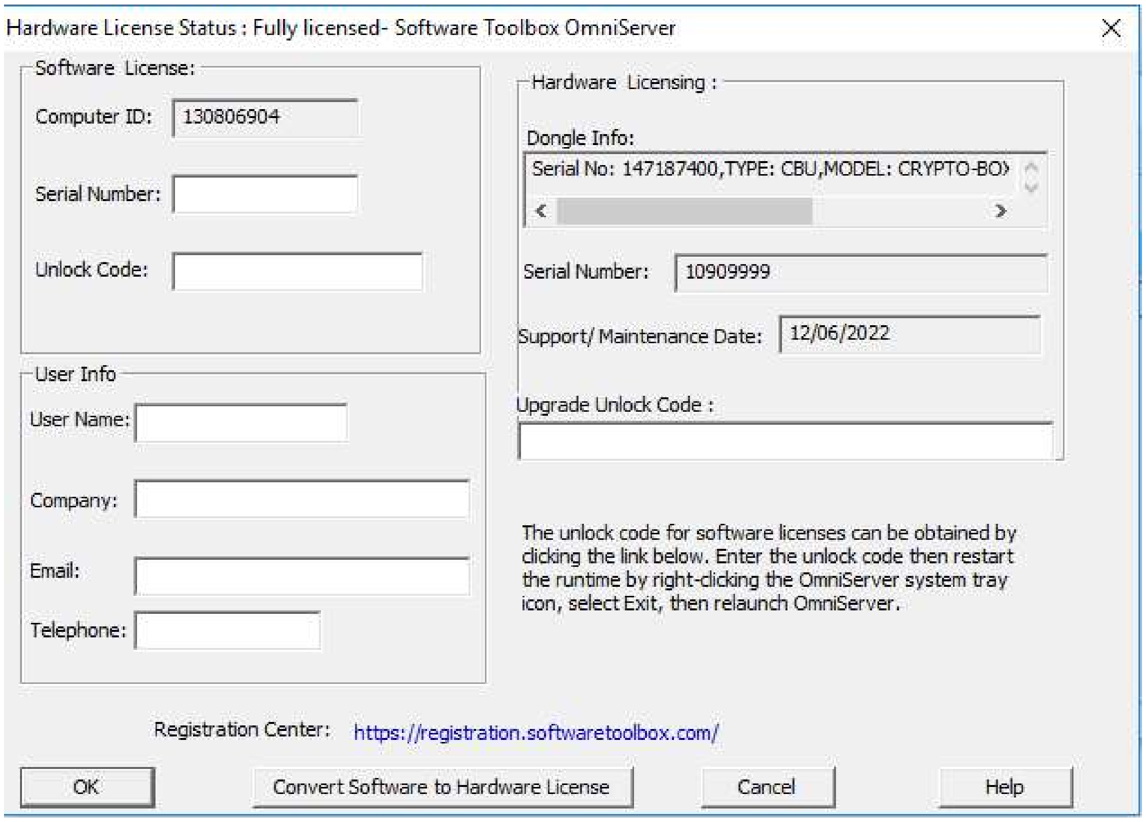
On inserting the hardware key dongle for the first time, it will take a few moments to install the required hardware level drivers allowing the Windows operating system to recognize the hardware key dongle. Once complete, you can verify that you are fully licensed as follows:

1. Open the OmniServer Configuration.

2. Go to Help -> Registration Code.



3. Under the Hardware Licensing section at the top right, you should see the Dongle Info associated with your hardware key and the associated Serial Number and Support/Maintenance Date of the license loaded on your hardware key. And the title bar should indicated a Hardware License Status of fully licensed.



Hardware License Status : Fully licensed- Software Toolbox OmniServer

Computer ID: 130806904
Serial Number:
Unlock Code:
User Name:
Company:
Email:
Telephone:

Dongle Info: Serial No: 147187400,TYPE: CBU,MODEL: CRYPTO-BOX
Serial Number: 10909999
Support/ Maintenance Date: 12/06/2022
Upgrade Unlock Code :

The unlock code for software licenses can be obtained by clicking the link below. Enter the unlock code then restart the runtime by right-clicking the OmniServer system tray icon, select Exit, then relaunch OmniServer.

Registration Center: https://registration.softwaretoolbox.com/

5.3 Converting Software to Hardware License

For OmniServer users who currently own an OmniServer software license (any edition) and prefer to switch to a hardware license option, OmniServer now supports the ability to transfer your existing registered OmniServer license to a Software Toolbox hardware key.

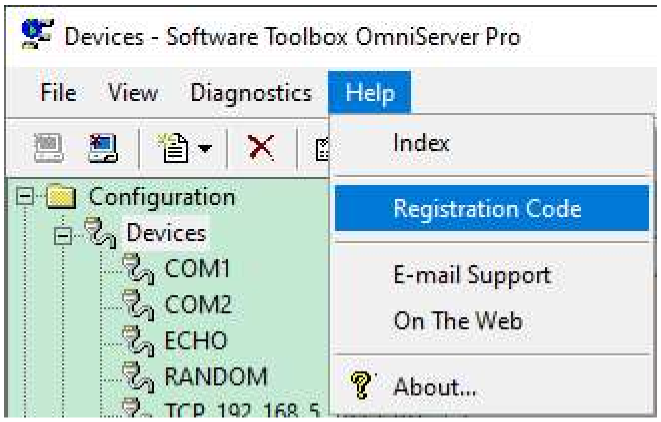
The first step is to contact Software Toolbox with your existing OmniServer software license information to purchase a Software Toolbox hardware key that is properly formatted to receive your OmniServer software license.

Once you receive your Software Toolbox hardware key, you just need to proceed with the following steps:

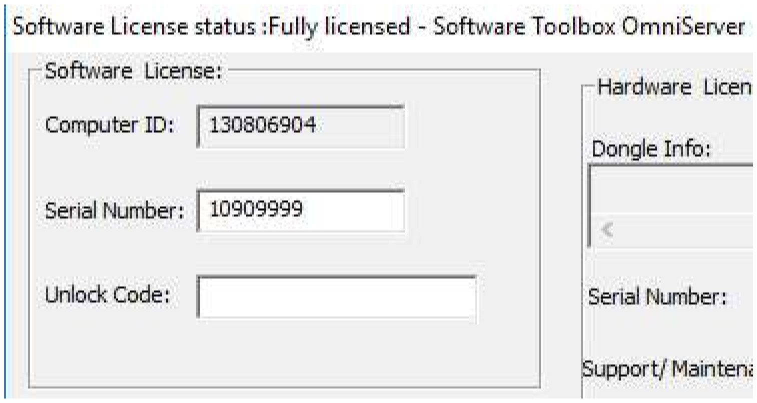
1. If you have not already done so, ensure that you have installed OmniServer V3.2-003 or newer (go to **Help -> About** in the

Omniserver Configuration to confirm). Current version of OmniServer is available here.

2. Insert the hardware key dongle (for the first time, it will take a few moments to install the required hardware level drivers allowing the Windows operating system to recognize the hardware key dongle.)
3. In the OmniServer Configuration, go to **Help -> Registration Code**.



4. Confirm that your OmniServer currently displays your software license information and the title bar indicates "Software License status: Fully licensed".


Software License status :Fully licensed - Software Toolbox OmniServer

Software License:

Computer ID: 130806904

Serial Number: 10909999

Unlock Code:

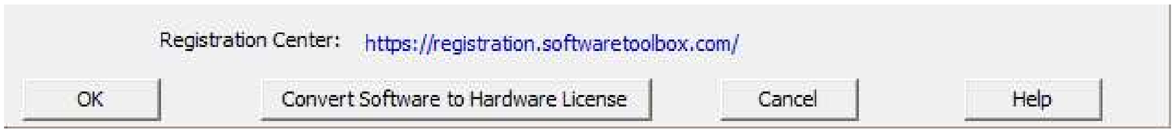
Hardware Licen

Dongle Info:

Serial Number:

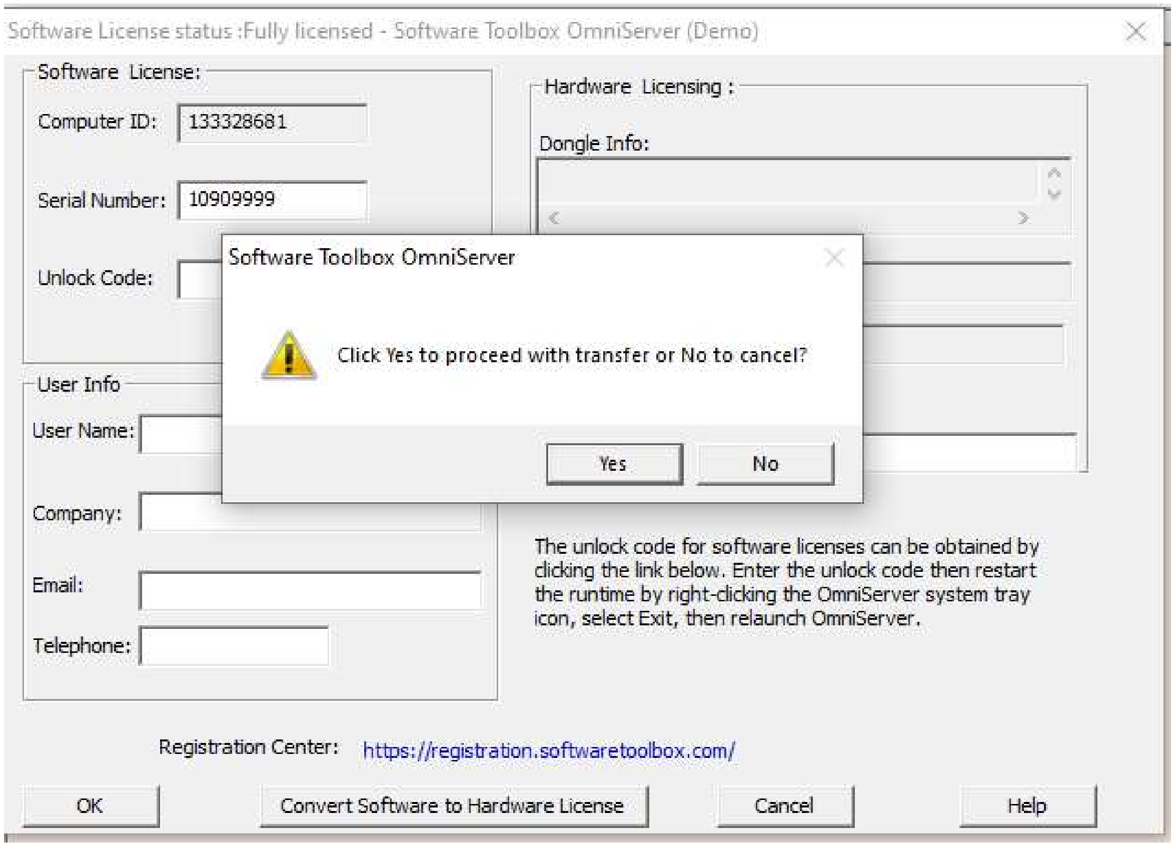
Support/ Maintena

5. Next, click the "Convert Software to Hardware License" button at the bottom of the window.


Registration Center: https://registration.softwaretoolbox.com/

6. You will be asked if you want to proceed with transferring your software license to the hardware key. Click "Yes" to proceed or "No" to stop the

transfer.


Software License status :Fully licensed - Software Toolbox OmniServer (Demo)

Software License:

Computer ID: 133328681

Serial Number: 10909999

Unlock Code:

User Info

User Name:

Company:

Email:

Telephone:

Hardware Licensing :

Dongle Info:

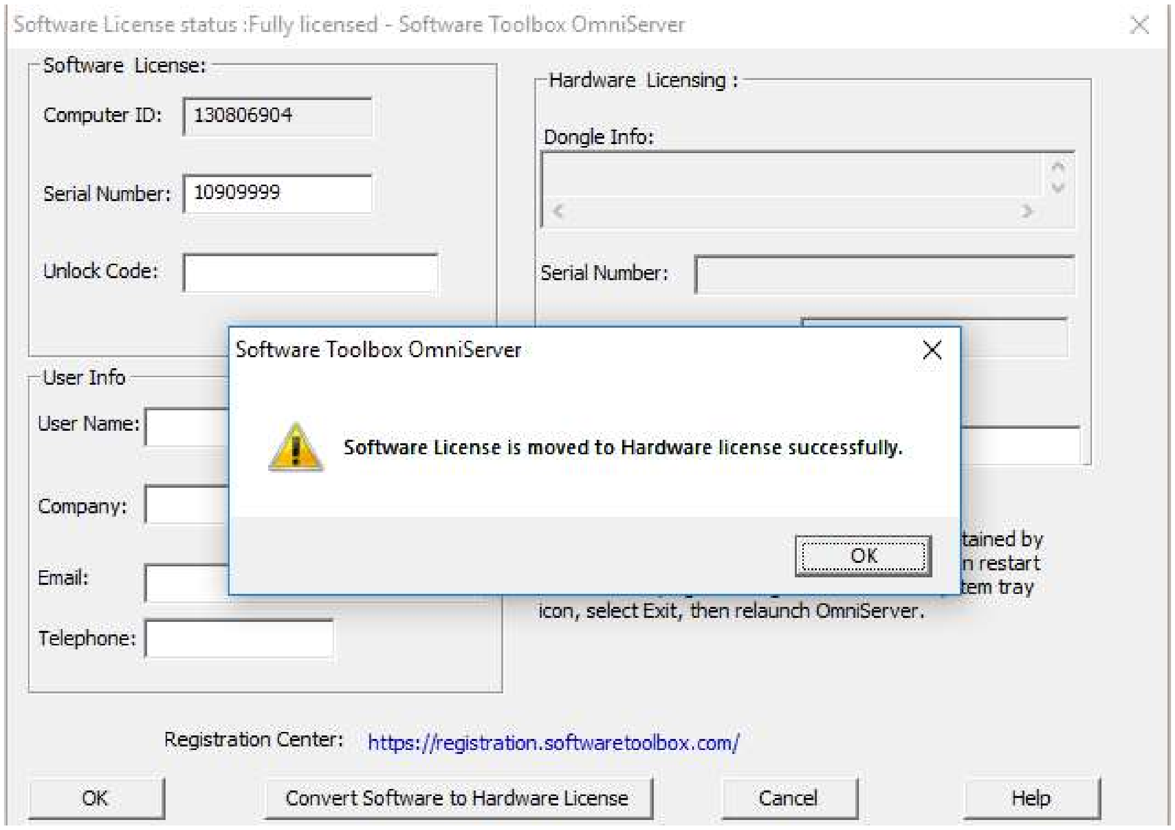
Software Toolbox OmniServer

Click Yes to proceed with transfer or No to cancel?

The unlock code for software licenses can be obtained by clicking the link below. Enter the unlock code then restart the runtime by right-clicking the OmniServer system tray icon, select Exit, then relaunch OmniServer.

Registration Center: https://registration.softwaretoolbox.com/

7. After clicking "Yes", the transfer will proceed and you'll get the following message after the transfer is successful:


Software License status :Fully licensed - Software Toolbox OmniServer

Software License:

Computer ID: 130806904

Serial Number: 10909999

Unlock Code:

User Info

User Name:

Company:

Email:

Telephone:

Hardware Licensing :

Dongle Info:

Serial Number:

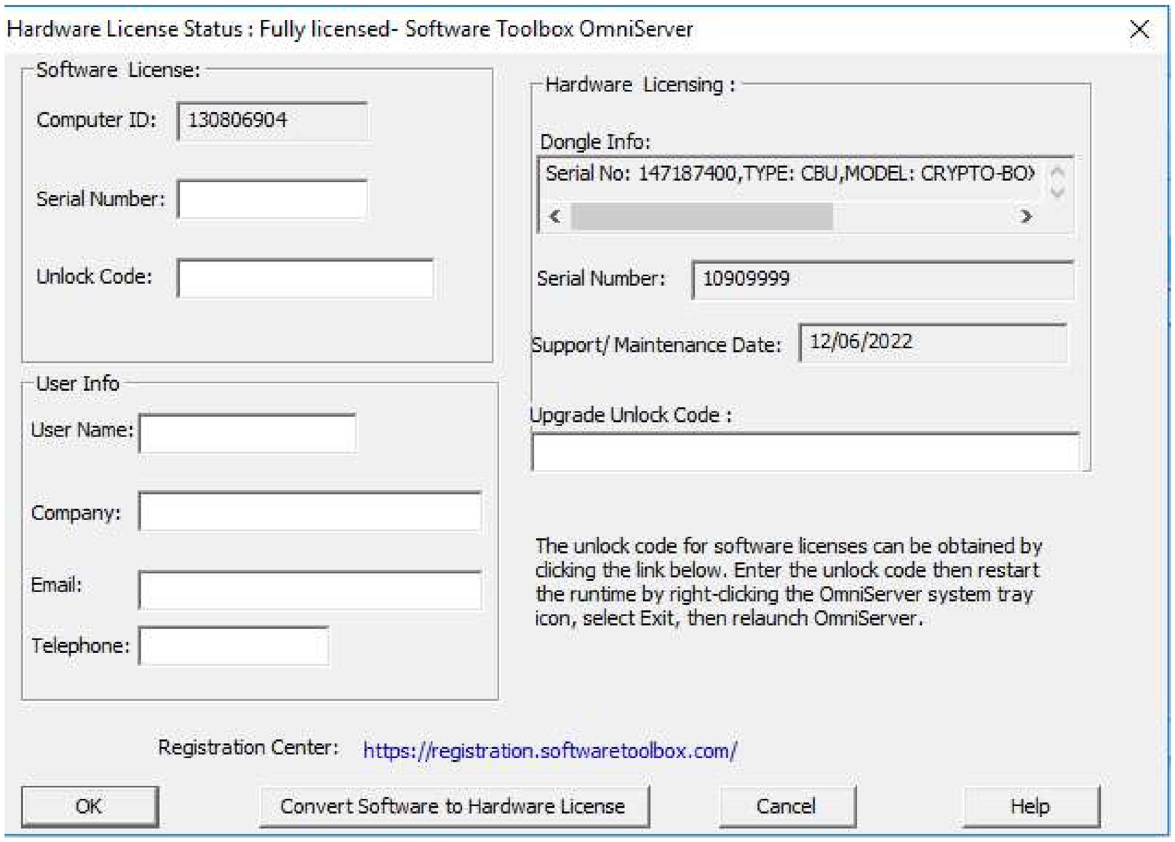
...tained by ...n restart ...tem tray icon, select Exit, then relaunch OmniServer.

Registration Center: https://registration.softwaretoolbox.com/

Software Toolbox OmniServer

Software License is moved to Hardware license successfully.

8. You can then confirm that the transfer was successful and that your hardware key is now working in the title bar, as in the below screenshot:



Hardware License Status : Fully licensed- Software Toolbox OmniServer

Software License:

Computer ID: 130806904

Serial Number:

Unlock Code:

User Info

User Name:

Company:

Email:

Telephone:

Hardware Licensing :

Dongle Info:

Serial No: 147187400,TYPE: CBU,MODEL: CRYPTO-BO›

Serial Number: 10909999

Support/ Maintenance Date: 12/06/2022

Upgrade Unlock Code :

The unlock code for software licenses can be obtained by clicking the link below. Enter the unlock code then restart the runtime by right-clicking the OmniServer system tray icon, select Exit, then relaunch OmniServer.

Registration Center: https://registration.softwaretoolbox.com/

Getting Up and Running

6 Getting Up and Running

The server has been designed to quickly send and receive data from any external device, and send that data back to any client program. However, there are still some basic steps you need to take to configure the server. This part will guide you along the steps required to get the server talking to your device an sending its data back to the client.

Online training resources

- Training Videos - over 20 videos covering basics to protocol configuration, client connectivity, device specific protocols, and more
- OmniServer Technical Blog - over 50 technical articles from simple to complex, along with user case studies
- Troubleshooting Guide (PDF)
- Exploring OmniServer Configuration
- OmniServer Diagnostic Tools Training

Step-by-Step Guide

If you would like to walk through the workings of the server on a step-by-step basis, here is the path you need to take:

- Working with the Server
- Working with the Client
- Bringing the Pieces Together

Quick Answers to Common Questions

If you'd like, you can review a series of commonly asked questions concerning the server. This will give you an idea of how it functions and how easy it is to configure:

⊟ What does the server do?

The server acts as a "bridge" between a client program and an device. It waits for a client program to request data, then determines the correct communications procedure with the device to obtain that data, and finally delivers the data to the client.

How can the server be used to communicate with my device?

The server uses standard Windows devices to communicate with attached devices. This includes Serial (COM) ports, Parallel (LPT) ports, and Ethernet (TCP, UDP, TELNET) ports. You can also use the Ethernet ports to access data on the Internet.

How do I get the data into other programs?

The server uses two main "transports" (in addition to a few variations) to send the data to another Windows program. These transports link the server with any client application than also supports these transports.

Currently, the server supports OPC-DA, OPC-UA, Wonderware Suitelink, and DDE, plus a few variations of DDE, such as FastDDE and PackedDDE, AdvancedDDE

How do I create a Protocol?

Included within the server is a **protocol editor**. You simply create a Protocol object, enter in the items you wish to transfer to the client and the messages you wish to send to the device, and the server takes care of all the polling for you.

For more information on creating a protocol, click here.

Where do I find Protocol information for my device?

What is generically known as a "Protocol Document" comes in a wide variety of formats and titles.

Basically, you are looking for any document that will have these critical pieces of information:

- Device Setup Information: For Serial ports, this will be the baud rate, parity, data bit, etc., information. For TCP/IP, this will be Address and Port number information.
- Communication Information: This section will describe how the device "talks" to an external computer, or host computer. For

example, barcode scanner documentation may say "The barcode outputs a nine digit number, followed by a Carriage Return". Or a pressure gauge may state "Once the device receives an STX character, it will respond with an STX, the data asked for, a checksum, and finally an ETX.

- Samples: Although some manuals do not include this information, the more samples showing how the device talks, the better chance there is to create a protocol.

Unfortunately, the names given to the manuals in which the information is written is as varied as the number of devices on the open market. However, the information you need can usually be found under one of these names:

- User's Guide (or Manual)
- Programmer's Manual
- Interface Manual
- Setup and Reference Manual

If you still cannot find the information, contact your device's manufacturer. They should be able to direct you to the correct source.

How do I interpret protocol information from my device?

The hardest part of the protocol design for the server may be in this step. Each device has it's own way of relaying protocol information to the user.

Basically, most documentation will boil down to one or two pages that might look like this:

Message from Host Computer

Description	Byte	Value
Start of Text	0	STX (0x02)
Module Address	1	Valid Module Address (0-6)

Point Address	2 - 3	Valid Point Address (0-500)
Command	4	See below for Commands
Data	5 - n	Device data (depends on your specific protocol)
End of Text	n + 1	ETX {0x03}
Checksum	n + 2	Sum of Bytes excl. STX

Response from the Device

Description	Byte	Value
Start of Text	0	STX (0x02)
Module Address	1	Valid Module Address (0-6)
Point Address	2 - 3	Valid Point Address (0-500)
Command	4	See below for Commands
Acknowledge	5	ACK {0x06}
Data	6 - n	Device data (depends on your specific protocol)
End of Text	n + 1	ETX {0x03}

Checksum	n + 2	Sum of Bytes excl. STX

Example: Read the point 400 in Module 3

From Host:
STX MAddr PAddr Cmd ETX CSUM
02 03 01 90 52 03 E6

Response from Device (Returns value of 257):
STX MAddr PAddr Cmd ACK Data ETX CSUM
02 03 01 90 52 06 01 01 03 F1

Given the example above, the important information you will need is the **Value** section and the **Byte** section. Any hard-coded bytes (like the STX above) can be entered into the server as-is. However, the data that could change - both the addresses and the data - will need special constructs called **data items**, which is the server's way of knowing to expect different values for each operation. The value can come from the client program (as in the addresses), or from the device (as in the data).

Please note: Without the information above telling you how the device "talks", the server cannot be "taught" how to talk to the device. Therefore, it is important that the above information is on-hand before you begin developing the server protocol.

How do I get the Server talking to my device?

The server "talks" to the device when these five conditions occur:

- A protocol has been designed (Click here for more information)
- A server device has been defined (Click here for more information)
- A server topic have been defined (Click here for more information)
- Client tags have been created (Click here for more information)

- The client requests the data (Click here for more information)

Once all of those have occurred, then the server will access the device, parse out the data, and return that data to the client.

⊟ How do I get the Client talking to the Server?

[This section applies only to the OmniServer Server Edition and the OmniServer Professional Edition.]

The server talks to the client through the use of client tags.

For more information on this, please click here.

⊟ Can I have the server talking to more than one device?

Yes. You are limited by the memory and processing capabilities of the computer, but few limitations have been coded into the server. Generally, you can only have a total of 2048 server objects, an object being a server topic, server device, a server client (for the Professional version), and server protocol. A combination of any of these that totals less than 2048 is permitted.

However, having the ability to create up to 2048 doesn't mean you should do that. The more objects you create, the more resources OmniServer requires from your computer. For example, in our testing, once we reached 1500 objects, OmniServer took 99% of the CPU and our polling rate began to diminish. So if you have 1000 devices, and each needs to be polled at, say 30 seconds, then you're probably OK. But if you need each of those machines polling at 100ms, that is likely not realistic given that typical machines where OmniServer is installed will have other applications competing for resources.

So what is a good "maximum" number of objects? It really depends upon the hardware specifications of your computer and the amount of polling OmniServer needs to each device. Experimentaion is your friend here. Here are some tips if you have multiple (100s) of devices:

- The more unsolicited messages and error messages you have in a protocol, the more processing time the server needs. Once you start going over 50 unsolicited messages in a single protocol, that will start affecting performance and eating CPU time. Try to consolidate unsolicited messages.

- While Command/Request (Host) Messages generally are not performance killers, each one does mean a trip to the device. Therefore, if you have hundreds of Command/Request (Host) Messages all wanting to be sent to the device at the same time, this will bottleneck your communications. Try to lessen the number of trips to the device by reading as many tags as possible from a single Command/Request (Host) Message.
- The slower your communications, the fewer devices the server can handle with efficiency. This is a hardware limitation. For example, 50 devices on ethernet had faster throughput that 50 devices at 9600 baud on a serial port - even if those serial ports are ethernet encapsulated. Obviously, this will limit the number of objects the server can handle effectively. Make sure your hardware infrastructure can handle the load.

6.1 Working with the Server

Once an device is connected to the computer, you need some way to get to its information. This is where a **server** comes in.

The primary function of a server (many times, it's only function) is to collect data from an device and "serve" it up to whatever client program is requesting the data. The Server takes away the responsibility of gathering the data from the client, which in turn gives the client more resources to process the data (the job for which it is best).

To get the best performance out of your server, it is necessary to go through a certain procedure. Skipping one of these steps will most likely degrade the server's performance, or more critically, causing it to deliver the wrong data.

Working with the Server: Procedures

- Gathering Device Information:
  Your first step in teaching the server to talk to the device, this

contains the information that helps the server to model the device's data steam. This is also called the device's protocol.

- Creating the Server Protocol
  The next step is to model the device's protocol in the server. That model is called the server's protocol.
- Creating the Server Device
  Next, the server needs to know which external ports are available for polling. These ports can be either Serial (COM - Virtual or Physical), Parallel (LPT), or Ethernet (TCP, UDP, Telnet, including serial encapsulation).
- Creating the Server Topic
  Finally, the Server Topic connects the server protocol with the server device. The topic is used by the server to tie in a protocol with one or more devices, insuring that the data is distributed correctly.

6.1.1 Gathering Device Information

The first step in getting the server to talk to the instrument is to find the instrument's protocol information. The instrument's protocol is normally found in either a user's guide or a programming manual that comes with the instrument. Each manufacturer places this information in different documents, so check with the manufacturer for additional information if you cannot find it.

The protocol information describes what sequence of characters needs to be sent to the instrument to perform certain tasks, such as returning the current temperature or changing the value of a set-point. It also tells the server what type of data to expect at random times (such as a barcode reader or weigh scale).

Please note: The server cannot "reverse engineer" a data stream to determine it's contents. If you do not have the documentation mentioned above, then the server will not help you at all. Also, if the documentation is proprietary in nature, it may be **illegal** to use the server to communicate to the instrument. Check with the manufacturer of your device for more information.

Locating and Interpreting Device Information

Unfortunately, there is no standard way of presenting this information, so you may have to do a little digging in your instrument's documentation to find it. However, there is some general help we can give you:

⊟ Where might I find my device's protocol documentation?

What is generically known as a "Protocol Document" comes in a wide variety of formats and titles.

Basically, you are looking for any document that will have these critical pieces of information:

- **Device Setup Information:** For Serial ports, this will be the baud rate, parity, data bit, etc., information. For TCP/IP, this will be Address and Port number information.
- **Communication Information:** This section will describe how the device "talks" to an external computer, or host computer. For example, barcode scanner documentation may say "The barcode outputs a nine digit number, followed by a Carriage Return". Or a pressure gauge may state "Once the device receives an STX character, it will respond with an STX, the data asked for, a checksum, and finally an ETX.
- **Samples:** Although some manuals do not include this information, the more samples showing how the device talks, the better chance there is to create a protocol.

Unfortunately, the names given to the manuals in which the information is written is as varied as the number of devices on the open market. However, the information you need can usually be found under one of these names:

- User's Guide (or Manual)
- Programmer's Manual
- Interface Manual
- Setup and Reference Manual

If you still cannot find the information, contact your device's manufacturer. They should be able to direct you to the correct source.

How do I interpret this information?

The hardest part of the protocol design for the server may be in this step. Each device has it's own way of relaying protocol information to the user.

Basically, most documentation will boil down to one or two pages that might look like this:

Message from Host Computer

Description	Byte	Value
Start of Text	0	STX (0x02)
Module Address	1	Valid Module Address (0-6)
Point Address	2-3	Valid Point Address (0-500)
Command	4	See below for Commands
{...data...}	5-n	Optional Data
End of Text	n+1	ETX {0x03}
Checksum	n+2	Sum of Bytes excl. STX

Response from the Device

Description	Byte	Value
Start of Text	0	STX (0x02)
Module Address	1	Valid Module Address (0-6)
Point Address	2-3	Valid Point Address (0-500)
Command	4	See below for Commands
Acknowledge	5	ACK {0x06}
{...data...}	6-n	Optional Data
End of Text	n+1	ETX {0x03}
Checksum	n+2	Sum of Bytes excl. STX

Example: Read the point 400 in Module 3

From Host:
STX MAddr PAddr Cmd ETX CSUM
02 03 01 90 52 03 E6

Response from Device (Returns value of 257):
STX MAddr PAddr Cmd ACK Data ETX CSUM
02 03 01 90 52 06 01 01 03 F1

Given the example above, the important information you will need is the **Value** section and the **Byte** section. Any hard-coded bytes (like the STX above) can be entered into the server as-is. However, the data that could change - both the addresses and the data - will need special constructs called **data items**, which is the server's way of knowing to expect different values for each operation. The value can come from the client program (as in the addresses), or from the device (as in the data).

Please note: Without the information above telling you how the device "talks", the server cannot be "taught" how to talk to the device. Therefore, it is important that the above information is on-hand before you begin developing the server protocol.

⊟ Where can I go if I need help?

Locating your device's protocol documents

Unfortunately, this is something that is outside of the scope of Software Toolbox. You will need to contact the manufacturer of your device to get the protocol documents.

Also, please note that the protocol might be proprietary, that is, it's not available to use by this server. If this is the case, it may be impossible (and illegal) to use this server to talk to the device.

Interpreting your device's protocol document.

If you are having extreme difficulty understanding the protocol document, we may be able to help. Click here to contact us and determine what support options are available.

6.1.2 Creating the Server Protocol

The **server protocol** is the method by which the server "knows" how to communicate with the device. Within each protocol is a series of instructions that tells the server exactly how to get data from the client, the exact format of the data to send to the device, how to interpret the data coming back from the device, and finally what data is to be sent back to the client.

⊟ How do I get started?

You will need to create a new protocol (or edit an old one). To do that, click here.

See also OmniServer Online Training Videos

What are the different parts of a protocol?

The are eight parts to a protocol. You may (or may not) use them all in your protocol, but it's always helpful to know that they do exist.

- Settings: The overall properties for the protocol. Translations, Message Length, and Binary Formats can be set here.
- Items: The data fields used to transfer information from the client to the device and back again. You can have four types: Integer; Real; String; and Discrete.
- Registers: Special data fields that are used to index Items. Primarily used to convey address or register information.
- Topic Variables: Special data fields that are used in association with Topics. Used almost exclusively to tie in a specific address with a topic.
- Error Detection Codes: User-defined calculations that produce a code that checks the validity of the data stream (also called **Frame Checks**, **CRCs**, and **Checksums**.
- Command/Request (Host) Messages: Used to poll an device for the value of data items, or to initialize an device. A Command/Request (Host) Message contains the actual description of the protocol for the device.
- Unsolicited Messages: These messages inform the server how to interpret messages that come in "unannounced" or "unexpectedly" from the device. Their primary use is to parse out message from devices like barcode scanners or weigh scales.
- Error Messages: Messages that are, in essence, alternate responses to Command/Request (Host) Messages. They operate much like Unsolicited Messages, except they can control the sequence of a Command/Request (Host) Message chain.

For detailed information on developing a protocol, click here.

How does the server use the protocol to "poll" for information?

The heart of the server's engine is located in the **Command/Request (Host) Message**. This protocol object describes the format of the data stream that is to be sent out to the device, in addition to the format of the data stream that comes back from the device, when applicable.

The server will only send out a message if a certain item is being asked for by a client (unless a Trigger has been defined). This is how it works:

- A Client program will request an item.
- The server will search through it's list of Hose Messages to find out if that item appears in any Command/Request (Host) Message's **Response** message.
- If the server finds a message, it will check to see if that message has already been queued to be sent to the device. If it has not, then the server queues up the message.
- Eventually, the message is sent to the device, and the expected response comes back. The server will then parse out the response message, extract the data, and send it back to the client.

How does the server use the protocol to gather "unsolicited" messages (such as data from barcode readers and scanners?

Since it is a common practice to use the server to gather information from devices like barcodes and weigh scales, the capability was built in to recognize data streams that come in "unsolicited", or "unannounced". The server handles these messages via the **Unsolicited Message**.

Basically, when a message comes in that does not match a response of a Command/Request (Host) Message, or is unexpected by the server, the message will be parsed according to any Unsolicited Messages that have been defined in the protocol. Any data item that is currently being requested by the client program which appears in the unsolicited message is then updated and sent back to the client program.

An important note: The server reports **changes** in data, rather than the receipt of data. Therefore, it is entirely possible that the server will not report the receipt of a data stream if the data contained in that stream is the same as the last data (like scanning the same barcode twice). Therefore, it is important to design your client program to where this can be taken into account. This can be done either by using other data in the stream as signals for new data (like a time stamp), or by changing the value of the data item after it is processed via writing back to the item from the client (thereby forcing the server to report the next data value no matter what the value might be).

6.1.3 Creating the Server Device

The **Server Device** is a "virtual" device. That is, it does not physically create devices or ports for you, but will tie itself into a device that already exists on the computer.

A **device** is defined as any port that connects the computer to the "outside world". In the server, you can define Serial Ports (commonly known as COM ports, which also includes USB device mapped to Virtual COM ports), LPT Ports (also known as Printer ports), or Ethernet Ports (commonly known as a TCP, UDP or Telnet ports and also includes serial encapsulation devices).

It is important to note that if the device does not exist on the computer, the Server will not communicate. For example, if you have COM1 and COM2 on your computer, you can create a COM3 in the server, but since COM3 does not exist, the server will fail when it attempts to open up the port.

Creating a server device is a relatively painless matter. For instructions on how to create a server device, click on the link below:

⊟ What is the purpose of the Server Device?

The server does not automatically know that devices (ports) are connected to your computer. Therefore, it is important that you define for the server to which devices it is allowed to communicate.

Although Serial (COM) and Parallel (LPT) devices are relatively easy, defining Ethernet (TCP, UDP, Telnet) devices takes some knowledge of the addressing and transport scheme for the device. You will need to consult your manufacturer for correct Ethernet information.

⊟ How do I create a Server Device?

Creating a server device is a very simple process. You will need to know this information before you create the device:

- Serial Ports (COM): Baud rate; Parity; Data Bits; Stop Bits; Flow Control; COM Port number.
- Parallel Ports (LPT): LPT Number.
- Ethernet: Address (either "xxx.xxx.xxx.xxx" or a friendly DNS name, if available, such as www.yourdevice.com; Port Number, Transport Type (TCP, UDP, or TELNET).

For a more detailed explanation in creating a Server Device, please click here.

See also OmniServer Online Training Videos

⊟ How many devices can I have?

As with many servers, there is a limit to the number of Server Devices you can have. With this release, here are your limits:

- Serial (COM): 256 (COM1 - COM256)
- Parallel (LPT): 9 (LPT1 - LPT9)
- Ethernet (Winsock): Generally unlimited. This is controlled more by the operating system than the server. Also be aware that most Microsoft non-server operating systems will only allow 10 open "requests" at any one time. This means that although you can have, for example, 995 open Ethernet connections, the server is only allowed to request 10 connections to be created per second - any additional attempts at creating a connection will fail and you will receive errors. The good news is that once those 10 connections have been created, the server is allowed to create 10 more, so having a total of 995 connections is possible, it will just take a

long time. This limitation is not seen on Server edition operating systems.

Now, there are some factors that can affect these numbers. The server is designed to where there is a maximum number of "objects" within the server, with each Server Device, Server Topic, Server Protocol, and Server Client representing an "object". Adding these objects together will give you the total number of objects created, and this number cannot be above 2048.

6.1.4 Creating the Server Topic

Before the server can talk to an device, it needs to know which protocol is associated with which server device. This is done through creating a **Server Topic**.

⊟ What is the function of a Topic?

The purpose of the Topic is to tie in a server device (or groups of server devices) to a particular protocol. Later, when you create the tags and/or points within your Client application, you will see again how the topic comes into play.

⊟ How do I create a Topic?

When you create a topic, you will need the following information on hand:

- Protocol: The name of the protocol you are going to use with this topic.
- Server Device: The name of the server device to use with this topic. You can use more than one device with a topic.
- Topic Variables: You create Topic Variables within the protocol, and set their values here. Topic variables can also be required by the currently set device.

For detailed information on creating a topic, please click here. You can also create a topic using the Topic Wizard.

See also OmniServer Online Training Videos

⊟ Is there a limit to the number of Topics?

The only limit to the number of topics you create is the limit to the total number of "objects" you can create within the server. A server object is defined as either a topic, a server device, a protocol, or a server client. The total number of these objects cannot surpass, currently, 2048.

6.2 Working with the Client

[This section applies only to the OmniServer Server Edition and the OmniServer Professional Edition.]

Without a client, the server will not operate. Clients provide the "front-end" interface to the data being delivered by the server. Clients are an integral part of any data acquisition project.

See also OmniServer Online Training Videos

Why you need a client

This is very important: **The Server cannot communicate with your device until a client program communicates with the server and tells it what data is required.**

The server acts when the client provides instructions. When there is no client, there is nothing to provide the server with instructions. Without the client asking for information the server does not interact with the device. Work must be done on the client side to

How the Client Accesses the Server

The server communicates with the client primarily through two types of transports: OPC and DDE.

Clients the Server Supports:
OPC-DA, OPC-UA, DDE, Wonderware SuiteLink
Packed DDE, Advanced DDE, Fast DDE

Note for Intellution WDMACS Users: Older versions of this server used a special FIX DMACS driver. This driver is no longer supported. Please use the Intellution OPC Client instead

The client initiates the communications (the server never does) by using one of the supported transports. The server then acknowledges the request, and will pass data back to the client each time the data requested by the client changes.

The number of clients that can be used with the server is large and diverse, but can basically be broken down into a few generic categories. Therefore, the purpose of this section is to explain the uses of client software and how to get them to talk to the server.

- Creating Client Tags/Points

For information specifically regarding the OPC UA Server interface, please see the following sections:

- What is OPC UA?

- OPC UA Configuration

6.2.1 Creating Client Tags/Points

[This section applies only to the OmniServer Server Edition and the OmniServer Professional Edition.]

Once the server protocol, device, and topic has been completed, it's time to setup the client program to talk to the server.

Important Note: The server will not deliver data nor talk to the device unless a client program requests the data. You must complete this section or the server will not communicate with the device.

This section will spell out how to access the server using some of the more popular types of clients. But before we go any further, one caveat: The examples below are presented based upon our experience with the products. It is not meant as a replacement to the support you get from each of the respective companies. In fact, if you do have some problems, you will save time (and frustration) by contacting them first and not the makers of the server.

So how do you set up your client tags and/or points? Here are some examples:

⊟ Generic OPC DA

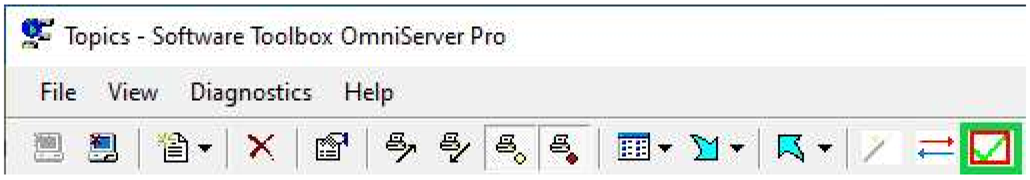
Note: use your topic name in place of **topicname** and your item name in place of **itemname**

Server Name: **SWToolbox.OmniServer**
Item Address: **topicname.itemname**

Example: To connect to Item "CODE" in topic "BARCODE":

Server: **SWToolbox.OmniServer**
Address: **BARCODE.CODE**

Launching the **Software Toolbox OPC Test Client** that installs with OmniServer from the toolbar will, by default, automatically connect to OmniServer and subscribe to all of the protocol items for your defined topics. (If you prefer to launch a blank configuration, you can disable automatically adding tags under View -> General Options.)



⊟ Generic DDE

Note: use your topic name in place of **topicname** and your item name in place of **itemname**

Application Name: **OSRVDDE**
Topic Name: **topicname**
Item Name: **itemname**

Example: To connect to Item "CODE" in topic "BARCODE":

Application: **OSRVDDE**
Topic: **BARCODE**
Item: **CODE**

⊟ GE Proficy (Formerly Intellution FIX) OPC

Please check the documentation for GE Proficy's OPC interface for instructions on how to connect to an OPC Server.

GE Fanuc CIMPlicity

Please check the documentation for GE Fanuc CIMPlicity for instructions on how to connect to a DDE Server.

AVEVA (Formerly Wonderware) InTouch

Please check the documentation for Wonderware Intouch for instructions on how to connect to a DDE, OPC, or Suitelink Server.

You can also view our video tutorials under Wonderware Client Connectivity on the OmniServer Video Resources page.

Rockwell Software RSView

Please check the documentation for Rockwell Software RSView for instructions on how to connect to a DDE Server.

Microsoft VBA

Note: use your topic name in place of **topicname** and your item name in place of **itemname**

Application Name: **OSRVDDE**
Topic Name: **topicname**
Item Name: **itemname**

VBA is a coding convention used in the Microsoft Office family of products (along with some programming languages). Although this example is in no way a substitute for an actual manual, it will give you an example on how to implement a connection using VBA. Should you need additional help, contact your client program's support. The support staff of this server will not be able to help you.

Example: To connect to Item "CODE" in topic "BARCODE":

```
Dim lChannel as Long
Dim vData as Variant
```

```
...
... some statements
...
lChannel = Application.DDEInitiate("OSRVDDE","BARCODE")
...
... some other statements
...
vData = Application.DDERequest(lChannel,"CODE")
...
... some more statements
...
Application.DDETerminate lChannel
```

Generic OPC UA

Note: Once an OPC UA connection to OmniServer is made in your OPC UA Client, all items available for all topics in OmniServer will be browseable and selectable.

OmniServer OPC UA Endpoint URL Format (Default):
opc.tcp://[your_machine_name]:27730

You will need to confirm that the following connection parameters in your OPC UA Client match those used by the desired OmniServer OPC UA Endpoint in the OPC UA Configuration.

- Endpoint URL and TCP Port #
- OPC UA Certificates exchanged between OmniServer and OPC UA Client (manually via export/import or you can trust rejected certificates in OmniServer)
- Endpoint Security Profile
- Authentication Enabled or Disabled (If Enabled, User/Password Information must match)

For testing OPC UA connectivity to OmniServer, we recommend Unified Automation's OPC UA Expert Test Client, available for download on the Unified Automation website.

6.2.2 What is OPC UA?

The OPC Unified Architecture, also known as OPC-UA, is the latest open-standard architecture developed by the OPC Foundation to improve and expand interoperability standards in the Industrial Automation Industry.

⊟ Why do we need a new architecture to begin with?

OPC-UA was the result of several advancements and changes in the way data was commonly being accessed and exchanged. Some changes that lead to the need for a new architecture include:

- Microsoft's COM and DCOM (the basis for previous standards) were deprecated and are now considered legacy technologies.
- Web services gained importance in data exchange between machines and for communications to factory floor devices.
- Earlier specifications were decoupled and did not integrate well, e.g. items in a Data Access server could not communicate directly with items in an Alarms and Events server.

⊟ What are the benefits of the new architecture?

- OPC-UA provides a way to connect clients and servers in a secure manner, without relying on Microsoft DCOM. This is a big advantage because it means that you are no longer saddled with the headaches associated with having to configure DCOM. This is because DCOM plays no role in data transport.
- OPC-UA can also allow users to make secure connections through firewalls and over VPN connections.
- OPC UA also expands the ability to provide factory floor information to other business systems, as a result of the object-oriented model described above.

For detailed information on configuring OPC UA settings in OmniServer, please visit the OPC UA Configuration section of the Help File.

6.3 Bringing it All Together

Now that the Server has the correct protocol, device, and topic assignments, and the Client has the correct tags or points configured to talk to the server, your final step is to tie the client and server together and retrieve the data from the instrument.

Remember that no data can be retrieved from the device without the client communicating with the server. Therefore, it is important to complete this step in the process.

- Getting the Client talking to the Server
- Troubleshooting

6.3.1 Getting the Client Talking to the Server

[This section applies only to the OmniServer Server Edition and the OmniServer Professional Edition.]

Once you save your protocol, the server is ready to begin polling. However, additional steps may need to be taken on the client side in order to get communications working. Most clients will automatically re-connect with the server. If that is not the case, then try one of the steps below.

Please Note: These instructions are not meant as a replacement for the actual product manual. In other words, they may not work. Please refer to your client's documentation for official procedures for making your selected type of connection (OPC DA, OPC UA, DDE or Suitelink).

Click on the list below to see what steps are required:

⊟ AVEVA (formerly Wonderware) InTouch

Once you go into **View** mode, communications should start automatically. If not, check to make sure the tag names are defined correctly. To get InTouch to re-connect with the server, select **Special | Reinitialize I/O** from the menu. Or, exit **View** mode, then re-enter View mode.

⊟ GE Proficy (formerly Intellution) OPC

Go into **Run** mode to begin communications. If communications does not happen, then check the definition of the tags. To get the Intellution OPC to re-connect with the server, drop out of **Run mode**, the re-enter Run mode.

⊟ GE Fanuc CIMPlicity

Go into **Run** mode to begin communications. If communications does not happen, then check the definition of the tags. To get the Intellution OPC to re-connect with the server, drop out of **Run mode**, the re-enter Run mode.

⊟ Rockwell Software RSView

Go into **Run** mode to begin communications. If communications does not happen, then check the definition of the tags. To get the Intellution OPC to re-connect with the server, drop out of **Run mode**, the re-enter Run mode.

⊟ Generic OPC UA Information

Please refer to the OPC UA Configuration section of this help file for specific information on configuration required for making an OPC UA connection.

If you still cannot get your client and the server to communicate, then click on the following link for more trouble-shooting information.

Troubleshooting

6.3.2 Troubleshooting Resources

We have a variety of resources to help you in solving problems with your OmniServer operations

1. Troubleshooting flowchart/wizard
2. Troubleshooting Guide (PDF)
3. OmniServer Videos
4. OmniServer Technical Blog
5. General Ethernet Troubleshooting Tips & Tools
6. OmniServer Knowledgebase
7. Open a Support Ticket

This page is intentionally left blank.
Remove this text from the manual
template if you want it completely blank.

Server Details

7 Server Details

This section of the server's Help file contains detailed information on the internal workings and operation of the server's Configuration and Runtime programs. It is divided into three sections:

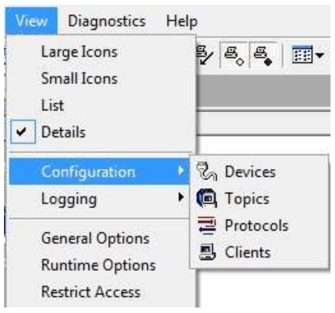
- The User Interface: This section contains detailed information on the operation of the server's GUI (Graphical User Interface) including the Wizard configurator.
- The Configuration Windows: This section explains each of the server's Configuration Windows (Device, Topic, Protocol, and Clients) in more detail.
- The Diagnostic Windows: This section contains an explanation of each of the four server diagnostic windows (I/O Monitor, Poll Statistics, Local Logger, and Item Values), how to use them, and the value of each one to the trouble shooting of protocol information.

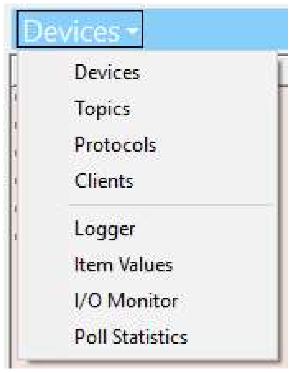
See also OmniServer Online Training Videos

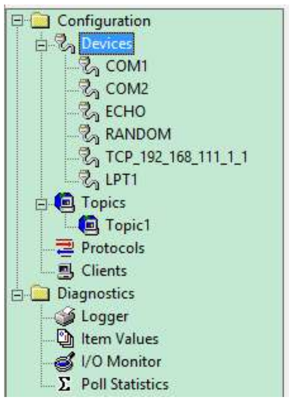
7.1 Configuration Windows

There are four server configuration windows that you need to successfully connect your device to your client using the server. They can be accessed in three different ways:

Server Menu	**View Window Selector**	**Configuration Tree View**







The four different configuration windows are:

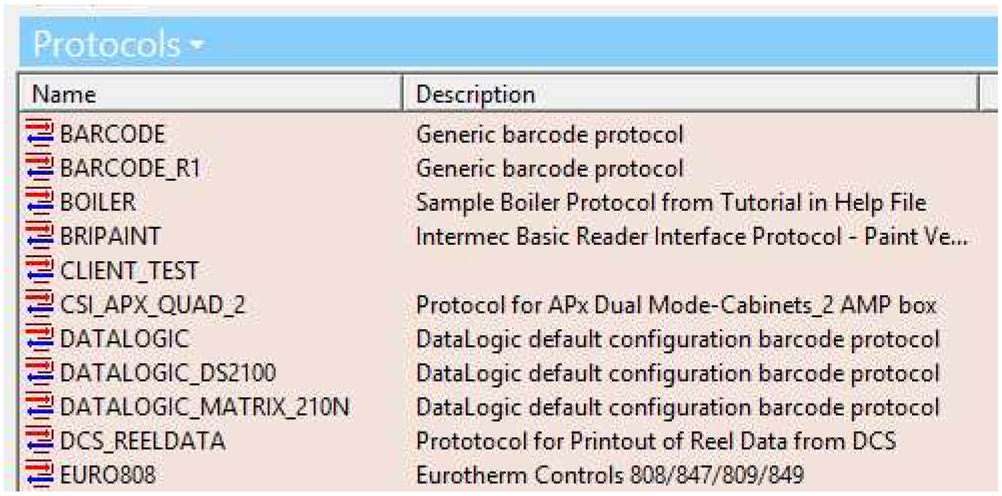
- Devices: Defines the server devices, which tells the server what physical I/O devices are available for communications (such as serial ports, Ethernet ports, etc.).
- Topics: Defines the server topics, which links a server device with a server protocol.
- Protocol: The heart of the server, which tells the server how to communicate with an device.
- Clients: Gives information on the supported client interfaces, specifically the transports used by the server to talk to a client.

7.1.1 Protocols

The heart of the server's operation is the **Protocol**. The protocol is responsible for:

- **For data coming from the client**: Formatting the data into data streams recognizable by the device.

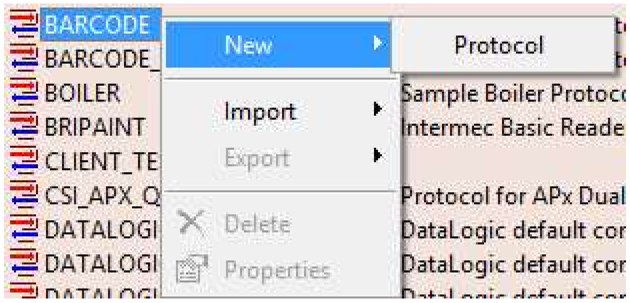
- **For data coming from the device**: Interpreting the data and sending it back to the client.



Protocols

Name	Description
BARCODE	Generic barcode protocol
BARCODE_R1	Generic barcode protocol
BOILER	Sample Boiler Protocol from Tutorial in Help File
BRIPAINT	Intermec Basic Reader Interface Protocol - Paint Ve...
CLIENT_TEST	
CSI_APX_QUAD_2	Protocol for APx Dual Mode-Cabinets_2 AMP box
DATALOGIC	DataLogic default configuration barcode protocol
DATALOGIC_DS2100	DataLogic default configuration barcode protocol
DATALOGIC_MATRIX_210N	DataLogic default configuration barcode protocol
DCS_REELDATA	Protototcol for Printout of Reel Data from DCS
EURO808	Eurotherm Controls 808/847/809/849

Main Protocol Screen

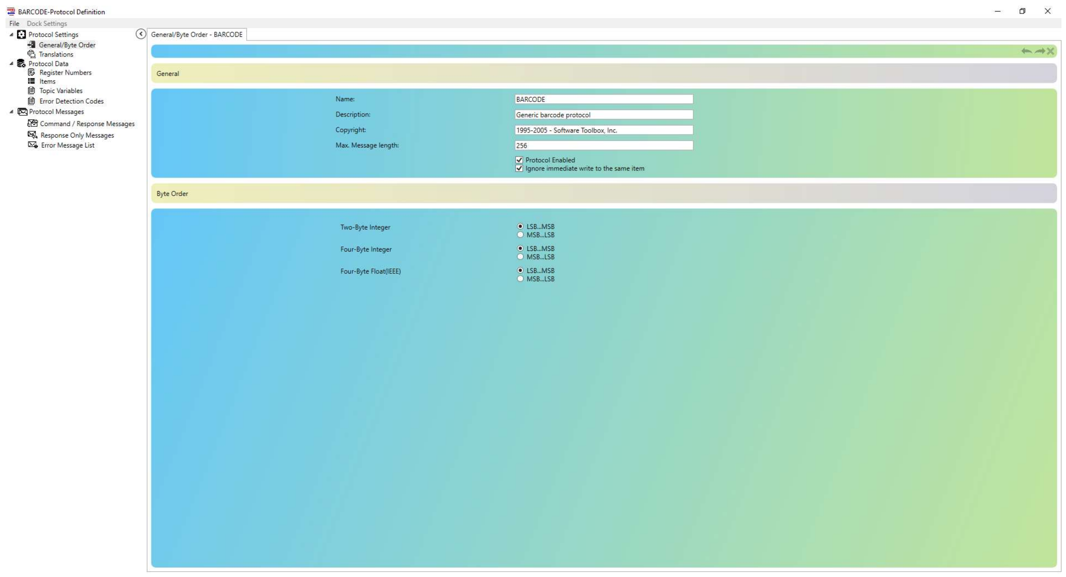


Main Protocol Screen - Popup Menu

Basically, the protocol's main job is to format and interpret the data moving between the client and the device. However, the protocol must first be taught how to do this, and this help documentation along with our online how-to videos can show you how.

You now have two options for creating new and editing existing protocols:

1. **Visual Protocol Editor** - An enhanced editor with greater ease-of-use features such as drag-and-drop support, tabular item, register and topic variable management, modular message configuration, configurable favorite sequences and more. *Provides increased productivity over the Legacy Protocol editor by as much as 35%.*



General

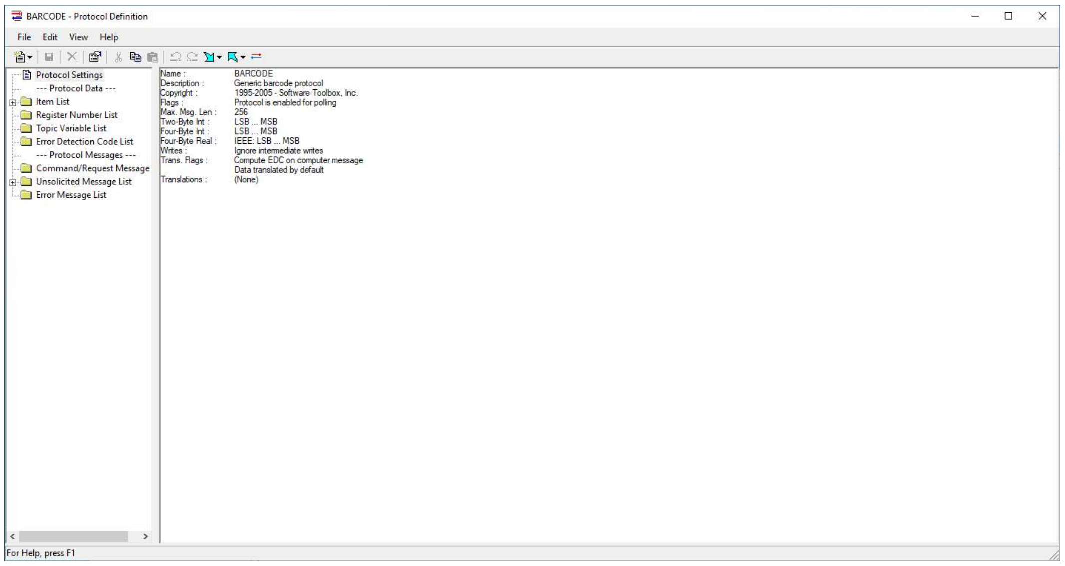
Name:	BARCODE
Description:	Generic barcode protocol
Copyright:	1995-2005 - Software Toolbox, Inc.
Max. Message length:	256

✓ Protocol Enabled
✓ Ignore immediate write to the same item

Byte Order

Two-Byte Integer	[✓ LSB...MSB] MSB...LSB
Four-Byte Integer	[✓ LSB...MSB] MSB...LSB
Four-Byte Float(IEEE)	[✓ LSB...MSB] MSB...LSB

2. **Legacy Protocol Editor** - The original editor for creating and editing protocols. For those who have used Omniserver in the past, this editor is what you are probably familiar with.

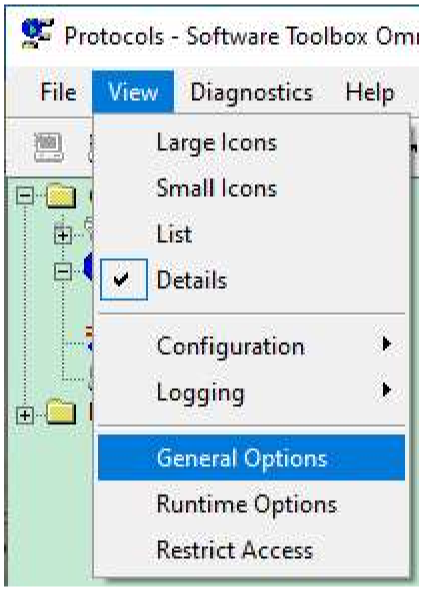


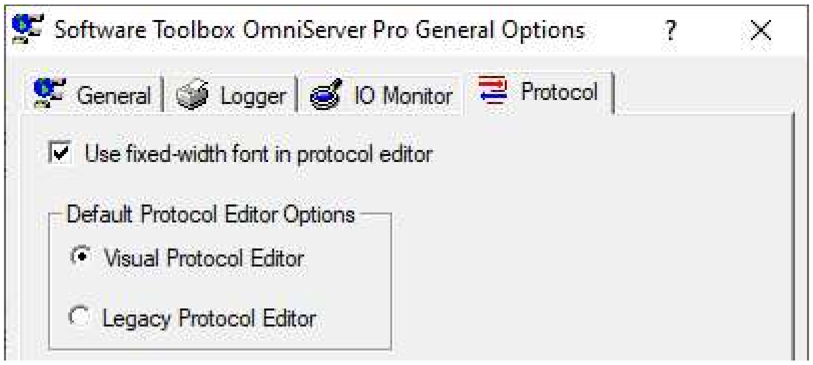
```
Name :              BARCODE
Description :       Generic barcode protocol
Copyright :         1995-2005 - Software Toolbox, Inc.
Flags :             Protocol is enabled for polling
Max. Msg. Len :     256
Two-Byte Int :      LSB ... MSB
Four-Byte Int :     LSB ... MSB
Four-Byte Real :    IEEE: LSB ... MSB
Writes :            Ignore intermediate writes
Trans. Flags :      Compute EDC on computer message
                    Data translated by default
Translations :      (None)
```

By default, the Visual Protocol Editor will be used to create new protocols or when you open an existing protocol. You should find the Visual Protocol Editor easier to use than the Legacy Protocol Editor.

However, for the best user experience, we have made it possible to change the server default protocol editor. You can control this in the following ways:

1. **From the OmniServer Configuration View Menu** - In the main OmniServer Configuration, go to the View Menu and select General Options then navigate to the Protocol section where you can toggle the default protocol editor.




Software Toolbox OmniServer Pro General Options

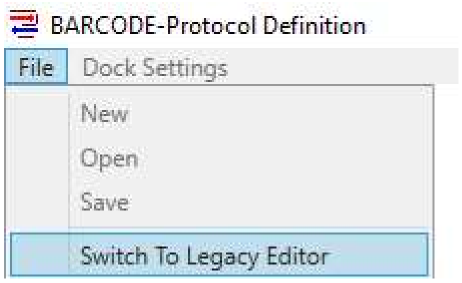
[✓] Use fixed-width font in protocol editor

Default Protocol Editor Options

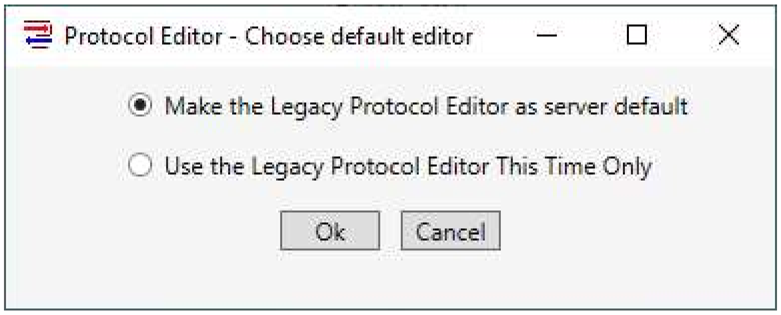
[● Visual Protocol Editor]

○ Legacy Protocol Editor

2. **From the Visual Protocol Editor** - To change from the Visual Protocol Editor to the Legacy Protocol Editor, go to the File menu and select Switch to Legacy Editor.



You'll then have the option of changing the default to the Legacy Protocol Editor or using the Legacy Protocol Editor this one time only (i.e. any future protocol creation or edits will still use the Visual Protocol Editor).

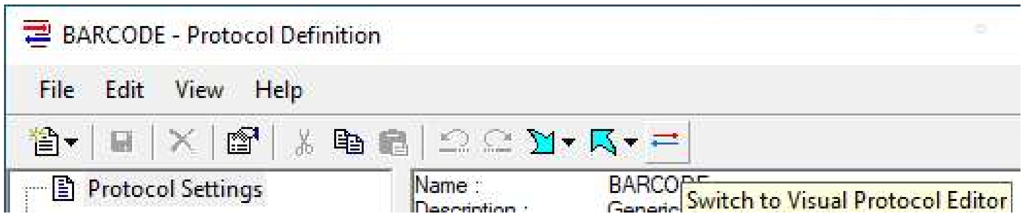

Protocol Editor - Choose default editor

(●) Make the Legacy Protocol Editor as server default

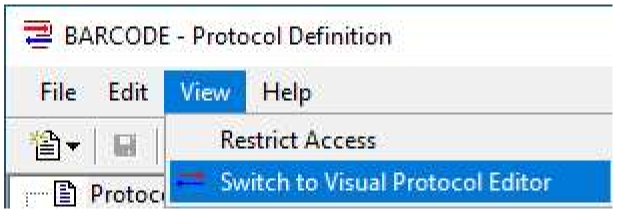
( ) Use the Legacy Protocol Editor This Time Only

3. **From the Legacy Protocol Editor** - To change from the Legacy Protocol Editor to the Visual Protocol Editor, you have two options in the Legacy Protocol Editor:

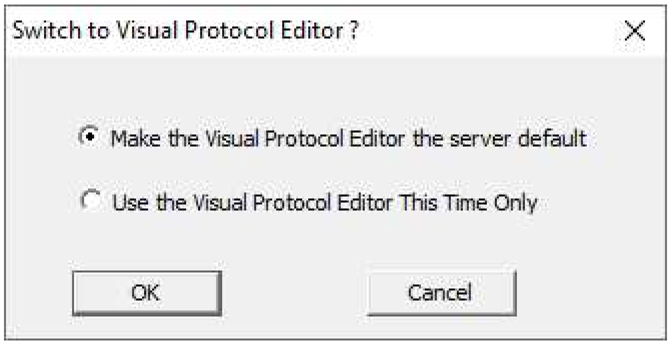
   a. **Toolbar Button** - Just click the toolbar button show in the screenshot below.



   b. **View Menu** - From the **View** menu, select **Switch to Visual Protocol Editor**.


BARCODE - Protocol Definition
File
Edit
View
Help
Restrict Access
Switch to Visual Protocol Editor
Protoco

After using either method above, you'll then have the option of changing the default to the Visual Protocol Editor or using the Visual Protocol Editor this one time only (i.e. any future protocol creation or edits will still use the Legacy Protocol Editor).

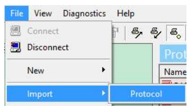

Switch to Visual Protocol Editor ?

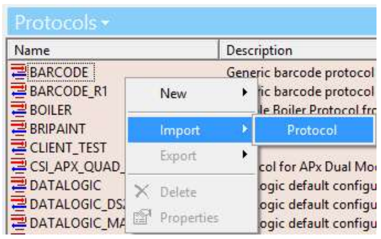
Make the Visual Protocol Editor the server default [✓]

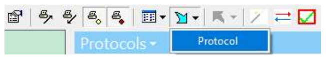
Use the Visual Protocol Editor This Time Only

Importing Existing Protocols

From the Protocols view, it is possible to access the Import option either from the File menu, by right-clicking in the white space of the Protocols view or by using the Import button in the toolbar.



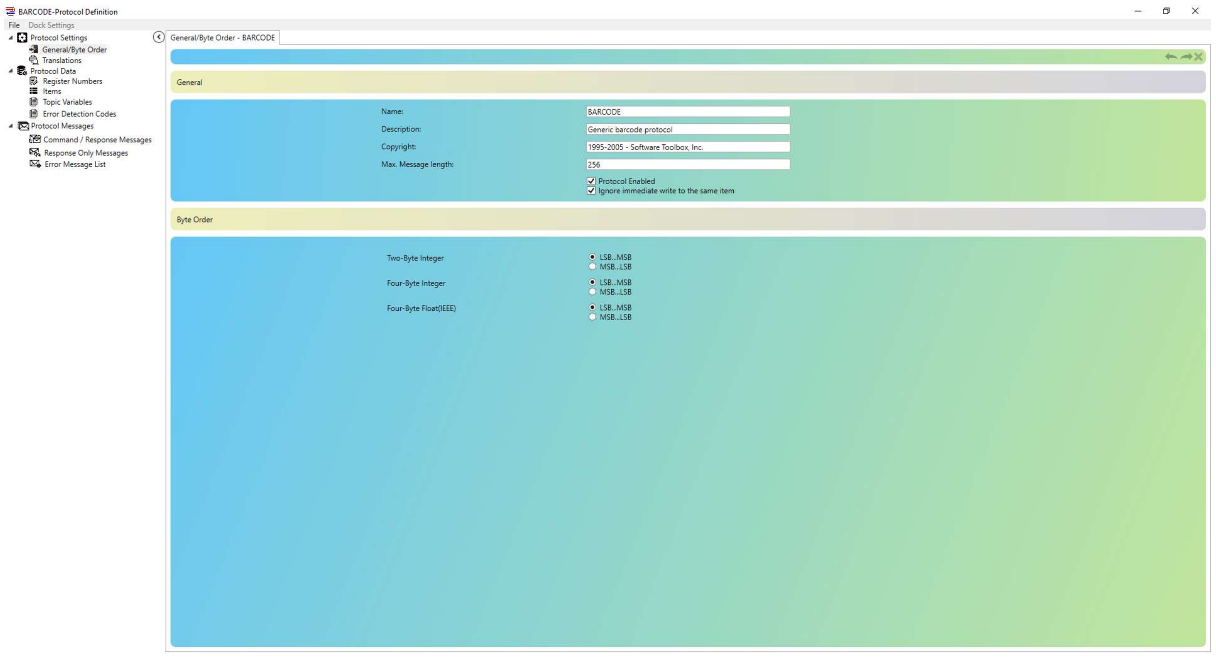




7.1.1.1 Visual Protocol Editor

An easier to use protocol builder with support for natural behaviors like dragging and dropping a message sequence, setting up your own layout with just the sequences you use the most when building an OmniServer protocol and even the ability to define your own custom "Favorite" sequences that you frequently use.

Navigate the underlying topics to explore the different components of an Omniserver protocol and how to configure them using the Visual Protocol Editor.



Name: BARCODE
Description: Generic barcode protocol
Copyright: 1995-2005 - Software Toolbox, Inc.
Max. Message length: 256
[✓] Protocol Enabled
[✓] Ignore immediate write to the same item

Two-Byte Integer: [✓ LSB...MSB] MSB...LSB
Four-Byte Integer: [✓ LSB...MSB] MSB...LSB
Four-Byte Float(IEEE): [✓ LSB...MSB] MSB...LSB

7.1.1.1.1 Menu Bar

You can access functions for saving the protocol and changing the layout configuration using the menu bar options.

The menu bar looks like this:

Placing your mouse pointer over one of the words and clicking the left mouse button will reveal a "pull-down" menu, which gives you an even greater choice in options.

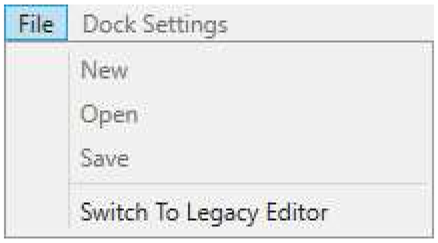
To discover what each of these menu selections do, click on the links below.

- The FILE Option
- The Dock Settings Option

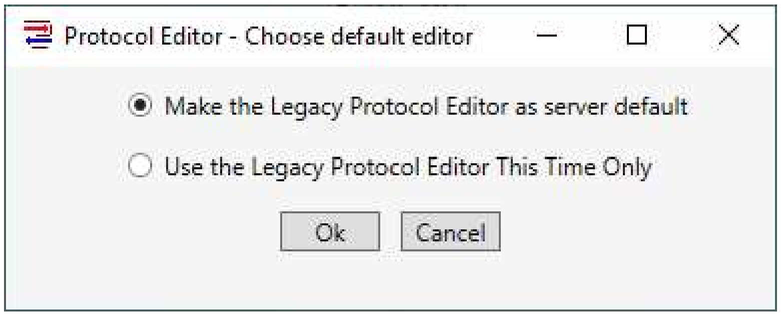
7.1.1.1.1.1 File

The first menu option available in a protocol is **File**. Here you will find the options for opening and saving a protocol file, as well as, switching to the Legacy Protocol Editor.

Some options, like "Dock Settings", are grayed out and unavailable whenever that particular option cannot be selected due to the current server state (Dock Settings only apply to Protocol Messages - so it will only be available when you're working in one of the underlying Protocol Messages sections).



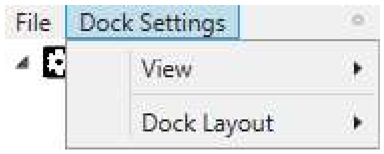
- **File | New** - _Currently Unavailable_ - This option is for creating a new protocol. Currently, it is only possible to create new protocols from the OmniServer Configuration. A future release will support starting a new protocol in a separate instance of the Visual Protocol Editor.
- **File | Open** - _Currently Unavailable_ - This option is for opening an existing protocol. Currently, it is only possible to open an additional existing protocol from the OmniServer Configuration. A future release will support opening an existing protocol in a separate instance of the Visual Protocol Editor.
- **File | Save** - Selecting this option saves the current changes to a protocol. This option will be unavailable if no changes have been made to the protocol.
- **File | Switch To Legacy Editor** - Select this option if you want to switch your current protocol over to the Legacy Protocol Editor. You will be presented with the following menu:


Protocol Editor - Choose default editor
Make the Legacy Protocol Editor as server default
Use the Legacy Protocol Editor This Time Only
Ok
Cancel

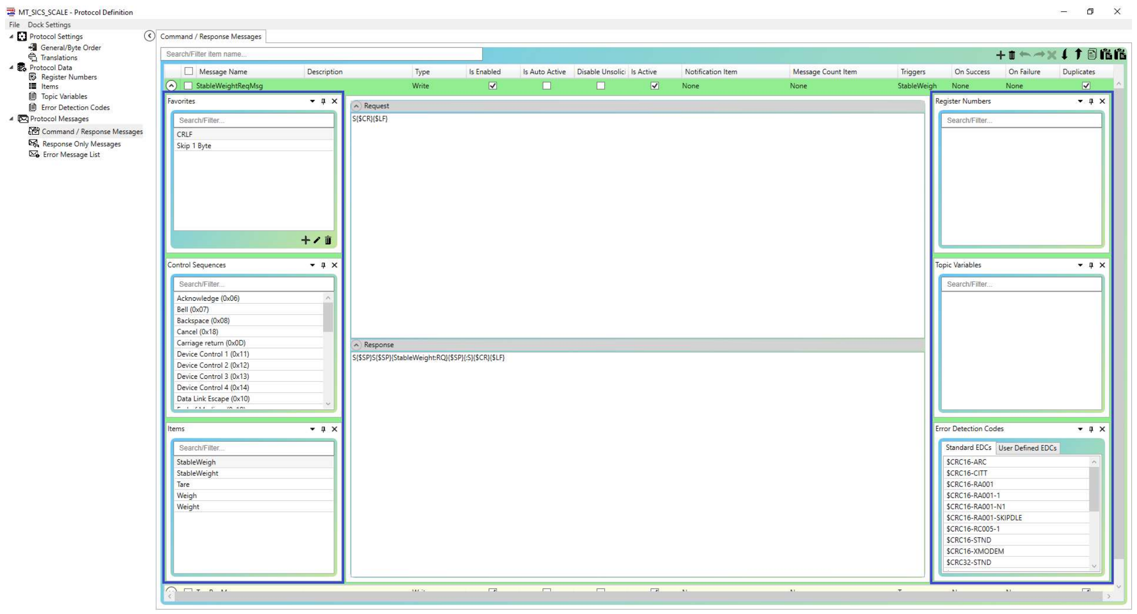
  - **Make the Legacy Protocol Editor as server default** - This option will change the server level default and opening protocols after this selection will use the Legacy Protocol Editor.
  - **Use the Legacy Protocol Editor This Time Only** - This option will NOT change the server level default and will use the Legacy Protocol Editor only until you close the current protocol.

7.1.1.1.1.2 Dock Settings

The second menu option available in the Visual Protocol Editor is **Dock Settings**. This menu option will be grayed out unless you are in one of the Protocol Messages sections (Command/Response Messages, Response Only Messages or Error Message List).



The Dock Settings provide options for hiding or making visible the different available modules used for adding items, registers, topic variables and other sequences to your protocol messages.



Message Name	Description	Type	Is Enabled	Is Auto Active	Disable Unsolici	Is Active	Notification Item	Message Count Item	Triggers	On Success	On Failure	Duplicates
StableWeightReqMsg		Write	✓			✓	None	None	StableWeigh	None	None	✓

Favorites: CRLF, Skip 1 Byte

Control Sequences: Acknowledge (0x06), Bell (0x07), Backspace (0x08), Cancel (0x18), Carriage return (0x0D), Device Control 1 (0x11), Device Control 2 (0x12), Device Control 3 (0x13), Device Control 4 (0x14), Data Link Escape (0x10)

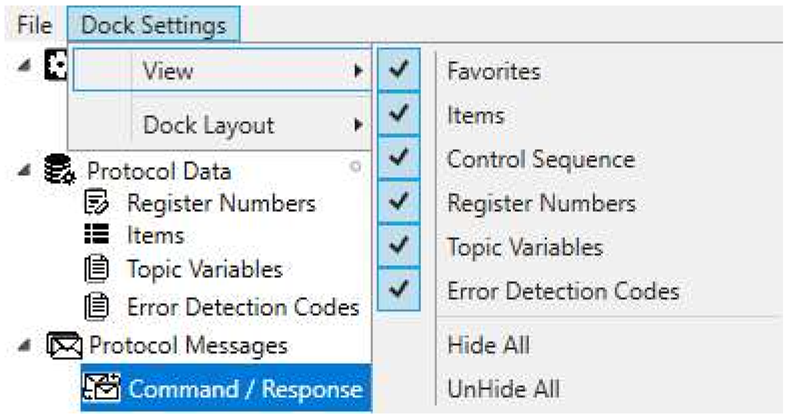
Items: StableWeigh, StableWeight, Tare, Weigh, Weight

Request: S{$CR}{$LF}

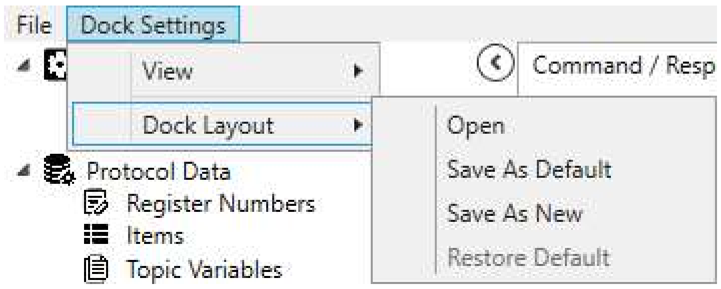
Response: S{$SP}S{$SP}{StableWeight:RQ}{$SP}{:S}{$CR}{$LF}

Standard EDCs: $CRC16-ARC, $CRC16-CITT, $CRC16-RA001, $CRC16-RA001-1, $CRC16-RA001-N1, $CRC16-RA001-SKIPDLE, $CRC16-RC005-1, $CRC16-STND, $CRC16-XMODEM, $CRC32-STND

These options allow you to display only the modules you need to use and save customized layouts for your own ease-of-use.



- **View** - provides options for customizing the view when editing Protocol Messages sections.
  - Favorites - Toggles the Favorites module to be visible or not visible.
  - Items - Toggles the Items module to be visible or not visible.
  - Control Sequence - Toggles the Control Sequence module to be visible or not visible.
  - Register Numbers - Toggles the Register Numbers module to be visible or not visible.
  - Topic Variables - Toggles the Topic Variables module to be visible or not visible.
  - Error Detection Codes - Toggles the Error Detection Codes module to be visible or not visible.
  - Hide All - Hides all sequence modules
  - Unhide All - Makes all sequence modules visible



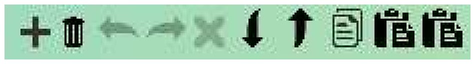
- **Dock Layout**

- Open - Allows you to select a saved configuration with the sequence modules you've chosen to be visible or not visible.
- Save As Default - Allows you to save the current layout as the default layout.
- Save As New - Allows you to save the current layout as a saved configuration that can be selected using the Open option.
- Restore Default - Allows you to restore the default layout/configuration (if you've used the Save As Default option, that new default will be loaded/restored).

7.1.1.1.2 Protocol Toolbars

You can access functions available in creating or maintaining parts of your protocol through the Tool Bar. Depending on which components of the protocol you are currently editing, the toolbar may have more or less options available

The following toolbar is available when configuring the following protocol components: Register Numbers, Items, Topic Variables, Error Detection Codes, and all Protocol Messages sections).



There are 10 toolbar selections:

+	Add	Create a new server object. This option will always be available.
🗑	Del ete	Delete the current object. This option is only available if an object has been selected.
←	Und o	Erases the last operation performed by the user and restores the previous condition of the protocol. This option is only available if the protocol has changed in some way.
→	Red o	"Undo" the last undo. Reverses the previous Undo operation. This option is only available if an Undo operation has been performed.

✖	Cancel	Cancels changes to the currently selected object.
⬇	Import	Allows import of certain protocol objects from a properly formatted CSV file. A Windows browser allows selection of the CSV file to be used for importing Items, Register Numbers, Topic Variables, Command/Response Messages, Response Only Messages and Error Messages.
⬆	Export	Allows export of certain protocol objects to a CSV file. A Windows browser allows selection of the name and location where the CSV file should be saved for exporting Items, Register Numbers, Topic Variables, Command/Response Messages, Response Only Messages and Error Messages.
📄	Copy	Copy an object to the internal clipboard. This option is only available if an object has been selected.
📋	Paste	Inserts an object copied to the internal clipboard into the protocol. This option is only available if an object has first been placed into the internal clipboard by a Copy operation.
📋	Bulk Paste	Inserts a copied object to the internal clipboard into the protocol multiple times. A secondary menu appears and allows you to specify how many instances you would like to be pasted. Useful for bulk adds of an existing item or other component that will be similar - an incrementing number is appended to the end of the name of the component.

The following toolbar is only available when configuring protocol Translations.

✚ 🗑 ↩ ↪ ✖

There are 5 toolbar selections:

➕	Add	Create a new translation. This option will only be available when a translation is not currently selected and being edited.
🗑	Del ete	Delete the current translation. This option is only available if a translation has been selected.
↶	Und o	Erases the last operation performed by the user and restores the previous condition of the protocol. This option is only available if the protocol has changed in some way.
↷	Red o	"Undo" the last undo. Reverses the previous Undo operation. This option is only available if an Undo operation has been performed.
✖	Can cel	Cancels changes to the currently selected translation.

7.1.1.1.3 Protocol Settings

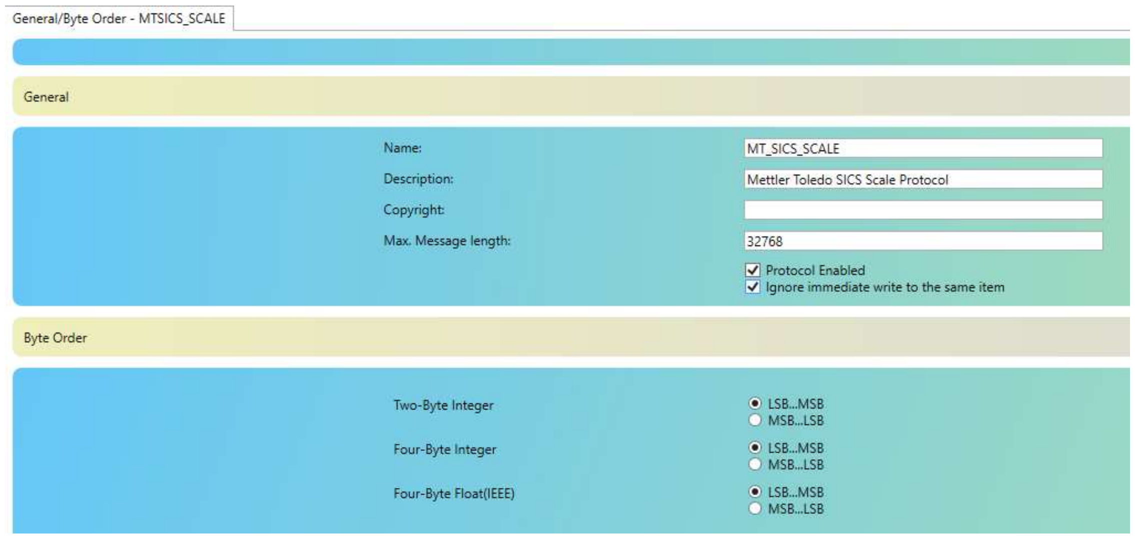
Protocol Settings are used to define basic properties global to your protocol, such as name and how binary data is to be interpreted.

When "Protocol Settings" is expanded there are two property sheets with which to define a protocol. They are:

- General/Byte Order: Defines the default operations and properties of the protocol and the default byte ordering for the protocol.
- Translations: Defines how to server will translate incoming or outgoing data.

7.1.1.1.3.1 General/Byte Order

The General /Byte Order section defines both identification and action properties of the protocol, as well as, the default byte ordering to be used (byte ordering can be overridden at the sequence level in messages, as well).


General/Byte Order - MTSICS_SCALE

General

Name: MT_SICS_SCALE
Description: Mettler Toledo SICS Scale Protocol
Copyright:
Max. Message length: 32768
[✓] Protocol Enabled
[✓] Ignore immediate write to the same item

Byte Order

Two-Byte Integer: [✓ LSB...MSB] MSB...LSB
Four-Byte Integer: [✓ LSB...MSB] MSB...LSB
Four-Byte Float(IEEE): [✓ LSB...MSB] MSB...LSB

The **General** options available are:

- **Name:** The name of the protocol, which must be unique within the server.
- **Description:** A description of the protocol. This is for documentation purposes only and is not used by the server.
- **Copyright:** Copyright information. All protocols are the property of the creators and not the manufacturer or licensor of the server. This is for documentation purposes only and is not used by the server.
- **Maximum Message Length:** The maximum number of bytes the server will read in and process from the device. This number can be anywhere between 1 and 32768.
- **Protocol is enabled:** Clearing this option will make the protocol unavailable to the server's runtime program, thus not transferring any data to the client. The default is **checked**, meaning that the protocol is available to the server.
- **Ignore intermediate writes to the same item:** When not checked, the server will send out all data to the device, even when that data repeats itself. The default is **checked**, meaning that the server will not send out the same write message twice, thus controlling the amount of unnecessary messaging to the device, or "spam".

The **Byte Order** options define the default order for the interpretation of two-byte integers, four-byte integers, and IEEE floating point real numbers.The following options available are:

- **Two-byte Integer:** Sets the processing of a two-byte integer. There are only two options:
  LSB...MSB: Also known as the "Little Endean" or "Intel" format, the second byte of the data stream is multiplied by 256 then added to the first byte in the data stream. This is the default.
  MSB...LSB: Also known as the "Big Endean" format, the first byte in the data stream is multiplied by 256 then added to the second byte in the data stream.

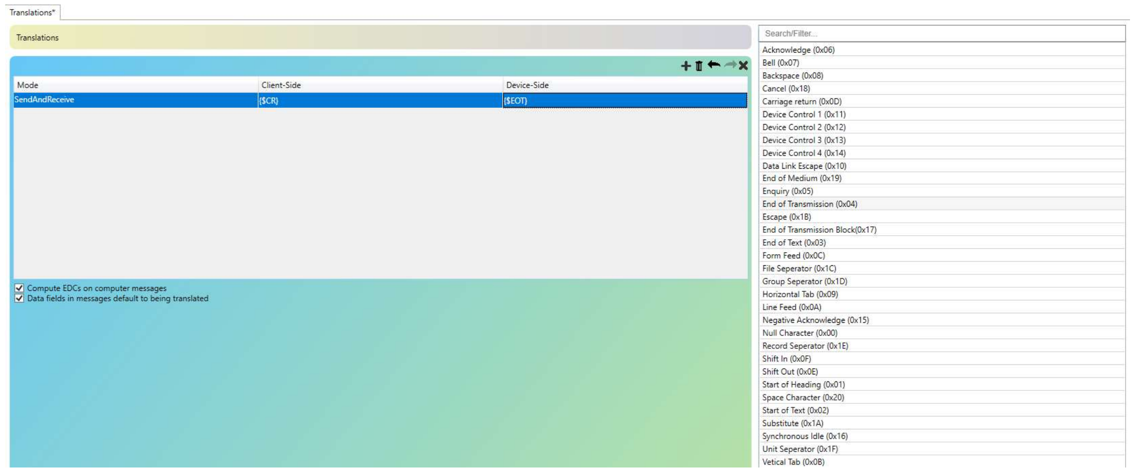
- **Four-byte Integer:** Sets the processing of a four-byte integer. There are only two options:
  LSB...MSB: Also known as the "Little Endean" or "Intel" format, the fourth byte in the data stream is multiplied by 16,777,216, the third byte in the data stream is multiplied by 65,536, the second byte in the data stream is multiplied by 256, and all these added together along with the first byte in to form the data stream the final number. This is the default.
  MSB...LSB: Also known as the "Big Endean" format, the first byte in the data stream is multiplied by 16,777,216, the second byte in the data stream is multiplied by 65,536, the third byte in the data stream is multiplied by 256, and all these added together along with the fourth byte in the data stream to form the final number.

- **Four-byte Float:** Sets the processing of a four-byte IEEE floating point integer. There are only two options:
  LSB...MSB: Also known as the "Little Endean" or "Intel" format, the IEEE floating point formula is applied to the data with the fourth byte of the data stream as the most significant byte (i.e., in reference to the data stream, 4th byte - 3rd byte - 2nd byte - 1st byte). This is the default.
  MSB...LSB: Also known as the "Big Endean" format, the IEEE floating point formula is applied to the data with the first byte of the data stream as the most significant byte (i.e. in reference to the data stream: 1st byte - 2nd byte - 3rd byte - 4th byte).

7.1.1.1.3.2 Translations

The Translations section defines how the server will translate incoming or outgoing data.

For a video tutorial on working with Translations, click here.



Mode	Client-Side	Device-Side
SendAndReceive	{$CR}	{$EOT}

☑ Compute EDCs on computer messages
☑ Data fields in messages default to being translated

Acknowledge (0x06)
Bell (0x07)
Backspace (0x08)
Cancel (0x18)
Carriage return (0x0D)
Device Control 1 (0x11)
Device Control 2 (0x12)
Device Control 3 (0x13)
Device Control 4 (0x14)
Data Link Escape (0x10)
End of Medium (0x19)
Enquiry (0x05)
End of Transmission (0x04)
Escape (0x1B)
End of Transmission Block(0x17)
End of Text (0x03)
Form Feed (0x0C)
File Seperator (0x1C)
Group Seperator (0x1D)
Horizontal Tab (0x09)
Line Feed (0x0A)
Negative Acknowledge (0x15)
Null Character (0x00)
Record Seperator (0x1E)
Shift In (0x0F)
Shift Out (0x0E)
Start of Heading (0x01)
Space Character (0x20)
Start of Text (0x02)
Substitute (0x1A)
Synchronous Idle (0x16)
Unit Seperator (0x1F)
Vetical Tab (0x0B)

The following toolbar is only available when configuring protocol Translations.

➕ 🗑 ⬅ ➡ ✖

There are 5 toolbar selections:

➕	Add	Create a new translation. This option will only be available when a translation is not currently selected and being edited.
🗑	Del ete	Delete the current translation. This option is only available if a translation has been selected.
⬅	Und o	Erases the last operation performed by the user and restores the previous condition of the protocol. This option is only available if the protocol has changed in some way.
➡	Red o	"Undo" the last undo. Reverses the previous Undo operation. This option is only available if an Undo operation has been performed.
✖	Can cel	Cancels changes to the currently selected translation.

Settings available when adding/editing a translation are:

- **Mode:** How the translation is applied to the message. "Send Only" will apply the translation to messages going from the server (here called the "computer") to the device (here called the "remote"). "Receive Only" will apply the translations to message coming in from the device (the "remote"). "Send and Receive" will apply the translation to both messages.

- **Client-Side (formerly Computer):** The text of the message as it will appear to the server. This is the pre-translation message on Send messages, and the post-translation message on Receive messages.
- **Device-Side (formerly Remote):** The text of the message as it will appear to the device. This is the pre-translation message on Receive messages, and the post-translation message on Send messages.

Settings always available that apply for translation server-wide:

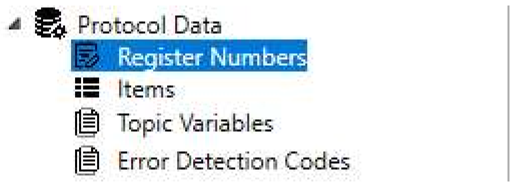
- **Compute EDCs on computer messages:** The server will compute all Error Detection Codes based upon the computer message, and not the message from the device. So for messages coming in from the device, the server will calculate EDCs based upon the message **after** translation has been done. For message going out to the device, EDCs are computed **before** the message is translated. The default is **checked**, which means that the EDCs are computed on the computer message.
- **Data fields in message default to being translated:** Clearing this field will instruct the server not to translate the data within messages unless that data is specifically formatted to translate (using the **Translate (T+)** format). The default is **checked,** which means that all data will be translated. This can be over-ridden on individual data items by using the **Do Not Translate (T-)** format.

Additionally, there is a module to the right of the Translations pane that contains available control character sequences. This module allows drag-and-drop of any available control characters to either of the translation fields (Client-Side or Device-Side).

The "Search/Filter" field may also be used to easily filter the list of control characters if you're having trouble finding the character you're interested in.

7.1.1.1.4 Protocol Data

The Protocol Data section contains sections for the main variable components of a protocol data message such as items which share data with client applications.

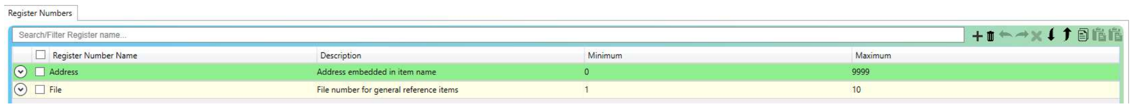


When "Protocol Data" is expanded there are four property sheets with which to define the following components:

- Register Numbers: A Register Number is a special server construct that extracts information from the name of the item.
- Items: An item is defined as the data that connects the client application with the server. The client will pass data to the server for further processing, and in turn, the server will parse data and send it back to the client through the item.
- Topic Variables: A Topic Variable is a user-defined server variable that is associated with a Topic. Instead of extracting information from an item, the Topic Variable is a constant number defined within a topic that gets passed into a protocol message. Topic variables make it easier to re-use protocols across multiple devices that have device-specific parameters such as a unique device ID per device.
- Error Detection Codes: An Error Detection Code is a calculation on a set of data that is used to insure that the data transmitted was without error. This section allows you to define custom EDCs for codes not covered by the pre-defined EDCs included with OmniServer.

7.1.1.1.4.1 Register Numbers

A Register Number is a special server construct that extracts information from the **name** of the item.



Register Number Name	Description	Minimum	Maximum
Address	Address embedded in item name	0	9999
File	File number for general reference items	1	10

For example, suppose you had ten items that all came from an Analog Input address bank of a device. Each one of these items was associated with addresses numbered 0 to 9999. Using the server, you could create items **AnalogInput0, AnalogInput1, ..., AnalogInput9999.** Or you can use a Register Number, and tell the server to expect a name that begins with **AnalogInput** and ends with a number. Therefore, your item name will look like this:

AnalogInput{Address}

Now each time the client requests a name like **AnalogInput245**, the server will extract the number **245** from the name and use that in the message. This allows you to make very generic messages without having to go to the trouble of creating a message for each address.

You create a new Register Number by clicking the **Add (+)** button in the toolbar.

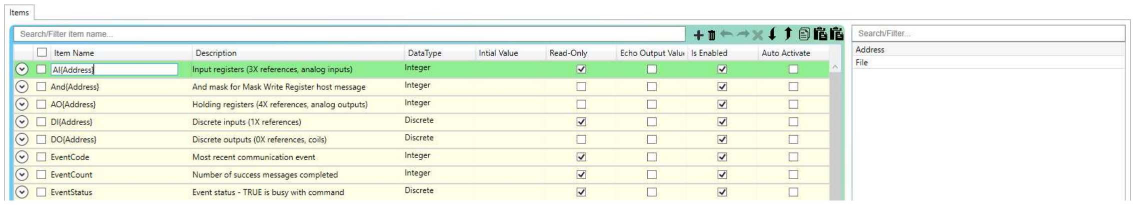
The settings associated with a Register Number are:

- **Register Number Name:** The name of the Register Number, which must be unique within the protocol. You will use this in conjunction with **Item Names** to specify banks of sequentially-numbered items.
- **Description:** A description of the register number. This is for documentation purposes only and is not used by the server.
- **Minimum:** The minimum numeric range accepted by the server. Used in conjunction with Maximum below, this tells the server not to accept any register numbers from the client lower than this number.
- **Maximum:** The maximum numeric range accepted by the server. Used in conjunction with **Range: Minimum** above, this tells the server not to accept any register numbers from the client higher than this number.

Register Numbers can also be imported and exported via CSV file using the Import ⬇ and Export ⬆ buttons in the toolbar.

7.1.1.1.4.2 Items

An **Item** is defined as the data that connects the client application with the server. The client will pass data to the server for further processing, and in turn, the server will parse data and send it back to the client through the item.



Item Name	Description	DataType	Intial Value	Read-Only	Echo Output Valu	Is Enabled	Auto Activate
AI{Address}	Input registers (3X references, analog inputs)	Integer		✓	☐	✓	☐
And{Address}	And mask for Mask Write Register host message	Integer		☐	☐	✓	☐
AO{Address}	Holding registers (4X references, analog outputs)	Integer		☐	☐	✓	☐
DI{Address}	Discrete inputs (1X references)	Discrete		✓	☐	✓	☐
DO{Address}	Discrete outputs (0X references, coils)	Discrete		☐	☐	✓	☐
EventCode	Most recent communication event	Integer		✓	☐	✓	☐
EventCount	Number of success messages completed	Integer		✓	☐	✓	☐
EventStatus	Event status - TRUE is busy with command	Discrete		✓	☐	✓	☐

You create a new Item by clicking the **Add (+)** button in the toolbar.

The settings associated with an Item are:

- **Item Name:** This name must be unique within the protocol. It will be used to identify the data variable between the client and the server. If you also use Register Numbers, then you can also specify a register number with name.

- **Description:** A description of the item. This is for documentation purposes only and is not used by the server.

- **Data Type:** The class of data used by this item. It can be one of four types:

  - Discrete: A single bit value. Usually expressed as Zero/One, or False/True.

  - Integer: A number expressed without any decimal places.

  - Real: A number expressed with or without decimal places. Also called a floating-point number.

  - String: Any sequence of alpha-numeric characters.

- **Initial Value:** The initial value for the item to be used for the item. This is not used if the field is left blank.

- **Read-Only:** When checked, the item can only pass data to the client. Any attempt by the client to write/change the data is ignored. The default is not-checked, which means the server will accept data from the client for the item.

- **Echo Output Value:** When checked, the server will copy the outgoing value of this item to its incoming value. The server actually keeps two copies of the data - one for the last value written by the client, and another for the last value read in from the device. By default, these two values are kept separate, so that when data is sent to the server by the client, the last value read in from the device is preserved.

- **Is Enabled:** When checked (the default) the server will make the item available to any client. By clearing this option, the item will never be made available to a client for reads or writes.
- **Auto Activate:** When checked, this option will cause the server to activate, or make available, this item when clients first connect to the server. The default is not-checked, meaning that the server will only activate this item if at least one client specifically requests the item, or the protocol itself automatically activates it.

Additionally, there is a module to the right of the Items pane that contains available Register Numbers. This module allows drag-and-drop of any available Register Numbers to the Item Name.

The "Search/Filter" field may also be used to easily filter the list of Register Numbers which is useful in situations where the protocol has a large number of Register Numbers.

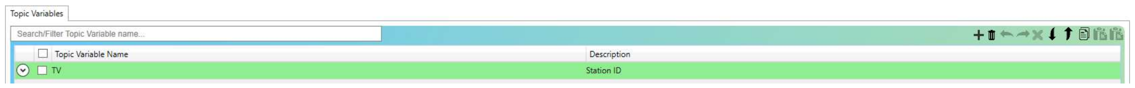
Items can also be imported and exported via CSV file using the Import and Export buttons in the toolbar.

7.1.1.1.4.3 Topic Variables

A Topic Variable is a user-defined server variable that is associated with a Topic. Instead of extracting information from an item, the Topic Variable is a constant number defined within a topic that gets passed into a protocol message wherever the Topic Variable has been inserted.

For example, Topic Variables are useful in protocols where there is a unique ID assigned to each device that is included in any commands/responses. A Topic Variable can be defined in place of a hard coded device ID, making the protocol reusable across devices.

Within the protocol, you define each Topic Variable, but the actual value of that topic variable is defined within the topic.



Topic Variable Name	Description
TV	Station ID

You create a new Topic Variable by clicking the **Add (+)** button in the toolbar.

The settings associated with a Topic Variable are:

- **Topic Variable Name:** The name of the Topic Variable, which must be unique within the protocol.

  Please note: If you change the name (not the description) of a topic variable, all references in the protocol will also be changed. **However**, any topics that use the protocol will have to have the value of the renamed topic variable set again.

- **Description:** A description of the topic variable.

Additionally, at the top of the Topic Variables pane, the "Search/Filter Topic Variable name" field may be used to easily filter the list of Topic Variables which is useful in situations where the protocol has a large number of Topic Variables.

Topic Variables can also be imported and exported via CSV file using the Import and Export buttons in the toolbar.

7.1.1.1.4.4 Error Detection Codes

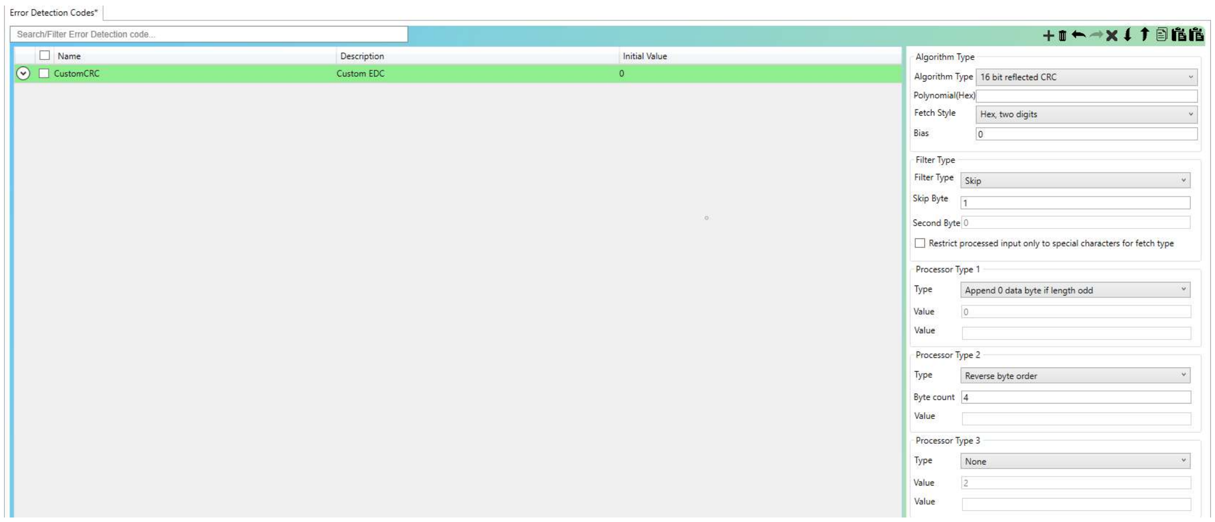
Server Details : Configuration : Protocol

Error Detection Code Builder

An **Error Detection Code** is a calculation on a set of data that is used to insure that the data transmitted was without error.

There a selection of error detection codes (or EDCs) pre-defined and available for your use. You can find a list here. However, there are times where you will need to define a variant of an existing code, or even a completely new code.

To help you in the design of a new EDC, OmniServer has what is known as an **EDC Builder**. To access the EDC builder, create a new Error Detection Code using the Add button in the toolbar.



Error Detection Codes*

Search/Filter Error Detection code...

Name	Description	Initial Value
CustomCRC	Custom EDC	0

Algorithm Type
- Algorithm Type: 16 bit reflected CRC
- Polynomial(Hex):
- Fetch Style: Hex, two digits
- Bias: 0

Filter Type
- Filter Type: Skip
- Skip Byte: 1
- Second Byte: 0
- ☐ Restrict processed input only to special characters for fetch type

Processor Type 1
- Type: Append 0 data byte if length odd
- Value: 0
- Value:

Processor Type 2
- Type: Reverse byte order
- Byte count: 4
- Value:

Processor Type 3
- Type: None
- Value: 2
- Value:

The list of Error Detection Codes can also be filtered for protocols where there are a large number of EDCs available.

- **Name:** The name of the EDC. This must be unique among the server. It is recommended to use a name that is meaningful for use when selecting the EDC in a protocol message.
- **Description:** A description of the EDC. This is for documentation purposes only, and is not required or used by the server.
- **Initial Value:** Starting seed value for the EDC.
- **Algorithm Type settings**
- **Filter Type settings**
- **Processor Type settings**

The Algorithm portion of a custom EDC is the basic calculation to be made on the data. The Algorithm types are:

- 8-Bit CRC:

  8 bit CRC with the polynomial specified as a 8 bit hex value. Requires a Polynomial.

- 8-Bit Reflected CRC:

  8 bit reflected CRC with the polynomial specified as a 8 bit hex value. Requires a Polynomial.

- 16-Bit CRC:

16 bit CRC with the polynomial specified as a 16 bit hex value. Requires a Polynomial.

- 16-Bit Reflected CRC:

  16 bit reflected CRC with the polynomial specified as a 16 bit hex value. Requires a Polynomial.

- 32-Bit CRC:

  32 bit CRC with the polynomial specified as a 32 bit hex value. Requires a Polynomial.

- 32-Bit Reflected CRC:

  32 bit reflected CRC with the polynomial specified as a 32 bit hex value. Requires a Polynomial.

- Honeywell CRC:

  Variation of the CRC-CCITT standard CRC used by many Honeywell devices. No other options are required.

- 8-Bit Checksum, 8-bit Accumulator:

  Checksum of data in 8 bit bytes using an 8 bit accumulator. No other options are required.

- 8-Bit Checksum, 32-bit Accumulator:

  Checksum of data in 8 bit bytes using a 32 bit accumulator

- 16-Bit Checksum, 16-bit Accumulator:

  Checksum of data in 16 bit words using a 16 bit accumulator

- 8-Bit LRC:

  8-bit LRC. Also known as XOR.

- 8-Bit LRC Rotated Left:

  8 bit LRC with result shifted left one bit after each byte is added to the LRC

- 8-Bit LRC Rotated Right:

  8 bit LRC with result shifted right one bit after each byte is added to the LRC

- 16-Bit LRC:

  16-bit LRC. Also known as XOR.

- 16-Bit LRC Rotated Left:

  16 bit LRC with result shifted left one bit after each byte is added to the LRC

- 16-Bit LRC Rotated Right:

  16 bit LRC with result shifted right one bit after each byte is added to the LRC

- Special Checksum Combo and LRC Algorithm 1:

  The high-order byte of the CSUML is the LRC value while the low-order byte is the standard checksum plus the value of the LRC byte.

- Special Checksum Combo and LRC Algorithm 2:

  LRC value as the low-order byte and the standard checksum in the high-order byte.

- Special Checksum Combo and LRC for UKCOMM:

  24-bit value consisting of the concatenation of a special 8-bit LRC, the standard checksum, and the standard LRC. The special LRC simply shifts the intermediate value up one bit after each exclusive or operation.

- Special 16-bit CRC 1:

  Based on the standard method of computing CRCs, this handles the XORing of the data into the intermediate CRC differently.

  Following is the code used to compute the this CRC:

```
ErrorCode := seedValue
DataCode := 0;
for each character c in the message
    ErrorCode := ErrorCode xor c
    do 8 times
        bit :=most significant bit of ErrorCode
        ErrorCode := up shift of ErrorCode
        if bit is on
            ErrorCode := ErrorCode xor generatingPolynomial
        end if
    end do
    DataCode := up shift of DataCode by 8 bits
    DataCode := DataCode or c
    ErrorCode := ErrorCode xor DataCode
end for
```

- Special 16-bit CRC 2:

  Ignores the leading ASCII DLE, or hexadecimal 10, of a pair of characters.

  Any character following a ASCII DLE, or hexadecimal 10 is biased by 1. This addition of 1 is allowed to overflow into high byte of the CRC computation.

  For example, the pair 10 FF would cause the 10 to be ignored and FF+1 or 100 to used in the CRC computation. Output Value: Ones complemented. Due to a bug in the implementation, the CRC is sign extended to a 32 bit value. To work around this problem, use signed formatting instead of unsigned formatting.

- Special Decimal-Encoded Checksum:

  8 bit checksum with the resulting value formatted for zero based encoding.

  Zero-based encoding takes a byte, splits it into two 4-bit hexadecimal digits, adds a hexadecimal 30 or ASCII 0 to each, and concatenates the two into a 16-bit result. For example, the hexadecimal byte value of 4F become the 34 3F, or ASCII 4?.

- Special Hex 40 Encoded 12-bit Checksum:

  Computes the ones complement of the standard 12-bit checksum. The 12-bit result is split into two 6-bit numbers, a hexadecimal 40 added to each number. Each of these numbers then are concatenated into a 16-bit result. This yields two characters in the printable ASCII range of hexadecimal 40 to 7F.

- Special Decimal-Encoded 8-bit CRC 1:

  8 bit checksum of the input with the 8th bit of each data byte removed. The upper two bits of the result are Exclusive ORed with the lower two bits of the sum. The upper two bits are then cleared. ASCII ‘0’ is then added to the final value.

- **Polynomial (hex):** The value applied to a CRC error detection code. Only applicable and used for CRC-type EDCs.
- **Fetch Style:** In addition to the translations, this tells the server how to modify the data before it goes to the device or after it comes in from the device.
    - Binary, One Byte:

      Each byte of the input stream is supplied to the EDC as a binary value 0-255.
    - Binary, Two Byte:

      Each pair of bytes are supplied to the EDC as a binary value 0-65535.
    - Decimal, One Digit:

      Each byte of the input stream is interpreted as a decimal value and supplied to the EDC as a binary value 0-9. Non-decimal digits are interpted as 0.
    - Hex, One Digit:

      Each byte of the input stream is interpreted as a hex value and supplied to the EDC as a binary value 0-16. Non-hex digits are interpreted as 0.
    - Hex, One Digit with Special Conversion:

      Each byte of the input stream is interpreted as a special hex character and sent to the EDC as a binary value 0-56. Digits 0-9 are interpreted as binary 0-9. Letters A-F are interpreted as binary 10-16. Letters a-f are interpreted as binary 10-16. Letters g-z are interpreted as "G" - "Z" or binary 47-56.
    - Hex, Two Digits:

      Pairs of hex digits from the input stream are supplied to the EDC as a binary value 0-255. Non-hex digits are interpreted as 0. The string "012315" is sent to the EDC as the values 1, 35 and 21.
    - Hex, Two Swapped Digits:

      Pairs of hex digits from the input stream are supplied to the EDC as a binary value 0-255. However, the digits are interpreted in reversed order. Non-hex digits are interpreted as 0. The string "012315" is sent to the EDC as the values 16, 50 and 81.

- Hex, Four Digits:

  4 hex digits from the input stream are supplied to the EDC as a binary value 0-65535. Non-hex digits are interpreted as 0.

- **Bias:** A bias set upon the calculation.

The **Filter Type section** of the EDC Builder is where you specify if and how the data will be "filtered" from the protocol message while the EDC is being calculated. Please note that the two text boxes underneath the **Filter Type** option may not be available, depending upon the Filter Type selected. Please refer to the documentation below to determine what's available at what time.

- **Filter Type:** The type of filter to be applied to the data. The types are:
  - **None:**

    No filters are applied to the data. The two text boxes are not available.
  - **Skip:**

    If a byte has the specified value (in decimal), it is ignored as part of the EDC.

    Enter the byte value of the number to skip. The second text box is unavailable.
  - **Skip Second Byte of a Pair:**

    If the pair of the first byte followed by the second byte (both in decimal) are found in the EDC data, then the second byte is ignored.

    Enter the byte value of the first byte in the first text box and the value of the second byte in the second text box.
  - **Bias:**

    Bias each byte of the input stream by the given value (in decimal).

Enter the bias. The second text box is unavailable.

- **Ignore Control Character (0-31):**

  Ignores all control characters, that is, characters with ASCII values from 00 to 31 (00h to 1Fh). The two text boxes are not available.

- **Restrict processed input...:** Restricts the filter process to only valid characters of the fetch type.

The **Processor Type (1 to 3) sections** allow post-processing to the EDC once the initial calculation has been completed. ***Please note:*** The final two text boxes are available based upon what has been selected in the pull-down boxes of the "Type" option.

The options available are:

- **Type:** The type of post-processing. The types are:
  - **None:**

    No post processing.

  - **Append 0 data byte if length odd:**

    If the length of the EDC data is odd, then append a 0 to the EDC. Text boxes are not available.

  - **One's Compliment:**

    Compute the one's complement of the end result. The Bitmask parameter is a bit mask in decimal of which bits to compute the one's complement. (i.e. 255 for a byte, 65535 for 2 bytes...). Second text box is not available.

  - **Two's Compliment:**

    Compute the two's complement of the end result. The Bitmask parameter is a bit mask in decimal of which bits to include in the two's complement. Second text box is not available.

  - **Reverse byte order:**

    Reverse the byte order of the end result. The Byte count parameter is the number of bytes. Only 2 and 4 is supported. Second text box is not available.

- **Reverse nibble order:**

  Reverse the nibble order of the end result. The Nibble count parameter is the number of nibbles. Only 2, 4, and 8 are supported. Second text box is not available.

- **AND result with value:**

  And the result with the given bit mask. The Value parameter is the bit mask in decimal. Second text box is not available.

- **If result less than value, add bias:**

  If the end result is less than the Test Value parameter, then the Value parameter is added to the end result.

- **Bias the result:**

  Add the Value parameter to the end result. Second text box is not available.

- **Return modulo of result and value:**

  Return the modulo of the end result and the Value parameter. Second text box is not available.

- **OR result with value:**

  Or the result with the given bit mask. The Value parameter is the bit mask in decimal. Second text box is not available.

- **Sum all bytes into a single byte value:**

  Sum the specified number of bytes of the end result into a final single byte. The Byte count parameter is the number of bytes. Only 1, 2, 3, and 4 are supported.

- **If result equals value, set to new value:**

  If the end result equals the Test Value parameter, then the Value parameter is used.

- **Extend the nibbles to bytes:**

  Shift the nibbles of the end result into their own bytes. The Nibble count parameter specifies the number of nibbles.

- **Reverse extend the nibbles to bytes:**

Shift the nibbles of the end result into their own bytes but in reverse order. The Nibble count parameter specifies the number of nibbles.

- **Add an even parity bit:**

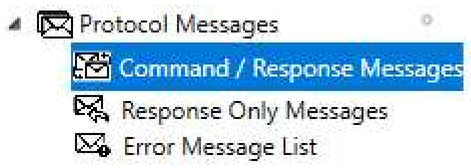
  Add an even parity bit to the end result. The Bit count parameter is the number of bits. Thus, if an parity bit is to be added to 7 bits of data, then 7 should be specified resulting in 8 bits of total data.

- **One's compliment combination:**

  The end result is masked by the number of bits specified in the Bit count parameter. Then a new value is computed being of 2*parameter bits long. The top half of the value is the masked end result. The lower half is the one's complement of that value.

7.1.1.1.5 Protocol Messages

The Protocol Messages section contains sections for the different types of messages that can be sent and/or received for a protocol.



When "Protocol Messages" is expanded there are three property sheets with which to define the following messages:

- Command/Response Messages: Formerly referred to as Host Messages, the heart of the server is the Command/Request Message. It is designed to solicit responses from or send data directly to the device. This is accomplished through Request and Response messages.

- Response Only Messages: A Response Only Message (formerly referred to as an Unsolicited Message) is designed to take care of data that comes in from the device without any prompting from the server - thus "unsolicited". The Response Only Message is primarily used for devices such as scales and barcode scanners where the server doesn't know when the data might come in from the device. However, Response Only Messages can also be used to trap secondary responses to Command/Request (Host) Messages (much in the same way Error Messages are used).

- Error Message List: An Error Message can also be defined as an alternative response to a Command/Request (Host) Message. It is checked against the data stream much like a Response Only Message, but whereas a Response Only Message is always accepted, you can set the acceptance level of an Error Message.

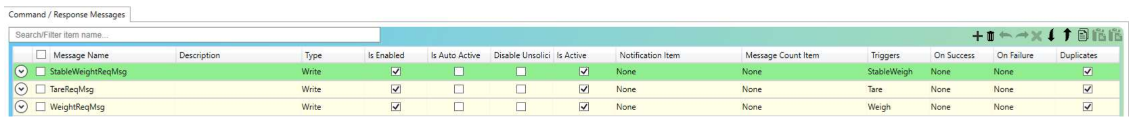
Aside from the types of messages, another component of configuring any message is the format of the "sequences" in those messages. A sequence can be any byte or combination of bytes that represents something, such as an Item, a Topic Variable, an EDC, etc.

Formatting, including length, data type, etc. is sometimes required - this is accomplished using the Sequence Formatter (which is similar in some ways to the Sequence Builder in the Legacy Protocol Editor).

7.1.1.1.5.1 Command/Request (Host) Message

The heart of the server is the Command/Request Message. It is designed to elicit responses from or send data directly to the device. This is accomplished through the Request and Response messages. This is how it works:

A **Request** is first sent to the device. If no response is expected, then the server moves on to another action. Otherwise, if a **Response** is expected, the server will wait for the device to send data back. If that data matches the **Response**, then the items (if any) are extracted from the messages and sent back to the client. If the **Response** is not satisfied, the server will, by default, retry the command/request message until it reports a failure or the device responds with appropriate data.


Command / Response Messages

Search/Filter item name...

Message Name	Description	Type	Is Enabled	Is Auto Active	Disable Unsolici	Is Active	Notification Item	Message Count Item	Triggers	On Success	On Failure	Duplicates
StableWeightReqMsg		Write	✓	☐	☐	✓	None	None	StableWeigh	None	None	✓
TareReqMsg		Write	✓	☐	☐	✓	None	None	Tare	None	None	✓
WeightReqMsg		Write	✓	☐	☐	✓	None	None	Weigh	None	None	✓

A new Command/Response Message can be added by clicking the Add ✚ button in the toolbar.

Prior to defining the actual request and response messages, the following properties of a Command/Response Message are defined:

- Message Name: The name of the message, which must be unique within the protocol.

- Description: A description of the message. This is for documentation purposes only and is not used by the server.

- Type: Controls the processing of the command/request message. There are two different types of command/request messages:
    - ***Read:*** "Reads" data from the device. If an item being requested by the client appears in the **Response** message of the Command/Request (Host) Message, then the server will send out the **Request** message in order to get the response from the device. This process happens without any interaction with the client or the user.
    - ***Write:*** "Writes" data to the device. If the client changes the value of an item in the server, the server will find any "write" Command/Request (Host) Messages that have the item in the **Request** message. Once found, that Command/Request (Host) Message is activated. This process happens without any interaction with the client or the user.

      The default for type is **Read**. However, the default operation of both the "Read" and "Write" messages can be over-ridden by setting options in the Triggers, On Success and/or On Failure columns.
- Is Enabled: The message is processed when this box is checked, otherwise, the message is never processed. The default for this option is checked, meaning that the message is processed.
- Is Auto Active: Setting this option tells the server to process this message immediately upon connection with the client, then process the command/request message normally. Set this option when you need to send initialization codes to the device upon startup. The default is **not set**, meaning that the command/request message is processed normally.
- Disable Unsolicited: This option informs the server that no Response Only (i.e. unsolicited) messages are to be processed when the command/request message response is being parsed. By default, the server will always compare all Response Only messages with the response of the command/request message (since error conditions can occur within a valid response). Setting this option disables this feature. The default is **not checked**, meaning that all unsolicited messages are processed normally.
- Activation: This option over-rides the default processing of command/request messages. Checking this option will mean that the command/response message will only be processed if the

Trigger is set or if this message is in a On Success or On Failure chain of another command/response message. The **Type** of the message (Read or Write) is ignored. The default is **not set**, meaning this message will be processed normally.

- Notification Item: The item selected in this box is set to One or True once the command/request message has been activated AND the message successfully completes. If the message does not complete, then this item is set to Zero or False. This is very useful in determining if a particular Command/Response (Host) Message is not working correctly. The default is **None**, meaning that there is no Notification Item associated with this message.
- Message Count Item: The item selected in this box is incremented whenever this message successfully completes.
- Triggers: An item that "triggers" this message. Whenever the server sees that the client has changed the specified trigger point item to anything other than Zero or False, the server will immediately process the command/response message, complete the parsing, then set the trigger point item back to Zero or False.

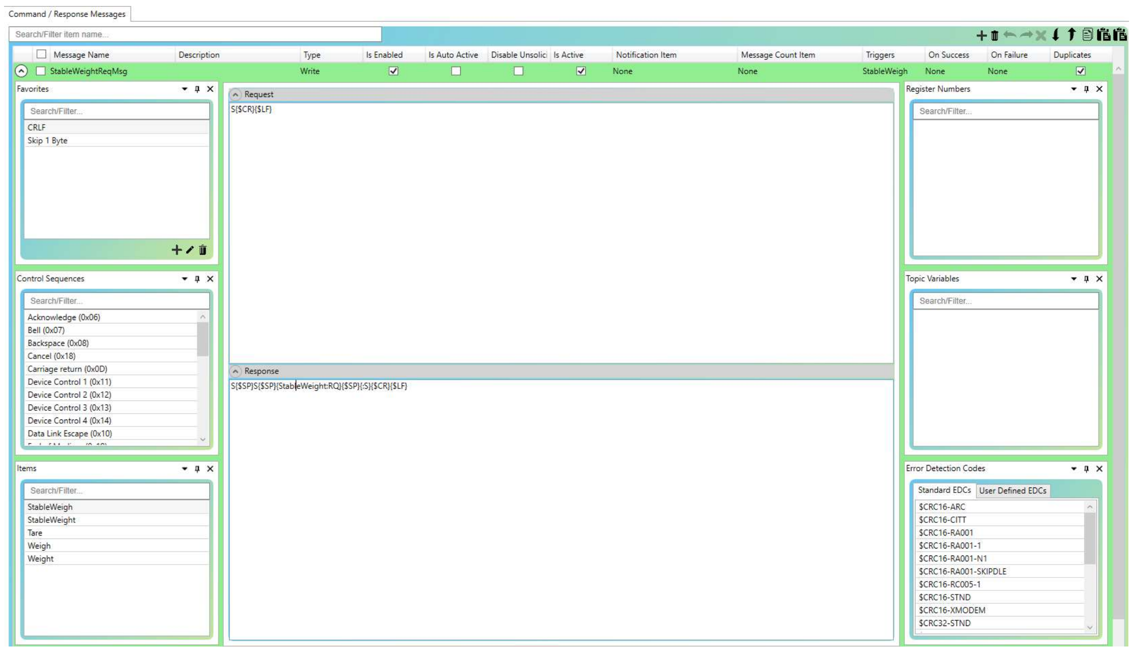
  This is useful in "push-button" situations, where one will need to send out a command/response message based upon a certain condition. The default is **None**, meaning that there is no item that will trigger this command/response message.
- On Success: Setting this option tells the server what command/response message to process if this command/response message successfully completes, meaning that the request and response were processed without errors.

  This is useful in sending more than one command/response message to the device based upon certain conditions. The default is **None**, meaning that no command/response message is processed after this one completed.
- On Failure: Setting this option tells the server what command/response message to process if this command/response message is unsuccessful in completing both the request and response side of the command/request message.

  This is useful in sending messages like initialization strings to recover from errors. The default is **None**, meaning that no command/response message is processed after this one completed.

- Duplicates: Only allow this message to have one outstanding triggered occurrence. When checked, the server will never queue up more than one instance of this message when triggered by the client. This prevents the server from overloading the message queue and causing low memory conditions.

Clicking the down arrow to the left of the new message expands to the actual Request/Response section of the Command/Response Message.



Message Name	Description	Type	Is Enabled	Is Auto Active	Disable Unsolici	Is Active	Notification Item	Message Count Item	Triggers	On Success	On Failure	Duplicates
StableWeightReqMsg		Write	✓	☐	☐	✓	None	None	StableWeigh	None	None	✓

Request

```
S{$CR}{$LF}
```

Response

```
S{$SP}S{$SP}{StableWeight:RQ}{$SP}{:S}{$CR}{$LF}
```

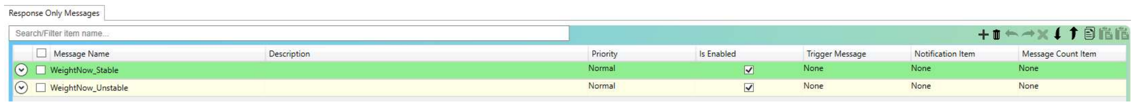
Here you build the actual messages to be sent to and/or received from the device. These sections support drag-and-drop of the different Sequence Modules to the left and right.

- Request: The Request field defines the format of the message to be sent to the device.
- Response: The Response field defines the format of the message expected in response to the request message. This message is sent by the device to the server.

Both message fields also support direct character entry for ASCII alphanumeric characters.

7.1.1.1.5.2 Response Only (Unsolicited) Message

A Response Only (Unsolicited) Message is designed to take care of data that comes in from the device without any prompting from the server - thus "unsolicited". The Response Only Message is primarily used for devices such as scales and barcode scanners where the server doesn't know when the data might come in from the device. However, they can also be used to trap secondary responses to Command/Response (Host) Messages (much in the same way Error Messages are used).


Response Only Messages

Search/Filter item name...

Message Name	Description	Priority	Is Enabled	Trigger Message	Notification Item	Message Count Item
WeightNow_Stable		Normal	✓	None	None	None
WeightNow_Unstable		Normal	✓	None	None	None

A new Response Only Message can be added by clicking the Add button in the toolbar.

Prior to defining the actual received and response messages, the following properties of a Response Only Message are defined:

- Message Name: The name of the message, which must be unique within the protocol.
- Description: A description of the message. This is for documentation purposes only and is not used by the server.
- Priority: Process the message given a certain Priority. This means that this unsolicited message will be evaluated before or after any others. The three different priorities are:

  Low: Process this unsolicited message after any other messages with priorities of Normal or High. If more than one message is marked Low, the messages are processed in the order of their creation.

  Normal: Process this unsolicited message after any messages marked as High, but before messages marked as Low. If more than one message is marked Normal, the messages are processed in the order of their creation.

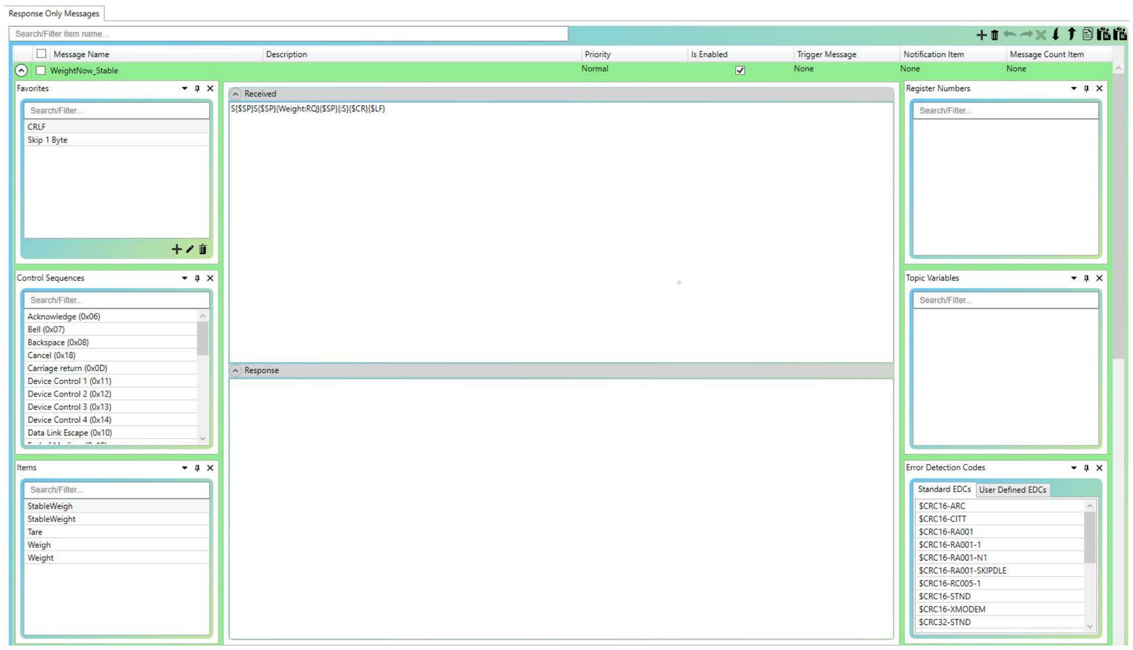
  High: Process this unsolicited message before any marked Normal or Low. If more than one message is marked High, the messages are processed in the order of their creation.
  The default for priority is **Normal**.

- Is Enabled: The message is processed when this box is checked, otherwise, the message is never processed. The default for this option is **checked**, meaning that the message is processed.
- Trigger Message: Defines the Command/Response (Host) Message to be processed once this unsolicited message has been activated. The Command/Response Message will get processed after the Response side of the unsolicited message is sent to the device, and after the Notification Item is set.

  Unsolicited triggered messages are queued to the current pending message list. Thus, if there are read or write messages due, they will be executed prior to the unsolicited message's triggered message.

  The default is **None**, meaning that no Command/Request (Host) Message will be activated.
- Notification Item: The item selected in this column is set to One or True once the unsolicited message has been activated. This is useful in knowing when an unsolicited message has arrived. The default is **None**, meaning that there is no Notification Item associated with this message
- Message Count Item: The item selected in this column is incremented whenever this message successfully completes. The default is **None**, meaning that there is no Notification Item associated with this message.

Clicking the down arrow to the left of the new message expands to the actual Received/Response section of the Response Only Message.



Response Only Messages

	Message Name	Description	Priority	Is Enabled	Trigger Message	Notification Item	Message Count Item
	WeightNow_Stable		Normal	✓	None	None	None

Received

```
S{$SP}S{$SP}{Weight:RQ}{$SP}{:S}{$CR}{$LF}
```

Response

Favorites: CRLF, Skip 1 Byte

Control Sequences: Acknowledge (0x06), Bell (0x07), Backspace (0x08), Cancel (0x18), Carriage return (0x0D), Device Control 1 (0x11), Device Control 2 (0x12), Device Control 3 (0x13), Device Control 4 (0x14), Data Link Escape (0x10)

Items: StableWeigh, StableWeight, Tare, Weigh, Weight

Error Detection Codes (Standard EDCs): $CRC16-ARC, $CRC16-CITT, $CRC16-RA001, $CRC16-RA001-1, $CRC16-RA001-N1, $CRC16-RA001-SKIPDLE, $CRC16-RC005-1, $CRC16-STND, $CRC16-XMODEM, $CRC32-STND

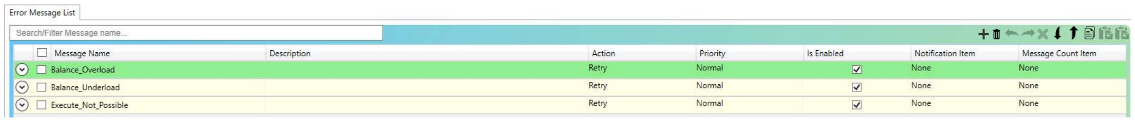
Here you build the actual messages to be received from and/or sent to the device. These sections support drag-and-drop of the different Sequence Modules to the left and right.

- Received: The Received field defines the format of the unsolicited message expected from the device.
- Response: The Response field defines the format of the message to be sent in response to the device's unsolicited message that was received by OmniServer.

Both message fields also support direct character entry for ASCII alphanumeric characters.

7.1.1.1.5.3 Error Messages

An Error Message can also be defined as an alternative response to a Command/Response (Host) Message. It is checked against the data stream much like a Response Only (Unsolicited) Message, but whereas a Response Only message is always accepted, you can set the acceptance level of an error message.



Error Message List

	Message Name	Description	Action	Priority	Is Enabled	Notification Item	Message Count Item
	Balance_Overload		Retry	Normal	✓	None	None
	Balance_Underload		Retry	Normal	✓	None	None
	Execute_Not_Possible		Retry	Normal	✓	None	None

A new Error Message can be added by clicking the Add ➕ button in the toolbar.

Prior to defining the actual Received message, the following properties of an Error Message are defined:

- Message Name: The name of the message, which must be unique within the protocol.
- Description: A description of the message. This is for documentation purposes only and is not used by the server.
- Action: One of four actions to take after the error message has been activated. Those actions are:

  Retry: Retry the message normally. If the message has exhausted all of it's retry attempts, then the message will fail.

  Retry Forever: Retry the message without charging this retry against the messages normal retry count. Thus, the message will retry as long as this or another retry forever error message is received as a response.

  Accept: Accept the error message as a valid response to the Command/Response (Host) Message and continue on to the **On Succeed** message chain of the Command/Response (Host) Message (if it exists).

  Fail: Consider the error message as a failed response to the Command/Response (Host) Message. Continue on to the **On Failure** message chain of the Command/Response (Host) Message (if it exists).

  The default for this action is **Retry**.
- Priority: Process the message given a certain Priority. This means that this error message will be evaluated before or after any others. Error messages are processed after the response part of a Command/Request (Host) Message is evaluated, and before an Unsolicited Message is processed.

  The three different priorities are:

  Low: Process this error message after any other messages with priorities of Normal or High. If more than one message is marked Low, the messages are processed in the order of their creation.

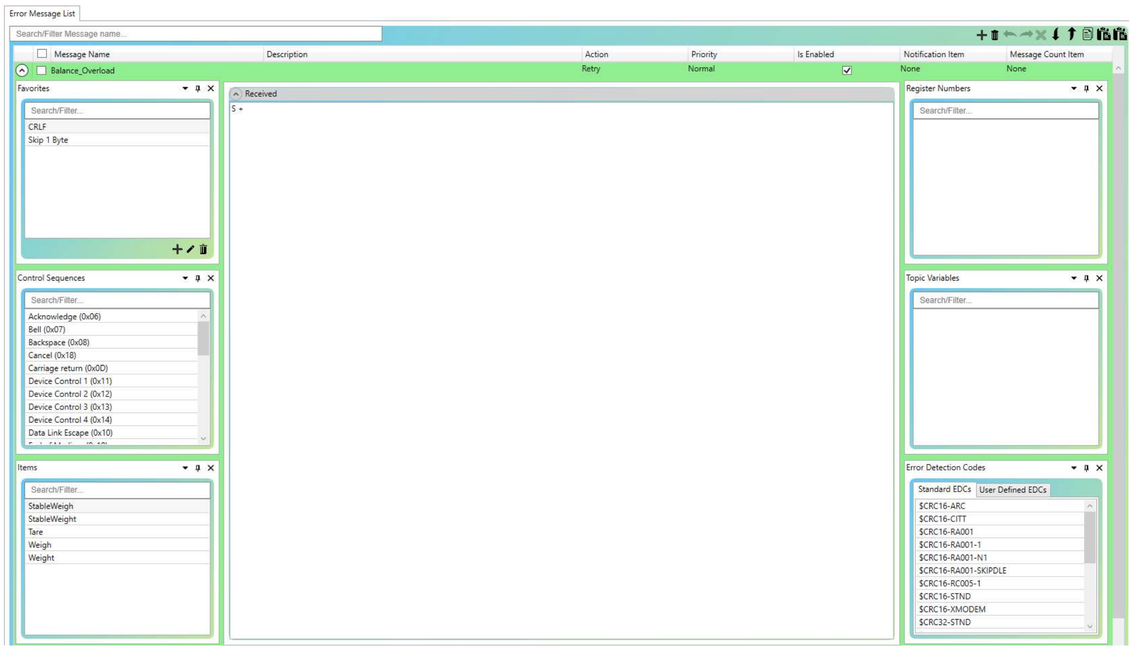
Normal: Process this error message after any messages marked as High, but before messages marked as Low. If more than one message is marked Normal, the messages are processed in the order of their creation.

High: Process this error message before any marked Normal or Low. If more than one message is marked High, the messages are processed in the order of their creation.

The default for priority is **Normal**.

- Is Enabled: The Error Message is processed when this box is checked, otherwise, the message is never processed. The default for this option is **checked**, meaning that the message is processed.
- Notification Item: The item selected in this column is set to One or True once the Command/Response (Host) Message has been activated AND the message successfully completes. If the message does not complete, then this item is set to Zero or False. This is very useful in determining if a particular error message is not working correctly. The default is **None**, meaning that there is no Notification Item associated with this message.
- Message Count Item: The item selected in this column is incremented whenever this message successfully completes.

Clicking the down arrow to the left of the new message expands to the actual Received section of the Error Message.



	Message Name	Description	Action	Priority	Is Enabled	Notification Item	Message Count Item
	Balance_Overload		Retry	Normal	✓	None	None

Received: S +

Favorites: CRLF, Skip 1 Byte

Control Sequences: Acknowledge (0x06), Bell (0x07), Backspace (0x08), Cancel (0x18), Carriage return (0x0D), Device Control 1 (0x11), Device Control 2 (0x12), Device Control 3 (0x13), Device Control 4 (0x14), Data Link Escape (0x10)

Items: StableWeigh, StableWeight, Tare, Weigh, Weight

Error Detection Codes: $CRC16-ARC, $CRC16-CITT, $CRC16-RA001, $CRC16-RA001-1, $CRC16-RA001-N1, $CRC16-RA001-SKIPDLE, $CRC16-RC005-1, $CRC16-STND, $CRC16-XMODEM, $CRC32-STND

Here you build the actual error messages to be received from the device. This section supporta drag-and-drop of the different Sequence Modules to the left and right.

- Received: The Received field defines the format of the error message expected from the device for a particular error state.

The Received message field also supports direct character entry for ASCII alphanumeric characters.

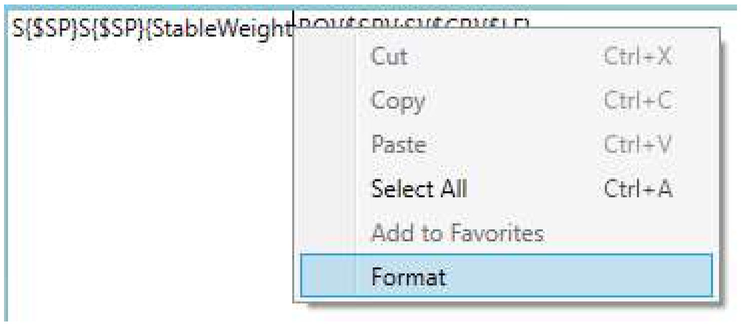
7.1.1.1.5.4 Sequence Formatting

How data sent to an device is built, or how it is parsed from the device is known as a sequence. This sequence describes the exact syntax of any messages channeling through the server. This sequence of characters is also known as a message.

To help you in the design of the message sequence, the Visual Protocol Editor supports drag-and-drop of many of the sequences you will need, including special/control characters, items, registers, topic variables, and Error Detection Codes.

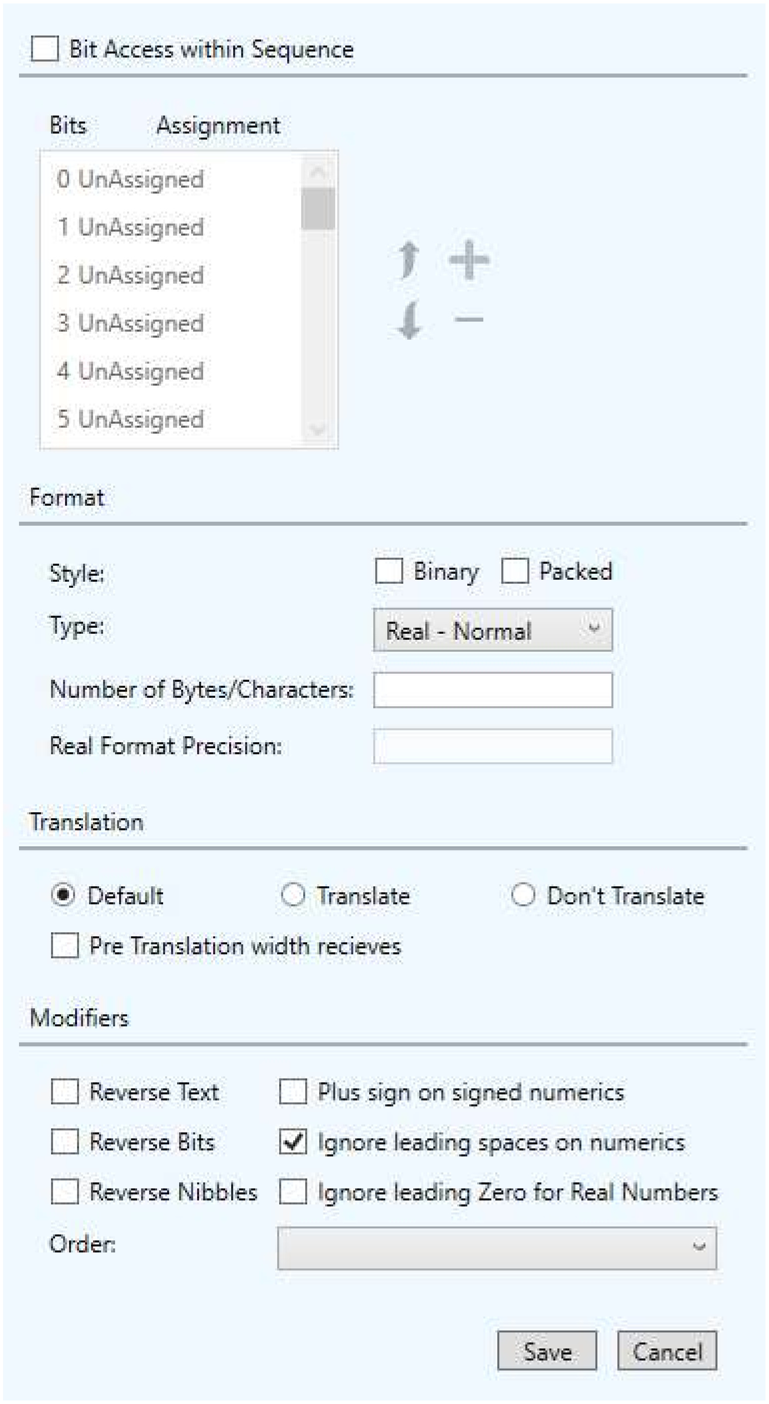
Beyond the sequences themselves, since there are different ways of formatting data, OmniServer also has what is known as a sequence formatter which is similar to the Sequence Builder that you may be familiar with in the Legacy Protocol Editor.

To access the sequence formatter, right-click the desired sequence you wish to format within the message box of any server message and select "Format", as shown below.



S{$SP}S{$SP}{StableWeight

The following menu will appear:



☐ Bit Access within Sequence

Bits	Assignment
0	UnAssigned
1	UnAssigned
2	UnAssigned
3	UnAssigned
4	UnAssigned
5	UnAssigned

Format

Style: ☐ Binary ☐ Packed

Type: Real - Normal

Number of Bytes/Characters:

Real Format Precision:

Translation

[✓ Default] Translate Don't Translate

☐ Pre Translation width recieves

Modifiers

☐ Reverse Text ☐ Plus sign on signed numerics

☐ Reverse Bits ☑ Ignore leading spaces on numerics

☐ Reverse Nibbles ☐ Ignore leading Zero for Real Numbers

Order:

The following formatting options for the currently selected sequence are available:

- Bit Access within Sequence: When enabled, you can assign individual bits in the sequence - this is useful for assigning bits within a byte to separate boolean items.

Just underneath the checkbox is the current selection for this sequence. The buttons to the right are the options available. They are:

- Up - Moves the currently selected assignment up by one position each time it is clicked.
- Down - Moves the currently selected assignment down by one position each time it is clicked.
- Plus/Add - Expands the number of bits assigned to a selection by one each time it is clicked. For example, if an item is assigned to Bit 0 and this button is clicked, the item would then be assigned to Bits 0-1.
- Minus/Remove - Decreases the number of bits assigned to a selection by one each time it is clicked. For example, if an item is assigned to Bits 0-1 and this button is clicked, the item would then be assigned only to Bit 0.

- Format Section: This informs the server how to interpret the data. By default, the data is interpreted in ASCII format, that is, data that can normally be viewed on a printer. This default can be changed in one of two ways:
  - Styles -
    - **Binary (Format Code: B):** Reads the data in Binary format. Binary reads in the data not as an ASCII letter or number, but as the actual internal numeric representation. For example, the letter "A" is read in ASCII format as "A", but in Binary format as the decimal number 65, which is the internal representation for the letter "A".
    - **Packed (Format Code: P):** Packing is a method of packing 4 characters into 3 bytes. Each byte of data (8) bits is shrunk to 6 bits by masking off the upper two bits. ASCII characters (uppercase) between 0x40 and 0x5F are mapped to 0x00 to 0x1F. ASCII characters (digits) between 0x20 and 0x3F retain their value. Thus, any characters 0x20 and below or 0x60 and above can not properly be represented using packing ASCII.
  - Type - You can also change the default format of the selected assignment by using the Type pull-down box. The default format is Default, meaning that the server will use the default. The list of supported formats are:
    - **Default (Format Code: None)**

      Uses the format specified by the assignment when created.

If a bit mask, or not assigned, or topic variable, or EDC, or a single number, default is "I" for ASCII and "U" for binary.

For items, it is based on the type. The server use "I" for Integer, "R" for Real, "S" for String, and "D" for Discrete.

For register numbers, the format used is "I"

- **Integer Signed (Format Code: I)**

  The data is treated as a Signed Integer. Based upon how the item is formatted, here is how a Signed Integer is parsed:

  For ASCII formats with a variable width: Allowed characters are "+-0123456789". All other are invalid

  For ASCII formats with a fixed width:: Skip leading spaces, +/- parsed, parse all digits. All other characters are invalid.

  For Binary formats, the bytes are treated as Signed, meaning that the most significant bit is the sign bit. So, for a one-byte item, the range of numbers is -127 to 127.

- **Integer Unsigned (Format Code: U)**

  Treats each number as an Unsigned Integer. Based upon how the item is formatted, here is how an Unsigned Integers parsed:

  For ASCII formats with a variable width: Allowed characters are "0123456789". All others are illegal.

  For ASCII formats with a fixed width: Skip leading spaces, +/- parsed, parse all digits. All other characters are illegal.

  For Binary formats, the bytes are treated as-is. So, for a one-byte item, the range of numbers is 0 to 255.

- **Integer Signed Hex (Format Code: Y)**

  Treats the data in Hexadecimal format, which is a base-16 counting system (Decimal format is a base-10 counting system). In Hexadecimal counting, the numbers 0 through 15 are represented by the numbers 0 through 9, then A, B, C, D, E, and F, where A=10, B=11, C=12, D=13, E=14, and F=15. Therefore, the decimal number 16 equals the hexadecimal number 10.

This format uses the most significant bit to determine the sign of the data.

For ASCII formats with a variable width: Allowed characters are "0123456789ABCDEFabcdef". All other characters are illegal. The value is sign extended based on number of digits processed.

For ASCII formats with a fixed width: Skip leading spaces, +/- parsed, parse all digits between 0-9, A-F, and a-f. Stop when an invalid character reached. Value is sign extended based on format width.

This format is ignored in Binary formats.

- **Integer Unsigned Hex (Format Code: X)**

  Treats the data in Hexadecimal format, which is a base-16 counting system (Decimal format is a base-10 counting system). In Hexadecimal counting, the numbers 0 through 15 are represented by the numbers 0 through 9, then A, B, C, D, E, and F, where A=10, B=11, C=12, D=13, E=14, and F=15. Therefore, the decimal number 16 equals the hexadecimal number 10.

  For ASCII formats with a variable width: Allowed characters are "0123456789ABCDEFabcdef". All other characters are illegal.

  For ASCII formats with a fixed width: Skip leading spaces, +/- parsed, parse all digits between 0-9, A-F, and a-f. Stop when an invalid character is reached.

  This format is ignored in Binary formats.

- **Integer BCD (Format Code: C)**

  Treats the data in Binary Coded Decimal (BCD) format. This format takes the two nibbles that makes up a byte and interprets the nibbles as a two-digit number. For example, the decimal number 65 returns the number "41" in BCD format (knowing that the hexadecimal representation of "65" is "41"). The decimal number 63 is ignored by the server, since "63" converts to "3F" in hexadecimal, and "F" is not a valid number.

  The equation for the calculation is "Value = Value * 100 + high nibble * 10 + low nibble".

In ASCII formats, only "0" through "9" are valid characters, all other cause the server to ignore the data. Also, you will normally need two bytes in the format, one for each nibble.

In Binary formats, any nibble value that contains "A" through "F" is rejected. This means that while "09" is valid, "0A" is not.

- **Real Normal (Format Code: R)**

  Treats the data as a Real (or Floating Point) number.

  For ASCII formats with a variable width: Allowed characters are "+-.0123456789". All others are illegal.

  For ASCII formats with a fixed width: Skip leading spaces, +/- parsed, parse all digits between 0-9. If a decimal point is found, then add that and all 0-9 digits following. If a "E" or "e" follows, then parse the +/- and the exponent digits. [+|-](0-9)[.(0-9)][e[+|-](0-9)]

  For Binary formats, the number is interpreted using IEEE floating point calculations. This must be four bytes in length, meaning that to insure the correct number, always use the format :4R.

- **Real Exponential (Format Code: E)**

  Treats the data as a Real (or Floating Point) number with scientific notation.

  For ASCII formats with a variable width: Allowed characters are "+-.0123456789Ee". All other characters are illegal.

  For ASCII formats with a fixed width: Skip leading spaces, +/- parsed, parse all digits between 0-9. If a decimal point is found, then add that and all 0-9 digits following. If a "E" or "e" follows, then parse the +/- and the exponent digits. [+|-](0-9)[.(0-9)][e[+|-](0-9)]

  This format is ignored in Binary formats.

- **Real Hex (Format Code: H)**

  Treats the data as a Real Hex number.

For ASCII formats with a variable width: This format is not allowed.

For ASCII formats with a fixed width: The 4 bytes of the IEEE number are treated as an integer and formatted as hexadecimal (Format: X)

For Binary formats: The 4 bytes of the IEEE number are treated as an integer and formatted as hexadecimal (Format: X).

- **Discrete Normal (Format Code: D)**

  Treats the data as a Discrete Value. The server will then break the assignment block into the individual bits so that you can assign items on a bit level, instead of a byte level.

  With the "D" format, all formats follow the same rules: Data is extracted as if "I" format. The resulting value is tested. If non-zero, then the value is set to TRUE (1). If zero, then the value is set to FALSE (0). When being formatted, it is the values of 1 or 0 are formatted as an "I" format.

- **String Normal (Format Code: S)**

  Treats the data as a series of characters. The String format is commonly used to discard or skip data.

  For ASCII formats with a variable width: The character following the sequence defines the end of the string. All characters are allowed. However, some clients might not be able to handle some character values (such as the "Null" (0x00) value).

  For ASCII formats with a fixed width: All characters are accepted.

  For Binary formats: Treated as ASCII

- **String Swap Word (Format Code: M)**

  Treats the incoming string as a series of two-byte data, and swaps those two bytes before processing it. The length of the data must be divisible by 2. This works the same as the String (S) format, except that every pair of bytes are swapped.

  For ASCII formats, both variable and fixed: All data is accepted.

  This format is ignored in Binary formats.

- **Unicode String (Format Code: W)**

  Treats the data as a series of Unicode characters (UTF-16). Unicode strings are only allowed in outgoing messages. By default, the characters in the Unicode string are output lest significant byte first. This can be changed to most significant byte first by using the byte ordering option of "O21".

  For ASCII formats with a variable width: All characters contained in the string are sent to the device.

  For ASCII formats with a fixed width: The width specifies the number of bytes to send which must be a multiple of two. If the length of the string is less than the width divided by two, then spaces are appended to the end of the string. If the length of the string is longer than the width divided by two, then the string is truncated.

  For Binary formats: Treated as ASCII

- **Other Linkage (Format Code: L)**

  The Linkage format is a special format that instructs the server to treat the item as if it was to read in data (thus triggering off the default operation on Read-Type Command/Request (Host) Messages), but not to expect nor parse any data for it. The primary use of the Linkage format is to have the server automatically send messages to the device (by using the default operation of the Read-Type Command/Request (Host) Message) without expecting data in return.

  For example, a heartbeat message can be sent to an device just to make sure that the server is still talking to it. Since no data is returned by the device, you use a Linkage format in the Response part of the Command/Request (Host) Message to trigger off the message. The server will then send out the message, but will not expect anything back into the assigned item.

  This format has no function in the ASCII nor Binary formats.

- Number of Bytes/Characters - The number of bytes. For data coming into the server from the device, this is the number of bytes to expect. The default is **Nothing**, meaning that the server will take all characters until a delimiter is found. This means that if you do not specify a width, the server will expect some hard-coded delimiter in the message stream after the assignment so that it will know when to stop putting characters into the selection. For this reason, you can never end a message stream with a selection that has not width (known as a **variable-length item**). For outputs, the server will send out just enough characters to fully express the item.

If there is a width, then the server will expect that many bytes from the device, or will send out that many bytes to the device. If the data is less than the number of bytes, the server will pad with spaces.

  - Real Format Precision - Applying to the Real Format only, this is the number of decimal places to expect from the device or the number to output to the device. The default is **Nothing**, meaning that a variable number of bytes is expected.

- Translation Section: This tells the server how to translate the data based upon the Translations set up within the protocol. The options are:
  - **Default** - Process translations according to the defaults set up in the Translations.
  - **Translate (Format Code: T+)** - Translate this data, according to the rules set up in the Translations.
  - **Don't Translate (Format Code: T-)** - Never translate this data.
  - **Pre Translation Width Receives** - If this checkbox is checked, then when counting the number of characters in the item, the pre-translation length is used. Otherwise, the post-translation length is use.

    The default is not checked, meaning that the server will use the post-translation length.

- Modifiers Section:
  - **Reverse Text (Format Code: V):** Reverses the text string. For example, if the data is "ABCD", then the server will reverse the data bytes and return "DCBA".

    This format cannot be used with the Binary or Linkage formats.
  - **Reverse Bits (Format Code: Z):** Reverses the bits within each byte. For example, if the bit pattern in a byte was "11000100", then the server will reverse these bytes and return "00100011".
  - **Reverse Nibbles (Format Code: N):** Reverses the nibbles within each byte. For example, if the bit pattern in a byte was "11000100", then the server will reverse these nibbles and return "00100011".

- **Ignore Leading Spaces on Numerics (Format Code: Q):** Ignores leading spaces on numeric items (Real and Integer) when using an ASCII variable-width format.
- **Order (Format Code: O):** Changes the order of bytes in the specified selection in the order spelled out in the text box. For example, suppose you had the string "ABCD", and you wanted the 2nd byte, then the 3rd byte, then the 1st, and finally the 4th byte. You will select the option "O2314" from the Order setting.

  This format is only available on 2, 3, or 4-byte strings.
- **Plus sign on signed numerics (Format Code: +):** Forces the server to output a plus sign for signed numerics. Used only when formatting data. When receiving data, this format is ignored.

  This format is only available with the I (Integer), R (Real), and E (Real-Exponential) formats.

  The default for this is not checked.
- **Ignore leading zero for real numbers (Format Code: ^):** Ignores leading zeroes on Real data types - only available when the sequence is formatted as a Real.

  The default for this is not checked.

7.1.1.1.5.5 Sequence Modules

The OmniServer Visual Protocol Editor makes it easier to build protocol messages by supporting drag-and-drop functionality of sequences to your messages. This is accomplished by providing a modular interface with a module for each type of sequence.

The following sequence modules are available (click on each option for further details:

- Favorites
- Control Sequences
- Items
- Register Numbers
- Topic Variables
- Error Detection Codes

It is also possible to only display the sequence modules that you choose - the reasoning behind this is that you shouldn't need to display Topic Variables and Register Numbers if you never plan to use them when building your messages.

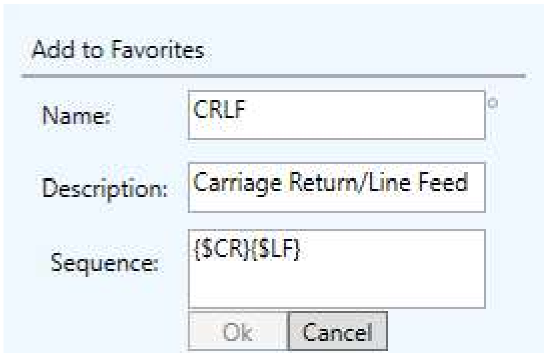
For full details on customizing your layout, click here.

The Favorites module allows you to define frequently used sequences to save you time when configuring your protocol messages.

OmniServer includes two pre-defined Favorites for you:

- **CRLF** - This Favorite sequence adds a Carriage Return and Line Feed when you drop it into a message.
- **Skip 1 Byte** - This Favorite sequence adds the sequence to skip/ignore a single byte - {:1S} - to ignore more than 1 byte, simply increase the numeric portion of the sequence.

To add your own Favorites, click the +/Add button, which opens the following:


Add to Favorites

Name: CRLF

Description: Carriage Return/Line Feed

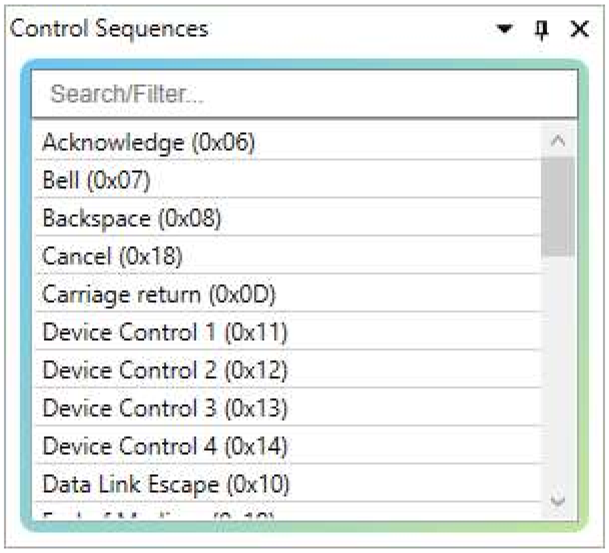
Sequence: {$CR}{$LF}

- **Name** - This is the display name of the Favorite in the module.
- **Description** - Provide an optional description of what the Favorite is/does.
- **Sequence** - This is the actual sequence that will be inserted into your message.

Any existing Favorite can be edited/updated by highlighting the desired Favorite in the list and clicking the Pencil/Edit button.

Any existing Favorite can be deleted by highlighting the desired Favorite in the list and clicking the Trash Can/Delete button.

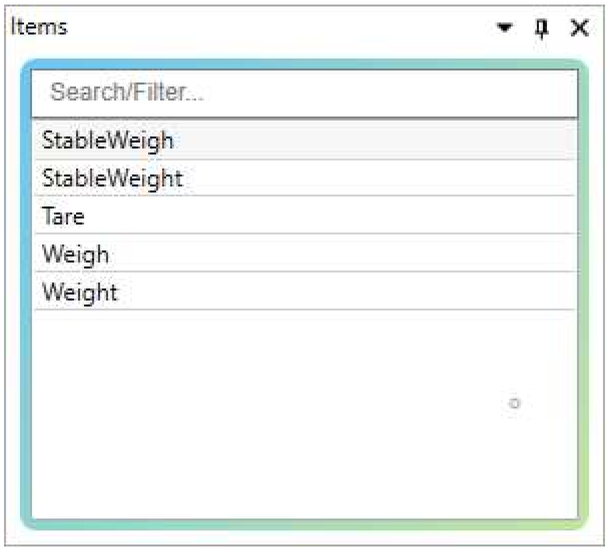
The Control Sequences module displays all reserved, pre-defined control characters available for use in a message sequence and can be easily dragged into any message.



Acknowledge (0x06)
Bell (0x07)
Backspace (0x08)
Cancel (0x18)
Carriage return (0x0D)
Device Control 1 (0x11)
Device Control 2 (0x12)
Device Control 3 (0x13)
Device Control 4 (0x14)
Data Link Escape (0x10)

A Search/Filter option is provided to easily search for a specific control character that you're looking for.

The Items Module displays all the items defined in the protocol and allows for easy drag-and-drop to any message.

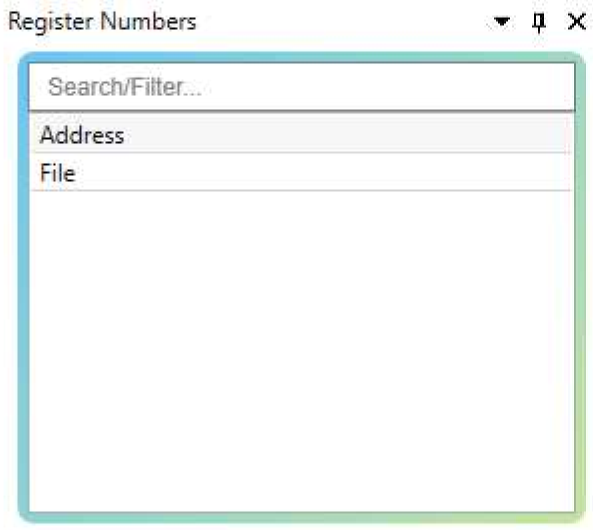


StableWeigh
StableWeight
Tare
Weigh
Weight

A Search/Filter option is provided to easily search for a specific item that you're looking for, which is useful in protocols with a large number of items.

Items are defined within the protocol here.

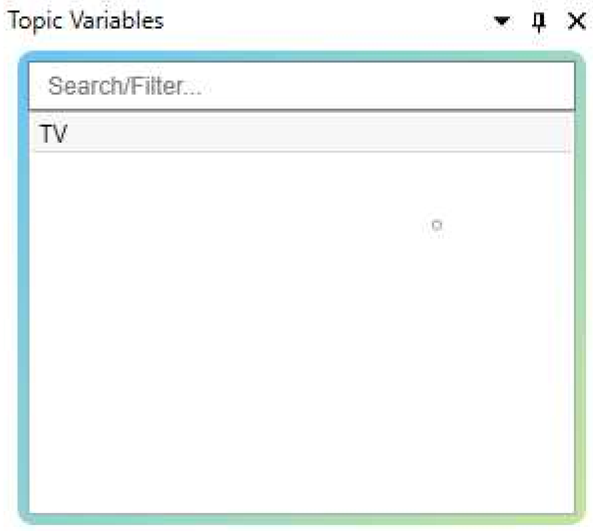
The Register Numbers Module displays all the Registers defined in the protocol and allows for easy drag-and-drop to any message.



A Search/Filter option is provided to easily search for a specific register that you're looking for, which is useful in protocols with a large number of registers.

Register Numbers are defined within the protocol here.

The Topic Variables Module displays all the Topic Variables defined in the protocol and allows for easy drag-and-drop to any message.

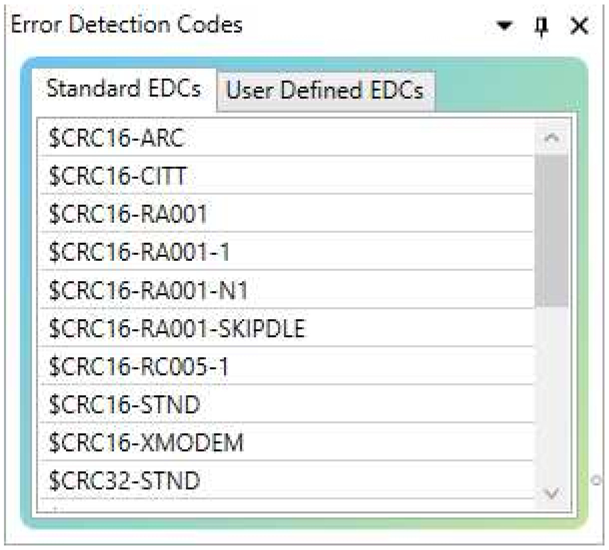


A Search/Filter option is provided to easily search for a specific topic variable that you're looking for, which is useful in protocols with a large number of topic variables.

Topic Variables are defined within the protocol here.

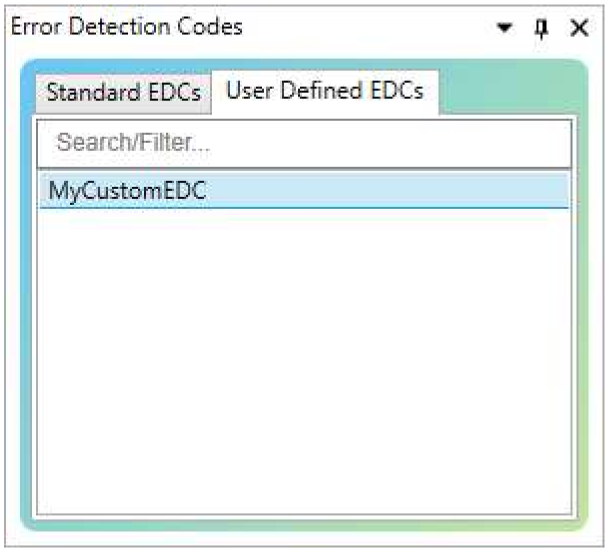
The Error Detection Codes module displays all of the EDCs available to the protocol and can be dropped into any message, including any user-defined EDCs and pre-defined EDCs.

Standard EDCs are defined by the server and can be located here.



$CRC16-ARC
$CRC16-CITT
$CRC16-RA001
$CRC16-RA001-1
$CRC16-RA001-N1
$CRC16-RA001-SKIPDLE
$CRC16-RC005-1
$CRC16-STND
$CRC16-XMODEM
$CRC32-STND

User-defined EDCs are defined within the protocol here.



MyCustomEDC

Error Detection Codes

When sending a message to a device, the server formats the message, replacing sequences with their formatted values. It then computes the error detection code from those characters specified between brackets in the message. The server then removes the brackets and replaces the error detection sequence with the formatted computed value.

When processing a message from a device, the server computes the error detection code. If the computed code matches the received code, the server accepts the message and processes it. If the codes do not match, the server ignores the message.

Rejection of a message can have an impact on performance unless the protocol handles the received message elsewhere. For Command/Response (Host) Messages, use an error message that mimics the expected response. For example, the error message can be designed like the response message - except that there are no brackets and a discarded item replaces the error detection sequence. The width of the string sequence must be the same as that of the error detection sequence. This will match the device's message with the incorrect code. Set the error message's action to either retry or accept. Similarly, the protocol can handle a failed error detection code in an unsolicited message with another unsolicited message of a lower priority.

Occasionally, you may implement a protocol that uses error detection but use it in an environment that does not cause interference in the media. You can use the same technique described above to process the messages. Simply use a discarded string item in place of the error detection code. This works only when you receive data from a device; you may still need to provide an error code for messages sent to the device.

In addition to user-defined EDCs (which you can read about here), the server supplies a number of pre-defined EDCs that are most commonly used:

- Cyclic Redundancy Check (CRC)
- Checksums
- Longitudinal Redundancy Check (LRC, XOR, BCC)
- Others

The checksum uses the sum of the values from the message to produce the error code. This is the simplest of all error detection codes. The standard checksum uses the binary value of each character of the message in computing its value. The following example shows the computation of the simple checksum on the message "01":

ASCII:	0	+	1	=	a
Decimal:	48	+	49	=	97
Hexadecimal:	30	+	31	=	61

List of Pre-Installed Checksums:

⊟ CSUM7

Standard 7-bit checksum. The most significant bit of each byte is ignored.

⊟ CSUM8

Standard 8-bit checksum.

⊟ CSUM8-DEC

Standard 8-bit checksum, decimal input. Each two-byte sequence represents a decimal number between 00 and 99.

⊟ CSUM8-HEX

Standard 8-bit checksum, hexadecimal input. Each two-byte sequence represents a hexadecimal number between 00 and FF.

⊟ CSUM8-TC

Two's compliment 8-bit checksum. The normal checksum is calculated. The checksum is then "complimented", that is, all the bits of the number are reversed (0 to 1 and 1 to 0). Finally, 1 is added to the number to give the final result.

For example, suppose a checksum of hexadecimal "94" is generated. Here is the calculation used to produce a two's compliment number:

Action	Hex	Binary
Starting #:	94	1001 0100
Compliment:	6B	0110 1011
Add One:	6C	0110 1100

⊟ CSUM8-TC16

16-bit checksum added to 8 bits, with a two's compliment. This is the same as CSUM8-TC above, but with a sixteen-bit number instead of an eight-bit number.

The cyclic redundancy check (CRC) produces the error code by considering the entire message as the coefficients to a base-2 polynomial, dividing it by a generating polynomial, and using the remainder as the error code.

List of Pre-Installed CRCs:

⊟ CRC16-ARC

- 16-bit Reflected CRC
- Polynomial: 800A
- Initial Value: 0

⊟ CRC16-CITT

- 16-bit CCITTCRC CRC
- Polynomial: 1021
- Initial Value: -1 (0xFFFF)

⊟ CRC16-RA001

- 16-bit Reflected CRC
- Polynomial: A001
- Initial Value: 0

⊟ CRC16-RA001-1

- 16-bit Reflected CRC
- Polynomial: A001
- Initial Value: 1

⊟ CRC16-RA001-N1

- 16-bit Reflected CRC
- Polynomial: A001
- Initial Value: -1 (0xFFFF)

⊟ CRC16-RA001-SKIPDLE

- 16-bit Reflected CRC
- Skips DLEs
- Polynomial: A001
- Initial Value: 0

⊟ CRC16-RC005

- 16-bit Reflected CRC
- Polynomial: C005
- Initial Value: 1

⊟ CRC16-STND

- 16-bit Standard CRC
- Polynomial: 8005
- Initial Value: 0

⊟ CRC16-XMODEM

- 16-bit Reflected CRC, XMODEM
- Polynomial: 8408
- Initial Value: 1

⊟ CRC32-STND

- 32-bit Reflected CRC
- Polynomial: 04C11DB7
- Initial Value: 0

Standard Implementation

In the standard implementation of the CRC algorithm, the CRC is computed by processing the incoming data most significant bit to least significant bit.

The most straightforward implementation is to loop eight times for each value added to the accumulator. The high-order bit is noted and the entire accumulator is shifted up by 1 bit. If the high-order bit was set, then the generating polynomial, less the x^16 term, is exclusive or'ed into the accumulator. This exclusive or'ing of the generating polynomial determines a remainder polynomial. The bit test version of the CRC implementation is as follows:

```
ErrorCode := seedValue
for each character c in the message
  ErrorCode := ErrorCode xor c
  do 8 times
    bit :=most significant bit of ErrorCode
    ErrorCode := up shift of ErrorCode
    if bit is on
      ErrorCode := ErrorCode xor
generatingPolynomial
    end if
  end do
end for
```

Some implementations recognize that the value of the high-order byte solely determines how the generating polynomial contributes to the accumulator. These implementations use a table lookup instead. The table lookup version of the CRC implementation is as follows:

```
ErrorCode:= seedValue
for each character c in the message
      IndexByte := high-order byte of ErrorCode
shifted into the low-order byte xor c
```

```
    ErrorCode := shift of low-order byte into high-order byte
    ErrorCode := ErrorCode xor TableLookup [IndexByte]
end for
```

Another method of computing the CRC is inspired by the hardware implementation:

```
CRCLowByte := 255
CRCHighByte := 255
for each character c in the string
    a := c xor CRCLowByte
    b := a * 8
    c := a * 16
    d := a * 128
    e := a and hexadecimal F
    f := (a / 16) and hexadecimal F
    g := (a / 32) and hexadecimal F
    h := (e / 2) and hexadecimal F
    CRCLowByte = CRCHighByte xor b xor f xor e xor d
    CRCHighByte = a xor g xor c xor h
end for
CRCHighByte := ones complement of CRCHighByte
CRCLowByte := ones complement of CRCLowByte
```

In almost all cases, the server uses the table lookup version to compute CRCs since it even outperforms the hardware inspired implementations. Even in cases where we receive a hardware inspired implementation for addition into the server, we extract the polynomial and seed values from the algorithm and convert the CRC to a table lookup.

Reflected Implementation

In the reflected implementation of the CRC algorithm, the CRC is computed by processing the incoming data least significant bit to most significant bit.

The difference between the standard and reflected implementation is that the polynomial is reversed, the accumulator is shifted down, and the least significant bit of the accumulator is inspected.

Generating Polynomials

As stated above, the generating polynomial is exclusive or'ed into the accumulator. The generating polynomial for a 16-bit CRC has 17 coefficients, for x^0 through x^16. However, the bit pattern representing the polynomial has only 16 coefficients. This is because the coefficient for x^16 is assumed instead of explicitly represented. This is acceptable since it, exclusive or'ed with the bit shifted out of the accumulator, cancels and contributes no more to the result.

In the case of the standard implementation, x^15 represents bit 15 and x^0 represents bit 0. For reflected implementations, this is reversed with x^15 representing bit 0 and x^0 representing bit 15.

For example:

Polynomial x^16 + x^15 + x^2 + 1
Standard Representation 1000 0000 0000 0101 (8005)
Reflected Representation 1010 0000 0000 0001 (A001)

Server Details : Configuration : Protocol

Error Detection Codes : LRC

The longitudinal redundancy check (LRC) uses the exclusive or (XOR) of the values from the message to produce the error code. The standard LRC uses the binary value of each character of the message in computing its value. The following example shows the computation of the simple LRC on the message "01":

```
ASCII:           0 XOR        1 = CTRL-A, SOH
Binary:   00110000 XOR 00110001 = 00000001
```

Hexadecimal: 30 XOR 31 = 01

Note: LRCs are commonly referred to as block check codes (BCC).

List of Pre-Installed LRCs:

⊟ LRC8

Standard 8-bit LRC.

⊟ LRC8-OCCOMBO

8-bit LRC combined with a one's compliment checksum.

⊟ LRC8-X30CRC

8-bit LRC encoded into two 0x30-biased bytes.

Server Details : Configuration : Protocol

Error Detection Codes : Others

The server supplies other special-case EDCs. They normally do not follow standard EDC calculations, but are used much like EDCs within the message.

List of Other EDCs:

⊟ KLEN

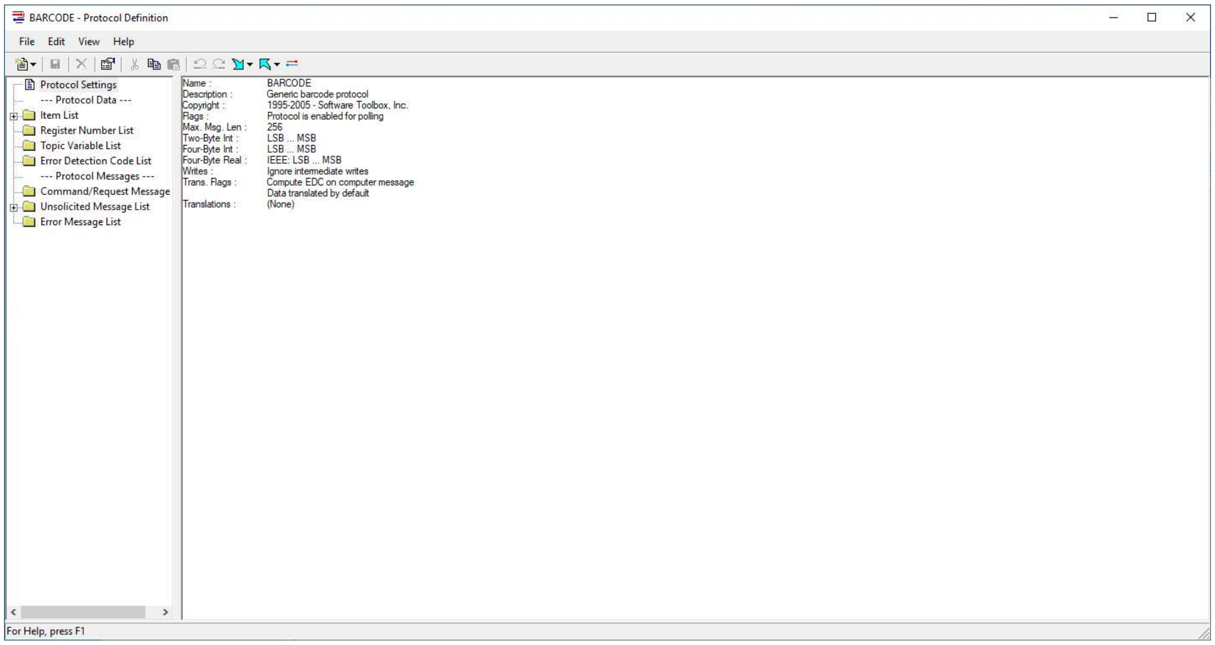
The length is the number of characters in the portion of the message that generates this code. Though this is not a true error detection code, it is useful in protocols that use counted, variable-length strings. KLEN returns just the length or byte count, so the KLEN of the message ABCDE is 5.

⊟ KLENA

The length is the number of characters in the portion of the message that generates this code plus the size of its own field. Though this is not a true error detection code, it is useful in protocols that use counted, variable-length strings. The KLENA of the message ABCDE with KLENA formatted as a 2X is 7.

7.1.1.2 Legacy Protocol Editor

This is the original Omniserver protocol editor that has been in use for over 20 years. The editor is still available as an alternative to the Visual Protocol Editor for existing users who are more comfortable with the Legacy look-and-feel.



```
Name :              BARCODE
Description :       Generic barcode protocol
Copyright :         1995-2005 - Software Toolbox, Inc.
Flags :             Protocol is enabled for polling
Max. Msg. Len :     256
Two-Byte Int :      LSB ... MSB
Four-Byte Int :     LSB ... MSB
Four-Byte Real :    IEEE: LSB ... MSB
Writes :            Ignore intermediate writes
Trans. Flags :      Compute EDC on computer message
                    Data translated by default
Translations :      (None)
```

7.1.1.2.1 The Menu Bar

You can access all the functions available in creating or maintaining your protocol through the Menu Bar.

The menu bar looks like this:

The Menu Bar

Placing your mouse pointer over one of the words and clicking the left mouse button will reveal a "pull-down" menu, which gives you an even greater choice in options.

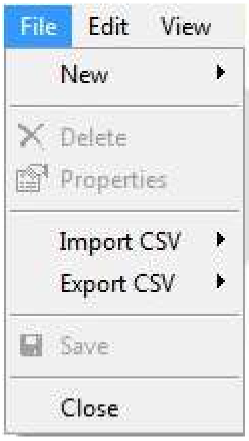
To discover what each of these menu selections do, click on the links below.

- The FILE Option
- The EDIT Option
- The VIEW Option
- The HELP Option

7.1.1.2.1.1 File

The first menu option available in a protocol is **File**. Here you will find the majority of the options associated with the management of protocol objects.

Some options, like "Delete" and "Properties" to the left, are grayed out and unavailable whenever that particular option cannot be selected due to the current server state, such as a component not being selected for deletion. For example, when an item is selected, the "Delete" and "Properties" options become available.



Sample | Menu

This selection is always active.

- **File | New** - Selecting this option creates protocol objects. A secondary menu appears and allows for the selection of the desired object:
    - Item
    - Register Number
    - Topic Variable
    - Error Detection Code
    - Command/Request (Host) Message
    - Unsolicited Message
    - Error Message
- **File | Delete** - Selecting this option deletes the currently selected protocol object and is only available when a protocol object is currently selected.
- **File | Properties** - Selecting this option opens the properties for the currently selected protocol object and is only available when a protocol object is currently selected.
- **File | Import CSV** - Selecting this option allows import of certain protocol objects from a properly formatted CSV file. A secondary menu appears and allows for the selection of the desired object to be imported from CSV, after which a Windows browser allows selection of the CSV file to be used for import:
    - Items - Item List must currently be selected/highlighted in the protocol for this import option to be available.
    - Register Numbers - Register Number List must currently be selected/highlighted in the protocol for this import option to be available.
    - Topic Variables - Topic Variable List must currently be selected/highlighted in the protocol for this import option to be available.
- **File |Export CSV** - Selecting this option allows export of certain protocol objects to a CSV file, which can then be edited and imported into the same or different OmniServer protocol. A secondary menu appears and allows for the selection of the desired

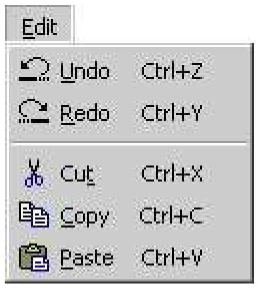
object to be exported to CSV, after which a dialog allows specifying the filename and location to save the CSV file:

- Items - Item List must currently be selected/highlighted in the protocol for this export option to be available.
- Register Numbers - Register Number List must currently be selected/highlighted in the protocol for this export option to be available.
- Topic Variables - Topic Variable List must currently be selected/highlighted in the protocol for this export option to be available.

- **File | Save** - Selecting this option saves the current changes to a protocol. This option will be unavailable if no changes have been made to the protocol.
- **File | Close** - Selecting this option closes the open protocol (if any changes have been made to the protocol, a prompt to save will be displayed).

7.1.1.2.1.2 Edit

The second menu option available on the server is **Edit**. Here you will find the majority of the options associated with the management of protocol objects.



For Help on any of these menu options, place your mouse over the options and click.

Undo the previous cut/copy/paste/edit operation. This option is only available once at least one change has been made in any server object.

"Undo" the Undo. Reserves the previous Undo operation. This option is only available when an Undo has been executed.

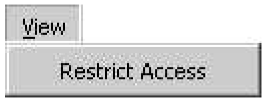
Deletes the currently selected server object and places it in the internal clipboard. This option is only available if a server object has been selected.

Copies a server object into the internal clipboard. This option is only available if there is a selected server object.

Insert any server object in the internal clipboard to the protocol. This option is only available if there has been an object placed in the internal clipboard via the Cut or Copy commands.

7.1.1.2.1.3 View

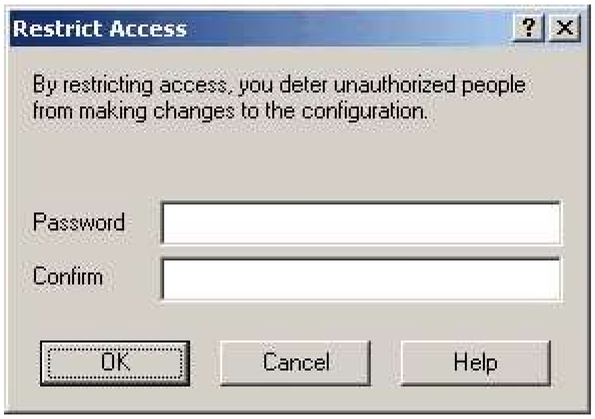
The third menu option available on the server is **View**. The View Menu option defines different options that can affect how the protocol is displayed or edited.



For Help on any of these menu options, place your mouse over the options and click

The Restrict Access option allows you to restrict the access of this protocol to anyone who does not know a password set up previously.

By default, the protocol is open and available to all for changes. To restrict the access, select this option. You will then be given this screen:


Restrict Access

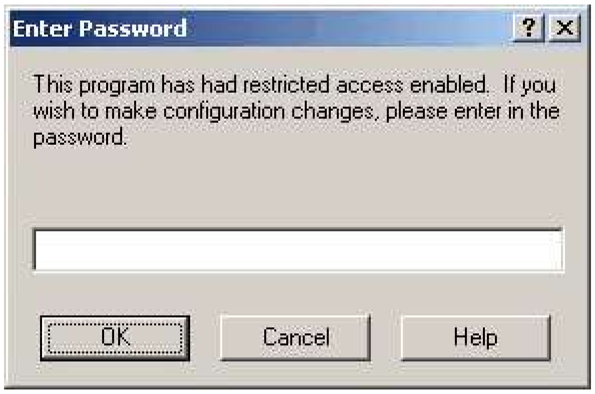
By restricting access, you deter unauthorized people from making changes to the configuration.

Password

Confirm

Set up a New Restriction

Type in a password and a confirmation password, then click OK. Once you save and close the protocol, any one attempting to access it will get this dialog box:


Enter Password

This program has had restricted access enabled. If you wish to make configuration changes, please enter in the password.

Confirming Access

No one will be able to change anything in the protocol until the correct password is entered.

To remove the restriction, select **View | Restrict Access** again, type in the confirming password, and you will receive a message that the restriction has been taken off.

7.1.1.2.1.4 Help

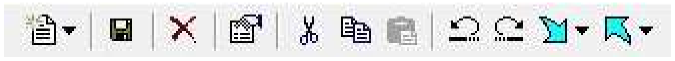
This is the Help File Menu for Protocol Definitions.

This option will open up this Help File.

For Help on any of these menu options, place your mouse over the options and click.

7.1.1.2.2 Protocol Toolbar

You can access all the functions available in creating or maintaining your protocol through the Tool Bar.


The Protocol Tool Bar

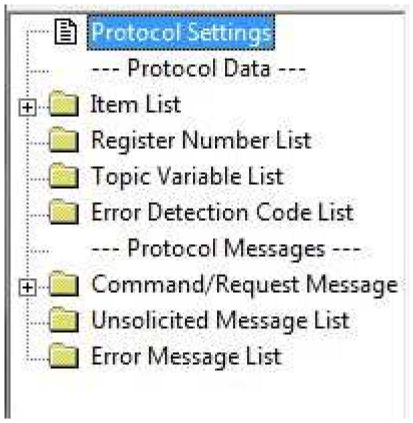
There are 11 tool bar selections:

	New	Create a new server object. This option will always be available.
	Save	Save the current protocol. This option is always available once at least one object has been changed in any way.
✕	Delete	Delete the current object. This option is only available if an object has been selected.
	Properties	Display the properties for the current object. This option is only available if an object has been selected. Clicking this option has the same effect as double-clicking on the object itself.
✂	Cut	Copy an object to the internal clipboard and delete the object. This option is only available if an object has been selected.
	Copy	Copy an object to the internal clipboard. This option is only available if an object has been selected.
	Paste	Copy an object from the internal clipboard to the protocol. This option is only available if an object has first been placed into the internal clipboard, either by the Cut or Copy operations.
	Undo	Erases the last operation performed by the user and restores the previous condition of the protocol. This option is only available if the protocol has changed in some way.
	Redo	"Undo" the last undo. Reverses the previous Undo operation. This option is only available if an Undo operation has been performed.
	Import	Allows import of certain protocol objects from a properly formatted CSV file. A secondary menu appears and allows for the selection of the desired object to be imported from CSV, after which a Windows browser allows selection of the CSV file to be used for import: • Items

		• Register Numbers • Topic Variables
	Export	Allows export of certain protocol objects to a CSV file. A secondary menu appears and allows for the selection of the desired object to be exported to CSV, after which a dialog allows specifying the filename and location to save the CSV file: • Items • Register Numbers • Topic Variables

7.1.1.2.3 Protocol Settings

Protocol Settings are used to define basic properties global to your protocol, such as name, how binary data is to be interpreted, and how to translate data.



- Protocol Settings
- --- Protocol Data ---
- Item List
- Register Number List
- Topic Variable List
- Error Detection Code List
- --- Protocol Messages ---
- Command/Request Message
- Unsolicited Message List
- Error Message List

The Protocol Settings Tree

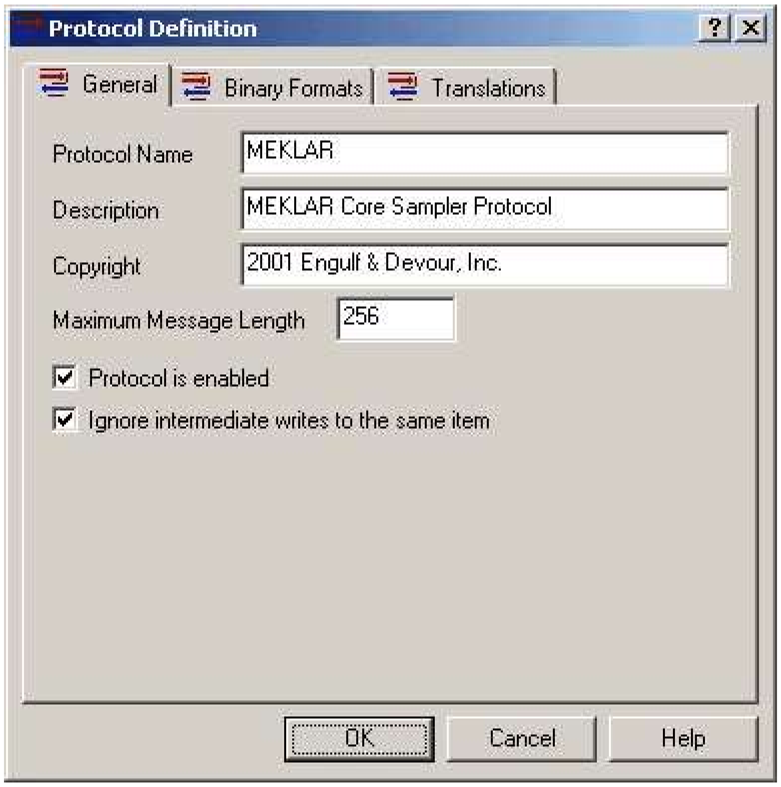
When you double-click on "Protocol Settings", you will see a dialog box. There are three property sheets with which to define a protocol. They are:

- General: Defines the default operations and properties of the protocol.
- Binary Formats: Defines the default byte-ordering of two-byte integers, four-byte integers, and IEEE floating point real numbers.

- Translations: Defines how to server will translate incoming or outgoing data.

7.1.1.2.3.1 General

The General Tab defines both identification and action properties of the protocol.



Protocol Name: MEKLAR
Description: MEKLAR Core Sampler Protocol
Copyright: 2001 Engulf & Devour, Inc.
Maximum Message Length: 256
[✓] Protocol is enabled
[✓] Ignore intermediate writes to the same item

The General Tab of a Protocol

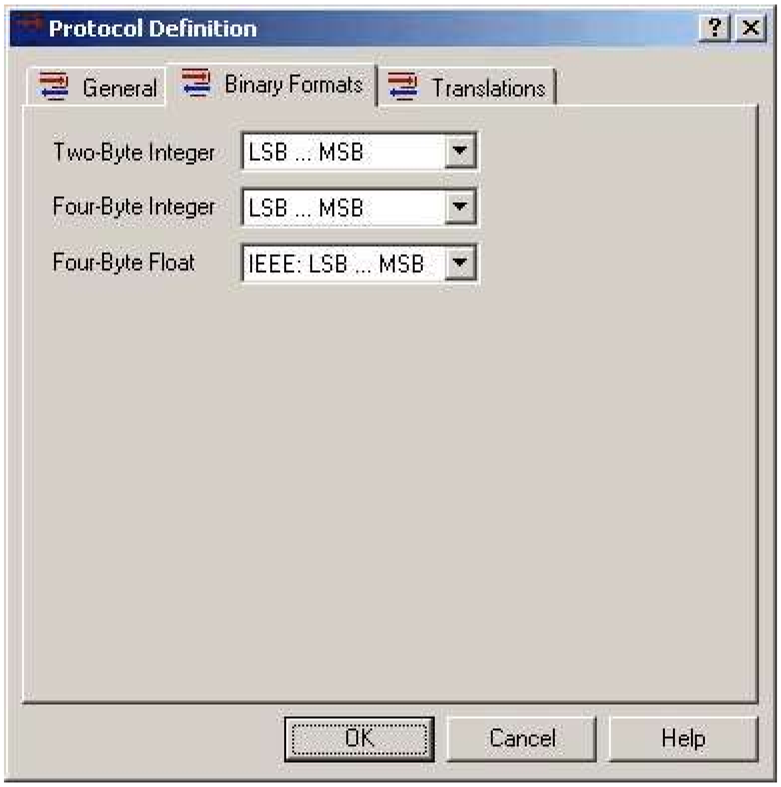
The options available are:

- **Protocol Name:** The name of the protocol, which must be unique within the server.
- **Description:** A description of the protocol. This is for documentation purposes only and is not used by the server.
- **Copyright:** Copyright information. All protocols are the property of the creators and not the manufacturer or licenser of the server. This is for documentation purposes only and is not used by the server.

- **Maximum Message Length:** The maximum number of bytes the server will read in and process from the device. This number can be anywhere between 1 and 32767.
- **Protocol is enabled:** Clearing this option will make the protocol unavailable to the server's runtime program, thus not transferring any data to the client. The default is **checked**, meaning that the protocol is available to the server.
- **Ignore intermediate writes to the same item:** When not checked, the server will send out all data to the device, even when that data repeats itself. The default is **checked**, meaning that the server will not send out the same write message twice, thus controlling the amount of unnecessary messaging to the device, or "spam".

7.1.1.2.3.2 Binary Formats

The Binary Formats Tab defines the default order for the interpretation of two-byte integers, four-byte integers, and IEEE floating point real numbers.



Two-Byte Integer: LSB ... MSB
Four-Byte Integer: LSB ... MSB
Four-Byte Float: IEEE: LSB ... MSB

The Binary Formats Tab

The options available are:

- **Two-byte Integer:** Sets the processing of a two-byte integer. There are only two options:
  LSB...MSB: Also known as the "Little Endean" or "Intel" format, the second byte of the data stream is multiplied by 256 then added to the first byte in the data stream. This is the default.
  MSB...LSB: Also known as the "Big Endean" format, the first byte in the data stream is multiplied by 256 then added to the second byte in the data stream.

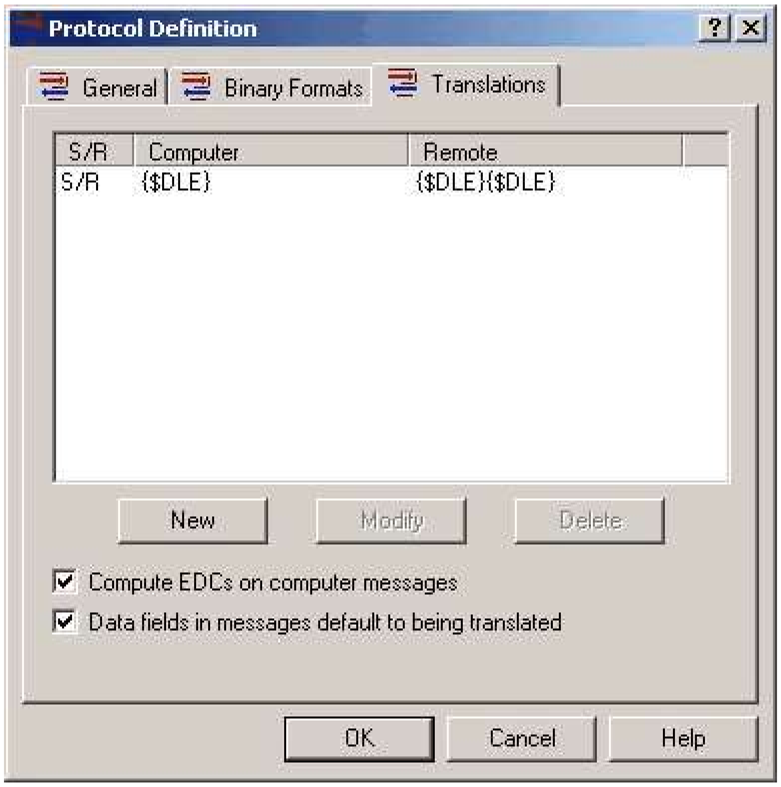
- **Four-byte Integer:** Sets the processing of a four-byte integer. There are only two options:
  LSB...MSB: Also known as the "Little Endean" or "Intel" format, the fourth byte in the data stream is multiplied by 16,777,216, the third byte in the data stream is multiplied by 65,536, the second byte in the data stream is multiplied by 256, and all these added together along with the first byte in to form the data stream the final number. This is the default.
  MSB...LSB: Also known as the "Big Endean" format, the first byte in the data stream is multiplied by 16,777,216, the second byte in the data stream is multiplied by 65,536, the third byte in the data stream is multiplied by 256, and all these added together along with the fourth byte in the data stream to form the final number.

- **Four-byte Float:** Sets the processing of a four-byte IEEE floating point integer. There are only two options:
  LSB...MSB: Also known as the "Little Endean" or "Intel" format, the IEEE floating point formula is applied to the data with the fourth byte of the data stream as the most significant byte (i.e., in reference to the data stream, 4th byte - 3rd byte - 2nd byte - 1st byte). This is the default.
  MSB...LSB: Also known as the "Big Endean" format, the IEEE floating point formula is applied to the data with the first byte of the data stream as the most significant byte (i.e. in reference to the data stream: 1st byte - 2nd byte - 3rd byte - 4th byte).

7.1.1.2.3.3 Translations

The Translations Tab defines how the server will translate incoming or outgoing data.

For a video tutorial on working with Translations, click here.



S/R	Computer	Remote
S/R	{$DLE}	{$DLE}{$DLE}

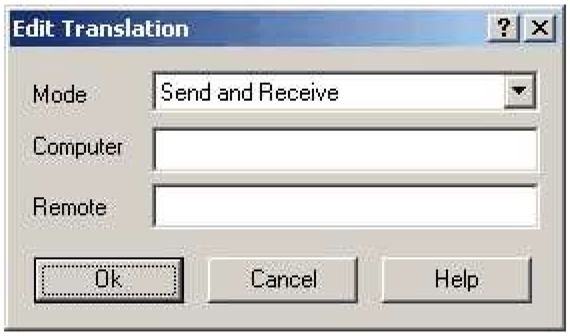
☑ Compute EDCs on computer messages

☑ Data fields in messages default to being translated

The Translations Tab

The options available are:

- **New:** Clicking on this button will create a new translations. This button is always active. When you click on the New button, you will get this dialog box:



Mode: Send and Receive

Computer:

Remote:

The New Translations Dialog Box

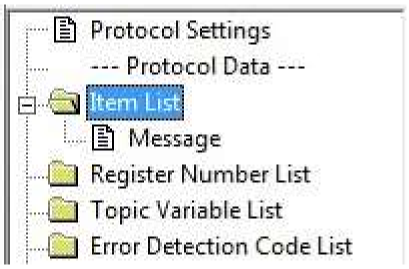
The options for this box are:

- **Mode:** How the translation is applied to the message. "Send Only" will apply the translation to messages going from the server (here called the "computer") to the device (here called the "remote"). "Receive Only" will apply the translations to message coming in from the device (the "remote"). "Send and Receive" will apply the translation to both messages.

- **Computer:** The text of the message as it will appear to the server. This is the pre-translation message on Send messages, and the post-translation message on Receive messages.
- **Remote:** The text of the message as it will appear to the device. This is the pre-translation message on Receive messages, and the post-translation message on Send messages.
- **Modify:** Allows you to modify a translation. This option is only active if a translation has been selected.
- **Delete:** Deletes a translation. This option is only active if a translation has been selected.
- **Compute EDCs on computer messages:** The server will compute all EDCs based upon the computer message, and not the message from the device. So for messages coming in from the device, the server will calculate EDCs based upon the message **after** translation has been done. For message going out to the device, EDCs are computed **before** the message is translated. The default is **checked**, which means that the EDCs are computed on the computer message.
- **Data fields in message default to being translated:** Clearing this field will instruct the server not to translate the data within messages unless that data is specifically formatted to translate (using the **Translate (T+)** format). The default is **checked,** which means that all data will be translated. This can be over-ridden on individual data items by using the **Do Not Translate (T-)** format.

7.1.1.2.4 Item

An **Item** is defined as the data that connects the client application with the server. The client will pass data to the server for further processing, and in turn, the server will parse data and send it back to the client through the item.



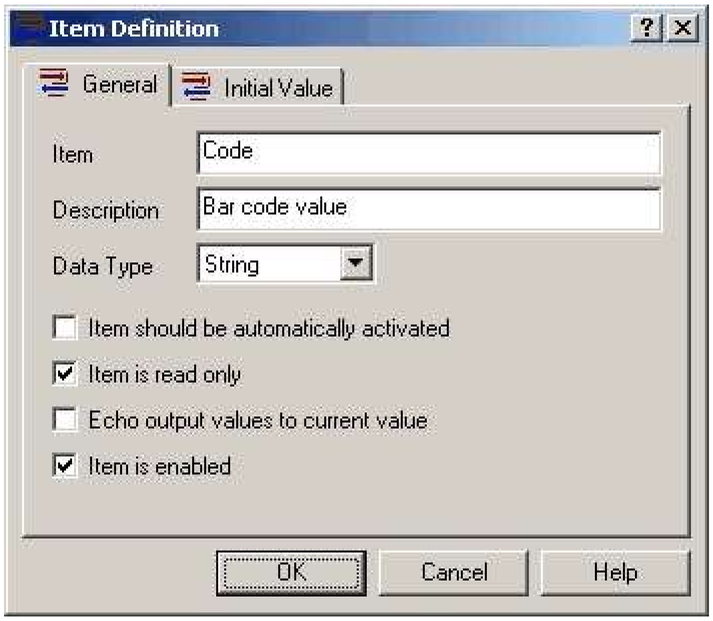
The Item Tree View

You create a new item by selecting **File | New | Item** from the menu, by clicking on the **New** icon in the toolbar, or by **right-clicking** inside the Tree List above and selecting **New**.

There are two property pages used with Items. Click on the links below for information on each:

Items Property Page - The General Tab

The General Tab contains the information needed by the server to uniquely identify the item.



Item: Code

Description: Bar code value

Data Type: String

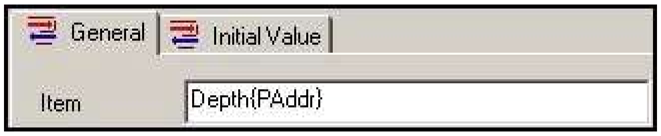
- [ ] Item should be automatically activated
- [✓] Item is read only
- [ ] Echo output values to current value
- [✓] Item is enabled

The General Tab

The different options are:

- Item: The Item name. This name must be unique within the protocol. It will be used to identify the data connection between the client and the server.

If you also use Register Numbers, then you can also specify a register number with name. This is how that will look:



Item Depth{PAddr}

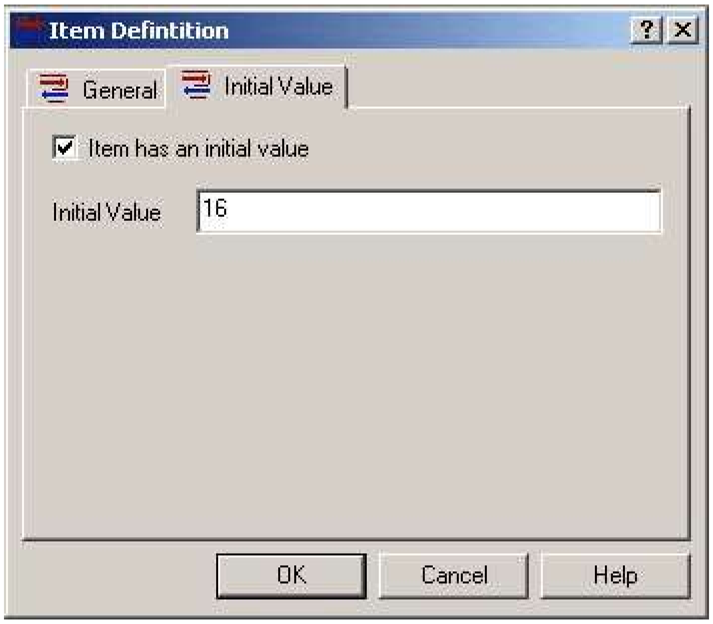
Using a Register Name with an Item Name

- Description: A description of the item. This is not used by the server, and is for informational purposes only.
- Data Type: The class of data used by this item. It can be one of four types:
  - Discrete: A single bit value. Usually expressed as Zero/One, or False/True.
  - Integer: A number expressed without any decimal places.
  - Real: A number express with or without decimal places. Also called a floating-point number.
  - String: Any sequence of characters.
- Item should be automatically activated: Checking this option will cause the server to activate, or make available, this item when client first connects to the server. The default is not-checked, meaning that the server will only activate this item if the client specifically request the item, or the protocol itself automatically activates it.
- Item is read only: When checked, the item can only pass data to the client. Any attempt by the client to change the data is ignored. The default is not-checked, which means the server will accept data from the client.
- Echo output values to current value. Setting this flag will copy the Outgoing value of this item to it's Incoming value. The server actually keeps two copies of the data - one for the last value written by the client, and another for the last value read in from the device. By default, these two values are kept separate, so that when data is sent to the server by the client, the last value read in from the device is preserved.

- Item is enabled: Checking this option (the default) will make the item available to any client. By clearing this option, the item will never be made available to a client.

Items Property Page - The Initial Values Tab

The Initial Values Tab lets you define a starting value for an item.



Item has an initial value: ✓

Initial Value: 16

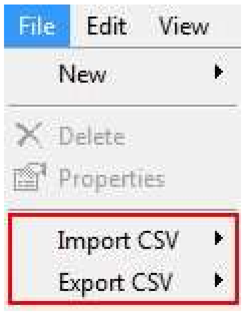
The Initial Values Tab

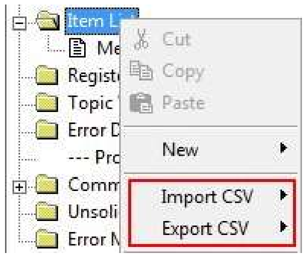
The different options are:

- Item has an initial value: Checking this box assigns the value in the **Initial Value** box to the item when the client first connects to the server. The default is not-checked, meaning that the item has no initial value.
- Initial Value: The initial value for the item. This is not used unless the **Item has an initial value** box is checked.

Item Import / Export

A protocol's Item List can be imported or exported via CSV. These options are available while the Item List is highlighted from the File menu, by right-clicking on the Item List or by clicking the Import or Export buttons in the toolbar. To ensure that you are using a properly formatted CSV file, it is recommended to configure at least one tag and export that tag to CSV.

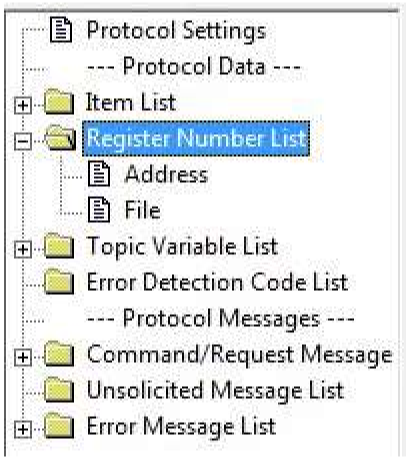




7.1.1.2.5 Register Number

A Register Number is a special server construct that extracts information from the **name** of the item.

For a video tutorial on working with Register Numbers, click here.



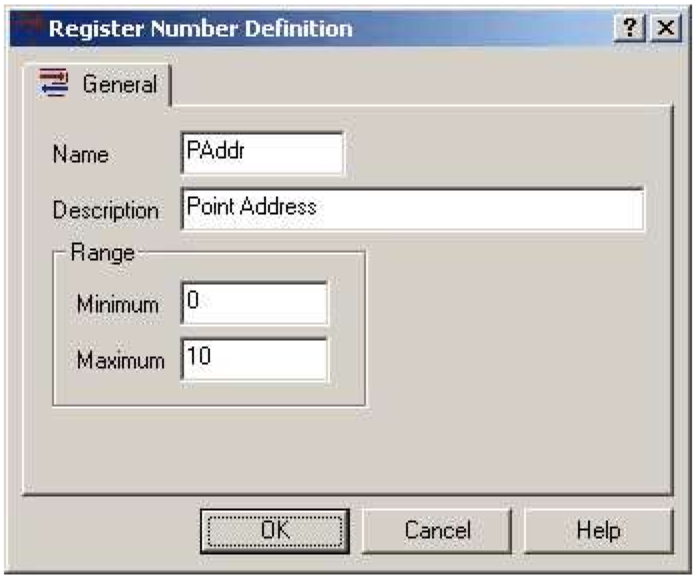
The Register Number Tree View

For example, suppose you had ten items all came from an Analog Input address bank of an device. Each one of these items was associated with addresses numbered 100 to 109. Using the server, you can create items **AnalogInput100, AnalogInput101, ..., AnalogInput109.** Or you can use a Register Number, and tell the server to expect a name that begins with **AnalogInput** and ends with a number. Therefore, your item name will look like this:

AnalogInput{Register}

Now each time the client requests a name like **AnalogInput245**, the server will extract the number **245** from the name and use that in the message. This allows you to make very generic messages without having to go to the trouble of creating a message for each address.

The Register Number Screen looks like this:



Name: PAddr

Description: Point Address

Range

Minimum: 0

Maximum: 10

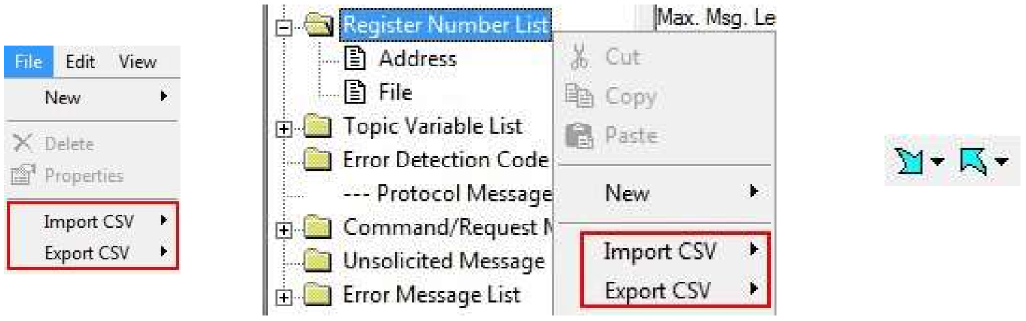
The General Tab for Register Numbers

The different options are:

- Name: The name of the Register Number, which must be unique within the protocol. You will use this in conjunction with **Item Names** to specify banks of sequentially-numbered items.
- Description: A description of the register number. This is for documentation purposes only and is not used by the server.
- Range: Minimum: The minimum numeric range accepted by the server. Used in conjunction with **Range: Maximum** below, this tells the server not to accept any register numbers from the client lower than this number.
- Range: Maximum: The maximum numeric range accepted by the server. Used in conjunction with **Range: Minimum** above, this tells the server not to accept any register numbers from the client higher than this number.

Item Import / Export

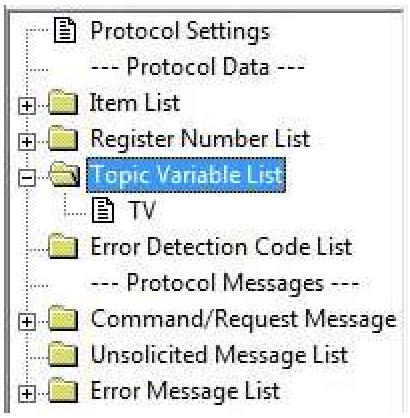
A protocol's Register Number List can be imported or exported via CSV. These options are available while the Register Number List is highlighted from the File menu, by right-clicking on the Register Number List or by clicking the Import or Export buttons in the toolbar. To ensure that you are using a properly formatted CSV file, it is recommended to configure at least one register number and export that to CSV.



7.1.1.2.6 Topic Variables

A Topic Variable is a user-defined server variable that is associated with a Topic. Instead of extracting information from an item, the Topic Variable is a constant number defined within a topic.

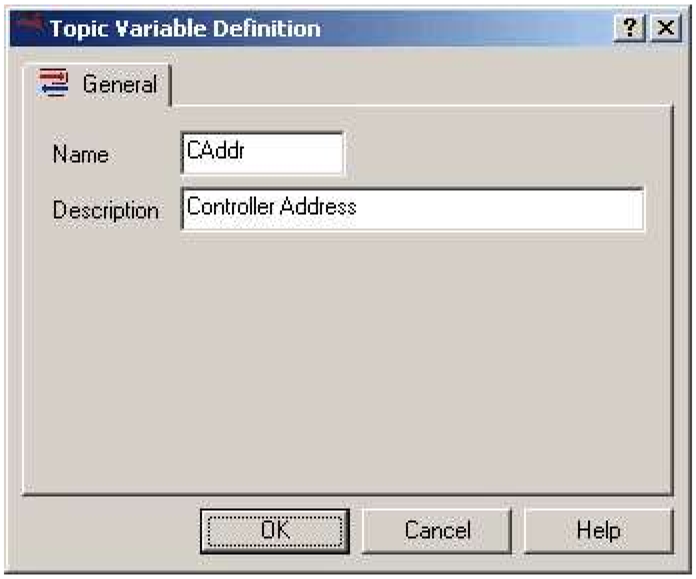
For a video tutorial on working with Topic Variables, click here.



The Topic Variable Tree

Within the protocol, you define each Topic Variable, but the actual value of that topic variable is defined within the topic.

The Topic Variable Screen looks like this:



Name: CAddr

Description: Controller Address

The General Tab for Topic Variables

The different options are:

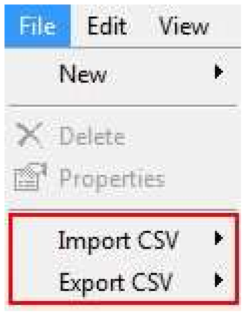
- Name: The name of the Topic Variable, which must be unique within the protocol.

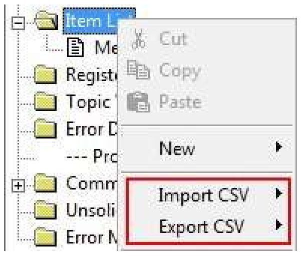
Please note: If you change the name (not the description) of a topic variable, all references in the protocol will also be changed. **However**, any topics that use the protocol will have to have the renamed topic variable set again.

- **Description:** A description of the topic variable.

Topic Variable Import / Export

A protocol's Topic Variable List can be imported or exported via CSV. These options are available while the Topic Variable List is highlighted from the File menu, by right-clicking on the Topic Variable List or by clicking the Import or Export buttons in the toolbar. To ensure that you are using a properly formatted CSV file, it is recommended to configure at least one topic variable and export it to CSV.

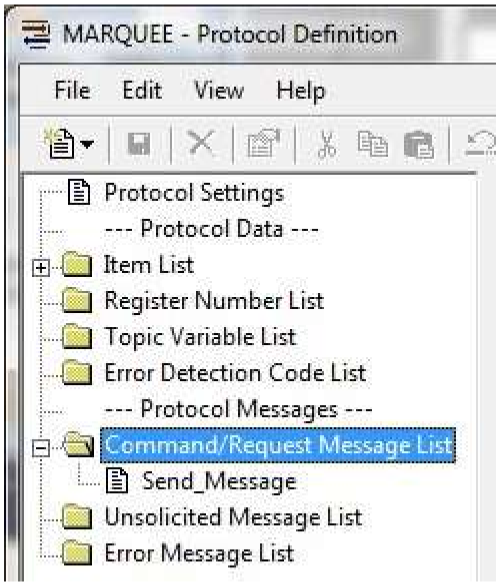




7.1.1.2.7 Command/Request (Host) Message

The heart of the server is the Command/Request Message. It is designed to elicit responses from or send data directly to the device. This is accomplished through the Request and Response messages. This is how it works:

A **Request** is first sent to the device. If no response is expected, then the server moves on to another action. Otherwise, if a **Response** is expected, the server will wait for the device to send data back. If that data matches the **Response**, then the items (if any) are extracted from the messages and sent back to the client. If the **Response** is not satisfied, the server will, by default, retry the command/request message until it reports a failure or the device responds with appropriate data.



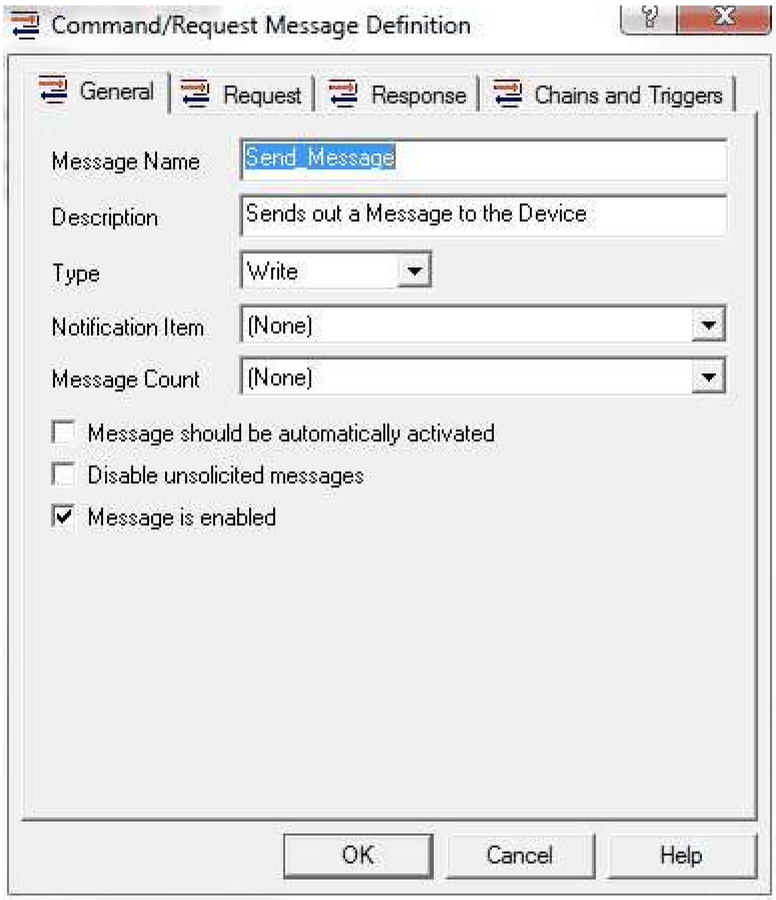
The Command/Request (Host) Message Tree

There are four property sheets with which to define a Command/Request Message. They are:

- General: Defines the default operations and properties of the command/request message.
- Request: Defines the format of the message to be sent to the device.
- Response: Defines the format of the message expect in response to the request message. This message is sent by the device to the server.
- Chains and Triggers: Defines actions taken by the server once the command/request message has been activated.

7.1.1.2.7.1 General Tab

The General Tab defines the default operations and properties of the command/request message.



Message Name: Send_Message
Description: Sends out a Message to the Device
Type: Write
Notification Item: (None)
Message Count: (None)
☐ Message should be automatically activated
☐ Disable unsolicited messages
☑ Message is enabled

The General Tab of a Command/Request (Host) Message

The options available are:

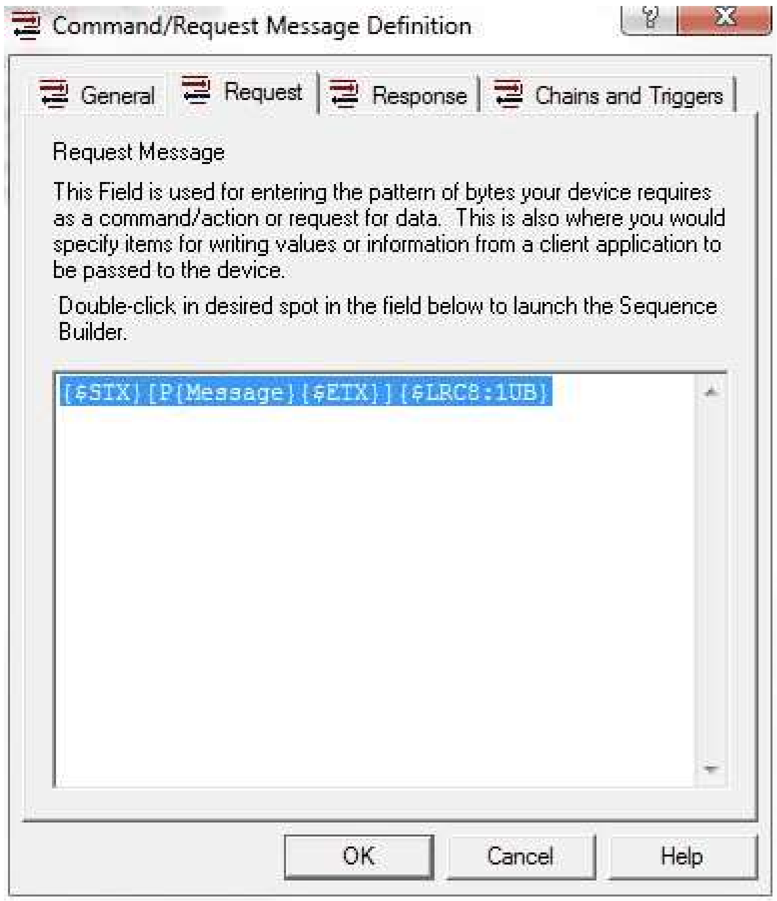
- Message Name: The name of the message, which must be unique within the protocol.
- Description: A description of the message. This is for documentation purposes only and is not used by the server.
- Type: Controls the processing of the command/request message. There are two different types of command/request messages::
  Read: "Reads" data from the device. If an item being requested by the client appears in the **Response** message of the Command/Request (Host) Message, then the server will send out the **Request** message in order to get the response from the device. This process happens without any interaction with the client or the user.
  Write: "Writes" data to the device. If the client changes the value of an item in the server, the server will find any "write" Command/Request (Host) Messages that have the item in the **Request** message. Once found, that Command/Request (Host) Message is activated. This process happens without any interaction with the client or the user.
  The default for type is **Read**. However, the default operation of both the "Read" and "Write" messages can be over-ridden by setting options in the **Chains and Triggers** section. See below for a more detailed explanation.
- Notification Item: The item selected in this box is set to One or True once the command/request message has been activated AND the message successfully completes. If the message does not complete, then this item is set to Zero or False. This is very useful in determining if a particular Command/Request (Host) Message is not working correctly. The default is **none**, meaning that there is no Notification Item associated with this message.
- Message Count: The item selected in this box is incremented whenever this message successfully completes.
- Message should be automatically activated: Setting this option tells the server to process this message immediately upon connection with the client, then process the command/request message normally. Set this option when you need to send initialization codes to the device upon startup. The default is **not set**, meaning that the command/request message is processed normally.
- Disable unsolicited messages: This option informs the server that no unsolicited messages are to be processed when the

command/request message response is being parsed. By default, the server will always compare all unsolicited messages with the response of the command/request message (since error conditions can occur within a valid response). Setting this option disables this feature. The default is **not checked**, meaning that all unsolicited messages are processed normally.

- Message is enabled: The message is processed when this box is checked, otherwise, the message is never processed. The default for this option is checked, meaning that the message is processed.

7.1.1.2.7.2 Request Tab

The Request Tab defines the format of the message to be sent to the device.



Request Message

This Field is used for entering the pattern of bytes your device requires as a command/action or request for data. This is also where you would specify items for writing values or information from a client application to be passed to the device.

Double-click in desired spot in the field below to launch the Sequence Builder.

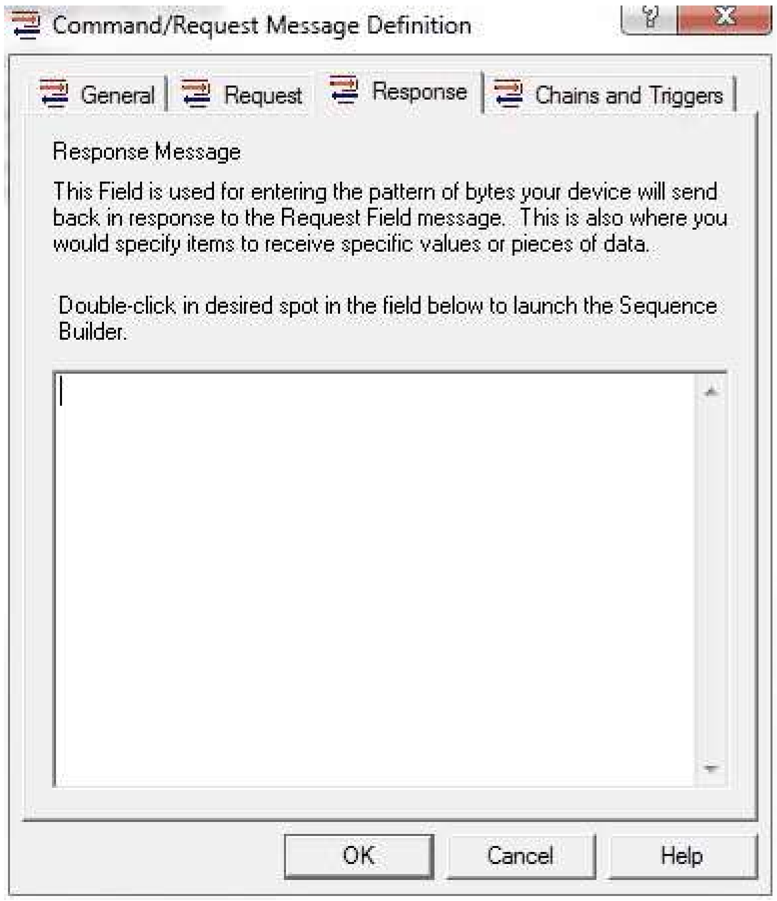
```
{$STX}[P{Message}{$ETX}]{$LRC8:1UB}
```

The Request Tab of a Command/Request (Host) Message

For a complete explanation of the use of the message box, please see the article on the Sequence Builder.

7.1.1.2.7.3 Response Tab

The Response Tab defines the format of the message expect in response to the request message. This message is sent by the device to the server.



Response Message

This Field is used for entering the pattern of bytes your device will send back in response to the Request Field message. This is also where you would specify items to receive specific values or pieces of data.

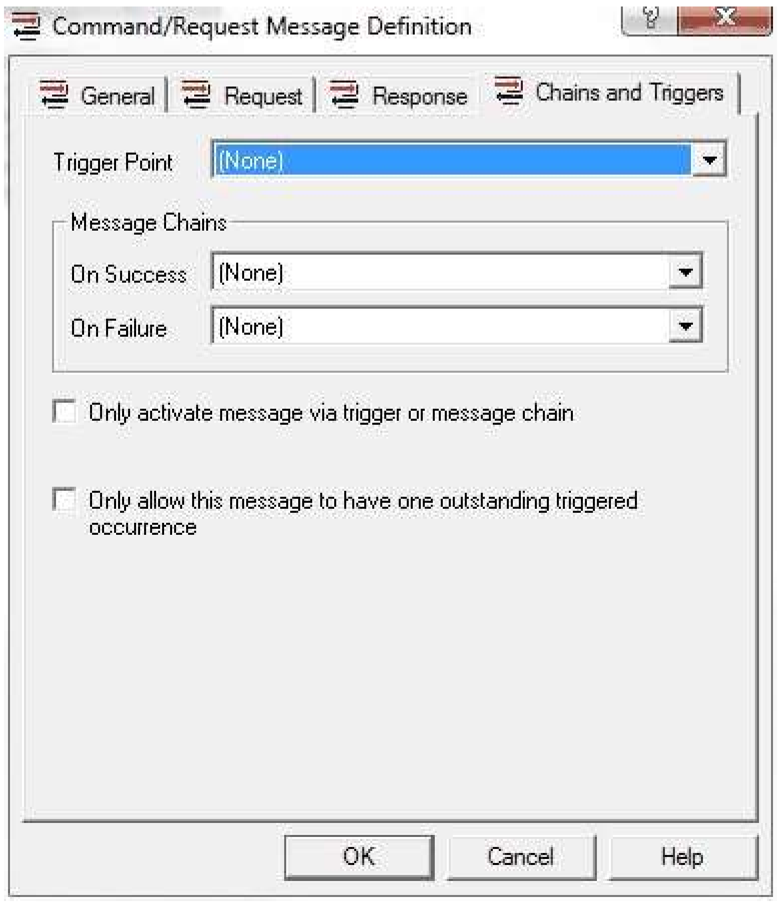
Double-click in desired spot in the field below to launch the Sequence Builder.

The Response Tab of a Command/Request Message

For a complete explanation of the use of the message box, please see the article on the Sequence Builder.

7.1.1.2.7.4 Chains and Triggers Tab

The Chains and Triggers Tab defines actions taken by the server once the Command/Request message has been activated.



Trigger Point: (None)

Message Chains

On Success: (None)

On Failure: (None)

☐ Only activate message via trigger or message chain

☐ Only allow this message to have one outstanding triggered occurrence

The Chains and Triggers Tab of a Command/Request Message

The options available are:

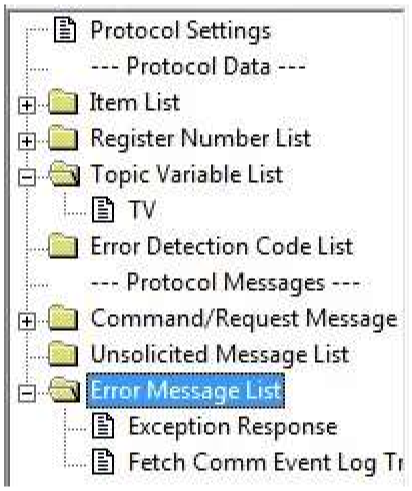
- Trigger Point: An item that "triggers" this message. Whenever the server sees that the client has changed the trigger point item to anything other than Zero or False, the server will immediately process the command/request message, complete the parsing, then set the trigger point item back to Zero or False. This is useful in "push-button" situations, where one will need to send out a command/request message based upon a certain condition. The default is **none**, meaning that there is no item that will trigger this command/request message.

- Message Chains: On Success: This option tells the server what command/request message to process if this command/request message successfully completes, that is, the request and response were processed without errors. This is useful in sending more than one command/request message to the device based upon certain conditions. The default is **none**, meaning that no command/request message is processed after this one completed.

- **Message Chains: On Failure:** This option tells the server what command/request message to process if this command/request message is unsuccessful in completing both the request and response side of the command/request message. This is useful in sending messages like initialization strings to recover from errors. The default is **none**, meaning that no command/request message is processed after this one completed.
- **Only activate message via trigger or message chain:** This option over-rides the default processing of command/request messages. Checking this option will mean that the command/request message will only be processed if the Trigger Point (mentioned above) is set or if this message is in a On Success or On Failure chain of another command/request message. The **type** of the message (see the General Tab above) is ignored. The default is **not checked**, meaning this message will be processed normally.
- Only allow this message to have one outstanding triggered occurrence. When checked, the server will never queue up more than one instance of this message when triggered by the client. This prevents the server from overloading the message queue and causing low memory conditions.

7.1.1.2.8 Error Message

An Error Message can also be defined as an alternative response to a Command/Request (Host) Message. It is checked against the data stream much like an Unsolicited Message, but whereas an unsolicited message is always accepted, you can set the acceptance level of an error message.



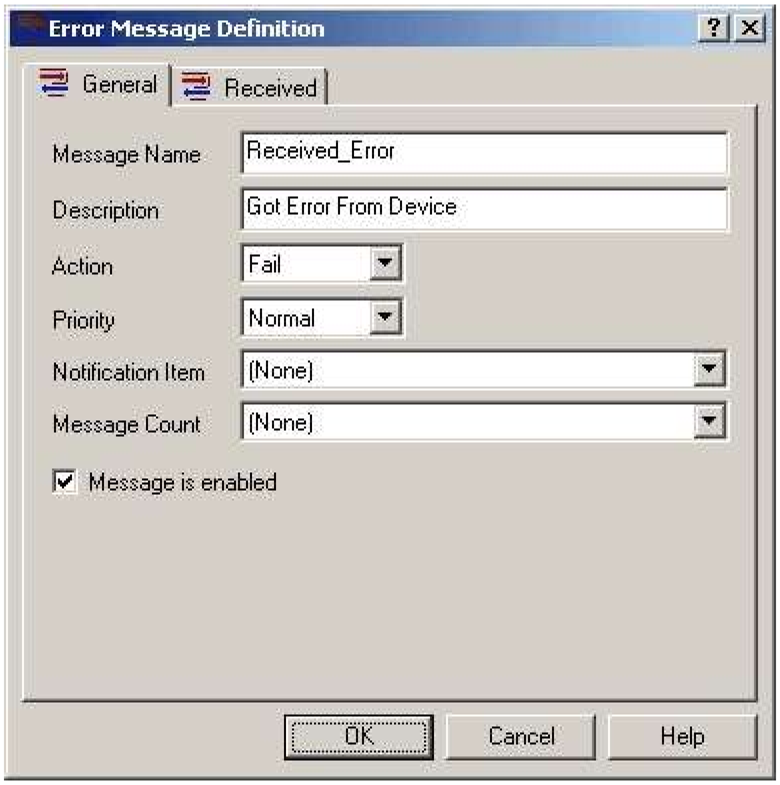
The Error Message Tree

There are two property sheets with which to define an Error Message. They are:

- General: Defines the default operations of the error message
- Received: Defines the format of the message received by the device.

7.1.1.2.8.1 General Tab

The General Tab defines both identification and action properties of the error message.



Message Name: Received_Error
Description: Got Error From Device
Action: Fail
Priority: Normal
Notification Item: (None)
Message Count: (None)
☑ Message is enabled

The General Tab of an Error Message

The options available are:

- **Message Name:** The name of the message, which must be unique within the protocol.
- **Description:** A description of the message. This is for documentation purposes only and is not used by the server.
- **Action:** One of four actions to take after the error message has been activated. Those actions are:
  Retry: Retry the message normally. If the message has exhausted all of it's retry attempts, then the message will fail.
  Retry Forever: Retry the message without charging this retry against the messages normal retry count. Thus, the message will retry as long as this or another retry forever error message is received as a response.
  Accept: Accept the error message as a valid response to the Command/Request (Host) Message and continue on to the **On**

Succeed message chain of the Command/Request (Host) Message (if it exists).
Fail: Consider the error message as a failed response to the Command/Request (Host) Message. Continue on to the **On Failure** message chain of the Command/Request (Host) Message (if it exists).
The default for this action is **Retry**.

- **Priority:** Process the message given a certain Priority. This means that this error message will be evaluated before or after any others. Error messages are processed after the response part of a Command/Request (Host) Message is evaluated, and before an Unsolicited Message is processed. The three different priorities are:
  Low: Process this error message after any other messages with priorities of Normal or High. If more than one message is marked Low, the messages are processed in the order of their creation.
  Normal: Process this error message after any messages marked as High, but before messages marked as Low. If more than one message is marked Normal, the messages are processed in the order of their creation.
  High: Process this error message before any marked Normal or Low. If more than one message is marked High, the messages are processed in the order of their creation.
  The default for priority is **Normal**.

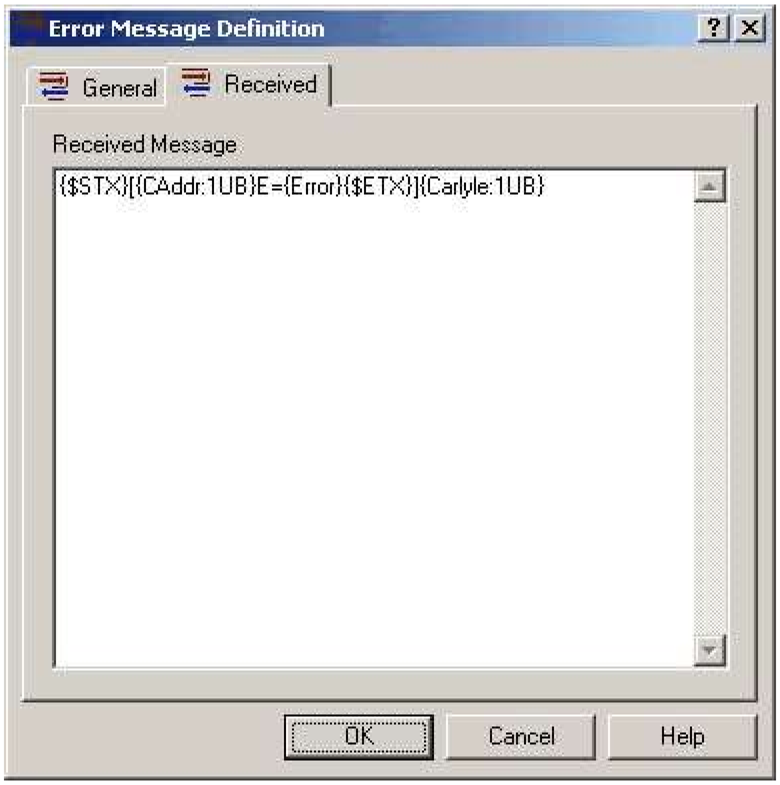
- **Notification Item:** The item selected in this box is set to One or True once the Command/Request (Host) Message has been activated AND the message successfully completes. If the message does not complete, then this item is set to Zero or False. This is very useful in determining if a particular error message is not working correctly. The default is **none**, meaning that there is no Notification Item associated with this message.

- **Message Count:** The item selected in this box is incremented whenever this message successfully completes.

- **Message is enabled:** The error is processed when this box is checked, otherwise, the message is never processed. The default for this option is checked, meaning that the message is processed.

7.1.1.2.8.2 Received Tab

The Received Tab defines the format of the message expected from the device.



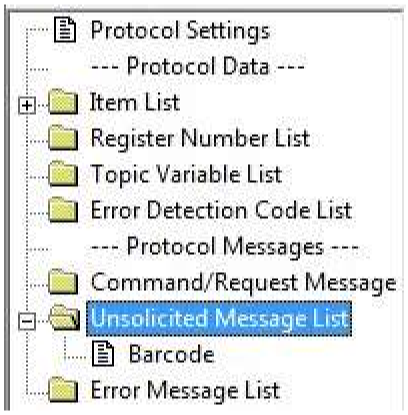
```
{$STX}[{CAddr:1UB}E={Error}{$ETX}]{Carlyle:1UB}
```

The Received Tab of an Error Message

For a complete explanation of the use of the message box, please see the article on the Sequence Builder.

7.1.1.2.9 Unsolicited Message

An Unsolicited Message is designed to take care of data that comes in from the device without any prompting from the server - thus "unsolicited". The unsolicited message is primarily used for devices such as scales and barcode scanners where the server doesn't know when the data might come in from the device. However, the unsolicited message can also be used to trap secondary responses to Command/Request (Host) Messages (much in the same way error messages are used).



The Unsolicited Message Tree

There are three property sheets with which to define an Unsolicited Message. They are:

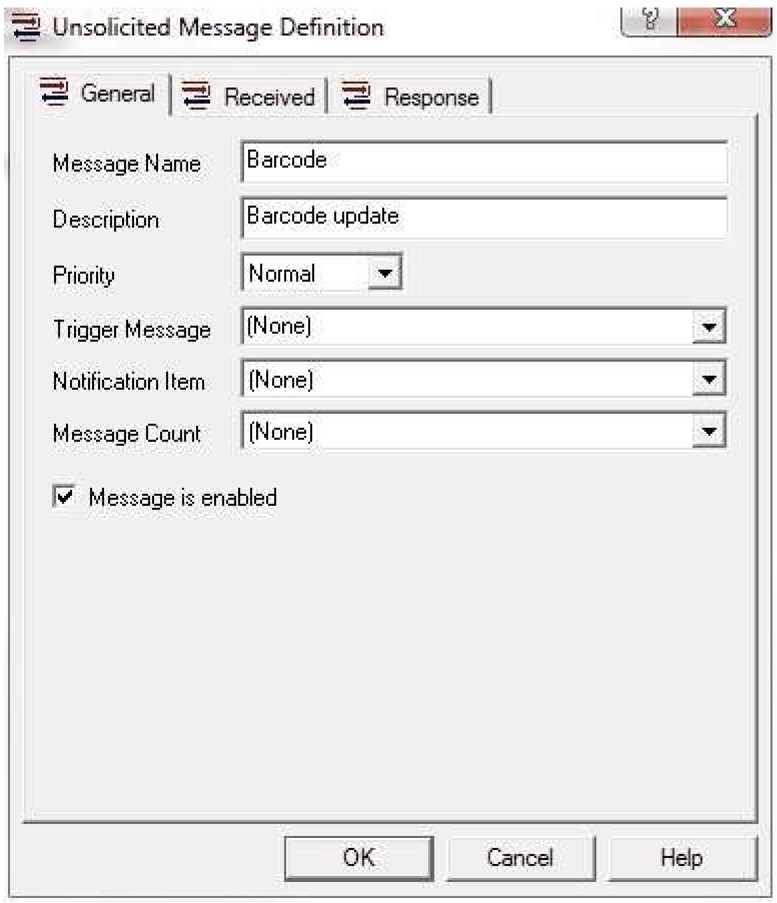
- General: Defines the default operations and properties of the unsolicited message.
- Received: Defines the format of the message received by the device.
- Response: Defines the format of the message to be sent in response to the device once the unsolicited message has been activated.

7.1.1.2.9.1 General Tab

Server Details : Configuration : Protocol : Unsolicited Message

General

The General Tab defines both identification and action properties of the unsolicited message.


Unsolicited Message Definition

General | Received | Response

Message Name	Barcode
Description	Barcode update
Priority	Normal
Trigger Message	(None)
Notification Item	(None)
Message Count	(None)

☑ Message is enabled

The General Tab of an Unsolicited Message

The options available are:

- Message Name: The name of the message, which must be unique within the protocol.
- Description: A description of the message. This is for documentation purposes only and is not used by the server.
- Priority: Process the message given a certain Priority. This means that this unsolicited message will be evaluated before or after any others. The three different priorities are:
  Low: Process this unsolicited message after any other messages with priorities of Normal or High. If more than one message is marked Low, the messages are processed in the order of their creation.
  Normal: Process this unsolicited message after any messages marked as High, but before messages marked as Low. If more than one message is marked Normal, the messages are processed in the order of their creation.

High: Process this unsolicited message before any marked Normal or Low. If more than one message is marked High, the messages are processed in the order of their creation.
The default for priority is **Normal**.

- Trigger Message: Defines the Command/Request (Host) Message to be processed once this unsolicited message has been activated. The Command/Request (Host) Message will get processed after the Response side of the unsolicited message is sent to the device, and after the Notification Item is set.

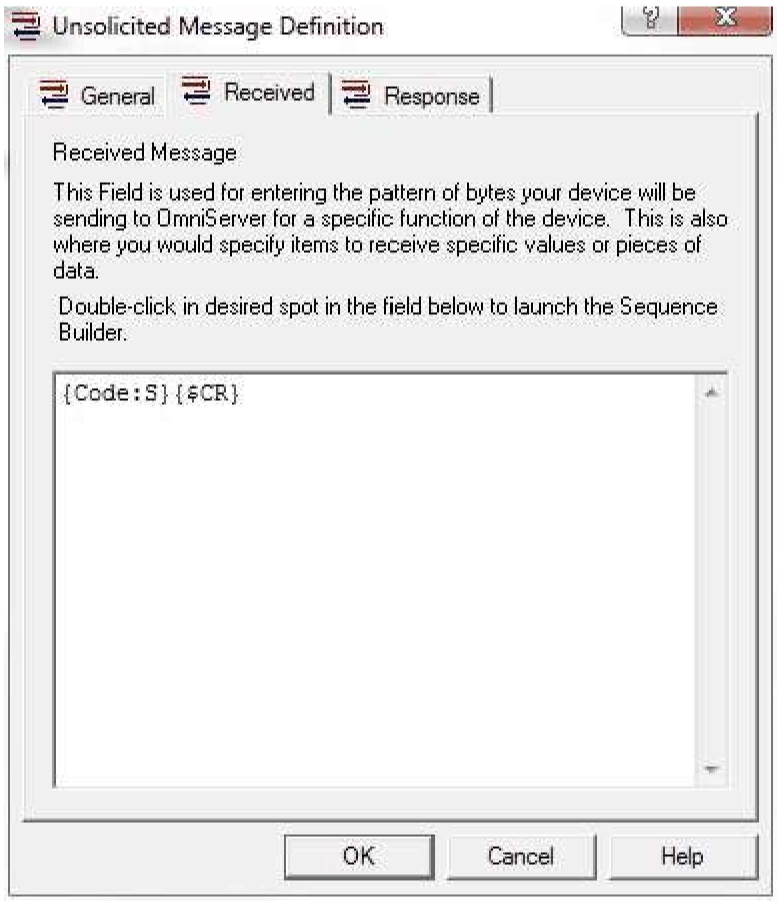
Unsolicited triggered messages are queued to the current pending message list. Thus, if there are read or write messages due, they will be executed prior to the unsolicited message's triggered message.

The default is **none**, meaning that no Command/Request (Host) Message will be activated.

- Notification Item: The item selected in this box is set to One or True once the unsolicited message has been activated. This is useful in knowing when an unsolicited message has arrived. The default is **none**, meaning that there is no Notification Item associated with this message.
- Message Count: The item selected in this box is incremented whenever this message successfully completes.
- Message is enabled: The message is processed when this box is checked, otherwise, the message is never processed. The default for this option is checked, meaning that the message is processed.

7.1.1.2.9.2 Received Tab

The Received Tab defines the format of the message expected from the device.



Received Message

This Field is used for entering the pattern of bytes your device will be sending to OmniServer for a specific function of the device. This is also where you would specify items to receive specific values or pieces of data.

Double-click in desired spot in the field below to launch the Sequence Builder.

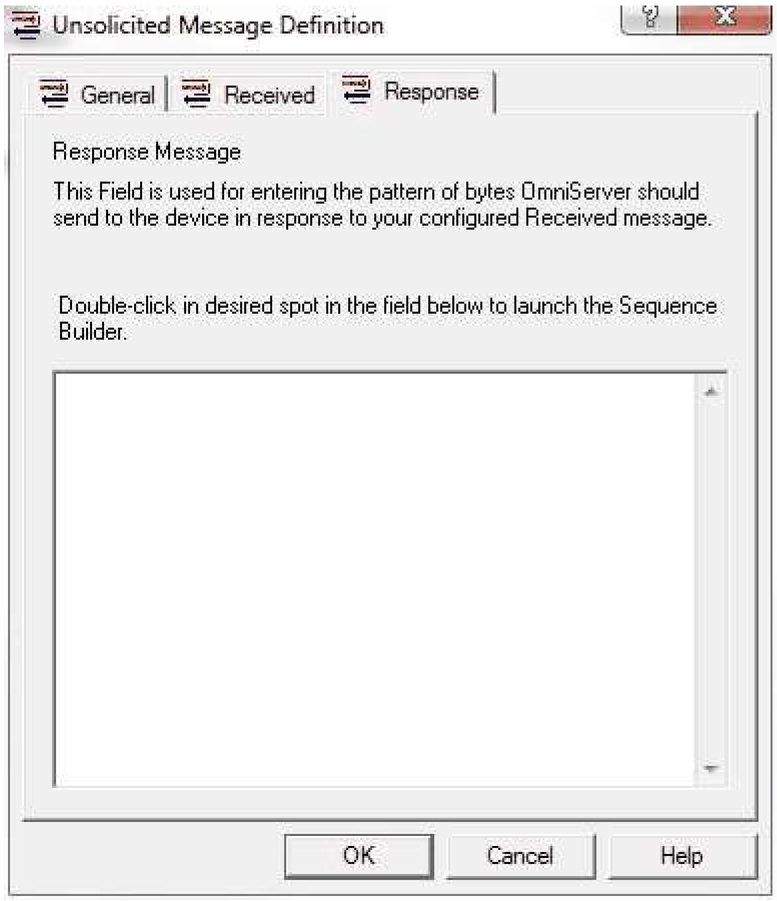
```
{Code:S}{$CR}
```

The Received Tab of an Unsolicited Message

For a complete explanation of the use of the message box, please see the article on the Sequence Builder.

7.1.1.2.9.3 Response Message

The Response Tab defines the format of the message to be sent in response to the device.


Unsolicited Message Definition

General | Received | Response

Response Message

This Field is used for entering the pattern of bytes OmniServer should send to the device in response to your configured Received message.

Double-click in desired spot in the field below to launch the Sequence Builder.

The Response Tab of an Unsolicited Message

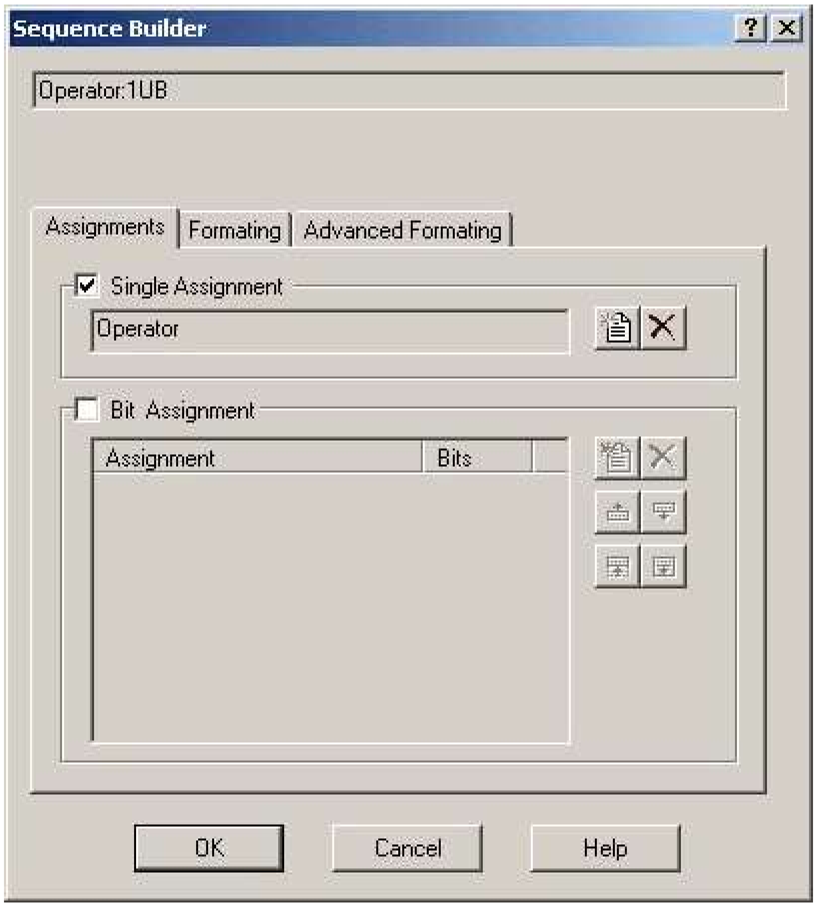
For a complete explanation of the use of the message box, please see the article on the Sequence Builder.

7.1.1.2.10 Sequence Builder

How data sent to an device is built, or how it is parsed from the device is known as a **sequence**. This sequence describes the exact syntax of any messages channeling through the server. This sequence of characters is also known as a **message**.

To help you in the design of the message sequence, this server has what is known as a **sequence builder**. To access the sequence builder, double-click within the message box of any server message.

When you double-click in that box, this is what you will see:



The Sequence Builder Screen

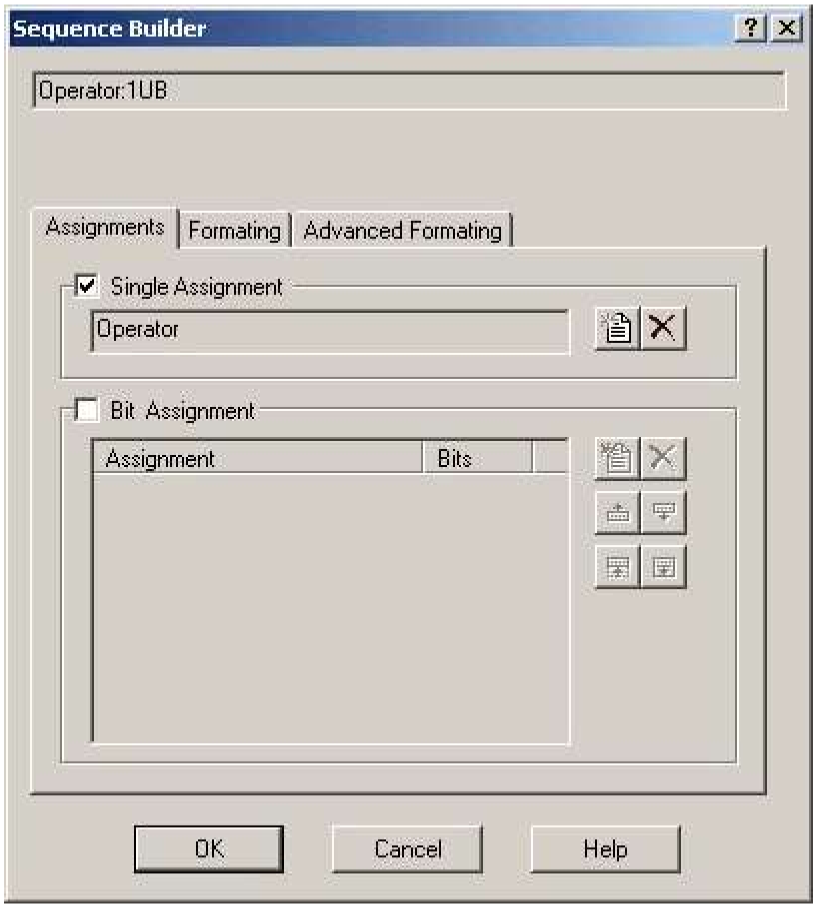
At the top of the screen is the final result of what you will be inserting into the message once you press OK. The server will also tell you if there is a problem with the combination of formats for the item.

To learn about the different aspects of the Sequence Builder, just follow one of the links below.

- The Assignments Tab: Here, you assign items, numbers, registers, controls, EDCs, and user-defined functions to your message.
- The Formatting Tab: Use this tab to format the basic types of data, such as Integer, String, Real, etc.
- The Advanced Formatting Tab: More formatting options, such as Reversed Text, Translations, etc.

7.1.1.2.10.1 Assignments

The **Assignments** tab of the Sequence Builder is where you assign items, registers, controls, numbers, EDCs, or user-defined objects to the message sequence.

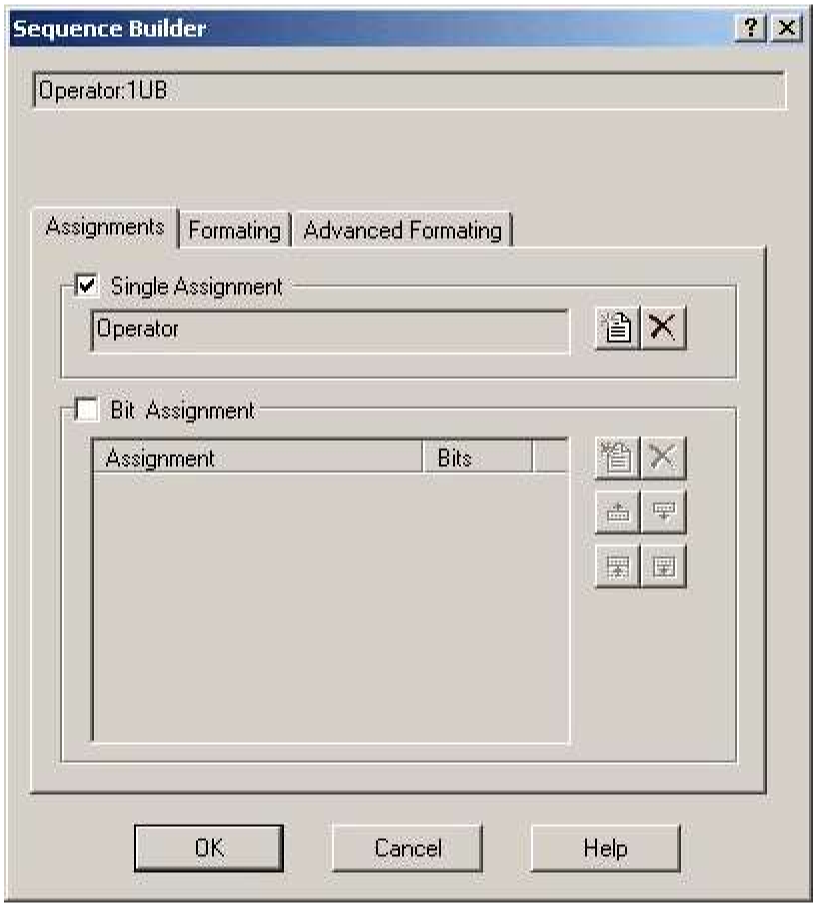


The Sequence Builder: Assignments Tab Screen

Making an Assignment

To start, there are two possible ways to make an assignment. To learn more about either, please click on the link below:

- Single Assignment: Assigns the selection to a sequence of data.



The Single Assignment Screen

Just underneath the check box is the current selection for this session. The buttons to the right are the options available. They are:

Select: Select the item, register, control, number, or user-defined control.

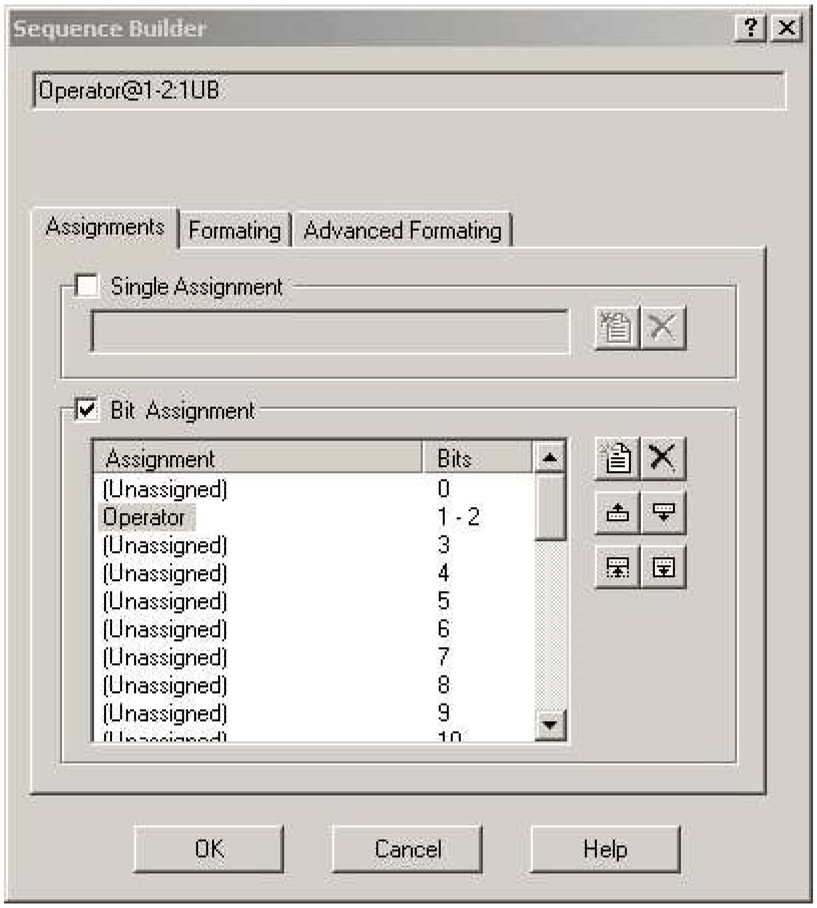
The Select icon brings up a property sheet dialog box in which you can select whatever items are available at that time. Those properties are:

- Items
- Register Numbers
- Control
- Others
- Number

⊟ ✕ Delete: Delete, or clear, the selected item, register, control, number, or user-defined control from the list.

The Delete button simply clears away any assignment previously made. This allows you to start over again.

⊟ Bit Assignment: Assigns a selection to a sequence of bits within the data stream.



Assignment	Bits
(Unassigned)	0
Operator	1 - 2
(Unassigned)	3
(Unassigned)	4
(Unassigned)	5
(Unassigned)	6
(Unassigned)	7
(Unassigned)	8
(Unassigned)	9

The Bit Assignment Screen

Just underneath the check box is the current selection for this session. The buttons to the right are the options available. They are:

⊟ Select: Select the item, register, control, number, or user-defined control.

The Select icon brings up a property sheet dialog box in which you can select whatever items are available at that time. Those properties are:

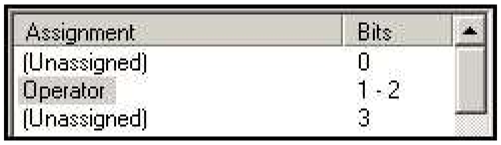
- Items
- Register Numbers
- Others
- Number

Delete: Delete, or clear, the selected item, register, control, number, or user-defined control from the list.

The Delete button simply clears away any assignment previously made. This allows you to start over again.

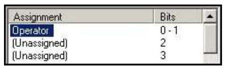
Move Up: Move the selection up in the bit list.

The Move Up button moves the selection up one spot in the bit listing.



Assignment	Bits
(Unassigned)	0
Operator	1 - 2
(Unassigned)	3

Before the Move Up Operation

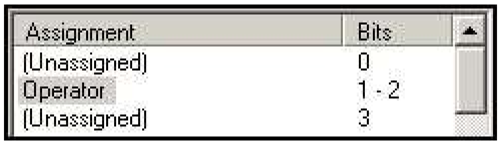


Assignment	Bits
Operator	0 - 1
(Unassigned)	2
(Unassigned)	3

After the Move Up Operation

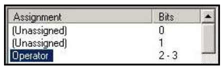
Move Down: Move the selection down in the bit list.

The Move Down button moves the selection down one spot in the bit listing.



Assignment	Bits
(Unassigned)	0
Operator	1 - 2
(Unassigned)	3

Before the Move Down Operation

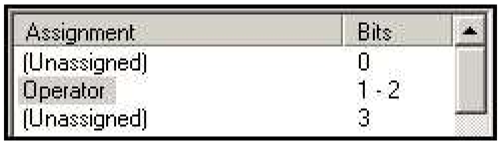


Assignment	Bits
(Unassigned)	0
(Unassigned)	1
Operator	2 - 3

After the Move Down Operation

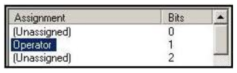
Shrink: Decrease the number of bits assigned to this selection.

The Shrink button decreases the number of bits assigned to a selection.



Assignment	Bits
(Unassigned)	0
Operator	1 - 2
(Unassigned)	3

Before the Shrink Operation

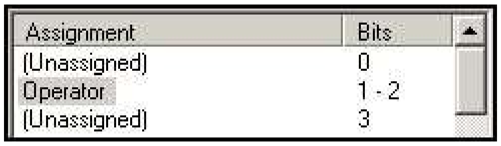


Assignment	Bits
(Unassigned)	0
Operator	1
(Unassigned)	2

After the Shrink Operation

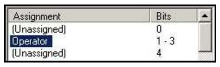
Expand: Increase the number of bits assigned to this selection.

The Expand button increases the number of bits assigned to the selection.



Assignment	Bits
(Unassigned)	0
Operator	1 - 2
(Unassigned)	3

Before the Expand Operation



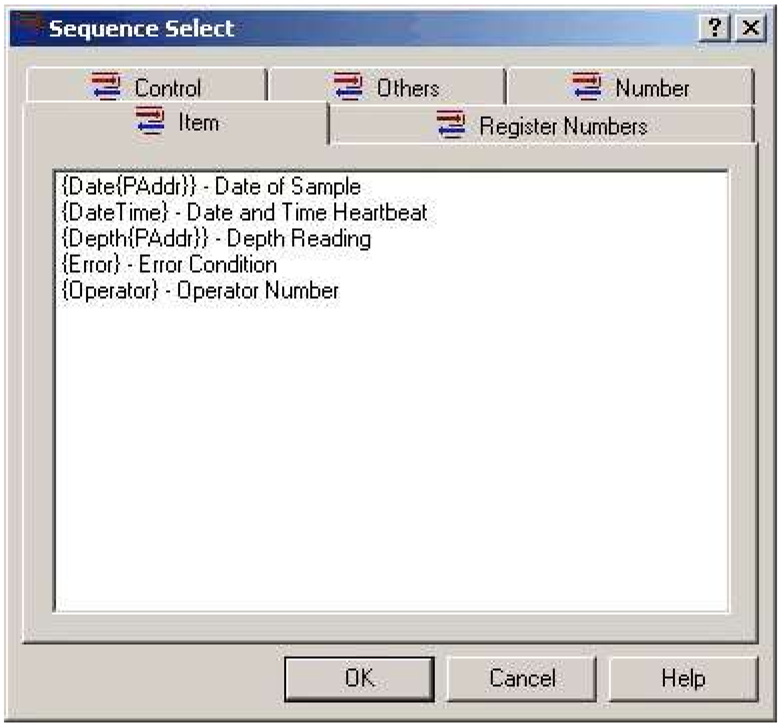
Assignment	Bits
(Unassigned)	0
Operator	1 - 3
(Unassigned)	4

After the Expand Operation

Server Details : Configuration : Protocol : Sequence Builder

Select Items

The Items screen displays all the items defined in the protocol.



{Date{PAddr}} - Date of Sample
{DateTime} - Date and Time Heartbeat
{Depth{PAddr}} - Depth Reading
{Error} - Error Condition
{Operator} - Operator Number

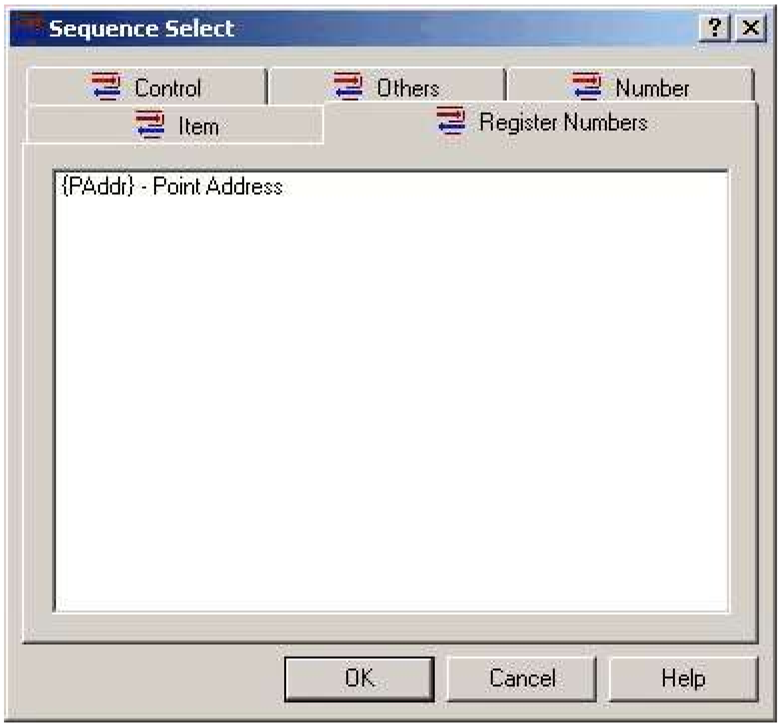
Sequence Select - Item

Items are defined within the protocol here.

Server Details : Configuration : Protocol : Sequence Builder

Select Register Numbers

The Registers screen displays all the registers defined in the protocol.



{PAddr} - Point Address

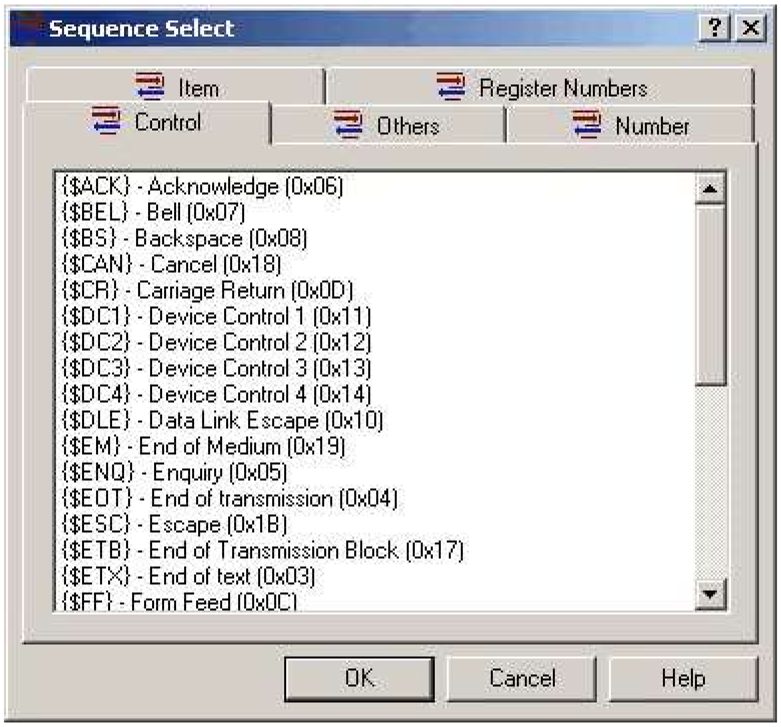
Sequence Select - Register

Register Numbers are defined within the protocol here.

Server Details : Configuration : Protocol : Sequence Builder

Select Control

The Controls screen displays all reserved, pre-defined control characters available for use in a message sequence.



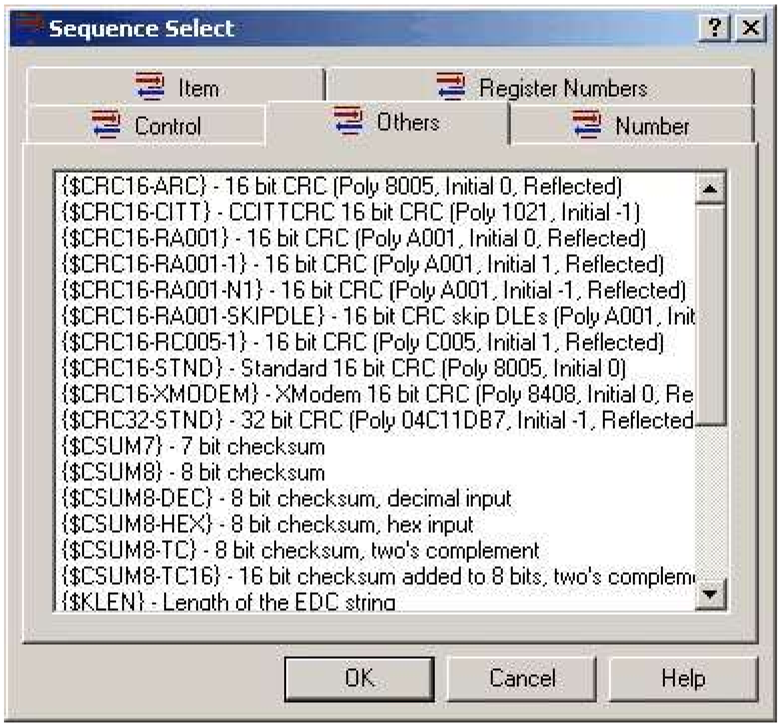
{$ACK} - Acknowledge (0x06)
{$BEL} - Bell (0x07)
{$BS} - Backspace (0x08)
{$CAN} - Cancel (0x18)
{$CR} - Carriage Return (0x0D)
{$DC1} - Device Control 1 (0x11)
{$DC2} - Device Control 2 (0x12)
{$DC3} - Device Control 3 (0x13)
{$DC4} - Device Control 4 (0x14)
{$DLE} - Data Link Escape (0x10)
{$EM} - End of Medium (0x19)
{$ENQ} - Enquiry (0x05)
{$EOT} - End of transmission (0x04)
{$ESC} - Escape (0x1B)
{$ETB} - End of Transmission Block (0x17)
{$ETX} - End of text (0x03)
{$FF} - Form Feed (0x0C)

Sequence Select - Controls

Server Details : Configuration : Protocol : Sequence Builder

Select Others

The Others screen displays all the EDCs available to the protocol, including any user-defined EDCs and Topic Variables.



{$CRC16-ARC} - 16 bit CRC (Poly 8005, Initial 0, Reflected)
{$CRC16-CITT} - CCITTCRC 16 bit CRC (Poly 1021, Initial -1)
{$CRC16-RA001} - 16 bit CRC (Poly A001, Initial 0, Reflected)
{$CRC16-RA001-1} - 16 bit CRC (Poly A001, Initial 1, Reflected)
{$CRC16-RA001-N1} - 16 bit CRC (Poly A001, Initial -1, Reflected)
{$CRC16-RA001-SKIPDLE} - 16 bit CRC skip DLEs (Poly A001, Init
{$CRC16-RC005-1} - 16 bit CRC (Poly C005, Initial 1, Reflected)
{$CRC16-STND} - Standard 16 bit CRC (Poly 8005, Initial 0)
{$CRC16-XMODEM} - XModem 16 bit CRC (Poly 8408, Initial 0, Re
{$CRC32-STND} - 32 bit CRC (Poly 04C11DB7, Initial -1, Reflected
{$CSUM7} - 7 bit checksum
{$CSUM8} - 8 bit checksum
{$CSUM8-DEC} - 8 bit checksum, decimal input
{$CSUM8-HEX} - 8 bit checksum, hex input
{$CSUM8-TC} - 8 bit checksum, two's complement
{$CSUM8-TC16} - 16 bit checksum added to 8 bits, two's complem
{$KLEN} - Length of the EDC string

Sequence Select - Others

Pre-defined EDCs are defined by the server and can be located here.

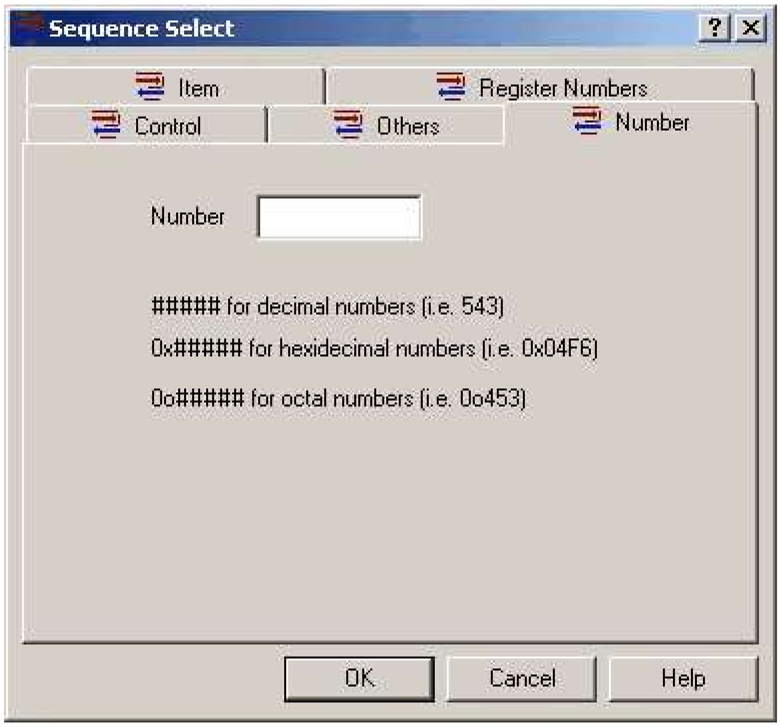
User-defined EDCs are defined within the protocol here.

Topic Variables are defined within the protocol here.

Server Details : Configuration : Protocol : Sequence Builder

Select Numbers

The Numbers screen allows you to enter in a Decimal, Hexadecimal, or Octal number.



Number

for decimal numbers (i.e. 543)

0x##### for hexidecimal numbers (i.e. 0x04F6)

0o##### for octal numbers (i.e. 0o453)

Sequence Select - Numbers

Server Details : Configuration : Protocol

Error Detection Codes

When sending a message to a device, the server formats the message, replacing sequences with their formatted values. It then computes the error detection code from those characters specified between brackets in the message. The server then removes the brackets and replaces the error detection sequence with the formatted computed value.

When processing a message from a device, the server computes the error detection code. If the computed code matches the received code, the server accepts the message and processes it. If the codes do not match, the server ignores the message.

Rejection of a message can have an impact on performance unless the protocol handles the received message elsewhere. For Command/Request (Host) Messages, use an error message that mimics the expected response. For example, the error message can be designed like the response message - except that there are no brackets and a discarded item replaces the error detection sequence. The width of the string sequence must be the same as that of the error detection sequence. This will match the device's message with the incorrect code. Set the error message's action to either retry or accept. Similarly, the protocol can handle a failed error detection code in an unsolicited message with another unsolicited message of a lower priority.

Occasionally, you may implement a protocol that uses error detection but use it in an environment that does not cause interference in the media. You can use the same technique described above to process the messages. Simply use a discarded string item in place of the error detection code. This works only when you receive data from a device; you may still need to provide an error code for messages sent to the device.

In addition to user-defined EDCs (which you can read about here), the server supplies a set number of EDCs that are most commonly used:

- Cyclic Redundancy Check (CRC)
- Checksums
- Longitudinal Redundancy Check (LRC, XOR, BCC)
- Others

The checksum uses the sum of the values from the message to produce the error code. This is the simplest of all error detection codes. The standard checksum uses the binary value of each character of the message in computing its value. The following example shows the computation of the simple checksum on the message "01":

ASCII:	0 + 1 = a
Decimal:	48 + 49 = 97
Hexadecimal:	30 + 31 = 61

List of Pre-Installed Checksums:

CSUM7

Standard 7-bit checksum. The most significant bit of each byte is ignored.

CSUM8

Standard 8-bit checksum.

CSUM8-DEC

Standard 8-bit checksum, decimal input. Each two-byte sequence represents a decimal number between 00 and 99.

CSUM8-HEX

Standard 8-bit checksum, hexadecimal input. Each two-byte sequence represents a hexadecimal number between 00 and FF.

CSUM8-TC

Two's compliment 8-bit checksum. The normal checksum is calculated. The checksum is then "complimented", that is, all the bits of the number are reversed (0 to 1 and 1 to 0). Finally, 1 is added to the number to give the final result.

For example, suppose a checksum of hexadecimal "94" is generated. Here is the calculation used to produce a two's compliment number:

Action	Hex	Binary
Starting #:	94	1001 0100
Compliment:	6B	0110 1011
Add One:	6C	0110 1100

CSUM8-TC16

16-bit checksum added to 8 bits, with a two's compliment. This is the same as CSUM8-TC above, but with a sixteen-bit number instead of an eight-bit number.

The cyclic redundancy check (CRC) produces the error code by considering the entire message as the coefficients to a base-2 polynomial, dividing it by a generating polynomial, and using the remainder as the error code.

List of Pre-Installed CRCs:

⊟ CRC16-ARC

- 16-bit Reflected CRC
- Polynomial: 800A
- Initial Value: 0

⊟ CRC16-CITT

- 16-bit CCITTCRC CRC
- Polynomial: 1021
- Initial Value: -1 (0xFFFF)

⊟ CRC16-RA001

- 16-bit Reflected CRC
- Polynomial: A001
- Initial Value: 0

⊟ CRC16-RA001-1

- 16-bit Reflected CRC
- Polynomial: A001
- Initial Value: 1

⊟ CRC16-RA001-N1

- 16-bit Reflected CRC
- Polynomial: A001
- Initial Value: -1 (0xFFFF)

⊟ CRC16-RA001-SKIPDLE

- 16-bit Reflected CRC
- Skips DLEs
- Polynomial: A001
- Initial Value: 0

⊟ CRC16-RC005

- 16-bit Reflected CRC
- Polynomial: C005
- Initial Value: 1

⊟ CRC16-STND

- 16-bit Standard CRC
- Polynomial: 8005
- Initial Value: 0

⊟ CRC16-XMODEM

- 16-bit Reflected CRC, XMODEM
- Polynomial: 8408
- Initial Value: 1

⊟ CRC32-STND

- 32-bit Reflected CRC
- Polynomial: 04C11DB7
- Initial Value: 0

Standard Implementation

In the standard implementation of the CRC algorithm, the CRC is computed by processing the incoming data most significant bit to least significant bit.

The most straightforward implementation is to loop eight times for each value added to the accumulator. The high-order bit is noted and the entire accumulator is shifted up by 1 bit. If the high-order bit was set, then the generating polynomial, less the x^16 term, is exclusive or'ed into the accumulator. This exclusive or'ing of the generating polynomial determines a remainder polynomial. The bit test version of the CRC implementation is as follows:

```
ErrorCode := seedValue
for each character c in the message
  ErrorCode := ErrorCode xor c
  do 8 times
    bit :=most significant bit of ErrorCode
    ErrorCode := up shift of ErrorCode
    if bit is on
      ErrorCode := ErrorCode xor
generatingPolynomial
    end if
  end do
end for
```

Some implementations recognize that the value of the high-order byte solely determines how the generating polynomial contributes to the accumulator. These implementations use a table lookup instead. The table lookup version of the CRC implementation is as follows:

```
ErrorCode:= seedValue
for each character c in the message
      IndexByte := high-order byte of ErrorCode
shifted into the low-order byte xor c
```

```
    ErrorCode := shift of low-order byte into high-
order byte
    ErrorCode := ErrorCode xor TableLookup
[IndexByte]
end for
```

Another method of computing the CRC is inspired by the hardware implementation:

```
CRCLowByte := 255
CRCHighByte := 255
for each character c in the string
    a := c xor CRCLowByte
    b := a * 8
    c := a * 16
    d := a * 128
    e := a and hexadecimal F
    f := (a / 16) and hexadecimal F
    g := (a / 32) and hexadecimal F
    h := (e / 2) and hexadecimal F
    CRCLowByte = CRCHighByte xor b xor f xor e
xor d
    CRCHighByte = a xor g xor c xor h
end for
CRCHighByte := ones complement of CRCHighByte
CRCLowByte := ones complement of CRCLowByte
```

In almost all cases, the server uses the table lookup version to compute CRCs since it even outperforms the hardware inspired implementations. Even in cases where we receive a hardware inspired implementation for addition into the server, we extract the polynomial and seed values from the algorithm and convert the CRC to a table lookup.

Reflected Implementation

In the reflected implementation of the CRC algorithm, the CRC is computed by processing the incoming data least significant bit to most significant bit.

The difference between the standard and reflected implementation is that the polynomial is reversed, the accumulator is shifted down, and the least significant bit of the accumulator is inspected.

Generating Polynomials

As stated above, the generating polynomial is exclusive or'ed into the accumulator. The generating polynomial for a 16-bit CRC has 17 coefficients, for x^0 through x^16. However, the bit pattern representing the polynomial has only 16 coefficients. This is because the coefficient for x^16 is assumed instead of explicitly represented. This is acceptable since it, exclusive or'ed with the bit shifted out of the accumulator, cancels and contributes no more to the result.

In the case of the standard implementation, x^15 represents bit 15 and x^0 represents bit 0. For reflected implementations, this is reversed with x^15 representing bit 0 and x^0 representing bit 15.

For example:

Polynomial x^16 + x^15 + x^2 + 1
Standard Representation 1000 0000 0000 0101 (8005)
Reflected Representation 1010 0000 0000 0001 (A001)

Server Details : Configuration : Protocol

Error Detection Codes : LRC

The longitudinal redundancy check (LRC) uses the exclusive or (XOR) of the values from the message to produce the error code. The standard LRC uses the binary value of each character of the message in computing its value. The following example shows the computation of the simple LRC on the message "01":

```
ASCII:          0 XOR        1 = CTRL-A, SOH
Binary:   00110000 XOR 00110001 = 00000001
```

Hexadecimal: 30 XOR 31 = 01

Note: LRCs are commonly referred to as block check codes (BCC).

List of Pre-Installed LRCs:

LRC8

Standard 8-bit LRC.

LRC8-OCCOMBO

8-bit LRC combined with a one's compliment checksum.

LRC8-X30CRC

8-bit LRC encoded into two 0x30-biased bytes.

Server Details : Configuration : Protocol

Error Detection Codes : Others

The server supplies other special-case EDCs. They normally do not follow standard EDC calculations, but are used much like EDCs within the message.

List of Other EDCs:

KLEN

The length is the number of characters in the portion of the message that generates this code. Though this is not a true error detection code, it is useful in protocols that use counted, variable-length strings. KLEN returns just the length or byte count, so the KLEN of the message ABCDE is 5.

KLENA

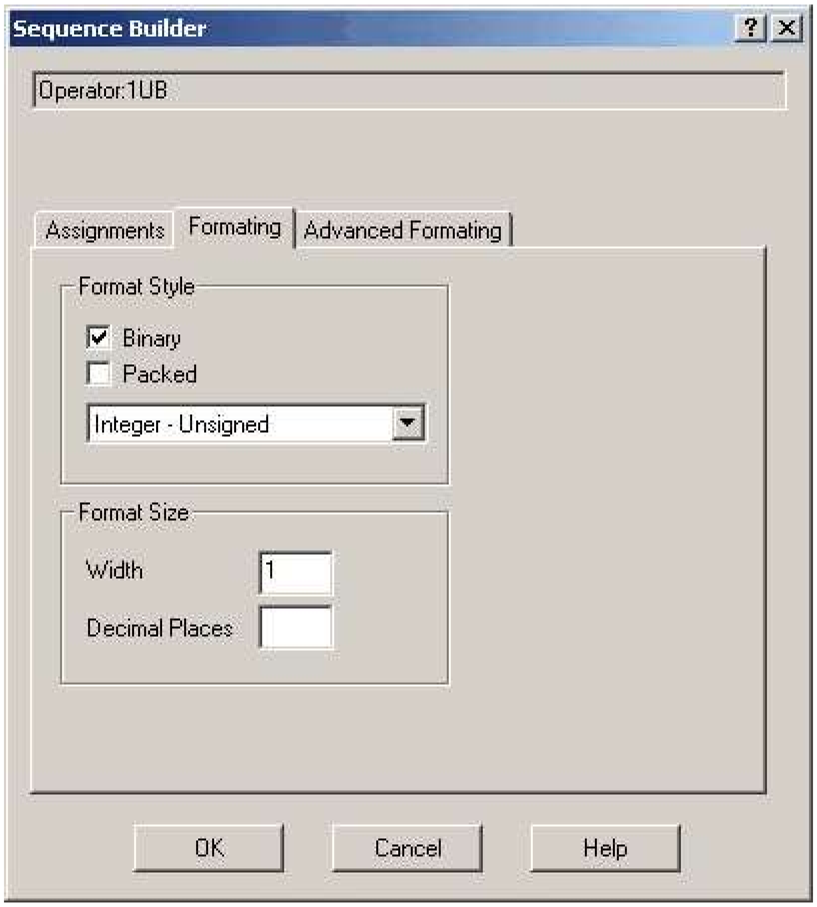
The length is the number of characters in the portion of the message that generates this code plus the size of its own field. Though this is not a true error detection code, it is useful in protocols that use counted, variable-length strings. The KLENA of the message ABCDE with KLENA formatted as a 2X is 7.

7.1.1.2.10.2 Formatting

Server Details : Configuration : Protocol : Sequence Builder

Formatting

The **Formatting** tab of the Sequence Builder is where you specify the format style and size of the selected assignment. Each format style has a letter representation, which appears at the top of the dialog box. In this way, once you learn what the different letters mean, you can tell at a glance what type of format is applied to the assignment.



Operator:1UB

Format Style

[✓] Binary

[ ] Packed

Integer - Unsigned

Format Size

Width: 1

Decimal Places:

The Sequence Builder: Formatting Tab Screen

The options available are:

- Format Style

  This informs the server how to interpret the data. By default, the data is interpreted in ASCII format, that is, data that can normally be viewed on a printer. This default can be changed in one of two ways:

  - **Binary (Format Code: B):** Reads the data in Binary format. Binary reads in the data not as an ASCII letter or number, but as the actual internal numeric representation. For example, the letter "A" is read in ASCII format as "A", but in Binary format as the decimal number 65, which is the internal representation for the letter "A".
  - **Packed (Format Code: P):** Packing is a method of packing 4 characters into 3 bytes. Each byte of data (8) bits is shrunk to

6 bits by masking off the upper two bits. ASCII characters (uppercase) between 0x40 and 0x5F are mapped to 0x00 to 0x1F. ASCII characters (digits) between 0x20 and 0x3F retain their value. Thus, any characters 0x20 and below or 0x60 and above can not properly be represented using packing ASCII.

- **Other Formats:** You can also change the default format of the selected assignment by using the Format Style pull-down box. The default format is **default**, meaning that the server will use the default format of the item.

  The list of supported formats are:

  - Default (Format Code: None)

    Uses the format specified by the assignment when created.

    If a bit mask, or not assigned, or topic variable, or EDC, or a single number, default is "I" for ASCII and "U" for binary.

    For items, it is based on the type. The server use "I" for Integer, "R" for Real, "S" for String, and "D" for Discrete.

    For register numbers, the format used is "I"

  - Integer Signed (Format Code: I)

    The data is treated as a Signed Integer. Based upon how the item is formatted, here is how a Signed Integer is parsed:

    For ASCII formats with a variable width: Allowed characters are "+-0123456789". All other are invalid

    For ASCII formats with a fixed width:: Skip leading spaces, +/- parsed, parse all digits. All other characters are invalid.

    For Binary formats, the bytes are treated as Signed, meaning that the most significant bit is the sign bit. So, for a one-byte item, the range of numbers is -127 to 127.

- Integer Unsigned (Format Code: U)

  Treats each number as an Unsigned Integer. Based upon how the item is formatted, here is how an Unsigned Integers parsed:

  For ASCII formats with a variable width: Allowed characters are "0123456789". All others are illegal.

  For ASCII formats with a fixed width: Skip leading spaces, +/- parsed, parse all digits. All other characters are illegal.

  For Binary formats, the bytes are treated as-is. So, for a one-byte item, the range of numbers is 0 to 255.

- Integer Signed Hex (Format Code: Y)

  Treats the data in Hexadecimal format, which is a base-16 counting system (Decimal format is a base-10 counting system). In Hexadecimal counting, the numbers 0 through 15 are represented by the numbers 0 through 9, then A, B, C, D, E, and F, where A=10, B=11, C=12, D=13, E=14, and F=15. Therefore, the decimal number 16 equals the hexadecimal number 10.

  This format uses the most significant bit to determine the sign of the data.

  For ASCII formats with a variable width: Allowed characters are "0123456789ABCDEFabcdef". All other characters are illegal. The value is sign extended based on number of digits processed.

  For ASCII formats with a fixed width: Skip leading spaces, +/- parsed, parse all digits between 0-9, A-F, and a-f. Stop when an invalid character reached. Value is sign extended based on format width.

  This format is ignored in Binary formats.

- Integer Unsigned Hex (Format Code: X)

  Treats the data in Hexadecimal format, which is a base-16 counting system (Decimal format is a base-10 counting system). In Hexadecimal counting, the numbers 0 through 15 are represented by the numbers 0 through 9, then A, B, C, D, E, and F, where A=10, B=11, C=12, D=13, E=14, and F=15. Therefore, the decimal number 16 equals the hexadecimal number 10.

  For ASCII formats with a variable width: Allowed characters are "0123456789ABCDEFabcdef". All other characters are illegal.

  For ASCII formats with a fixed width: Skip leading spaces, +/- parsed, parse all digits between 0-9, A-F, and a-f. Stop when an invalid character is reached.

  This format is ignored in Binary formats.

- Integer BCD (Format Code: C)

  Treats the data in Binary Coded Decimal (BCD) format. This format takes the two nibbles that makes up a byte and interprets the nibbles as a two-digit number. For example, the decimal number 65 returns the number "41" in BCD format (knowing that the hexadecimal representation of "65" is "41"). The decimal number 63 is ignored by the server, since "63" converts to "3F" in hexadecimal, and "F" is not a valid number.

  The equation for the calculation is "Value = Value * 100 + high nibble * 10 + low nibble".

  In ASCII formats, only "0" through "9" are valid characters, all other cause the server to ignore the data. Also, you will normally need two bytes in the format, one for each nibble.

  In Binary formats, any nibble value that contains "A" through "F" is rejected. This means that while "09" is valid, "0A" is not.

- Real Normal (Format Code: R)

  Treats the data as a Real (or Floating Point) number.

  For ASCII formats with a variable width: Allowed characters are "+-.0123456789". All others are illegal.

  For ASCII formats with a fixed width: Skip leading spaces, +/- parsed, parse all digits between 0-9. If a decimal point is found, then add that and all 0-9 digits following. If a "E" or "e" follows, then parse the +/- and the exponent digits. [+|-](0-9)[.(0-9)][e[+|-](0-9)]

  For Binary formats, the number is interpreted using IEEE floating point calculations. This must be four bytes in length, meaning that to insure the correct number, always use the format **:4R.**

- Real Exponential (Format Code: E)

  Treats the data as a Real (or Floating Point) number with scientific notation.

  For ASCII formats with a variable width: Allowed characters are "+-.0123456789Ee". All other characters are illegal.

  For ASCII formats with a fixed width: Skip leading spaces, +/- parsed, parse all digits between 0-9. If a decimal point is found, then add that and all 0-9 digits following. If a "E" or "e" follows, then parse the +/- and the exponent digits. [+|-](0-9)[.(0-9)][e[+|-](0-9)]

  This format is ignored in Binary formats.

- Real Hex (Format Code: H)

  Treats the data as a Real Hex number.

  For ASCII formats with a variable width: This format is not allowed.

For ASCII formats with a fixed width: The 4 bytes of the IEEE number are treated as an integer and formatted as hexadecimal (Format: X)

For Binary formats: The 4 bytes of the IEEE number are treated as an integer and formatted as hexadecimal (Format: X).

- Discrete Normal (Format Code: D)

  Treats the data as a Discrete Value. The server will then break the assignment block into the individual bits so that you can assign items on a bit level, instead of a byte level.

  With the "D" format, all formats follow the same rules: Data is extracted as if "I" format. The resulting value is tested. If non-zero, then the value is set to TRUE (1). If zero, then the value is set to FALSE (0). When being formatted, it is the values of 1 or 0 are formatted as an "I" format.

- String Normal (Format Code: S)

  Treats the data as a series of characters. The String format is commonly used to discard or skip data.

  For ASCII formats with a variable width: The character following the sequence defines the end of the string. All characters are allowed. However, some clients might not be able to handle some character values (such as the "Null" (0x00) value).

  For ASCII formats with a fixed width: All characters are accepted.

  For Binary formats: Treated as ASCII

- String Swap Word (Format Code: M)

  Treats the incoming string as a series of two-byte data, and swaps those two bytes before processing it. The length of the data must be divisible by 2. This works the same as the String (S) format, except that every pair of bytes are swapped.

For ASCII formats, both variable and fixed: All data is accepted.

This format is ignored in Binary formats.

- Unicode String (Format Code: W)

  Treats the data as a series of Unicode characters (UTF-16). Unicode strings are only allowed in outgoing messages. By default, the characters in the Unicode string are output lest significant byte first. This can be changed to most significant byte first by using the byte ordering option of "O21".

  For ASCII formats with a variable width: All characters contained in the string are sent to the device.

  For ASCII formats with a fixed width: The width specifies the number of bytes to send which must be a multiple of two. If the length of the string is less than the width divided by two, then spaces are appended to the end of the string. If the length of the string is longer than the width divided by two, then the string is truncated.

  For Binary formats: Treated as ASCII

- Other Linkage (Format Code: L)

  The Linkage format is a special format that instructs the server to treat the item as if it was to read in data (thus triggering off the default operation on Read-Type Command/Request (Host) Messages), but not to expect nor parse any data for it. The primary use of the Linkage format is to have the server automatically send messages to the device (by using the default operation of the Read-Type Command/Request (Host) Message) without expecting data in return.

For example, a heartbeat message can be sent to an device just to make sure that the server is still talking to it. Since no data is returned by the device, you use a Linkage format in the Response part of the Command/Request (Host) Message to trigger off the message. The server will then send out the message, but will not expect anything back into the assigned item.

This format has no function in the ASCII nor Binary formats.

- Format Size:

  This tells the server the number of bytes to expect for this format, or the number of bytes to output for this format.

  - **Width (Format Code: <number>):**

    The number of bytes. For data coming into the server from the device, this is the number of bytes to expect. The default is **Nothing**, meaning that the server will take all characters until a delimiter is found. This means that if you do not specify a width, the server will expect some hard-coded delimiter in the message stream after the assignment so that it will know when to stop putting characters into the selection. For this reason, you can never end a message stream with a selection that has not width (known as a **variable-length item**). For outputs, the server will send out just enough characters to fully express the item.

    If there is a width, then the server will expect that many bytes from the device, or will send out that many bytes to the device. If the data is less than the number of bytes, the server will pad with spaces.

  - **Decimal Places (Format Code: .<number>):**

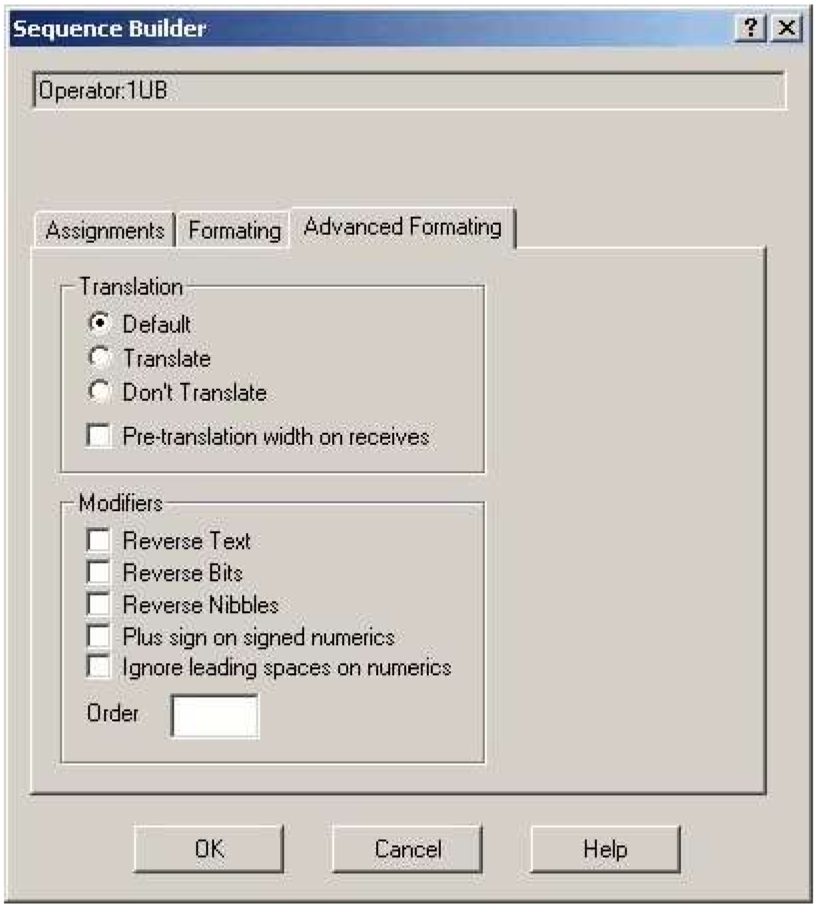
    Applying to the Real Format only, this is the number of decimal places to expect from the device or the number to output to the device. The default is **Nothing**, meaning that a variable number of bytes is expected.

7.1.1.2.10.3 Advanced Formatting

Server Details : Configuration : Protocol : Sequence Builder

Advanced Formatting

The **Advanced Formatting** tab of the Sequence Builder is where you specify how the server is to treat the data, that is, how the server is to format the data after reading it in or how to format it before it goes out.



The Sequence Builder: Advanced Formatting Tab Screen

The options available are:

- **Translation:** This tells the server how to translate the data based upon the Translations set up within the protocol. The options are:

  - Default (Format Code: None):

    Process translations according to the defaults set up in the Translations.

    - Translate (Format Code: T+):

      Translate this data, according to the rules set up in the Translations.

    - Don't Translate (Format Code: T-):

      Never translate this data.

    - Pre-translation width on receives (Format Code: K):

      If this checkbox is checked, then when counting the number of characters in the item, the pre-translation length is used. Otherwise, the post-translation length is use.

      The default is **not checked**, meaning that the server will use the post-translation length.

- **Modifiers:** In addition to the translations, this tells the server how to modify the data before it goes to the device or after it comes in from the device.

    - Reverse Text (Format Code: V):

      Reverses the text string. For example, if the data is "ABCD", then the server will reverse the data bytes and return "DCBA".

      This format cannot be used with the Binary or Linkage formats.

    - Reverse Bits (Format Code: Z):

      Reverses the bits within each byte. For example, if the bit pattern in a byte was "11000100", then the server will reverse these bytes and return "00100011".

    - Reverse Nibbles (Format Code: N):

Reverses the nibbles within each byte. For example, if the bit pattern in a byte was "11000100", then the server will reverse these nibbles and return "00100011".

- Ignore Leading Spaces on Numerics (Format Code: Q):

  Ignores leading spaces on numeric items (Real and Integer) when using an ASCII variable-width format.

- Order (Format Code: O):

  Changes the order of bytes in the specified selection in the order spelled out in the text box. For example, suppose you had the string "ABCD", and you wanted the 2nd byte, then the 3rd byte, then the 1st, and finally the 4th byte. You will then type in the number "2314" in the Order box.

  This format is only available on 2, 3, or 4-byte strings.

- Plus sign on signed numerics (Format Code: +):

  Forces the server to output a plus sign for signed numerics. Used only when formatting data. When receiving data, this format is ignored.

  This format is only available with the I (Integer), R (Real), and E (Real-Exponential) formats.

  The default for this is **not checked**.

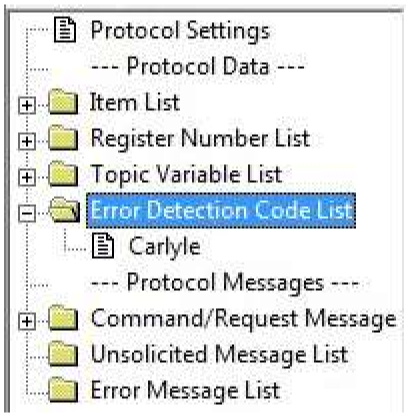
7.1.1.2.11 Error Detection Code Builder

Server Details : Configuration : Protocol

Error Detection Code Builder

An **Error Detection Code** is a calculation on a set of data that is used to insure that the data transmitted was without error.

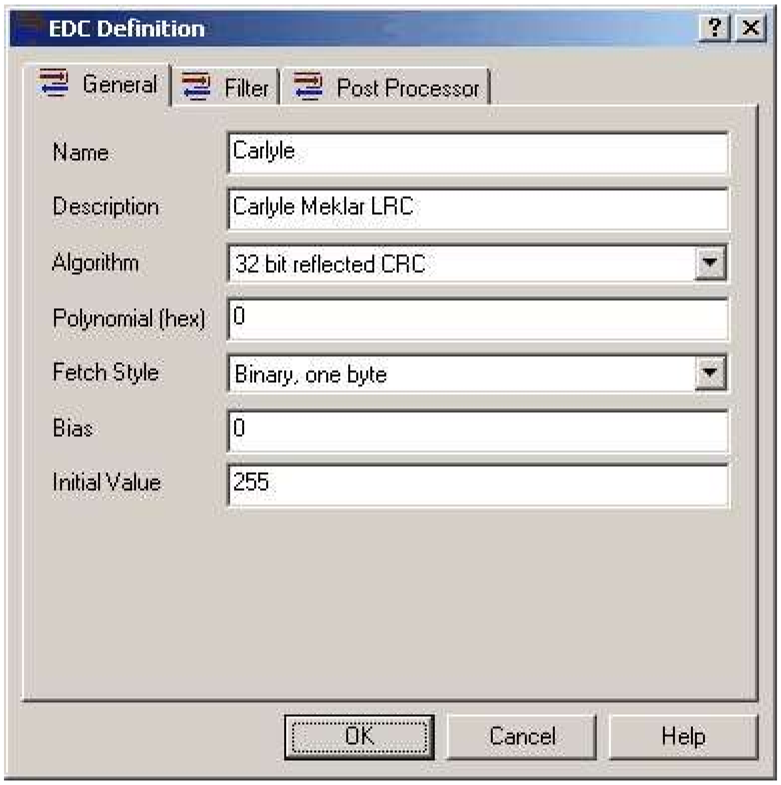
There are a base number of these error detection codes (or EDCs) available for your use. You can find a list here. However, there are times where you will need to define a variant of an existing code, or even a completely new code.



The EDC Tree

To help you in the design of a new EDC, this server has what is known as an **EDC Builder**. To access the EDC builder, create a new Error Detection Code object from the menu bar, tool bar, or the popup menu.

The initial EDC Builder screen will look something like this:



Name	Carlyle
Description	Carlyle Meklar LRC
Algorithm	32 bit reflected CRC
Polynomial (hex)	0
Fetch Style	Binary, one byte
Bias	0
Initial Value	255

The Initial EDC Builder Screen

To learn about the different aspects of the EDC Builder, just follow one of the links below.

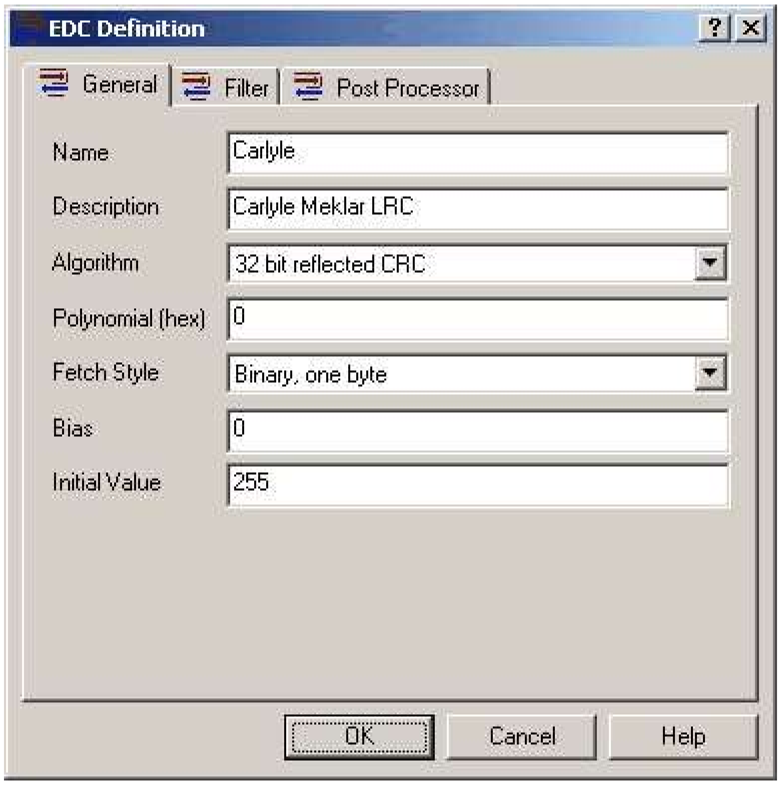
- **The General Tab:** Here, you name the EDC and specify the basic type of EDC.
- **The Filter Tab:** Use this tab to specify any filters used in the EDC.
- **The Post Processor Tab:** Informs the server what additional calculations will need to be done after the initial calculation and filters are applied.

7.1.1.2.11.1 General Tab

Server Details : Configuration : Protocol

EDC Builder : General

The **General** tab of the EDC Builder is where you specify the basic type of EDC, and the data that goes with it.



Name: Carlyle
Description: Carlyle Meklar LRC
Algorithm: 32 bit reflected CRC
Polynomial (hex): 0
Fetch Style: Binary, one byte
Bias: 0
Initial Value: 255

The EDC Builder: General Tab Screen

The options available are:

- **Name:** The name of the EDC. This must be unique among the server.
- **Description:** A description of the EDC. This is for documentation purposes only, and is not used by the server.
- **Algorithm:** The basic calculation to be made on the data. The types are:

  - 8-Bit CRC:

    8 bit CRC with the polynomial specified as a 8 bit hex value. Requires a Polynomial.

  - 8-Bit Reflected CRC:

    8 bit reflected CRC with the polynomial specified as a 8 bit hex value. Requires a Polynomial.

  - 16-Bit CRC:

    16 bit CRC with the polynomial specified as a 16 bit hex value. Requires a Polynomial.

  - 16-Bit Reflected CRC:

    16 bit reflected CRC with the polynomial specified as a 16 bit hex value. Requires a Polynomial.

  - 32-Bit CRC:

    32 bit CRC with the polynomial specified as a 32 bit hex value. Requires a Polynomial.

  - 32-Bit Reflected CRC:

    32 bit reflected CRC with the polynomial specified as a 32 bit hex value. Requires a Polynomial.

- Honeywell CRC:

  Variation of the CRC-CCITT standard CRC used by many Honeywell devices. No other options are required.

- 8-Bit Checksum, 8-bit Accumulator:

  Checksum of data in 8 bit bytes using an 8 bit accumulator. No other options are required.

- 8-Bit Checksum, 32-bit Accumulator:

  Checksum of data in 8 bit bytes using a 32 bit accumulator

- 16-Bit Checksum, 16-bit Accumulator:

  Checksum of data in 16 bit words using a 16 bit accumulator

- 8-Bit LRC:

  8-bit LRC. Also known as XOR.

- 8-Bit LRC Rotated Left:

  8 bit LRC with result shifted left one bit after each byte is added to the LRC

- 8-Bit LRC Rotated Right:

  8 bit LRC with result shifted right one bit after each byte is added to the LRC

- 16-Bit LRC:

  16-bit LRC. Also known as XOR.

- 16-Bit LRC Rotated Left:

  16 bit LRC with result shifted left one bit after each byte is added to the LRC

- 16-Bit LRC Rotated Right:

  16 bit LRC with result shifted right one bit after each byte is added to the LRC

- Special Checksum Combo and LRC Algorithm 1:

  The high-order byte of the CSUML is the LRC value while the low-order byte is the standard checksum plus the value of the LRC byte.

- Special Checksum Combo and LRC Algorithm 2:

  LRC value as the low-order byte and the standard checksum in the high-order byte.

- Special Checksum Combo and LRC for UKCOMM:

  24-bit value consisting of the concatenation of a special 8-bit LRC, the standard checksum, and the standard LRC. The special LRC simply shifts the intermediate value up one bit after each exclusive or operation.

- Special 16-bit CRC 1:

  Based on the standard method of computing CRCs, this handles the XORing of the data into the intermediate CRC differently.

  Following is the code used to compute the this CRC:

  ```
  ErrorCode := seedValue
  DataCode := 0;
  for each character c in the message
  ```

```
    ErrorCode := ErrorCode xor c
    do 8 times
        bit :=most significant bit of ErrorCode
        ErrorCode := up shift of ErrorCode
        if bit is on
            ErrorCode := ErrorCode xor generatingPolynomial
        end if
    end do
    DataCode := up shift of DataCode by 8 bits
    DataCode := DataCode or c
    ErrorCode := ErrorCode xor DataCode
end for
```

- Special 16-bit CRC 2:

  Ignores the leading ASCII DLE, or hexadecimal 10, of a pair of characters.

  Any character following a ASCII DLE, or hexadecimal 10 is biased by 1. This addition of 1 is allowed to overflow into high byte of the CRC computation.

  For example, the pair 10 FF would cause the 10 to be ignored and FF+1 or 100 to used in the CRC computation. Output Value: Ones complemented. Due to a bug in the implementation, the CRC is sign extended to a 32 bit value. To work around this problem, use signed formatting instead of unsigned formatting.

- Special Decimal-Encoded Checksum:

  8 bit checksum with the resulting value formatted for zero based encoding.

  Zero-based encoding takes a byte, splits it into two 4-bit hexadecimal digits, adds a hexadecimal 30 or ASCII 0 to each, and concatenates the two into a 16-bit result. For example, the hexadecimal byte value of 4F become the 34 3F, or ASCII 4?.

- Special Hex 40 Encoded 12-bit Checksum:

  Computes the ones complement of the standard 12-bit checksum. The 12-bit result is split into two 6-bit numbers, a hexadecimal 40 added to each number. Each of these numbers then are concatenated into a 16-bit result. This yields two characters in the printable ASCII range of hexadecimal 40 to 7F.

- Special Decimal-Encoded 8-bit CRC 1:

  8 bit checksum of the input with the 8th bit of each data byte removed. The upper two bits of the result are Exclusive ORed with the lower two bits of the sum. The upper two bits are then cleared. ASCII ‘0’ is then added to the final value.

- **Polynomial (hex):** The value applied to a CRC error detection code. Only available for the "CRC"-style codes.
- **Fetch Style:** In addition to the translations, this tells the server how to modify the data before it goes to the device or after it comes in from the device.

  - Binary, One Byte:

    Each byte of the input stream is supplied to the EDC as a binary value 0-255.

  - Binary, Two Byte:

    Each pair of bytes are supplied to the EDC as a binary value 0-65535.

  - Decimal, One Digit:

    Each byte of the input stream is interpreted as a decimal value and supplied to the EDC as a binary value 0-9. Non-decimal digits are interpted as 0.

  - Hex, One Digit:

Each byte of the input stream is interpreted as a hex value and supplied to the EDC as a binary value 0-16. Non-hex digits are interpreted as 0.

- Hex, One Digit with Special Conversion:

  Each byte of the input stream is interpreted as a special hex character and sent to the EDC as a binary value 0-56. Digits 0-9 are interpreted as binary 0-9. Letters A-F are interpreted as binary 10-16. Letters a-f are interpreted as binary 10-16. Letters g-z are interpreted as "G" - "Z" or binary 47-56.

- Hex, Two Digits:

  Pairs of hex digits from the input stream are supplied to the EDC as a binary value 0-255. Non-hex digits are interpreted as 0. The string "012315" is sent to the EDC as the values 1, 35 and 21.

- Hex, Two Swapped Digits:

  Pairs of hex digits from the input stream are supplied to the EDC as a binary value 0-255. However, the digits are interpreted in reversed order. Non-hex digits are interpreted as 0. The string "012315" is sent to the EDC as the values 16, 50 and 81.

- Hex, Four Digits:

  4 hex digits from the input stream are supplied to the EDC as a binary value 0-65535. Non-hex digits are interpreted as 0.

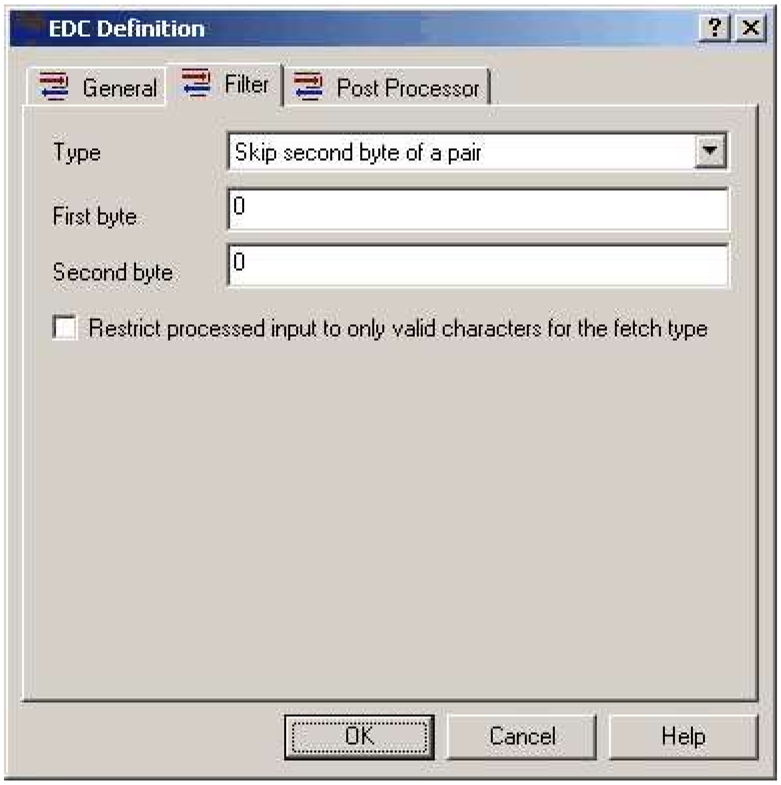
- **Bias:** A bias set upon the calcualtion.
- **Initial Value:** Starting seed value for the EDC.

7.1.1.2.11.2 Filter Tab

Server Details : Configuration : Protocol

EDC Builder : Filter

The **Filter** tab of the EDC Builder is where you specify how the data will be "filtered" from the data while the EDC is being calculated.



Type: Skip second byte of a pair
First byte: 0
Second byte: 0
☐ Restrict processed input to only valid characters for the fetch type

The EDC Builder: Filter Tab Screen

Please note that the two text boxes underneath the **Type** option may not be available, depending upon the option selection. Please refer to the documentation below to determine what's available at what time.

The options available are:

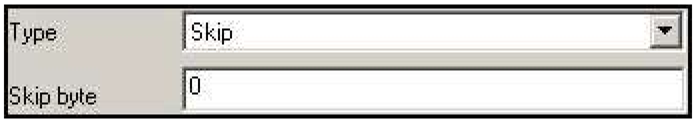
- **Type:** The type of filter to be applied to the data. The types are:

  - **None:**

    No filters are applied to the data. The two text boxes are not available.

- **Skip:**

  If a byte has the specified value (in decimal), it is ignored as part of the EDC.



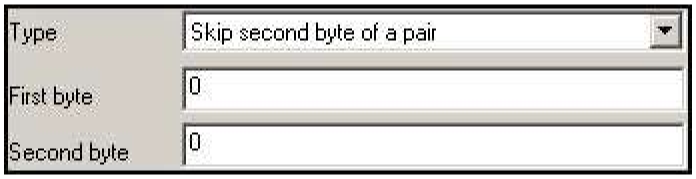
Type: Skip
Skip byte: 0

The Skip Byte Filter

  Enter the byte value of the number to skip. The second text box is unavailable.

- **Skip Second Byte of a Pair:**

  If the pair of the first byte followed by the second byte (both in decimal) are found in the EDC data, then the second byte is ignored.



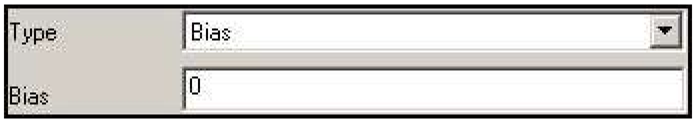
Type: Skip second byte of a pair
First byte: 0
Second byte: 0

The Skip Second Byte Filter

  Enter the byte value of the first byte in the first text box and the value of the second byte in the second text box.

- **Bias:**

  Bias each byte of the input stream by the given value (in decimal).



Type: Bias
Bias: 0

The Bias Filter

  Enter the bias. The second text box is unavailable.

- **Ignore Control Character (0-31):**

Ignores all control characters, that is, characters with ASCII values from 00 to 31 (00h to 1Fh). The two text boxes are not available.

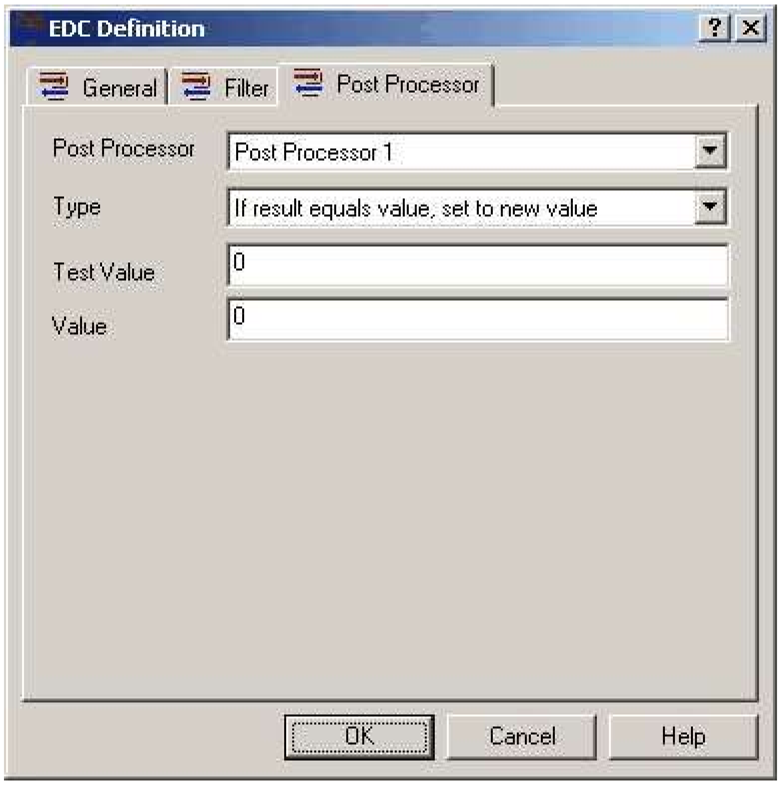
- **Restrict processed input...:** Restricts the filter process to only valid characters of the fetch type.

7.1.1.2.11.3 Post Processing Tab

Server Details : Configuration : Protocol

EDC Builder : Post Processor

The **Post-Processor** tab of the EDC Builder does additional processing to the EDC once the initial calculation has been completed.



Post Processor: Post Processor 1
Type: If result equals value, set to new value
Test Value: 0
Value: 0

The EDC Builder: Post Processor Tab Screen

Please note: The final two text boxes are only available depending upon what has been selected in the pull-down boxes of the "Type" option. The options available are:

- **Post-Processor:** The number of the post-processing. You can have two post-processors defined.

- **Type:** The type of post-processing. The types are:

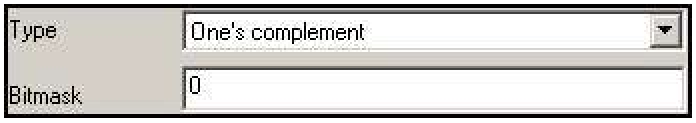
  - **None:**

    No post processing.

  - **Append 0 data byte if length is odd:**

    If the length of the EDC data is odd, then append a 0 to the EDC.

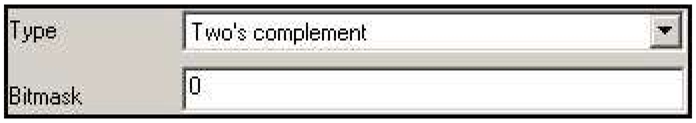
  - **One's Compliment:**

Type	One's complement
Bitmask	0

    Compute the one's complement of the end result. The parameter is a bit mask in decimal of which bits to compute the one's complement. (i.e. 255 for a byte, 65535 for 2 bytes...)

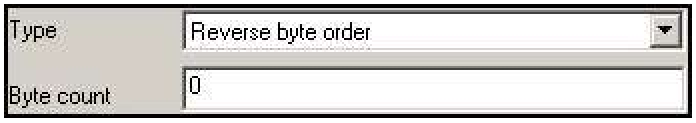
  - **Two's Compliment:**

Type	Two's complement
Bitmask	0

    Compute the two's complement of the end result. The parameter is a bit mask in decimal of which bits to include in the two's complement.

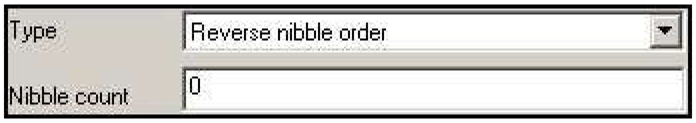
  - **Reverse byte order:**

Type	Reverse byte order
Byte count	0

    Reverse the byte order of the end result. The parameter is the number of bytes. Only 2 and 4 is supported.

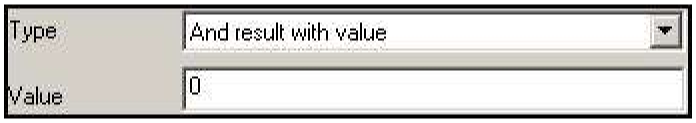
- **Reverse nibble order:**



Type	Reverse nibble order
Nibble count	0

Reverse the nibble order of the end result. The parameter is the number of nibbles. Only 2, 4, and 8 are supported.

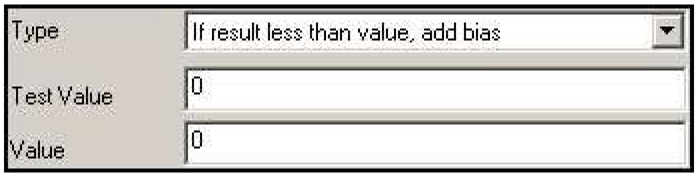
- **AND result with value:**



Type	And result with value
Value	0

And the result with the given bit mask. The parameter is the bit mask in decimal.

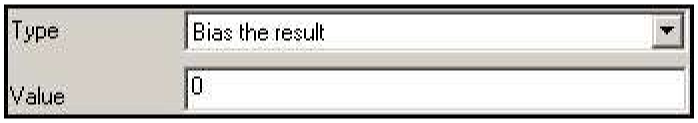
- **If result less than value, add bias:**



Type	If result less than value, add bias
Test Value	0
Value	0

If the end result is less than the first parameter, then the second parameter is added to the end result.

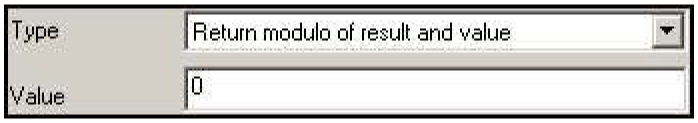
- **Bias the result:**



Type	Bias the result
Value	0

Add the first parameter to the end result.

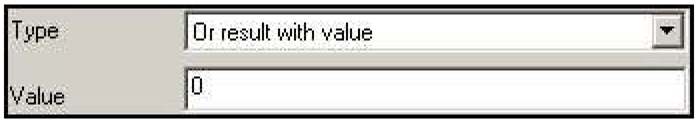
- **Return modulo of result and value:**



Type	Return modulo of result and value
Value	0

Return the modulo of the end result and the first parameter.

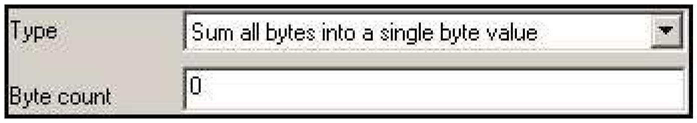
- **OR result with value:**



Type	Or result with value
Value	0

Or the result with the given bit mask. The parameter is the bit mask in decimal.

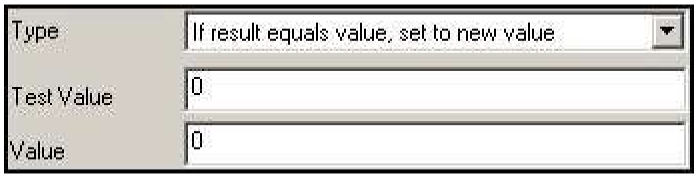
- **Sum all bytes into a single byte value:**



Type	Sum all bytes into a single byte value
Byte count	0

Sum the specified number of bytes of the end result into a final single byte. The parameter is the number of bytes. Only 1, 2, 3, and 4 are supported.

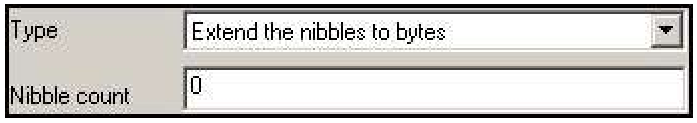
- **If result equals value, set to new value:**



Type	If result equals value, set to new value
Test Value	0
Value	0

If the end result equals the first value, then the second value is used.

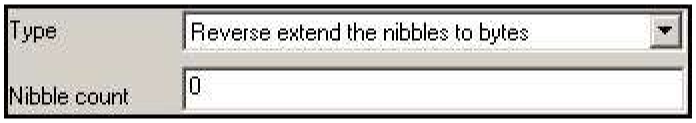
- **Extend the nibbles to bytes:**



Type	Extend the nibbles to bytes
Nibble count	0

Shift the nibbles of the end result into their own bytes. The parameter specifies the number of bytes. Only 1, 2, 3, and 4 are supported.

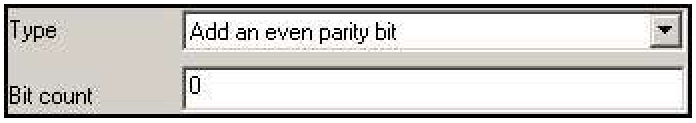
- **Reverse extend the nibbles to bytes:**



Type	Reverse extend the nibbles to bytes
Nibble count	0

Shift the nibbles of the end result into their own bytes but in reverse order. The parameter specifies the number of bytes. Only 1, 2, 3, and 4 are supported.

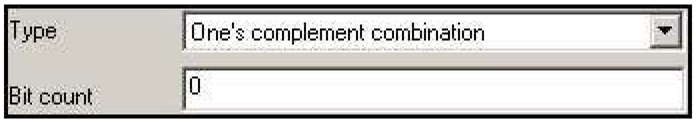
- **Add an even parity bit:**



Type	Add an even parity bit
Bit count	0

Add a even party bit to the end result. The parameter is the number of bits. Thus, if an parity bit is to be added to 7 bits of data, then 7 should be specified resulting in 8 bits of total data.

- **One's compliment combination:**



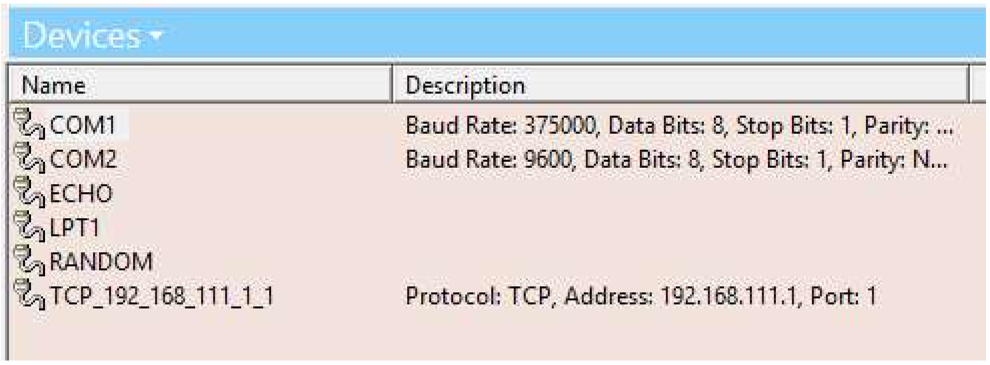
Type	One's complement combination
Bit count	0

The end result is masked by the number of bits specified in the parameter. Then a new value is computed being of 2*parameter bits long. The top half of the value is the masked end result. The lower half is the one's complement of that value.

7.1.2 Devices

So what is a server device? A Server Device is a connection to a physical or virtual communications port.

For example, if you had one physical serial port (COM1) and one virtual serial port mapped to a USB device (COM3) in your computer, then you can create two OmniServer devices, also called COM1 and COM3, respectively. Or, if the device is an Ethernet device, you would have create an OmniServer Ethernet device for each IP address of a device on the network.



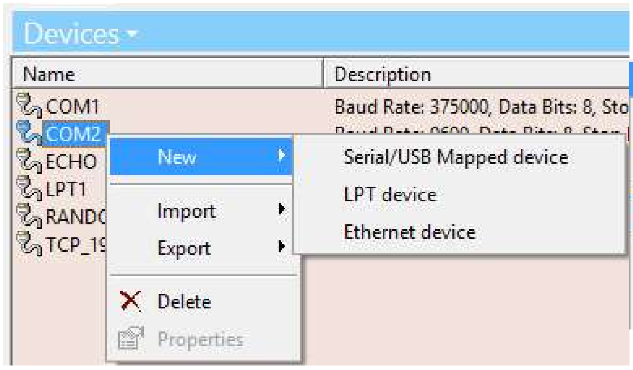
Devices

Name	Description
COM1	Baud Rate: 375000, Data Bits: 8, Stop Bits: 1, Parity: ...
COM2	Baud Rate: 9600, Data Bits: 8, Stop Bits: 1, Parity: N...
ECHO	
LPT1	
RANDOM	
TCP_192_168_111_1_1	Protocol: TCP, Address: 192.168.111.1, Port: 1

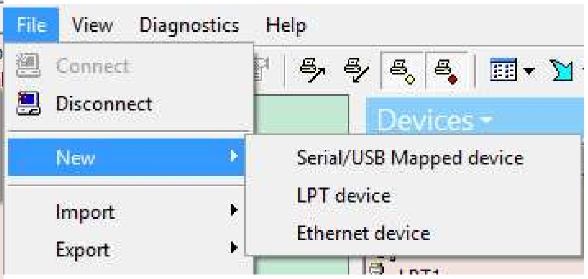
Sample Devices Screen

It is important to note that the OmniServer device is only a **representation of the physical device** in that it doesn't physically create the device for you. For example, you can define up to 256 Serial devices, labeled COM1, COM2, ..., COM256. But if only COM1 and COM2 actually exist on your computer (in Device Manager), you will get errors for the other 254 devices.

You can create three different kinds of devices by selecting one of the following:



This menu is available by right-clicking inside the View Window.

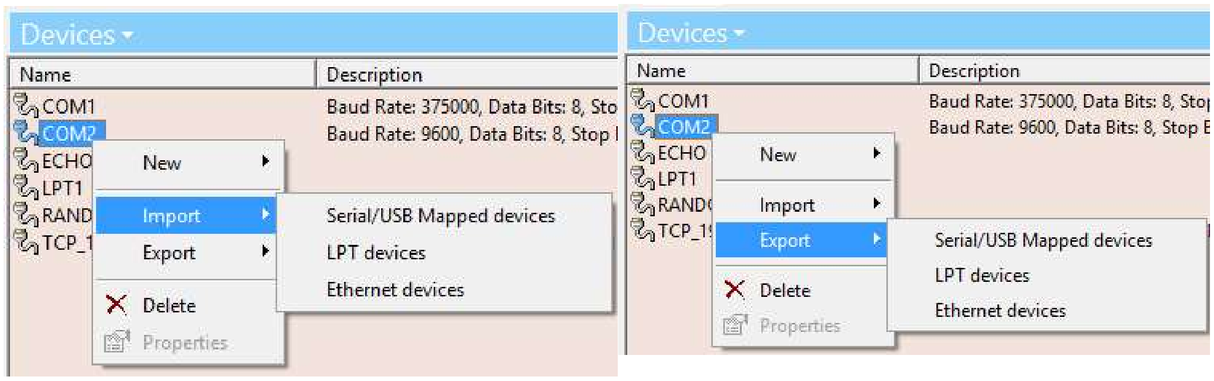


This menu is available from the server's menu bar.

Device Import / Export

Devices can be imported from a properly formatted CSV file or exported to a CSV file. Devices are imported/exported by type, so all COM devices can be imported or exported and all Winsock devices can be imported or exported, etc. This functionality makes it easier to merge configurations and to perform mass edits.

Import/Export options are available either from the File menu while in the Devices view, by right-clicking in the white space in the Devices view or by clicking the Import button or Export button in the toolbar while in the Devices view.



To ensure you have a properly formatted CSV file, it is recommended to create one instance of the device type you require in OmniServer (Serial/USB Mapped, Ethernet or LPT) and then Export that device to CSV.

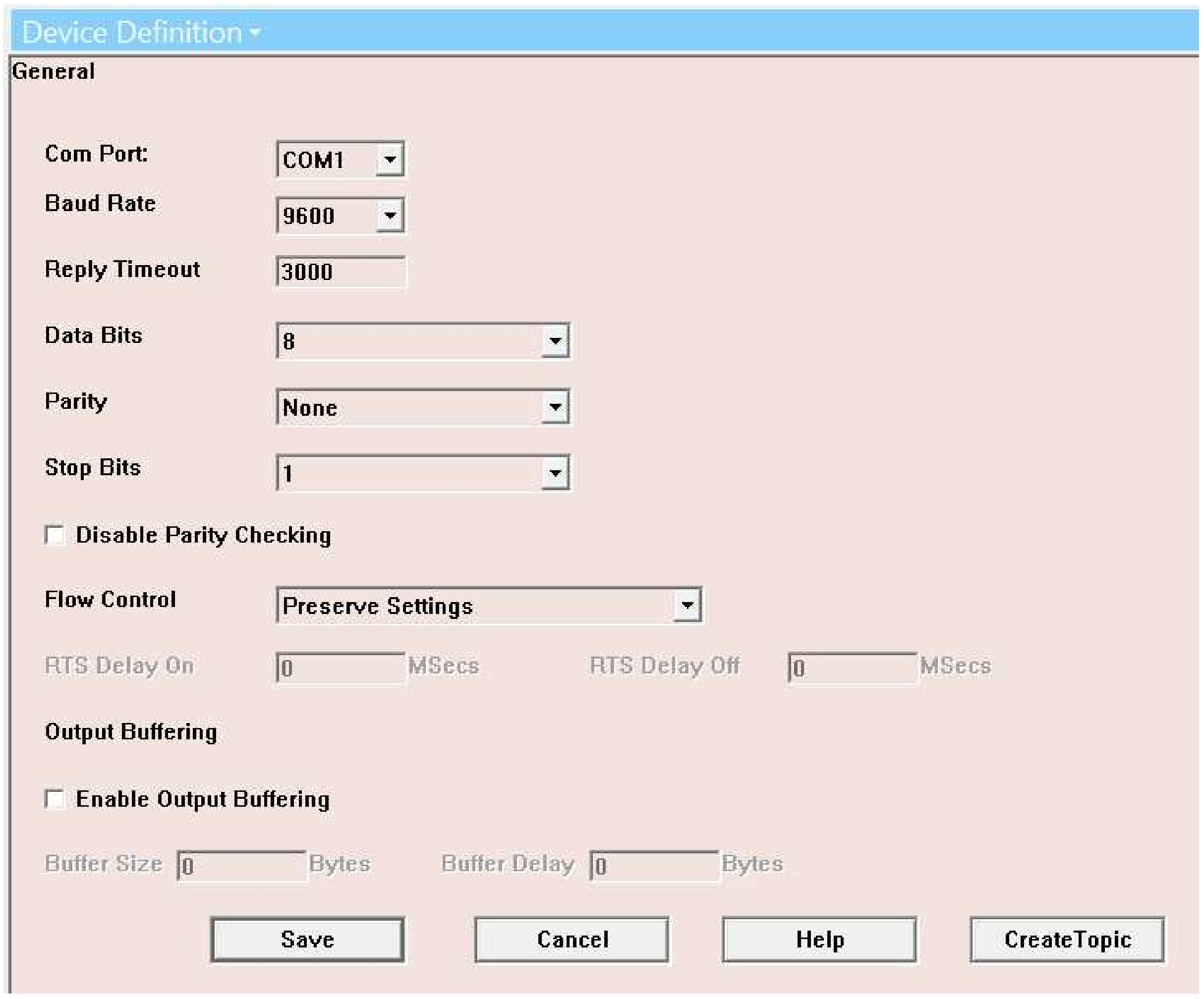
The three possible server devices are:

- Serial/USB Mapped: Also called COM ports (this can also include USB devices mapped to a virtual COM port in the operating system).
- LPT: Also called Printer ports, these devices are write-only. You cannot read data from these devices.
- Ethernet: Also known as Winsock, you can have three different connections: TCP; UDP; and TELNET (including serial encapsulation devices).

7.1.2.1 Serial/USB Mapped

A serial, or COM, device is used when you need RS-232, RS-422, or RS-485 communications or when a USB device has been mapped to a virtual

COM port in the operating system. Setting up a serial device is relatively easy.



Com Port:	COM1
Baud Rate	9600
Reply Timeout	3000
Data Bits	8
Parity	None
Stop Bits	1
☐ Disable Parity Checking	
Flow Control	Preserve Settings
RTS Delay On	0 MSecs
RTS Delay Off	0 MSecs
Output Buffering	
☐ Enable Output Buffering	
Buffer Size	0 Bytes
Buffer Delay	0 Bytes

The General Screen.

The options are:

- **COM Port:** You can select up to 256 different COM ports. Please note that this is a **representative device** in that it doesn't actually create the port if it doesn't exist on your computer.

- **Baud Rate, Data Bits, Parity, and Stop Bits:** The communications parameters for the port. This is dependent upon the type of device connected to this port. Check with your device's documentation for details on what values these parameters must contain.

- **Reply Timeout:** This value informs the server how long to wait for a response from the device until it gives up and continues normal operations. By default, the server will try three times to get a successful response from the device (if one is required) before it aborts the attempt.

- **Disable Parity Checking:** Some devices do not care about the Parity Bit setting, and therefore will send out random bits (or always a Zero or One bit) in place of the Parity bit. If the server is checking Parity, the message will fail and the server will report and error where none exist. Checking this box will force the server not to check for parity.

  Disable Parity Checking is a **rare** option. This box should always remain unchecked unless the device's documentation specifically mentions that Parity is not used.

⊟ Flow Control: The server supports seven different flow control models. Click on Flow Control for the list.

The Seven Flow Control Settings:

Preserve Settings: Keep the settings currently used by the operating system.

None: No flow control.

DTR/DSR: Use Data Terminal Ready / Data Set Ready flow control. This is a hardware flow control that monitors the DTR and DSR lines on the serial port.

RTS/CTS: Use Read to Send / Clear to Send flow control. This is a hardware flow control that controls the RTS line on the serial port.

XON/XOFF: Use Transmit On / Transmit Off flow control. This is a software flow control that sends a CTRL-Q (0x16) to signal a start of a transmission (XON), and a CTRL-S (0x18) to signal the end of a transmission (XOFF).

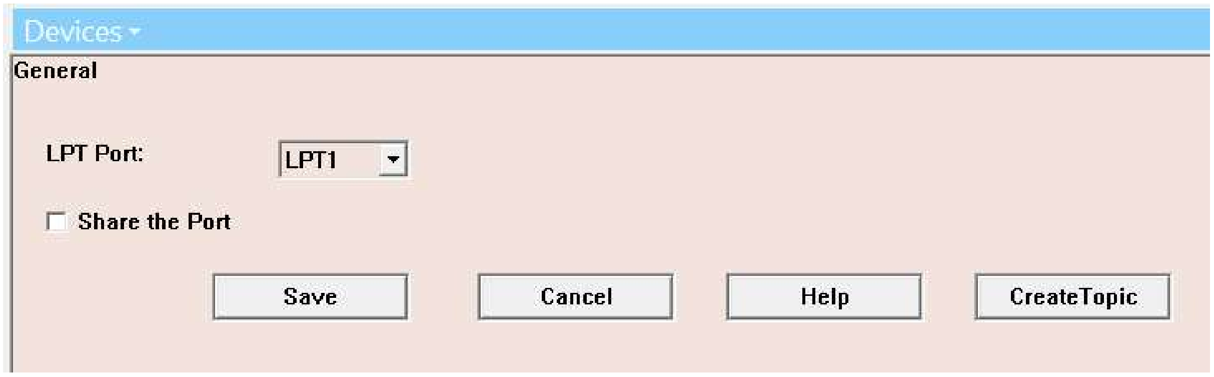
RTS Toggle: Signals that the RTS line is toggle between each server transmission.

RTS Delay: Signals that a delay is encountered in conjunction with the RTS line being toggled on or off. The delay times are specified with the RTS Delay numbers (see below).

- **RTS Delay:** Used only by the RTS Delay flow control. Sets up a delay between the toggles of the RTS line.
- **Output Buffering:** In some instances, an device can only take a certain number of characters, even though an individual data stream may contain more characters. Setting this option will tell the server the size of that buffer, and how long to wait after processing that buffer before asking the device for more data.
- **Create Topic button:** Clicking this button will initiate creation of a new Topic using the current device (You will then need to define the other parameters of a topic including the topic name and associated protocol for the topic).

7.1.2.2 LPT

A Parallel Port is more commonly called the Printer, or LPT, port. The server uses this port as a **write-only** port. That is, data can be sent from the server to this device, but no data can be sent from the device to the server.



LPT Port: LPT1
Share the Port: [ ]

The LPT Device Settings View

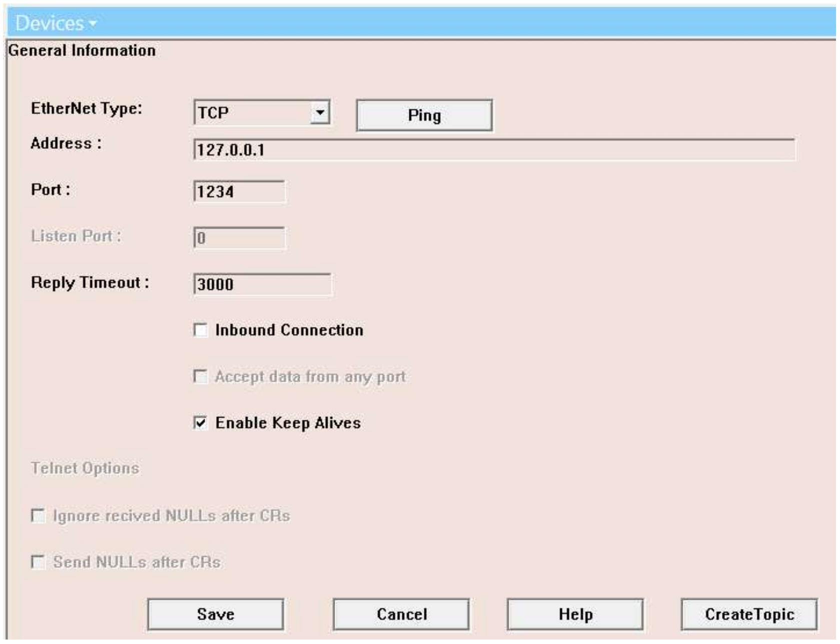
You can select up to nine different LPT ports (numbered 1 to 9).

- **Share the Port:** when enabled, this tells the server that other programs will want to use this port. By default, the server takes exclusive control of the port, not allowing other programs (such as a word processor) to access the port. While this desirable for Serial ports, it is not for Parallel ports, simply because other programs will need to print. Set this option to allow other programs to use the port.
- **Create Topic button:** Clicking this button will initiate creation of a new Topic using the current device (You will then need to define the other

parameters of a topic including the topic name and associated protocol for the topic).

7.1.2.3 Ethernet

An Ethernet device is simply a connection to a network using what is known as an IP address or DNS name. The number of these devices that can be created is limited by the operating system and available computer resources. Serial devices being accessed through an Ethernet encapsulation device are also configured via the OmniServer Ethernet device.



General Information

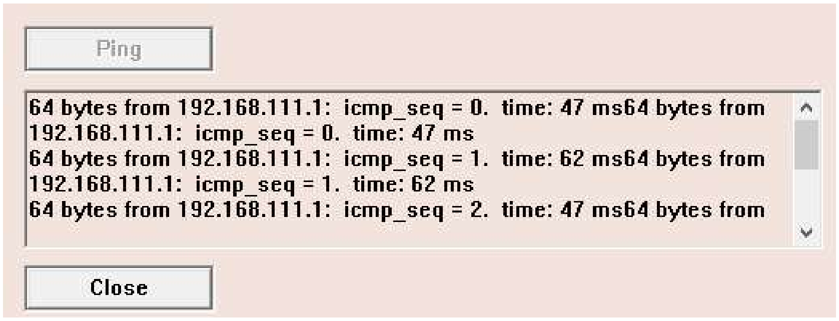
EtherNet Type: TCP
Address : 127.0.0.1
Port : 1234
Listen Port : 0
Reply Timeout : 3000
☐ Inbound Connection
☐ Accept data from any port
☑ Enable Keep Alives
Telnet Options
☐ Ignore recived NULLs after CRs
☐ Send NULLs after CRs

Sample Ethernet Device Settings View

The options on this screen are:

- **Type:** You can select one of three types of Ethernet connections based on the transport required by your Ethernet device: TCP, UDP or Telnet.
- **Ping:** The Ping button will "ping" the address to check if the server can reach it. When you click

on the Ping button, you will receive this results view:



```
64 bytes from 192.168.111.1:  icmp_seq = 0.  time: 47 ms64 bytes from
192.168.111.1:  icmp_seq = 0.  time: 47 ms
64 bytes from 192.168.111.1:  icmp_seq = 1.  time: 62 ms64 bytes from
192.168.111.1:  icmp_seq = 1.  time: 62 ms
64 bytes from 192.168.111.1:  icmp_seq = 2.  time: 47 ms64 bytes from
```

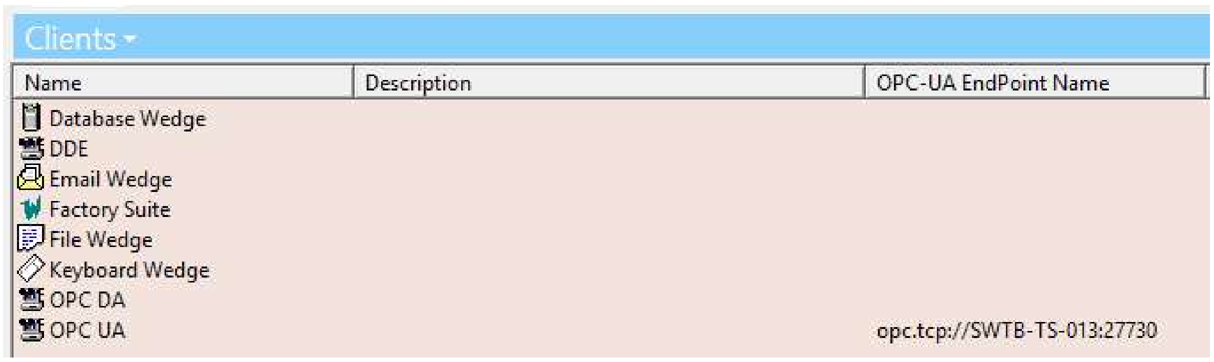
The Ping Results View

- **Address:** This can be any IP address, either in the "xxx.xxx.xxx.xxx" form, or the DNS address form, like "www.omnidssi.com".
- **Port:** The port number for the device. This is not active with TCP or TELNET types if the Inbound Connection box is checked.
- **Listen Port:** Only active with a UDP connection, or when the Inbound Connection box is checked. Tells the server in which port to "listen".
- **Reply Timeout:** Tells the server how long to wait for a response from the device until it gives up and continues on to the next server operation.
- **Inbound connection:** This instructs the server to accept connections from remote devices instead of actively going out and making the connection. This only applies to TCP/TELNET
- **Accept Data from Any Port:** Ignores all port assignments, and will accept data from any port at the address. This is useful for devices in which no port configuration is available. This only applies to UDP
- **Enable Keep Alives:** Sends a TCP KEEP ALIVE signal to the Winsock connection on a periodic basis. This is primarily useful to protocols which only accept Unsolicited Messages. Since no signal will be sent from the device when it goes down,

the server normally will not know when that device goes down. Enabling this option will cause the server to send a signal to see if the device is still on-line.

- **Ignore Received NULLs after CR:** In TELNET communications, some devices will send a NULL character after each CR. This checkbox instructs the server to ignore this character.
- **Send NULLs after CR in binary mode:** In TELNET communications, this instructs the server to send out a NULL character after each CR.
- **Create Topic button:** Clicking this button will initiate creation of a new Topic using the current device (You will then need to define the other parameters of a topic including the topic name and associated protocol for the topic).

7.1.3 Clients



Name	Description	OPC-UA EndPoint Name
Database Wedge		
DDE		
Email Wedge		
Factory Suite		
File Wedge		
Keyboard Wedge		
OPC DA		
OPC UA		opc.tcp://SWTB-TS-013:27730

Clients View

The Clients screen shows the types of clients to which the server can communicate.

Current Clients Supported by the Server

[This section applies only to the OmniServer Server Edition and the OmniServer Professional Edition.]

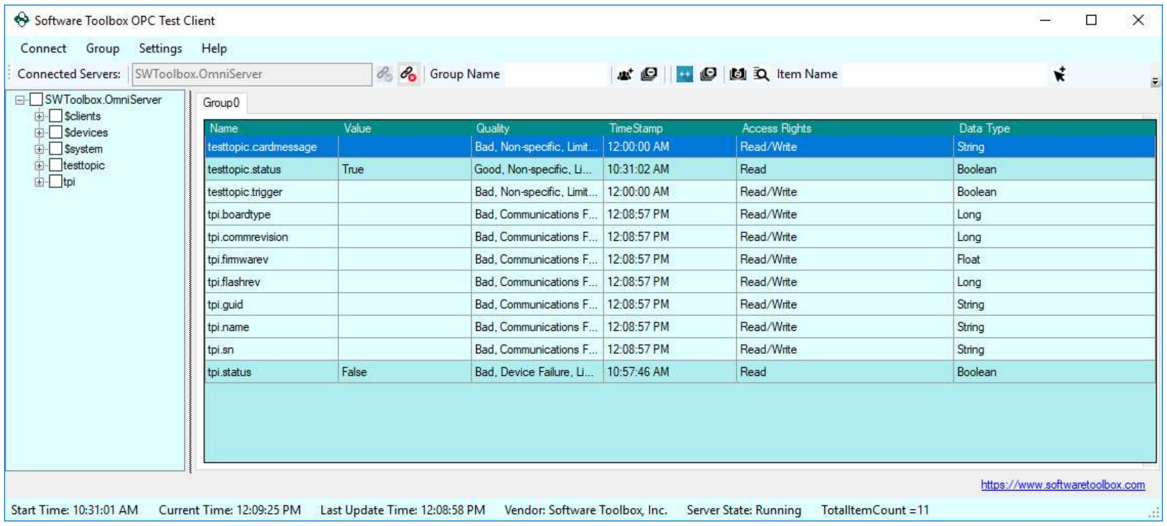
- DDE - Displays information about the application name for DDE communications.

- Factory Suite (Wonderware Suitelink) - Displays information concerning the application name for communicating using Wonderware Suitelink.

- OPC DA - Displays information about the application name (ProgID) for OPC DA communications.

  - **NOTE:** OmniServer installs with the Software Toolbox OPC Test Client which support OPC DA for testing purposes. It can be launched using the toolbar button in the OmniServer Configuration interface or from the Windows Start/Programs menu under Software Toolbox.

    By default, launching from the toolbar will auto-populate all protocol tags for your configured topics in OmniServer. (Automatically adding tags can be disabled under View -> General Options.)



Name	Value	Quality	TimeStamp	Access Rights	Data Type
testtopic.cardmessage		Bad, Non-specific, Limit...	12:00:00 AM	Read/Write	String
testtopic.status	True	Good, Non-specific, Li...	10:31:02 AM	Read	Boolean
testtopic.trigger		Bad, Non-specific, Limit...	12:00:00 AM	Read/Write	Boolean
tpi.boardtype		Bad, Communications F...	12:08:57 PM	Read/Write	Long
tpi.commrevision		Bad, Communications F...	12:08:57 PM	Read/Write	Long
tpi.firmwarev		Bad, Communications F...	12:08:57 PM	Read/Write	Float
tpi.flashrev		Bad, Communications F...	12:08:57 PM	Read/Write	Long
tpi.guid		Bad, Communications F...	12:08:57 PM	Read/Write	String
tpi.name		Bad, Communications F...	12:08:57 PM	Read/Write	String
tpi.sn		Bad, Communications F...	12:08:57 PM	Read/Write	String
tpi.status	False	Bad, Device Failure, Li...	10:57:46 AM	Read	Boolean

https://www.softwaretoolbox.com

Start Time: 10:31:01 AM Current Time: 12:09:25 PM Last Update Time: 12:08:58 PM Vendor: Software Toolbox, Inc. Server State: Running TotalItemCount = 11

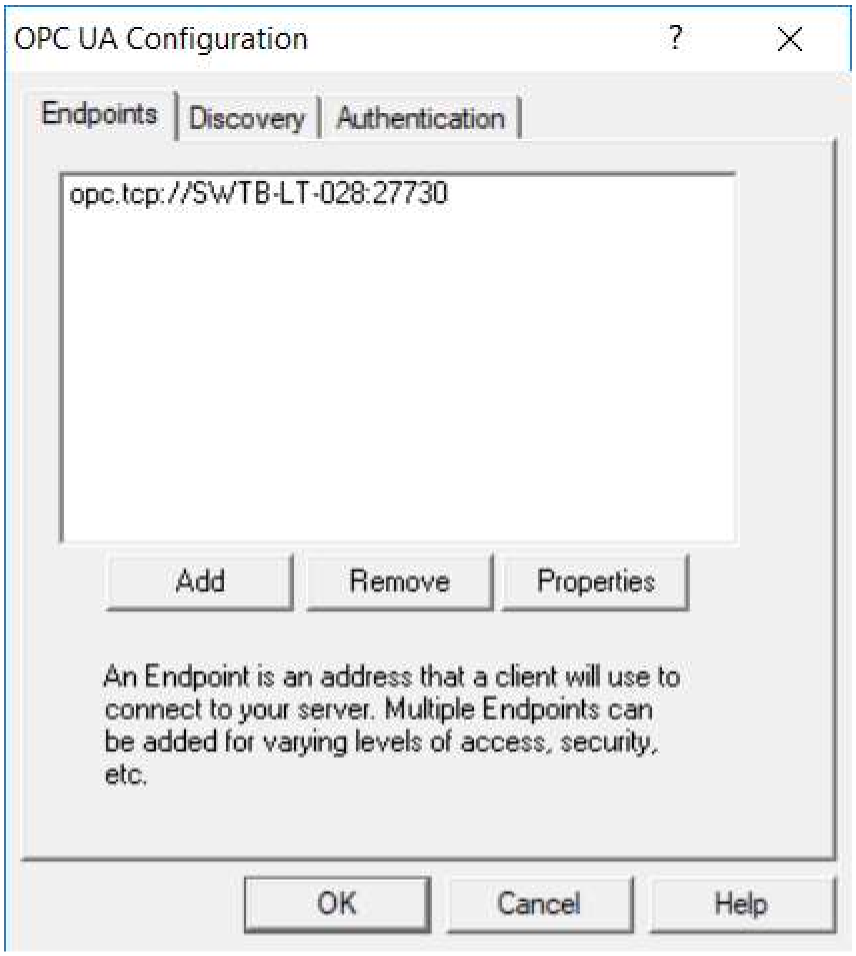
- OPC UA - Displays information about the OPC UA endpoint and port information for OPC UA client connectivity. This icon can also be double-clicked to open the OPC UA Configuration dialog where OPC UA settings for OmniServer are configured, including endpoints, ports, discovery and security features. Please see the OPC UA Configuration section of the help file for full details.

Server Plug-ins / Wedges

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

Included with the Professional Edition of OmniServer are client plug-ins known as Wedges. For more information on plug-ins / wedges and how they operate, please see this help file section on Plug-ins / Wedges.

7.1.3.1 OPC UA Configuration



opc.tcp://SWTB-LT-028:27730

An Endpoint is an address that a client will use to connect to your server. Multiple Endpoints can be added for varying levels of access, security, etc.

OPC UA Configuration Dialog

Accessed by double-clicking the OPC UA icon in the Clients window of OmniServer, this section allows you to define and edit OPC UA endpoints and ports, as well as Encryption and Authentication options and Discovery services.

OPC UA Configuration Sections:

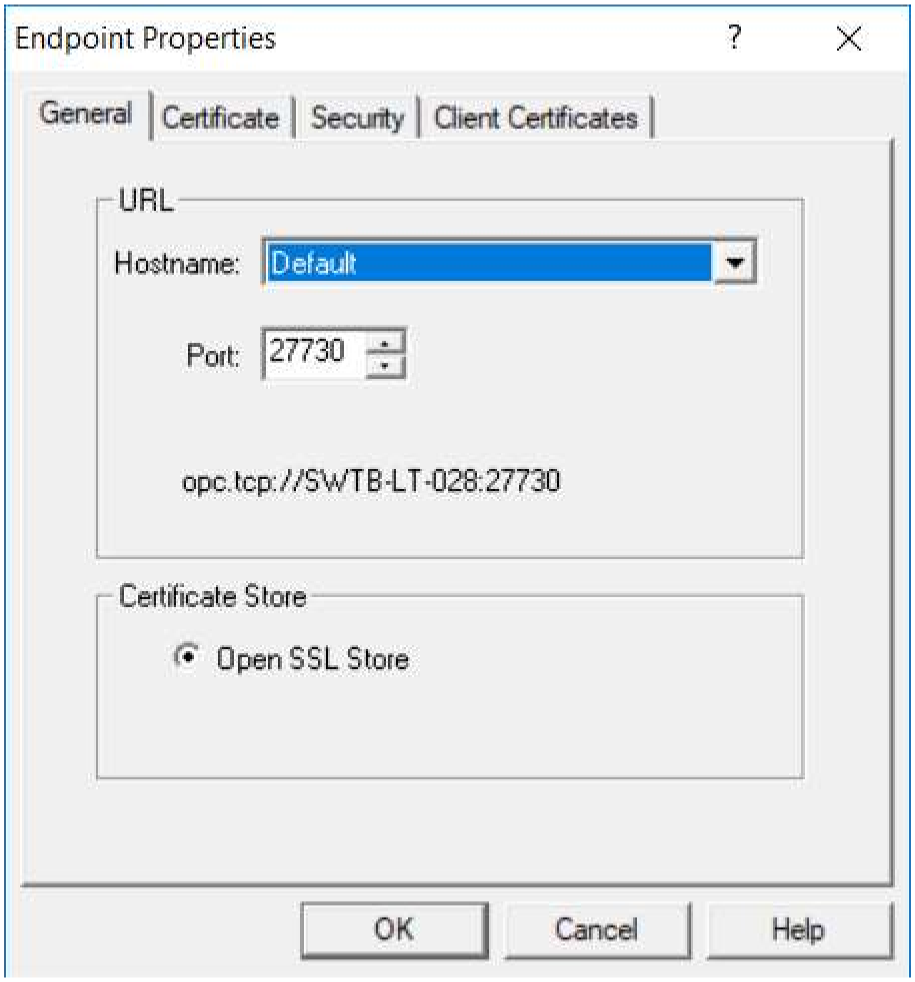
- Endpoints
- Discovery

- Authentication

For testing OPC UA connectivity to OmniServer, we recommend Unified Automation's OPC UA Expert Test Client, available for download on the Unified Automation website.

7.1.3.1.1 Endpoint Properties

An OPC UA Endpoint defines an OPC Server in similar fashion of an OPC DA Program ID. An OPC UA Server endpoint is used by an OPC UA Client application to connect to the OPC UA Server.



URL

Hostname: Default

Port: 27730

opc.tcp://SWTB-LT-028:27730

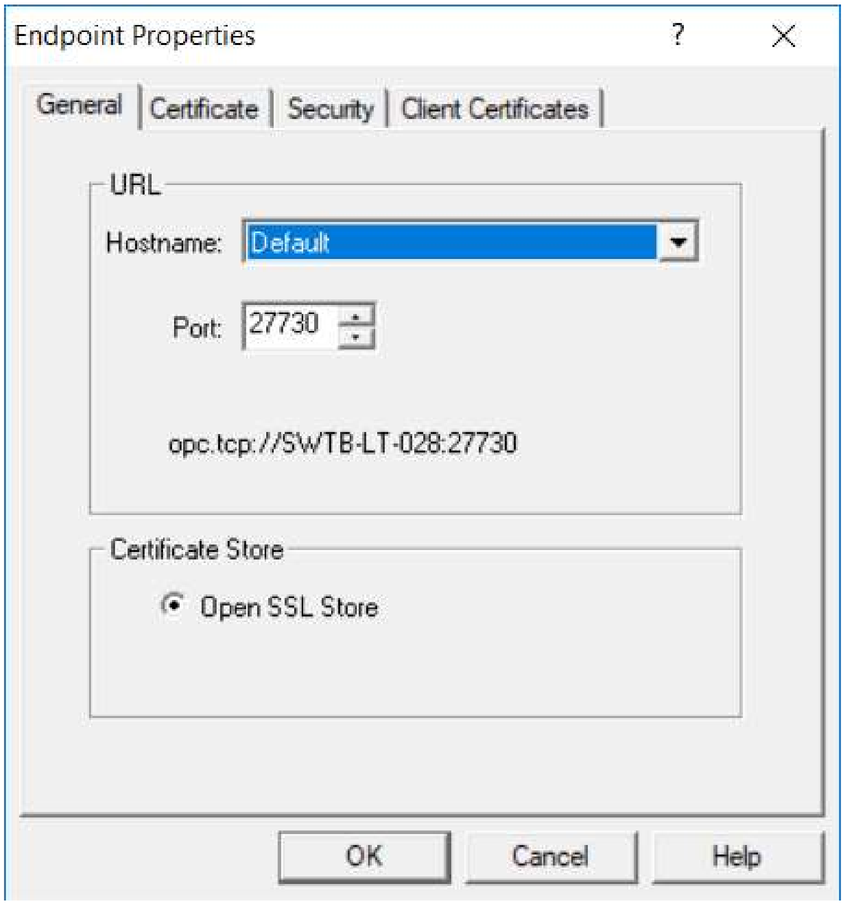
Certificate Store

[✓ Open SSL Store]

- Endpoint Properties - General
- Endpoint Properties - Certificate
- Endpoint Properties - Security
- Endpoint Properties - Client Certificates

7.1.3.1.1.1 Endpoint Properties - General

General properties of an OPC UA Endpoint in OmniServer are as follows:



URL

Hostname: Default

Port: 27730

opc.tcp://SWTB-LT-028:27730

Certificate Store

(●) Open SSL Store

- **Hostname** - allows you to select the name of the computer that will be used in the endpoint definition.
    - **Default** - will use the DNS hostname of the local computer in the endpoint definition. This will allow the server to be accessible by a remote OPC UA client application
    - **Local Only** - will use the loopback address (127.0.0.1) in the endpoint definition. This will allow the server to be accessible only by a local OPC UA client application

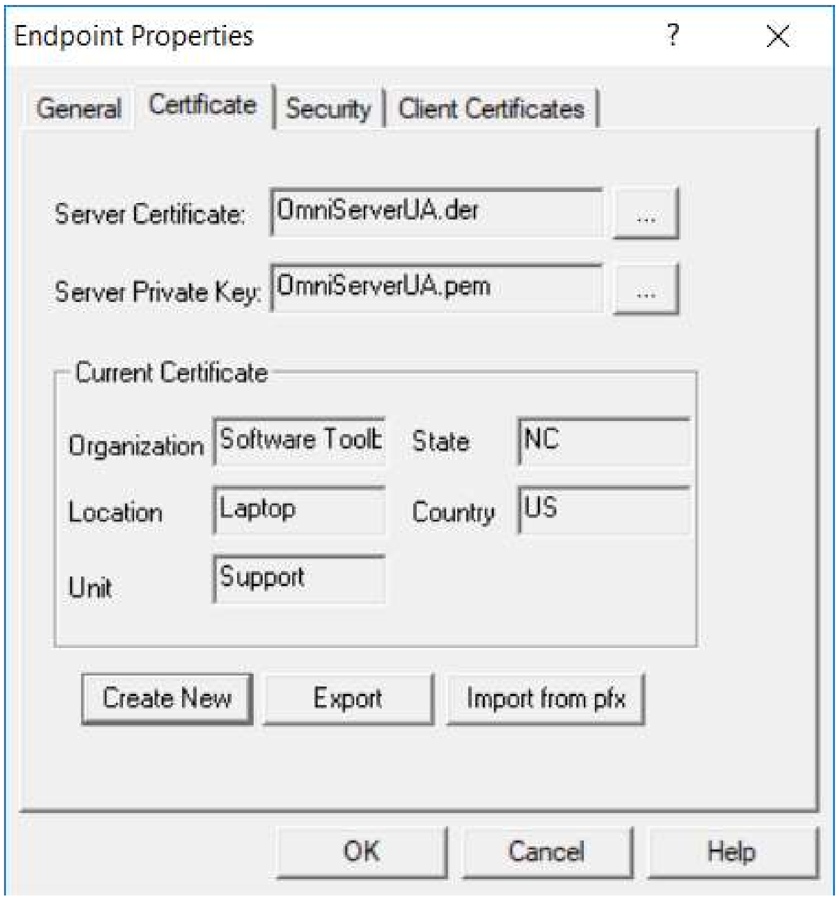
- **Port** - specifies the TCP port number that an OPC UA Client can connect to for this endpoint
- **Endpoint** - fully-qualified OPC UA endpoint definition displays at the bottom of the URL section - this is what needs to be specified in your OPC UA client application.

Certificate Store:

- **Open SSL Store** - Open SSL is used for the creation of self-issued OmniServer OPC UA certificates. The Open SSL Certificate Store is the location where these certificates are stored, as well as trusted certificates from OPC UA Clients.

7.1.3.1.1.2 Endpoint Properties - Certificate

The following sections contain the setting details for OmniServer's OPC UA Server certificate.



Server Certificate: OmniServerUA.der

Server Private Key: OmniServerUA.pem

Organization: Software Toolt
State: NC
Location: Laptop
Country: US
Unit: Support

The following fields are available.

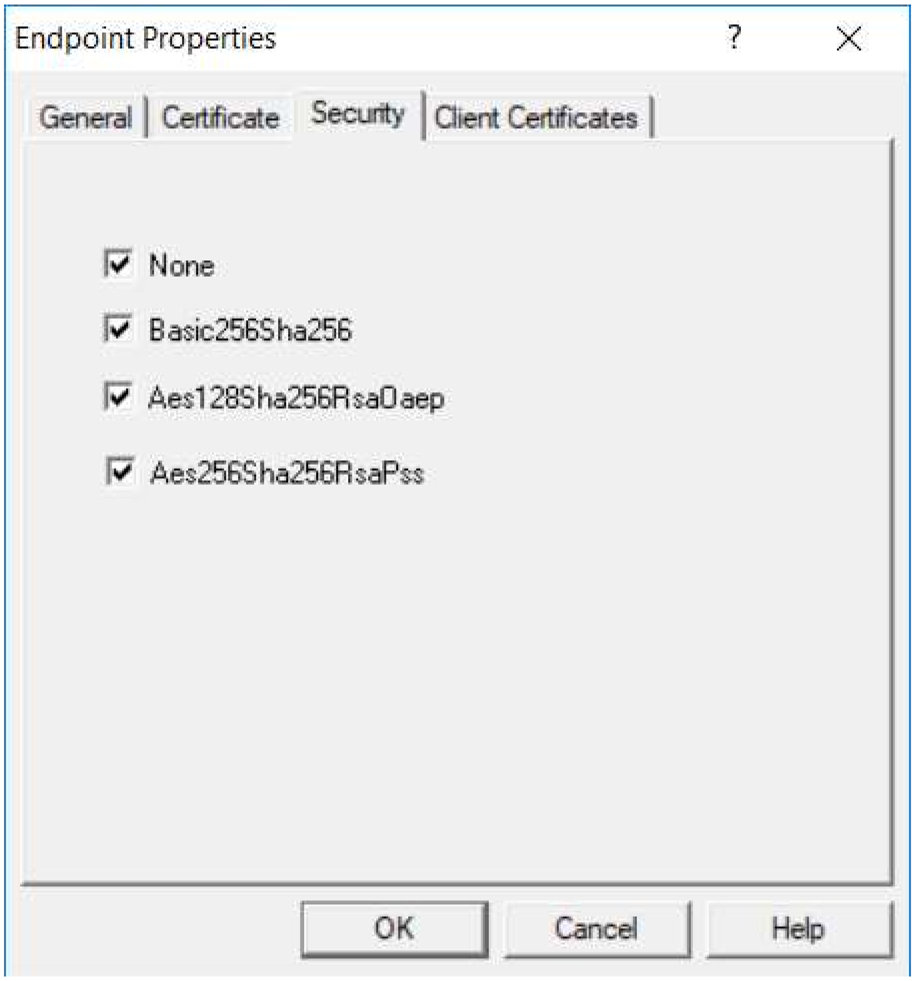
- **Server Certificate** - specifies the name of the current OmniServer OPC UA certificate in the Open SSL Store. Clicking on the ellipse button to the right will allow you to browse to an existing server certificate.
- **Server Private Key** - specifies the name of the current server private key. Clicking on the ellipse button to the right will allow you to browse to an existing Server Private Key.
- **Current Certificate** - displays the details of the current OmniServer OPC UA Certificate in the Open SSL Store.

Buttons:

- **Create New** - allows you to create a new server certificate to be saved in the Open SSL Store. The default location for Open SSL Store certificates is: C:\Documents and Settings\All Users\Application Data\Software Toolbox\OmniServer\PKI\CA\certs
- **Export** - allows you to export the current OmniServer OPC UA Certificate to a specified location on your local machine, or network, as a .der certificate file, which can be imported into any OPC UA client.
- **Import from .pfx** - a .pfx file is a composite security file containing information for both the security certificate and security private key. Importing from a .pfx file will separate this information into a corresponding .der security certificate and .pem security private key to be used by OmniServer for this endpoint.

7.1.3.1.1.3 Endpoint Properties - Security

The following types of security profiles are supported by OmniServer for OPC UA Clients:



☑ None
☑ Basic256Sha256
☑ Aes128Sha256RsaOaep
☑ Aes256Sha256RsaPss

- **None** - no security profiles will be utilized. OPC UA Clients not utilizing a security profile will be able to connect to OmniServer when this is enabled
- **Basic256Sha256** - allows OPC UA clients using Basic 256 bit encryption with an SHA256 hash for security to connect to OmniServer
- **Aes128Sha256RsaOaep** - allows OPC UA clients using AES 128 bit encryption with an SHA256 hash and RSA and OAEP encryption security to connect to OmniServer
- **Aes256Sha256RsaPss** - allows OPC UA clients using AES 256 bit encryption with an SHA256 hash and RSA-PSS encryption security to connect to OmniServer

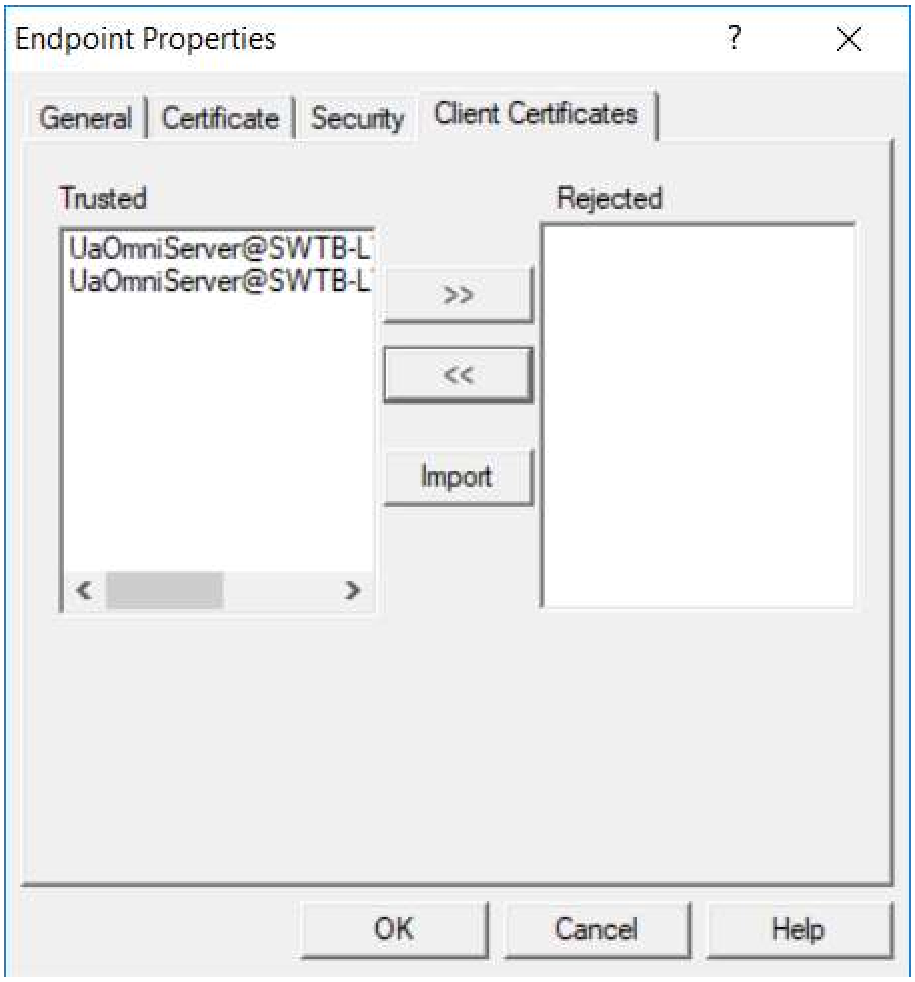
All four profiles are enabled by default. If a particular profile is disabled, an OPC UA Client using that security profile specifically will not be able to successfully connect to OmniServer.

NOTE: If you discover that your OPC UA client application have an option for one of the above supported security policies (other than None), check with the client manufacturer to see if they have an updated version with support for one of these security policies.

7.1.3.1.1.4 Endpoint Properties - Client Certificate

Endpoint Properties - Client Certificates

This section shows both OmniServer Server Certificates and any OPC UA Client certificates that have been either imported or accumulated when an OPC UA Client attempted a connection to OmniServer.



UaOmniServer@SWTB-L
UaOmniServer@SWTB-L

- **Trusted** - shows a list of trusted Client Certificates. An OPC UA Client using Security (does NOT apply if No Security is used) must have its instance certificate in the Trusted List here to successfully connect. You can transfer a trusted certificate to the rejected list by clicking on the [>>] button.

The trusted certificates are stored at:

C:\Documents and Settings\All Users\ Application Data\Software Toolbox\OmniServer\PKI\CA\certs

- **Rejected** - shows a list of rejected Client Certificates. If an OPC UA Client using Security (does NOT apply if No Security is used) has its instance certificate listed here and not in the Trusted List, the OPC UA Client will not be able to connect! You can transfer a rejected certificate to the trusted list by clicking on the [<<] button.

The rejected certificates are stored at:

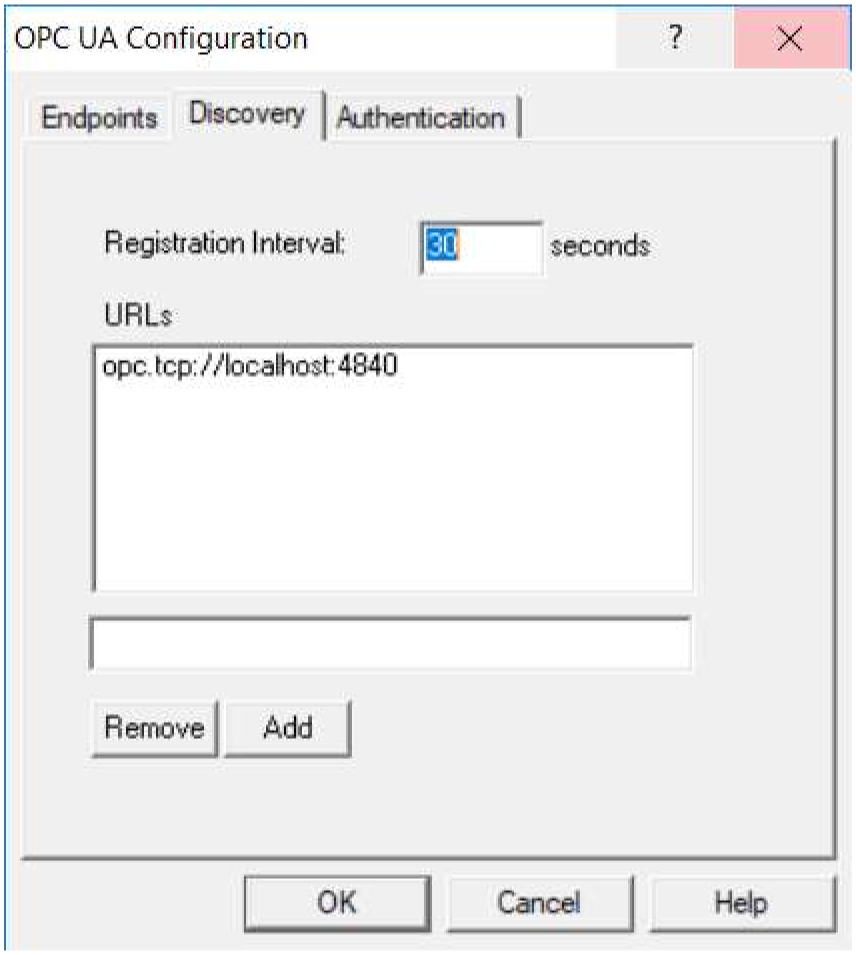
C:\Documents and Settings\All Users\ Application Data\Software Toolbox\OmniServer\PKI\CA\rejected

- Import - allows you to browse to an existing client certificate exported from an OPC UA client for import into the Trusted section.

NOTE: The certificate management in this section only pertains when an OPC UA Client application is using Security.

7.1.3.1.2 Discovery Properties

This section provides settings allowing you to add OPC UA Discovery Server Services on the local or remote machines that OmniServer will register with periodically, allowing an OPC UA Client to find OmniServer when it is browsing with one of these defined Discovery URLs.



Registration Interval: 30 seconds

URLs

opc.tcp://localhost:4840

- **Registration Interval** - how frequently the OmniServer registers with the specified OPC UA Discovery Servers so it can be located by OPC UA client applications.
- **URLs** - list of OPC UA Discovery Server endpoints with which OmniServer will register at the specified Registration Interval.

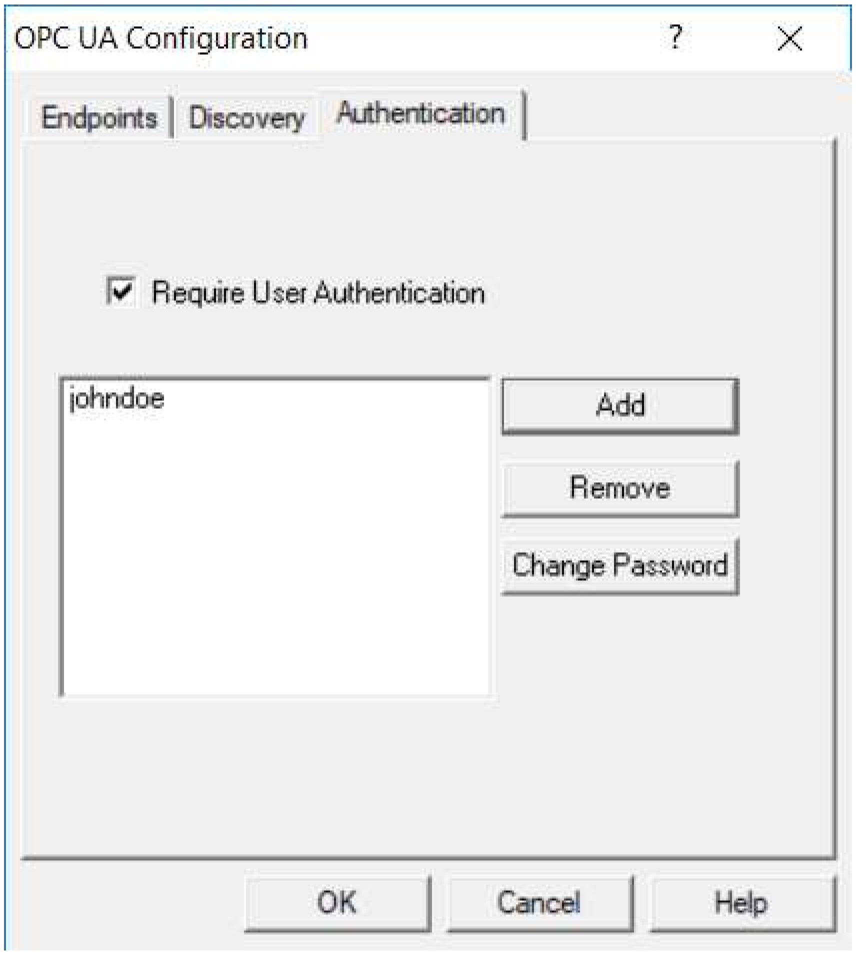
Buttons:

- **Remove** - deletes the selected Discovery Server URL from the list.
- **Add** - adds a specified Discovery Server URL to the list.

7.1.3.1.3 Authentication Properties

This section allows you to enable OPC UA User Authentication and create and manage user accounts that can be used by OPC UA Client to make authenticated connections.

NOTE: User Authentication is disabled by default: OPC UA Clients should not specify a username and password when User Authentication is disabled in OmniServer.

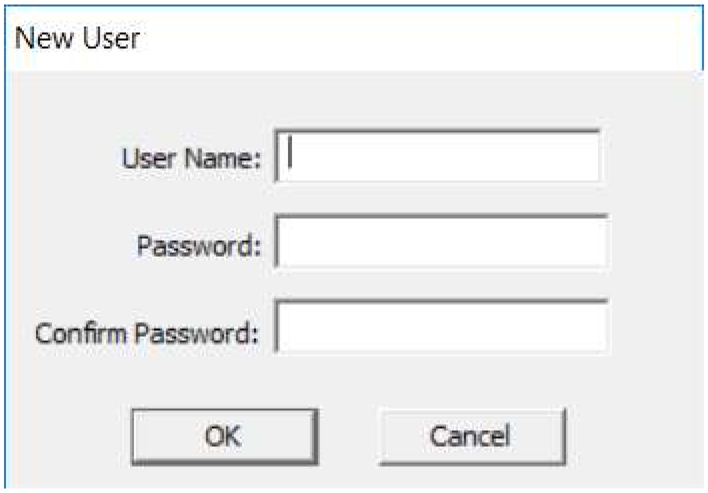


☑ Require User Authentication

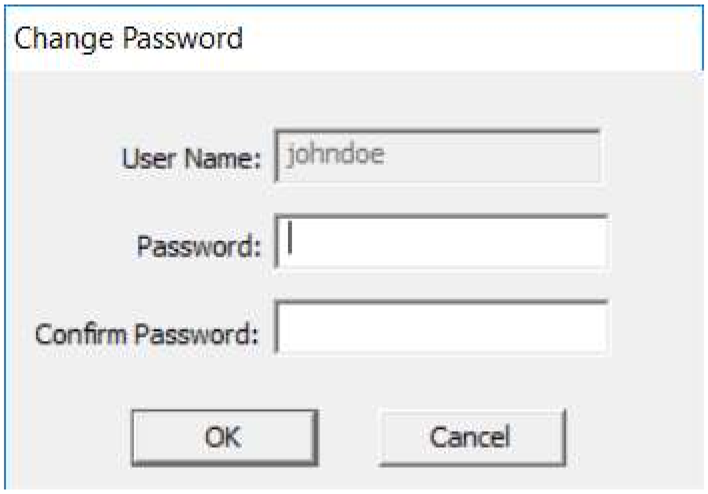
johndoe

- **Require User Authentication** - when checked, allows you to create user profiles with a user name and password

The following buttons are used for adding, removing and managing Users and Passwords.



- **Add** - clicking this button accesses the "New User" dialog seen above, allowing you to create a new user profile.
  - **User Name** - the user name specified in an OPC UA Client using Authentication.
  - **Password** - the password specified with the above user name in an OPC UA Client using Authentication.
  - **Confirm Password** - requires re-entry of the exact same password for verification purposes.
- **Remove** - deletes the selected user profile from the list.



- **Change Password** - clicking this button accesses the "Change Password" dialog seen above, allowing you to change the password associated with the selected existing user profile.

- **Password** - enter the new password to be associated with this user profile.
- **Confirm Password** - re-enter the new password for verification purposes.

7.1.3.2 Plug-ins / Wedges

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

This section of the server's Help file contains detailed information on the Server Plug-ins (also known as Wedges).

What is a Server Plug-in / Wedge?

A server plug-in is a client object that passes data from the server to an external operating system program and/or device. Internally, the plug-in acts just like a client program in that it polls the server for information. Depending upon that information, the plug-in will then format and send that data to another process on that computer based upon the definitions within the plug-in.

What plug-ins / wedges are available?

Currently, these are the plug-ins defined:

- Database Wedge - Sends server data to any pre-defined ODBC-compliant database (OmniServer Professional Edition only).
- Keyboard Wedge - Sends server data to any Windows program via "keystrokes".
- File Wedge - Sends server data to any file (OmniServer Professional Edition only).
- Email Wedge - Sends server data as an email message (OmniServer Professional Edition only).

Diagnostic Tools for Plug-ins / Wedges

Some additions have been made to the Item Values window to show diagnostic information for each of the plug-ins / wedges. To discover what changes have been made, please follow the week below.

Plug-in / Wedge Diagnostic Tools in the Item Values Window

Specifying Expressions

Each plug-in / wedge specifies the format of the data by the use of Expressions. Once the data has been received from the server, the expression is used to manipulate the data before it is sent further along by the action defined within the plug-in / wedge.

For more information on how expressions are used within each type of plug-in / wedge, please follow the link below:

How to specify Expressions.

7.1.3.2.1 Using the Item Values Window with Wedges

Enhancements have been made to the Item Values Window to assist the the diagnostics of using the server wedges.

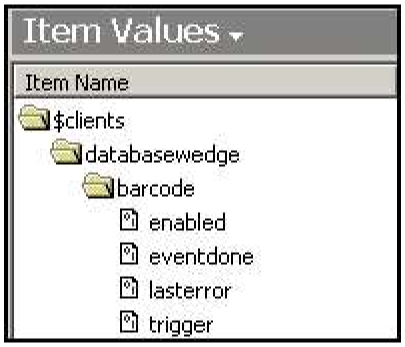
How to access the Item Values Windows.

There are two ways to bring up the Item Values Window:

- Select the Item Values icon from the server's Diagnostic Toolbar.
- Select **View | Diagnostics | Item Values** from the server's menu.

What information is contained the window?

Each wedge type (database, keyboard, file, etc.) is located under the "Sclients" section of the Item Values window. Within each wedge type will be listed every wedge created. And finally, within each wedge created, there are four items available for manipulation by the client. program. This is a sample of the window you will get:



Sample Item Values Windows

In this example, a previous database wedge called "Barcode" has been created.

What do the different items mean?

Each wedge will have four diagnostic items associated with it. Each of these items can be read from and written to by the client program:

- **Enabled**: This item enables or disables the wedge. Setting this value to Logical False (0) will disable the triggering of the wedge, while setting the value to Logical One (non-zero) will enable the triggering. **Please note:** Disabling the wedge will **not** stop the wedge from activating the topic/protocol. It will only prevent the triggering of the wedge. Any items mentioned in the Trigger, Reset, Event (for Keyboard and File wedges) or Database and Data (for Database wedges) tabs will still be activated.
- **Eventdone**: This item is set the Logical One (non-zero) once the wedge event, or action, has been completed.
- **Lasterror**: This item reports the last error reported by the wedge. This item does **not** report the current error status, but rather the last error that has occurred, no matter when it occurred.
- **Trigger**: Setting this item to Logical One (non-zero) forces a trigger of the wedge (unless the wedge has been disabled).

Can I access any of these items with my client?

Each of the these items can accessed by the client by using this format:

$clients.*wedge_type.wedge_name.diagnostic_item*

where...

- ***wedge_type*** is the type of the wedge. This value is any one of the defined wedges within the server. As of this release, here are the wedge types defined:

Wedge	***wedge_type*** **name**
DatabaseWedge	databasewedge
FileW	filewe

edge	dge
KeyboardWedge	keyboardwedge
EmailWedge	emailwedge

- ***wedge_name*** is the name of the wedge you have defined.
- ***wedge_item*** is the name of any one of the four item defined above.

Please note: The Database and File wedges are only available in the OmniServer Professional Edition.

7.1.3.2.2 Using Expressions with Wedges

[This section applies only to Plug-Ins/Wedges and the OmniServer Professional Edition.]

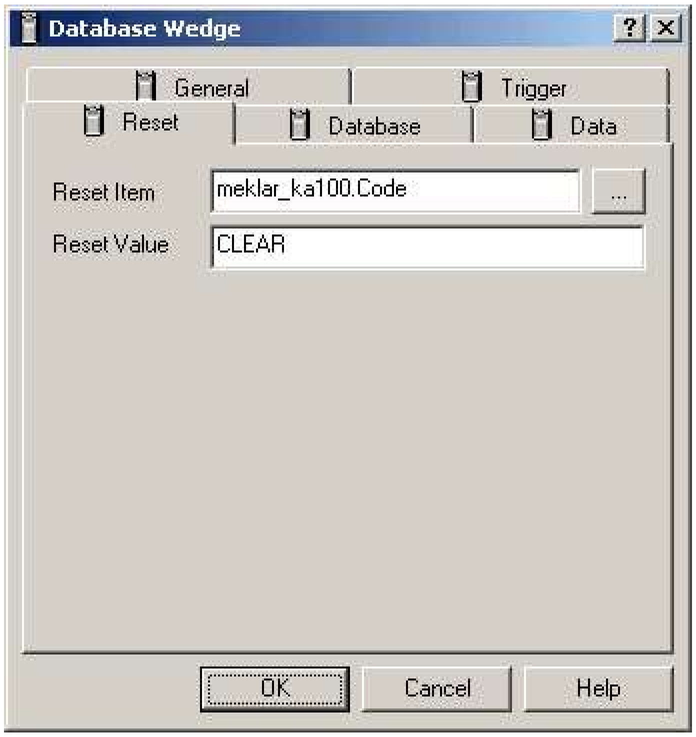
To transfer the data to from the server through the plug-in / wedge, the plug-in / wedge uses Expressions. Expressions describe the format of the data after the plug-in / wedge has received it from the server.

See also Using Expressions & Valid Operators

How an Expression is formatted.

Currently, these are the plug-ins / wedges that use expressions:

⊟ All Plug-ins / Wedges - The RESET Tab



Reset Item: meklar_ka100.Code
Reset Value: CLEAR

The Reset Tab - in this case, for the Database Wedge

You type in the expression within the **Reset Value** text box. A plug-in / wedge currently only supports four types of expressions:

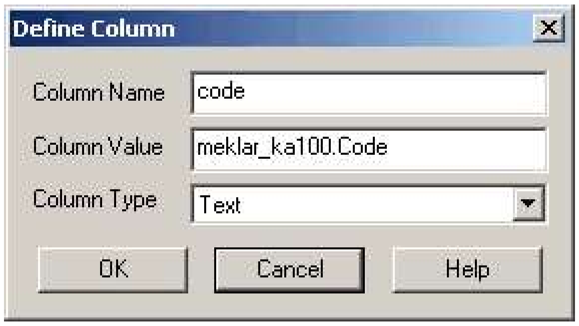
Item Addresses: The address of an item in the server, in the format *topic.item*. If the topic/item combination is not valid, the plug-in / wedge will return an error.

String Literals: A string of characters. Do not put quotes around the value.

Numeric Literals: An integer or floating point number. You do not have to place quotes around the numbers. Scientific, hexadecimal, binary, or octal notation is not valid.

Database Plug-in / Wedge

[This section applies only to the OmniServer Professional Edition.]



Column Name: code
Column Value: meklar_ka100.Code
Column Type: Text

The Database Wedge Expression Window

You type in the expression within the **Column Value** text box. The database wedge currently only supports four types of expressions:

Item Addresses: The address of an item in the server, in the format *topic.item*. If the topic/item combination is not valid, the wedge will return an error.

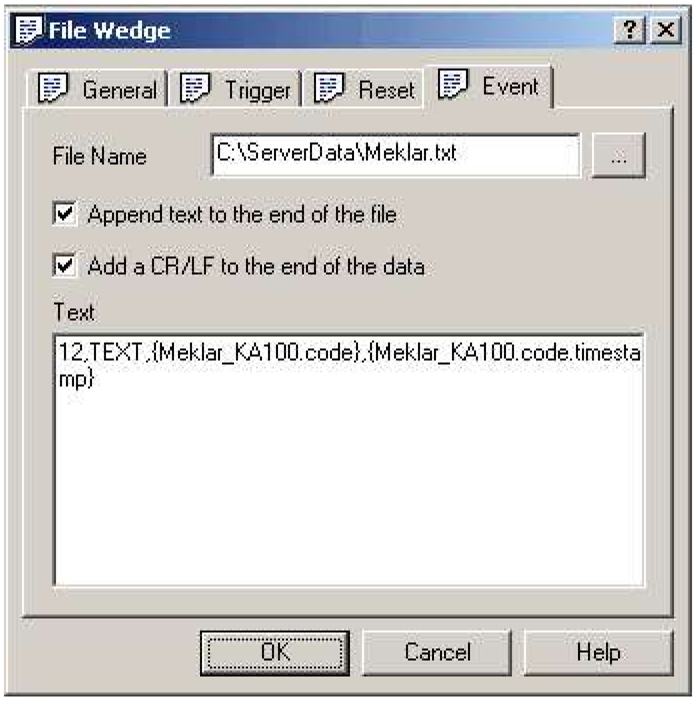
String Literals: A string of characters. You must place quotes (") around the string, otherwise the wedge returns an error.

Numeric Literals: An integer or floating point number. You do not have to place quotes around the numbers. Scientific, hexadecimal, binary, or octal notation is not valid.

Complex Expressions: Using the expression parser, you can enter in an expression using any legal combinations of item addresses, string literals, and numeric literals. For a complete explanation of valid expressions, please click here.

File Plug-in / Wedge

[This section applies only to the OmniServer Professional Edition.]



File Name: C:\ServerData\Meklar.txt

☑ Append text to the end of the file

☑ Add a CR/LF to the end of the data

Text:

```
12,TEXT,{Meklar_KA100.code},{Meklar_KA100.code.timestamp}
```

The File Wedge Events Window

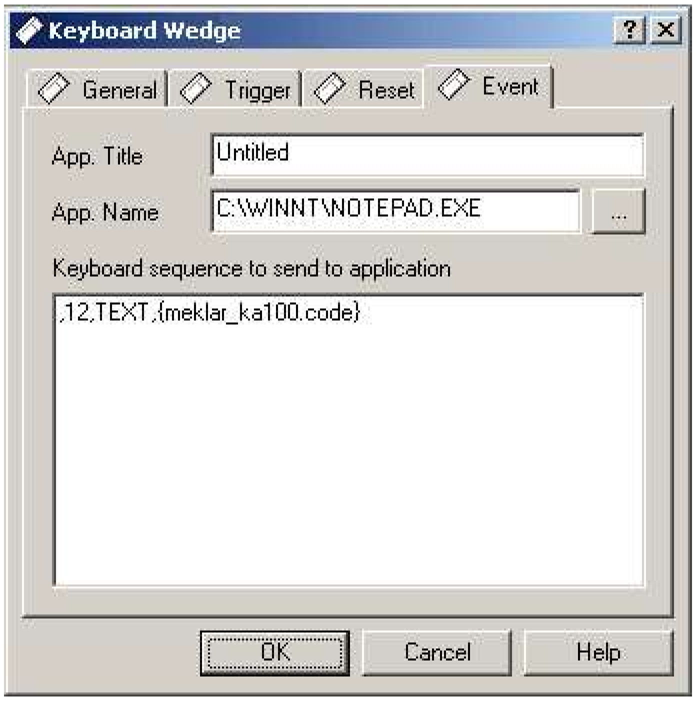
You type in the expression within the **Text** text box. The rules for entering information into the **text** box are as follows:

All **Item Address** data must be enclosed within braces (**{ }**).

Any characters outside of the braces are sent as-is to the file.

All **Complex Expressions** must be enclosed within braces (**{ }**). Using the expression parser, you can enter in an expression using any legal combinations of item addresses, string literals, and numeric literals. For a complete explanation of valid expressions, please click here.

⊟ Keyboard Plug-in / Wedge



App. Title: Untitled

App. Name: C:\WINNT\NOTEPAD.EXE

Keyboard sequence to send to application:

```
,12,TEXT,{meklar_ka100.code}
```

The Keyboard Wedge Events Window

You type in the expression within the **Text** text box. The rules for entering information into the **text** box are as follows:

All **Item Address** data must be enclosed within braces (**{ }**).

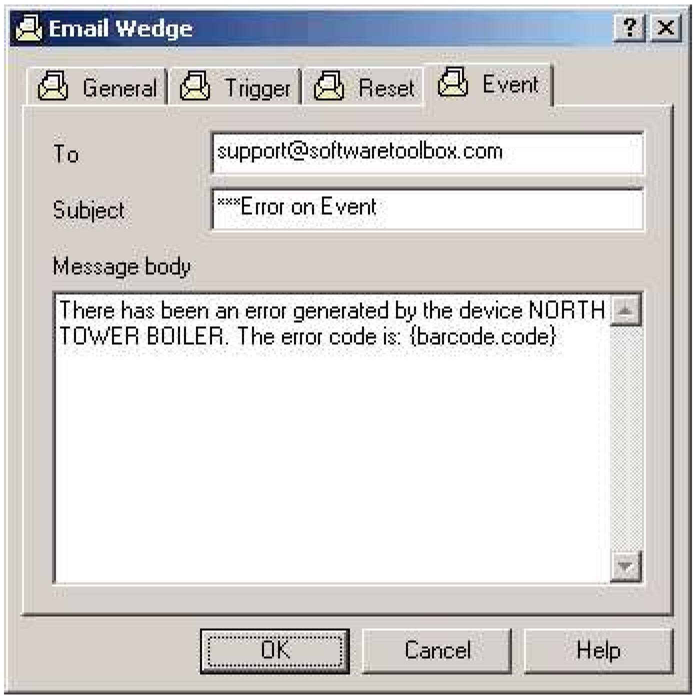
Any characters outside of the braces are sent as-is to the file.

All **Complex Expressions** must be enclosed within braces (**{ }**). Using the expression parser, you can enter in an expression using any legal combinations of item addresses, string literals, and numeric literals. For a complete explanation of valid expressions, please click here.

Special formats can be used to represent keystrokes that cannot be typed in on a normal keyboard (for example, the ESCAPE key). You can find a complete list of keystrokes on the Special Keyboard Expressions page.

Email Plug-in / Wedge

[This section applies only to the OmniServer Professional Edition.]



To: support@softwaretoolbox.com

Subject: ***Error on Event

Message body: There has been an error generated by the device NORTH TOWER BOILER. The error code is: {barcode.code}

The Email Wedge Event Window

You type in the expression within the **Text** text box. The rules for entering information into the **text** box are as follows:

All **Item Address** data must be enclosed within braces (**{ }**).

Any characters outside of the braces are sent as-is to the file.

All **Complex Expressions** must be enclosed within braces (**{ }**). Using the expression parser, you can enter in an expression using any legal combinations of item addresses, string literals, and numeric literals. For a complete explanation of valid expressions, please click here.

Special Topic/Item Addresses to use in Expressions.

With all expressions, you can use the Topic/Item combination to access any data value within the server. However, you can also access certain properties of that item.

Currently, you can use these expressions in your wedge to access these item properties:

topic.item - Returns the current value of the item.

***topic.item*.value** - Returns the current value of the item.

***topic.item*.timestamp** - Returns the current date/time stamp of the item.

7.1.3.2.3 Email Plug-in / Wedge

[This section applies only to the OmniServer Professional Edition.]

The server's **Email Plug-in / Wedge** is designed to output data from the server as an email message to a specified email address. The email plug-in / wedge is activated by some change in a data value (called a **Trigger**) which causes some data to be sent out as an email message.

For a video tutorial on working with the Email Plug-in / Wedge, click here.

How it works:

Once polling is activated, the server will search through the listing of plug-ins / wedges and perform the following operations:

- The plug-in / wedge is checked to see if it is enabled. If the plug-in / wedge is disabled, then the processes that follow are not executed, and the server continues to the next plug-in / wedge
- The **Trigger** information is then checked to find out if the data logging of the plug-in / wedge is performed. If the trigger conditions are not satisfied, the plug-in / wedge is not processed further and the server continues to the next plug-in / wedge.
- The plug-in / wedge will then look at the **Reset** section to see if any server item's value will be reset to a specific value.
- Finally, the plug-in / wedge will look at the **Event** section, compose the email message based upon the message body, subject, and email address, and then send that email message to either the default SMTP server or one defined during configuration. Should any errors occur at this time, the error message will be logged to the item specified in the **General** section under the **Last Error** item.
- Finally, if there is an item in in the **Notification Item** box in the **General** section, the server will set the value of that item to Logical One.

How to use the Email Plug-in / Wedge

You can find the email plug-in / wedge in the **Clients** section of the server's Configuration program. Locate the Email Wedge Icon (shown below) and double-click on it.

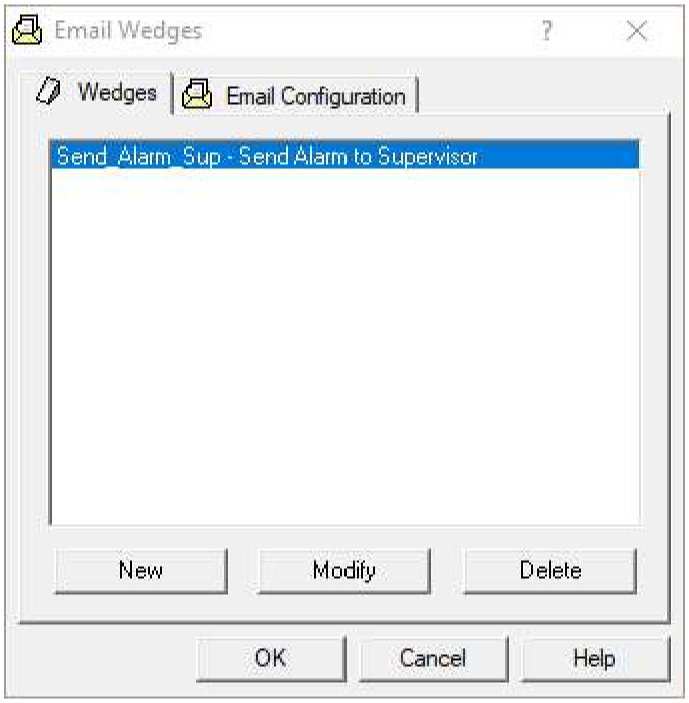
The Client Email Wedge Icon

- Creating, Modifying or Deleting a Wedge
- The General Tab
- The Trigger Tab
- The Reset Tab
- The Event Tab

7.1.3.2.3.1 Creating an Email Wedge

[This section applies only to the OmniServer Professional Edition.]

The first screen you see allows one to create, modify, or delete an email instance.



The Email Plug-in / Wedge Selection Screen

The selections are:

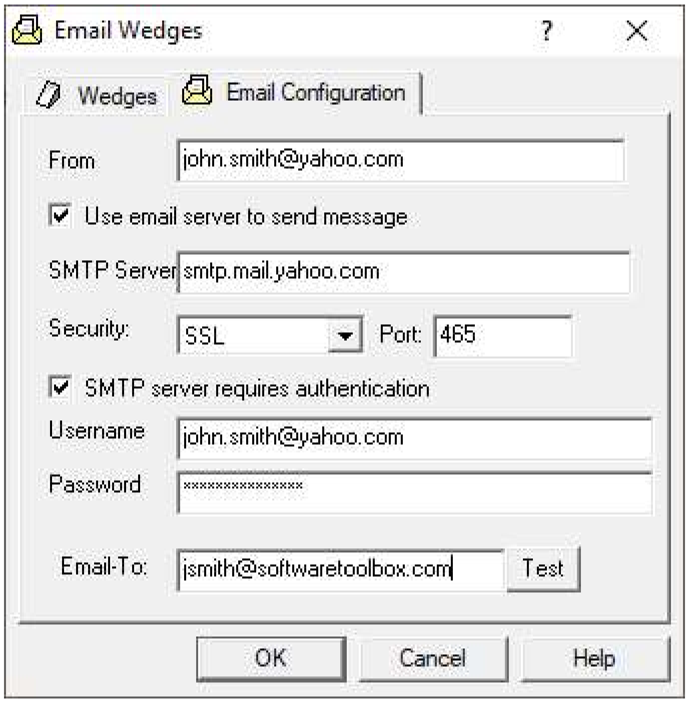
- **New**: Creates a new email instance. You will then be immediately taken the the **General** tab, where you can name your email instance.
- **Modify**: Opens up the selected email instance and allows you to begin changing its properties. This option is not available if there is no email selected.
- **Delete**: Deletes the currently selected email instance. This option is not available if there is no email instance selected.

You can also setup your Email Configuration from this screen by selecting the Email Configuration tab.

7.1.3.2.3.2 The Configuration Tab

[This section applies only to the OmniServer Professional Edition.]

The **Configuration** tab is used to specify what email services are used by the wedge when performing a wedge action.


Email Wedges

Wedges | Email Configuration

From: john.smith@yahoo.com

☑ Use email server to send message

SMTP Server: smtp.mail.yahoo.com

Security: SSL Port: 465

☑ SMTP server requires authentication

Username: john.smith@yahoo.com

Password: ****************

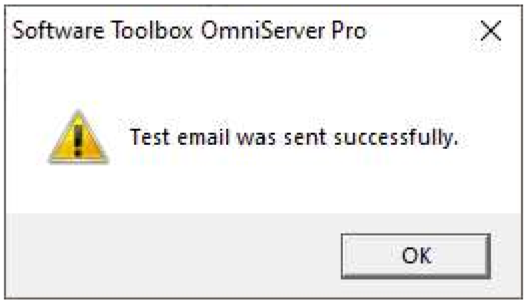
Email-To: jsmith@softwaretoolbox.com

Email Plug-in / Wedge - The Configuration Tab

The options available are:

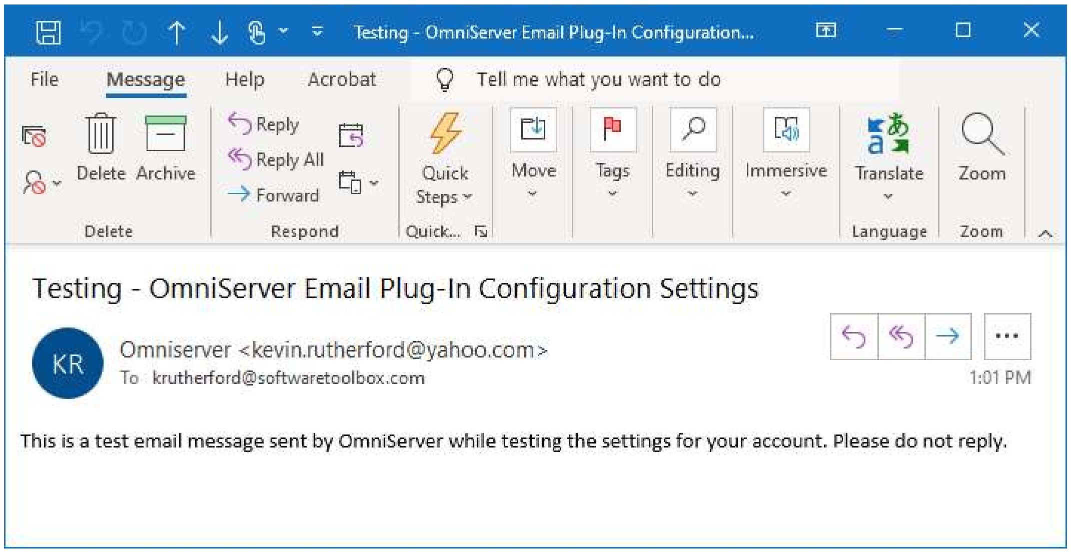
- **From**: This is the email address "From" which this plug-in / wedge sends an email. The email address must be valid (name@name.ext), but does not actually have to exist. However, it is the name that will appear in the From list of the email client receiving the email, so it's a good idea to make it a real email address.
- **Use email server to send message**: If checked, this lets you specify a particular email server on which to send the email message (such as a specific company's mail server or a public mail server such as Google or Yahoo). By default, this is not checked, meaning that the wedge will use whatever default email server has been configured for the particular computer.
- **SMTP Server**: The SMTP server to use if the "Use email server to send message" checkbox is checked. If you do not know the name of the server, first **do not** check the above checkbox. Otherwise, check with your systems administrator for instruction.
  - **NOTE:** If you choose to use a public email server (such as Google or Yahoo) for sending emails, there are certain security considerations that have to be taken in your public email account before it will work with OmniServer. Click here for recommendations.

- **Security:** Select the secure encryption method that the connection to the email server will use (whether you're specifying a mail server or using the default, these settings will be used). Select from SMARTTLS (default), SSL or TLS, as required by your mail server.
- **SMTP server requires authentication**: If checked, the plug-in / wedge will send log-in information (see the below options) to the SMTP server. Some SMTP servers require what is called "SMTP Authentication", which this option gives you. If you are unsure, check with your systems administrator to confirm what your mail server requires. If you're using a public mail server such as Google or Yahoo, authentication will always be required.
- **Username**: The Username to pass along to the SMTP server for SMTP Authentication (see above).
- **Password**: The Password to pass along to the SMTP server for SMTP Authentication (see above).
- **Email-To:** This field specifies the email address to be used only for the test email to confirm your email settings are correct. Enter an email address for an account that you have access to so you can confirm receipt, which ensures your email settings will work correctly for your emails.
- **Test:** Once you've completed your Email Configuration and specified an "Email-To", click this button to send a test email to the specified account for verifying that your email settings are correct, after which you will receive the following confirmation (or a related error will be displayed if your settings are not correct):


Software Toolbox OmniServer Pro

Test email was sent successfully.

- An example of the test message that will be sent to the specified "Email-To" address is shown here:



Testing - OmniServer Email Plug-In Configuration Settings

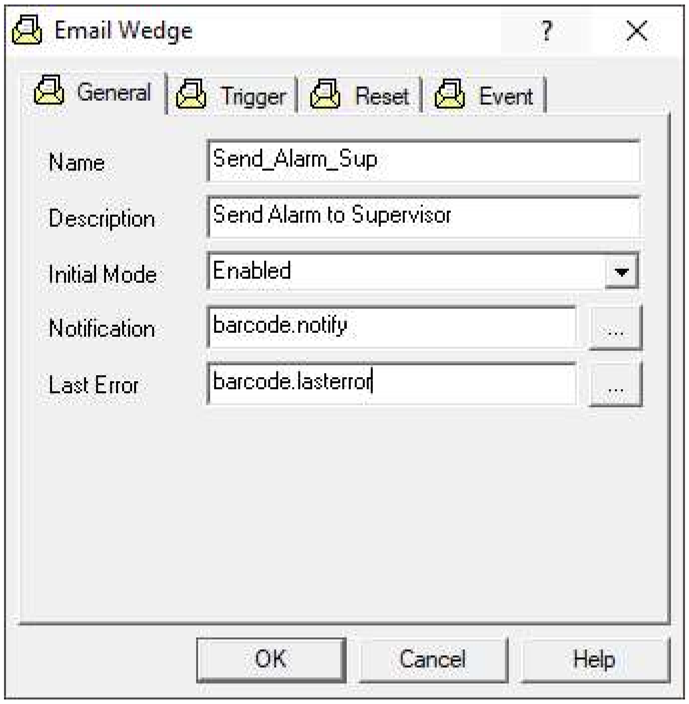
KR Omniserver <kevin.rutherford@yahoo.com>
To krutherford@softwaretoolbox.com
1:01 PM

This is a test email message sent by OmniServer while testing the settings for your account. Please do not reply.

7.1.3.2.3.3 The General Tab

[This section applies only to the OmniServer Professional Edition.]

The General Tab is where you define the basic properties of the Email plug-in / wedge instance.



Name	Send_Alarm_Sup
Description	Send Alarm to Supervisor
Initial Mode	Enabled
Notification	barcode.notify
Last Error	barcode.lasterror

Email Plug-in / Wedge Instance - The General Tab

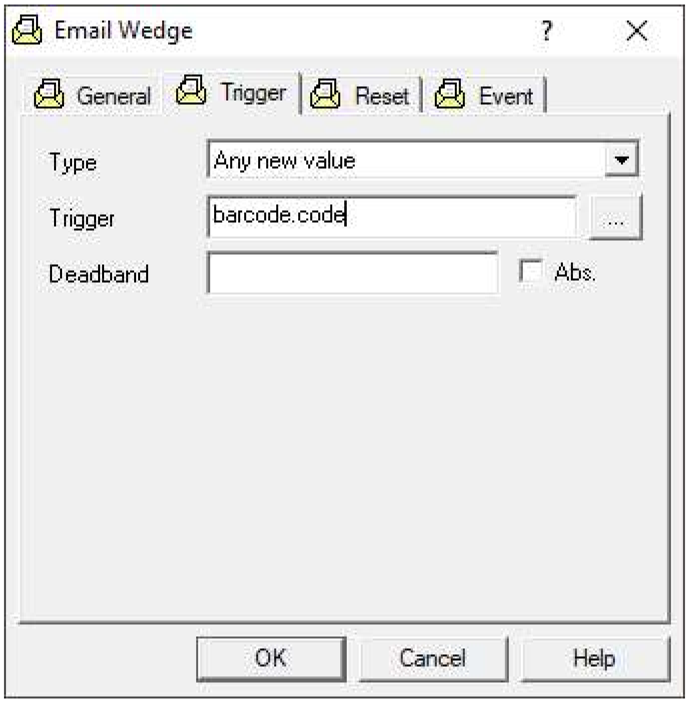
The options available are:

- **Name**: The name of the email instance.
- **Description**: A one-line description of the email instance. This is for documentation purposes only and has no effect on the operation of the email instance.
- **Initial Mode**: Whether or not this email instance is active or not: **Enabled** activates the email instance upon start-up of the server; **Disabled** disables all email activity for this instance. The default is **Enabled**.
- **Notification**: The item used to notify the client that an email action has completed for this particular instance. To select an item, click on the **Browse** button to the right of the text box. If there is an item listed, the value of this item will be set to Logical One upon the successful completion of any action by this email instance (see the **Trigger** tab for a list of actions available).
- **Last Error**: The item used to notify the client of any errors encountered by an action of the email instance. To select an item, click on the **Browse** button to the right of the text box. If there is an item listed, any error encountered by the email instance will be recorded to this item.

7.1.3.2.3.4 The Trigger Tab

[This section applies only to the OmniServer Professional Edition.]

The **Trigger** tab is used to specify how the server is to perform an email instance action. How the action is triggered is based upon the **type** of action on a certain **item**.



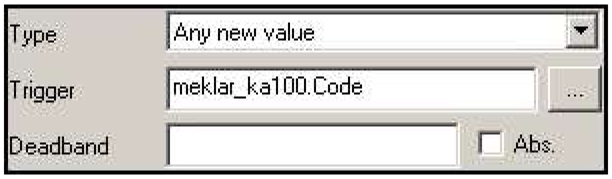
Type: Any new value
Trigger: barcode.code
Deadband:
Abs.

Trigger Plug-in / Wedge - The Trigger Tab (Any New Value Type)

Depending upon the **Type** selected below, you will have available different options. Please click on the links below for the different types:

- **Type**: The type of trigger for this email instance. The type specifies the condition on which the trigger activates. The available types are:

  - Any New Value



Type: Any new value
Trigger: meklar_ka100.Code
Deadband:
Abs.

Email Plug-in / Wedge: Trigger: Any New Value Selection

The options available are:

- **Type**: Causes the email to be triggered upon the receipt of any new value for the trigger. Note that this is not any value - if two identical values are received consecutively, then only the first value will trigger the wedge.
- **Trigger**: The item the server polls for the trigger.

- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the email is readied again to be triggered.

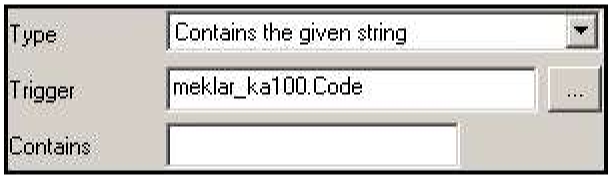
  Normally, an email instance is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the email instance for another trigger. Any values falling within this range will not cause the email to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the email to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

  The default for this field is **<blank>**, meaning that there is no deadband value and any new value will trigger the email to be sent.

- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the email to be sent.

- Contains the Given String



Type	Contains the given string
Trigger	meklar_ka100.Code
Contains	

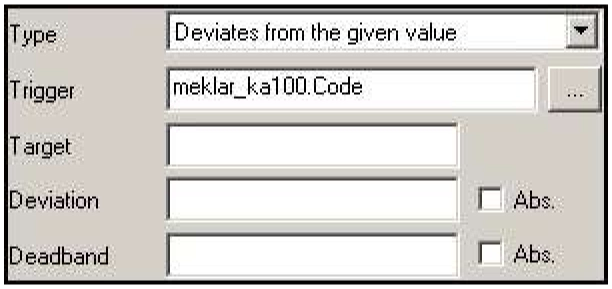
Email Plug-in / Wedge: Trigger: Contains Given String Selection

The options available are:

- **Type**: Causes the email to be triggered when the trigger contains a given string.

- **Trigger**: The item the server polls for the trigger.
- **Contains**: The string that the trigger is to be checked against. The default for this field is **<blank>**, meaning that there is no string to check against. If you use this option, you will need to specify a relevant string for comparison in order to properly trigger the email to be sent for the desired condition.

- Deviates from the given value



Type	Deviates from the given value	
Trigger	meklar_ka100.Code	...
Target		
Deviation		☐ Abs.
Deadband		☐ Abs.

Email Plug-in / Wedge: Trigger: Deviates Selection

The options available are:

- **Type**: Causes the email to be triggered when the value of the trigger deviates from a given value.
- **Trigger**: The item the server polls for the trigger.
- **Target**: The target value for the trigger. This is the base value from which the server computes the deviation.
- **Deviation**: The deviation specifies a percentage above and below the **Target** that the value of the trigger is compared to in order to determine if the email is to be triggered. For example, if the target was "200", and the Deviation is set to "5" (meaning 5%), then for the email to trigger, the next value of the trigger must be less than "190" or greater than "210". The default for this field is **<blank>**, meaning that there is no deviation value and any new value will trigger the email.
- **Deviation Abs.**: The ABS flag tells the server that the number in the **Deviation** text box is to be interpreted as an absolute value and not a percentage. Using the

example above, if the ABS flag is set, then the Deviation value is treated as the number "5", and not 5%. That means the value of the trigger must be less than "195" or greater than "205" to trigger the email (since the target is set for "200").

- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the email is ready again to be triggered.

  Normally, an email instance is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the email instance for another trigger. Any values falling within this range will not cause the email to be triggered.

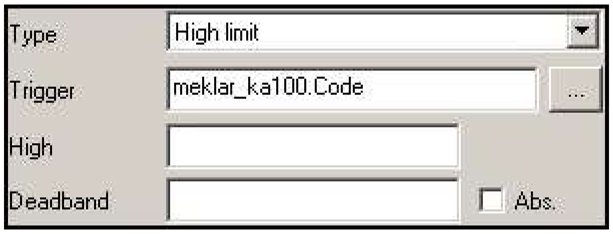
  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the email to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

  Once the email instance is re-activated for triggering, the deviation comes back into effect.

  The default for this field is **<blank>**, meaning that there is no deadband value and any new value will be compared against the target and the deviation as described above.

- **Deadband Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the email.

- High Limit



Type	High limit
Trigger	meklar_ka100.Code
High	
Deadband	
☐ Abs.	

Email Plug-in / Wedge: Trigger: High Limit Selection

The options available are:

- **Type**: Causes the email to be triggered when the value of the trigger moves above the given value.
- **Trigger**: The item the server polls for the trigger.
- **High**: The value on which the server compares the trigger value. If the value of the trigger moves above the value given in this box, the email will be triggered.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below before the email instance is ready again to be triggered.

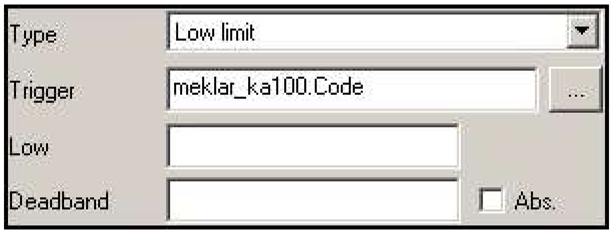
  Normally, an email instance is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a value in which the value of any subsequent triggers must fall below in order to clear the email instance for another trigger. Any values falling above this value will not cause the email to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the email to be triggered again, a value of the trigger must fall below "190". Any values above and equal to "190" will not cause a trigger.

  The default for this field is **<blank>**, meaning that there is no deadband value, and any new value must fall below the target value before the email will be ready for triggering again.
- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an

absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the email.

- Low Limit



Type	Low limit
Trigger	meklar_ka100.Code
Low	
Deadband	☐ Abs.

Email Plug-in / Wedge: Trigger: Low Limit Selection

The options available are:

- **Type**: Causes the email instance to be triggered when the value of the trigger moves below the given value.
- **Trigger**: The item the server polls for the trigger.
- **Low**: The value on which the server compares the trigger value. If the value of the trigger moves below the value given in this box, the email will be triggered.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must move above before the email instance is readied again to be triggered.

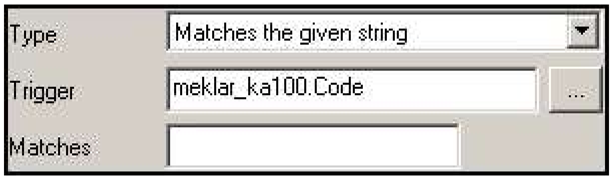
  Normally, an email instance is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a value in which the value of any subsequent triggers must move above in order to clear the email instance for another trigger. Any values falling below this value will not cause the email to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the email to be triggered again, a value of the trigger must move above "210". Any values below and equal to "210" will not cause a trigger.

The default for this field is **<blank>**, meaning that there is no deadband value, and any new value must be above the target value before the email will be ready for triggering again.

- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the email.

- Matches the Given String



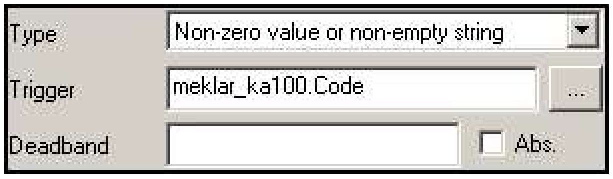
Type: Matches the given string
Trigger: meklar_ka100.Code
Matches:

Email Plug-in / Wedge: Trigger: Matches Given String Selection

The options available are:

- **Type**: Causes the email instance to be triggered when the value of the trigger matches the string specified in the **Matches** box. This string is case sensitive and must match exactly.
- **Trigger**: The item the server polls for the trigger.
- **Matches**: The value on which the server compares the trigger value. If the value of the trigger exactly matches this value, then the email will be triggered.

- Non-zero Value or non-empty string



Type	Non-zero value or non-empty string	
Trigger	meklar_ka100.Code	...
Deadband		☐ Abs.

Email Plug-in / Wedge: Trigger: Non-zero Value Selection

The options available are:

- **Type**: Causes the email instance to be triggered when the value of the trigger contains a non-zero value or a non-empty string.
- **Trigger**: The item the server polls for the trigger.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the email is ready again to be triggered.

  Normally, an email instance is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the email instance for another trigger. Any values falling within this range will not cause the email to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the email to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

  Please note that the deadband will not work properly with non-numeric strings.

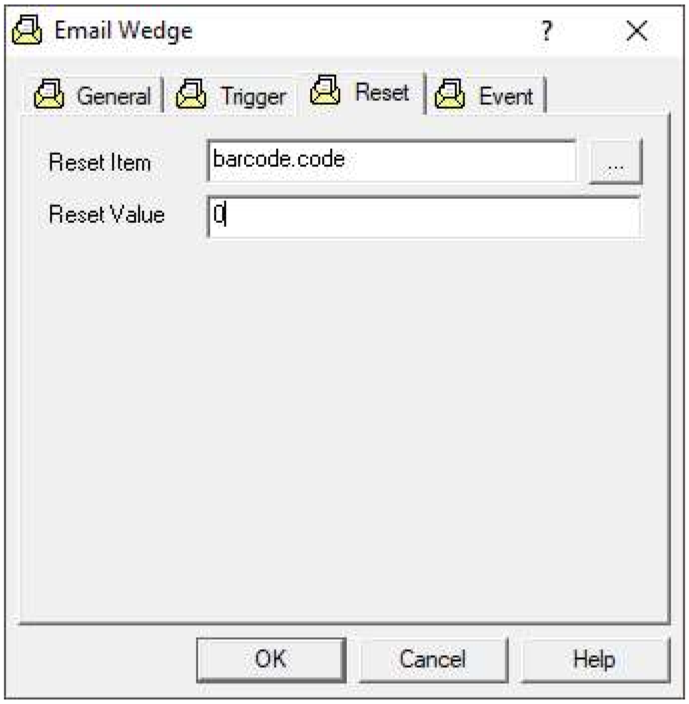
  The default for this field is **<blank>**, meaning that there is no deadband value and any new value will trigger the email.
- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is

treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the email.

7.1.3.2.3.5 The Reset Tab

[This section applies only to the OmniServer Professional Edition.]

The Reset Tab is where you tell the email instance what item to reset (and to what value) after the email has been triggered. One use for the Reset value is to reset the value of a trigger once the wedge has completed its action, thereby forcing the wedge to record all occurrences of the trigger and not just new values (useful when triggering on items that will frequently be updated with identical values).



Reset Item: barcode.code
Reset Value: 0

Email Plug-in / Wedge - The Reset Tab

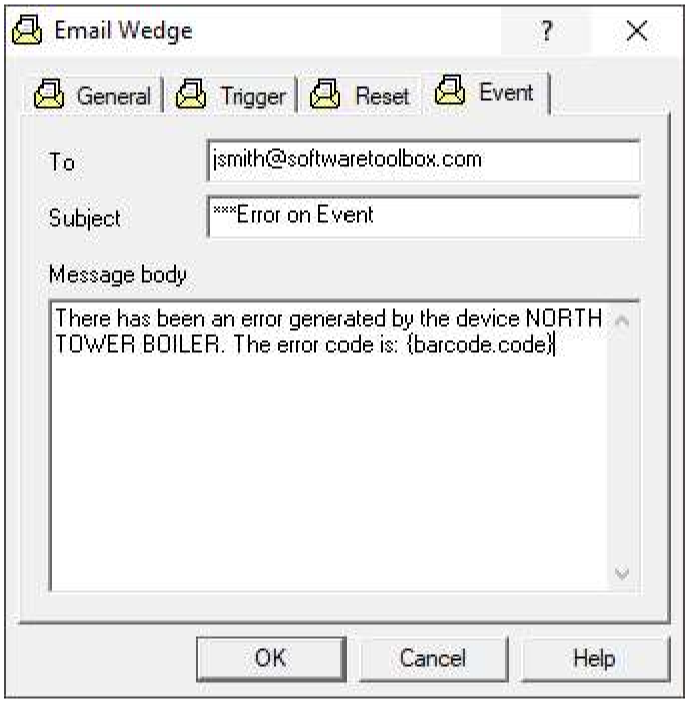
The options available are:

- **Reset Item**: The item that the **Reset Value** will be applied to after completing the triggered action. Use the **Browse** button to select your item.
- **Reset Value**: The value that will be written to the **Reset Item**.

7.1.3.2.3.6 The Event Tab

[This section applies only to the OmniServer Professional Edition.]

The Events Tab defines the email and data format to which the the server will send the trigger information.



To: jsmith@softwaretoolbox.com

Subject: ***Error on Event

Message body: There has been an error generated by the device NORTH TOWER BOILER. The error code is: {barcode.code}

Email Plug-in / Wedge - The Events Tab

The options available are:

- **To**: The email address to which this event will be sent.
- **Subject**: Text that will appear on the Subject line of the email.
- **Message**: This text box contains the format of the message that will be sent to the email once the email instance is triggered. **Please note that it is your responsibility to insure that this email is not considered Spam by email servers.** For a complete explanation of how the data must be formatted, including how to insert values, timestamps and qualities for OmniServer protocol items, please see the section on Expressions.

7.1.3.2.4 Database Plug-in / Wedge

[This section applies only to the OmniServer Professional Edition.]

The server's **Database Wedge** is designed to output data from the server to any ODBC-compliant database. A database wedge is activated by some change in a data value (called a **Trigger**) which causes some data to be logged to a pre-defined ODBC database.

For a video tutorial on working with the Database Wedge, click here.

For information on ODBC databases, please see your Windows documentation.

How it works:

Once the server's runtime engine is activated, the server will search through the listing of wedges and perform the following operations:

- The wedge is checked to see if it is enabled. If the wedge is disabled, then the processes that follow are not executed, and the server continues to the next wedge
- The **Trigger** information is then checked to find out if the data logging of the wedge is performed. If the trigger conditions are not satisfied, the wedge is not processed further and the server continues to the next wedge.
- The wedge will then look at the **Reset** section to see if any server item's value will be reset to a specific value.
- The **Database** and **Data** sections are then used to determine: 1) The database and table to which the data will be logged; 2) How the information will be posted (Add or Replace); 3) The column that will be affected by the change; and 4) The actual data that will be logged, by either an Item or constant data, such as a string. Should any errors occur at this time, the error message will be logged to the item specified in the **General** section under the **Last Error** item.
- Finally, if there is an item in in the **Notification Item** box in the **General** section, the server will set the value of that item to Logical One.

How to use the Database Wedge

You can find the database wedge in the **Clients** section of the server's Configuration program. Locate the Database Wedge Icon (shown below) and double-click on it.

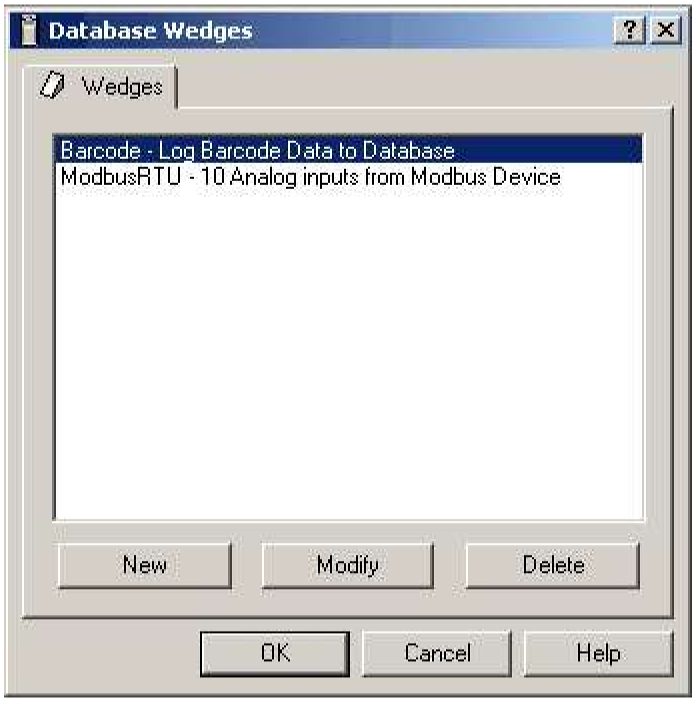
The Client Database Icon

- Creating, Modifying or Deleting a Wedge
- The General Tab
- The Trigger Tab
- The Reset Tab
- The Database Tab
- The Data Tab

7.1.3.2.4.1 Creating a Database Wedge

[This section applies only to the OmniServer Professional Edition.]

The first screen you see allows one to create, modify, or delete a wedge.



Barcode - Log Barcode Data to Database
ModbusRTU - 10 Analog inputs from Modbus Device

The Database Wedge Selection Screen

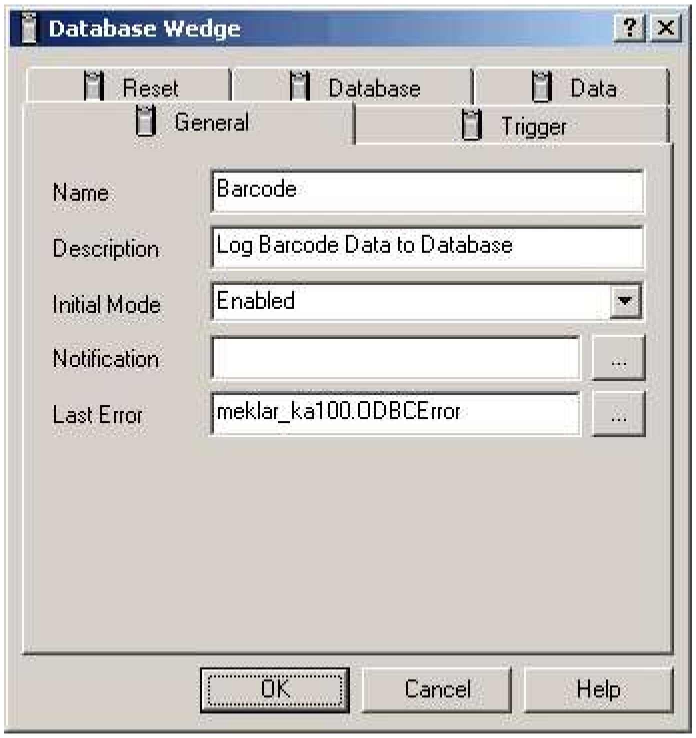
The selections are:

- **New**: Creates a new Database Wedge. You will then be immediately taken the the **General** tab, where you can name your wedge.
- **Modify**: Opens up the selected wedge and allows you to begin changing it's properties. This option is not available if there is no wedge selected.
- **Delete**: Deletes the currently selected wedge. This option is not available if there is no wedge selected.

7.1.3.2.4.2 The General Tab

[This section applies only to the OmniServer Professional Edition.]

The General Tab is where you define the basic properties of the wedge.



Name: Barcode
Description: Log Barcode Data to Database
Initial Mode: Enabled
Notification:
Last Error: meklar_ka100.ODBCError

Database Wedge - The General Tab

The options available are:

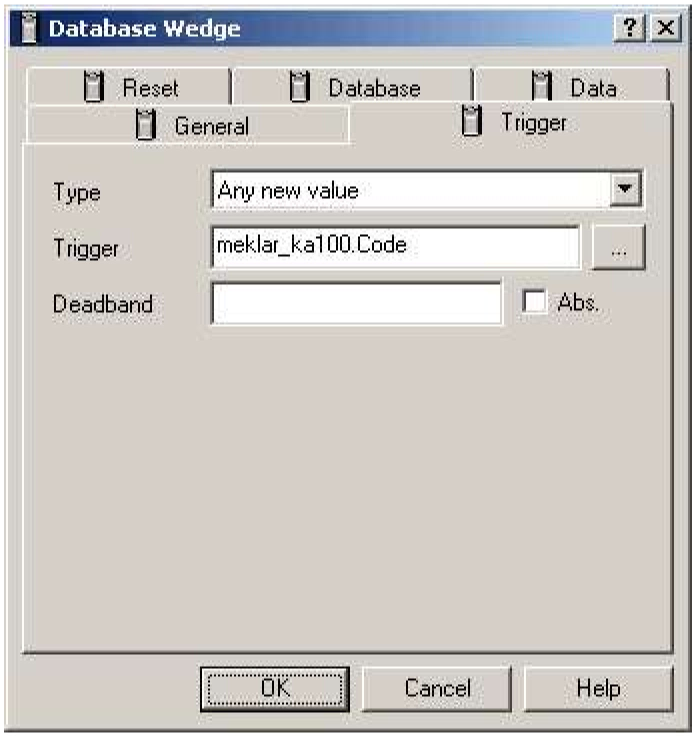
- **Name**: The name of the wedge.
- **Description**: A one-line description of the wedge. This is for documentation purposes only and has no effect on the operation of the wedge.

- **Initial Mode**: How the wedge starts up. There are two options available: **Enabled** activates the wedge upon startup of the server; **Disabled** disables all wedge activity for this wedge. The default is **Enabled**.
- **Notification**: The item used to notify the client that a wedge action has completed. To select an item, click on the **Browse** button to the right of the text box. If there is an item listed, the value of this item will be set to Logical One upon the successful completion of any action by this wedge (see the **Trigger** tab for a list of actions available).
- **Last Error**: The item used to notify the client of any errors encountered by an action of the wedge. To select an item, click on the **Browse** button to the right of the text box. If there is an item listed, any error encountered by the wedge will be recorded to this item.

7.1.3.2.4.3 The Trigger Tab

[This section applies only to the OmniServer Professional Edition.]

The **Trigger** tab is used to specify how the server is to perform a wedge action. How the action is triggered is based upon the **type** of action on a certain **item**.



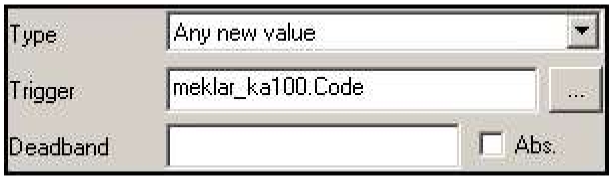
Type: Any new value
Trigger: meklar_ka100.Code
Deadband:
Abs.: ☐

Database Wedge - The Trigger Tab (Any New Value Type)

Depending upon the **Type** selected below, you will have available different options. Please click on the links below for the different types:

- **Type**: The type of trigger for this wedge. The type specifies the condition on which the trigger activates. The available types are:

  - Any New Value



Type: Any new value
Trigger: meklar_ka100.Code
Deadband:
Abs.: ☐

Database Wedge: Trigger: Any New Value Selection

The options available are:

- **Type**: Causes the wedge to be triggered upon the receipt of any new value for the trigger. Not that this is not any value - if two like values are received consecutively, then only the first value will trigger the wedge.

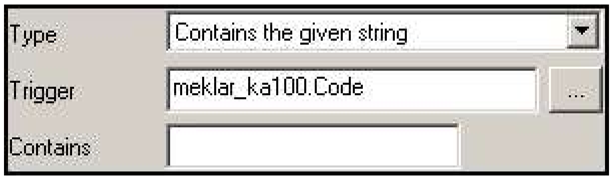
- **Trigger**: The item the server polls for the trigger.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is readied again to be triggered.

  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

  The default for this field is **<blank>**, meaning that there is no deadband value and any new value will trigger the wedge.
- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

- Contains the Given String

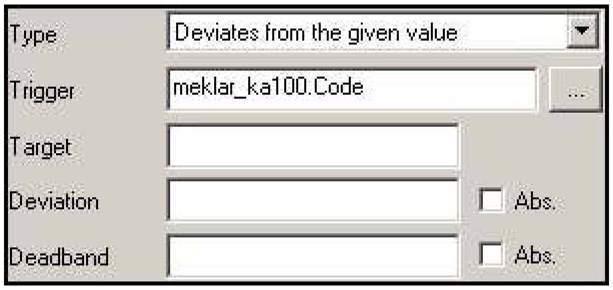

Type: Contains the given string
Trigger: meklar_ka100.Code
Contains:

Database Wedge: Trigger: Contains Given String Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the trigger contains a given string.
- **Trigger**: The item the server polls for the trigger.
- **Contains**: The string that the trigger is to be checked against. The default for this field is **<blank>**, meaning that there is no string to check against. If you use this option, it's a good idea to place some type of data in this field.

- Deviates from the given value



Type	Deviates from the given value	
Trigger	meklar_ka100.Code	...
Target		
Deviation		☐ Abs.
Deadband		☐ Abs.

Database Wedge: Trigger: Deviates Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger deviates from a given value.
- **Trigger**: The item the server polls for the trigger.
- **Target**: The target value for the trigger. This is the base value from which the server computes the deviation.
- **Deviation**: The deviation specifies a percentage above and below the **Target** that the value of the trigger is compared to in order to determine if the wedge is to be triggered. For example, if the target was "200", and the Deviation is set to "5" (meaning 5%), then for the wedge to trigger, the next value of the trigger must be less than "190" or greater than "210". The default for this field is **<blank>**, meaning that there is no deviation value and any new value will trigger the wedge.

- **Deviation Abs.**: The ABS flag tells the server that the number in the **Deviation** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deviation value is treated as the number "5", and not 5%. That means the value of the trigger must be less than "195" or greater than "205" to trigger the wedge (since the target is set for "200").

- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is readied again to be triggered.

  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

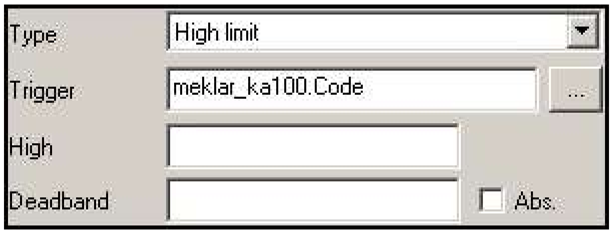
  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

  Once the wedge is re-activated for triggering, the deviation comes back into effect.

  The default for this field is **<blank>**, meaning that there is no deadband value and any new value will be compared against the target and the deviation as described above.

- **Deadband Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

- High Limit



Type: High limit
Trigger: meklar_ka100.Code
High:
Deadband:

Database Wedge: Trigger: High Limit Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger moves above the given value.
- **Trigger**: The item the server polls for the trigger.
- **High**: The value on which the server compares the trigger value. If the value of the trigger moves above the value given in this box, the wedge will be triggered.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below before the wedge is readied again to be triggered.

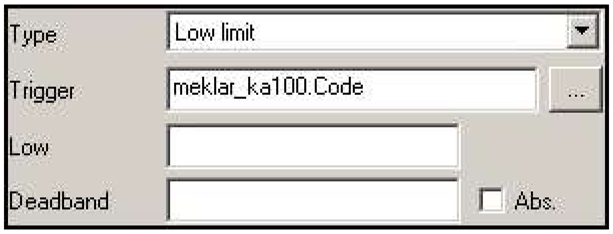
  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a value in which the value of any subsequent triggers must fall below in order to clear the wedge for another trigger. Any values falling above this value will not cause the wedge to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190". Any values above and equal to "190" will not cause a trigger.

  The default for this field is **<blank>**, meaning that there is no deadband value, and any new value must fall below the target value before the wedge will be readied for triggering again.

- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

- Low Limit



Type: Low limit
Trigger: meklar_ka100.Code
Low:
Deadband:
Abs. [ ]

Database Wedge: Trigger: Low Limit Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger moves below the given value.
- **Trigger**: The item the server polls for the trigger.
- **Low**: The value on which the server compares the trigger value. If the value of the trigger moves below the value given in this box, the wedge will be triggered.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must move above before the wedge is readied again to be triggered.

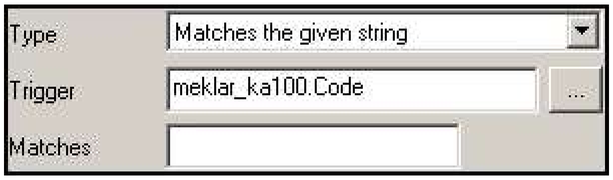
  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a value in which the value of any subsequent triggers must move above in order to clear the wedge for another trigger. Any values falling below this value will not cause the wedge to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must move above "210". Any values below and equal to "210" will not cause a trigger.

The default for this field is **<blank>**, meaning that there is no deadband value, and any new value must be above the target value before the wedge will be readied for triggering again.

- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

- Matches the Given String



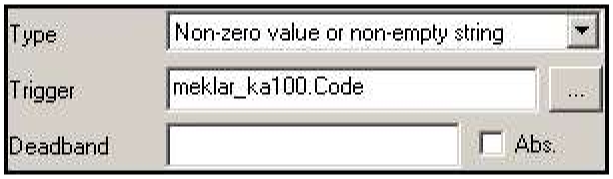
Type: Matches the given string
Trigger: meklar_ka100.Code
Matches:

Database Wedge: Trigger: Matches Given String Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger matches the string specified in the **Matches** box. This string is case sensitive and must match exactly.
- **Trigger**: The item the server polls for the trigger.
- **Matches**: The value on which the server compares the trigger value. If the value of the trigger exactly matches this value, then the wedge will be triggered.

- Non-zero Value or non-empty string



Type	Non-zero value or non-empty string	
Trigger	meklar_ka100.Code	...
Deadband		☐ Abs.

Database Wedge: Trigger: Non-zero Value Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger contains a non-zero value or a non-empty string.
- **Trigger**: The item the server polls for the trigger.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is readied again to be triggered.

  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

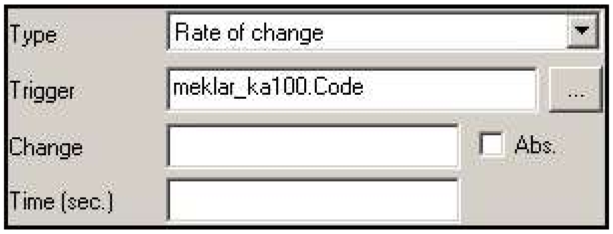
  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

  Please note that the deadband will not work properly with non-numeric strings.

  The default for this field is **<blank>**, meaning that there is no deadband value and any new value will trigger the wedge.
- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is

treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

- Rate of Change (Not Available at this Time)



Database Wedge: Trigger: Rate of Change Selection

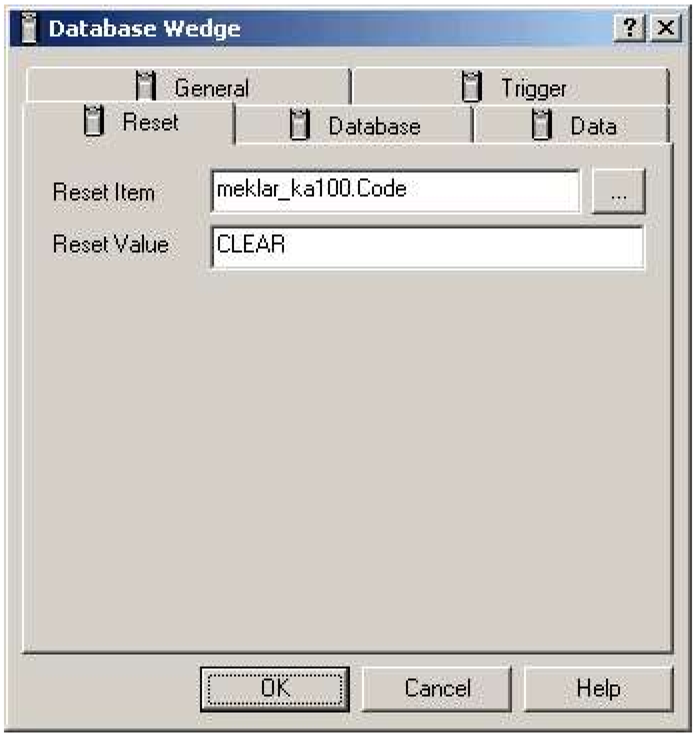
The options available are:

- **Type**: Causes the wedge to be triggered when the rate of change of the trigger over time exceeds the value in the **Change** box.
- **Trigger**: The item the server polls for the trigger.
- **Change**: The percentage value on which the server compares the trigger value in relationship to the **Time** value. If the value of the trigger has changed more than this value under or equal to the time value in **Time**, then the wedge will be triggered. For example, if the value of Change was "5", then the value of the trigger must have changed by 5% within the **Time** specified.
- **Abs.**: The ABS flag tells the server that the Change value is to be interpreted as a number and not a percentage.
- **Time (sec.)**: The amount of time in determining if the trigger value has changed.

7.1.3.2.4.4 The Reset Tab

[This section applies only to the OmniServer Professional Edition.]

The Reset Tab is where you tell the wedge what item to reset (and to what value) just before the action of the wedge is executed. One use for the Reset value is to reset the value of a trigger once the wedge has completed its action, thereby forcing the wedge to record all occurrences of the trigger and not just new values.



Reset Item: meklar_ka100.Code

Reset Value: CLEAR

Database Wedge - The Reset Tab

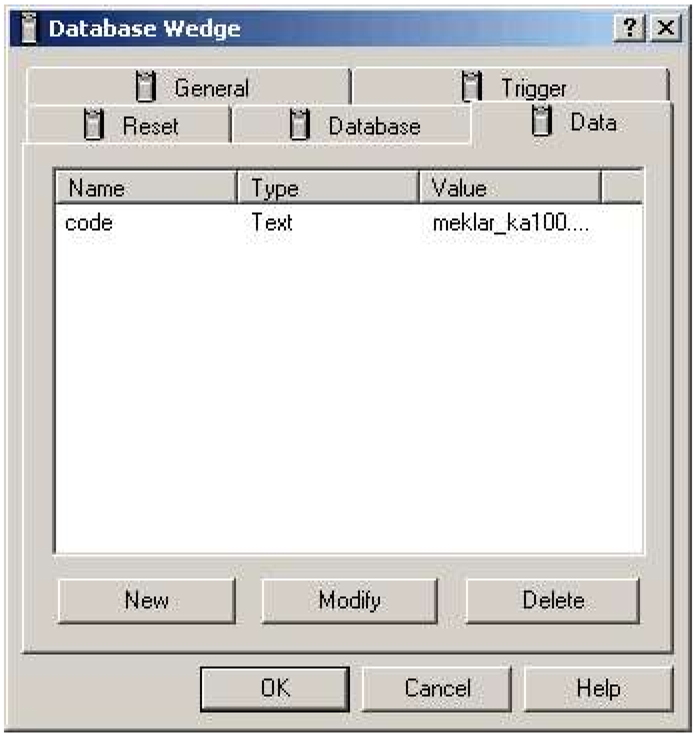
The options available are:

- **Reset Item**: The item that the **Reset Value** will be applied just before the wedge has completed its action. Use the **Browse** button to select your item.
- **Reset Value**: The value that will be inserted into the **Reset Item**. This can be any legal expression. For an explanation of expressions, please see this page.

7.1.3.2.4.5 The Data Tab

[This section applies only to the OmniServer Professional Edition.]

The Data Tab defines the column data that will be added to the ODBC database once the wedge has been triggered.



Name	Type	Value
code	Text	meklar_ka100....

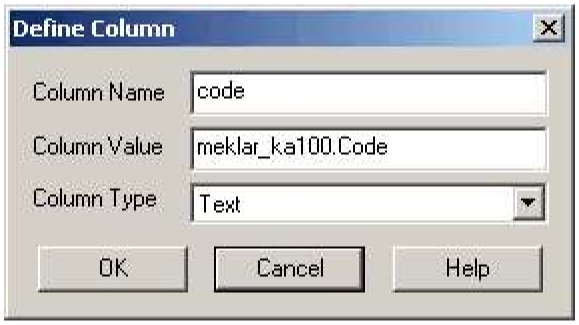
Database Wedge - The Data Tab

The listing gives all the columns in the table that will be changed once the wedge action is executed. Clicking on the **New** button will create a new column; clicking on the **Modify** button will allow one to change a selected column; and **Delete** will delete the reference to that column.

Whenever you click on the **New** or **Modify** buttons, you can then define a new column or change an old one. That dialog box is called the **Define Column** dialog box. Help for that dialog is located here.

[This section applies only to the OmniServer Professional Edition.]

The **Define Column** dialog box is used whenever you click on **New** or **Modify** when defining columns on the **Data Tab** of the Database Wedge.


Define Column

Column Name: code

Column Value: meklar_ka100.Code

Column Type: Text

Database Wedge - The Data Tab: Define Column

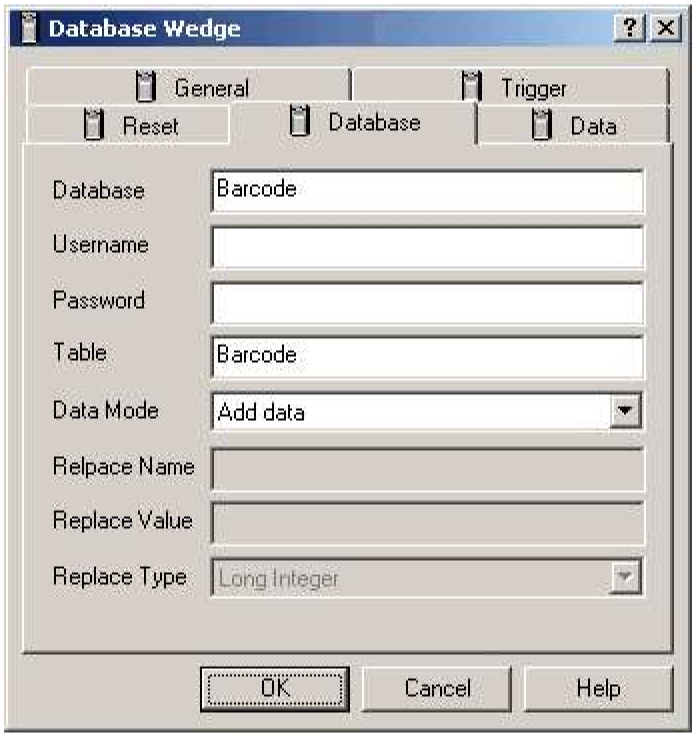
The options available are:

- **Column Name**: The name of the column that will receive the data. This column name must appear in the table specified in the **Database** tab.
- **Column Value**: The value to set the column. This is a regular expression recognized by the wedge. For an explanation of what expressions are allowed, please see the section on Expressions.
- **Column Type**: The data type of the column.

7.1.3.2.4.6 The Database Tab

[This section applies only to the OmniServer Professional Edition.]

The Database Tab defines the database to which the the server will send the trigger information.



Database	Barcode
Username	
Password	
Table	Barcode
Data Mode	Add data
Relpace Name	
Replace Value	
Replace Type	Long Integer

Database Wedge - The Database Tab

The options available are:

- **Database**: The name of the database. This name must also appear in the ODBC (Data Sources) database list.
- **Username**: The username needed to access the database. If left blank, no username will be used.
- **Password**: The password to use with the username for access to the database. If left blank, no password will be used.
- **Table**: The table on which all the wedge action will take place. This table must be within the database mentioned in the **Database** field.
- **Data Mode**: Tells the wedge the action to perform on the table. There are two possible values: **Add Data** adds data into the table, while **Replace Data** replaces a specific column with a specific value.
- **Replace Name**: If the **Data Mode** is set to "Replace Data", this field specifies the column in the table that a value will be replaced. Instead of simply adding records to the end of the database, this will search the database using this field as the name of a column

- **Replace Value**: If the **Data Mode** is set to "Replace Data", this field specifies the data to search on the **Replace Name** field. If this value cannot be found in the database, no replacement will be made.
- **Replace Type:** The data type of the replaced value.

7.1.3.2.5 Keyboard Plug-in / Wedge

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

The server's **Keyboard Wedge** is designed to output data from the server to any Windows program that will accept keystrokes via the SendKeys function. The keyboard wedge is activated by some change in a data value (called a **Trigger**) which causes some data to be sent to a pre-defined program via keystrokes.

How it works:

Once polling is activated, the server will search through the listing of wedges and perform the following operations:

- The wedge is checked to see if it is enabled. If the wedge is disabled, then the processes that follow are not executed, and the server continues to the next wedge
- The **Trigger** information is then checked to find out if the data logging of the wedge is performed. If the trigger conditions are not satisfied, the wedge is not processed further and the server continues to the next wedge.
- The wedge will then look at the **Reset** section to see if any server item's value will be reset to a specific value.
- Finally, the wedge will look at the **Events** section and determine not only the program to which to send the information, but also the format of the data. Should any errors occur at this time, the error message will be logged to the item specified in the **General** section under the **Last Error** item.
- Finally, if there is an item in in the **Notification Item** box in the **General** section, the server will set the value of that item to Logical One.

How to use the Keyboard Wedge

You can find the keyboard wedge in the **Clients** section of the server's Configuration program. Locate the keyboard Wedge Icon (shown below) and double-click on it.

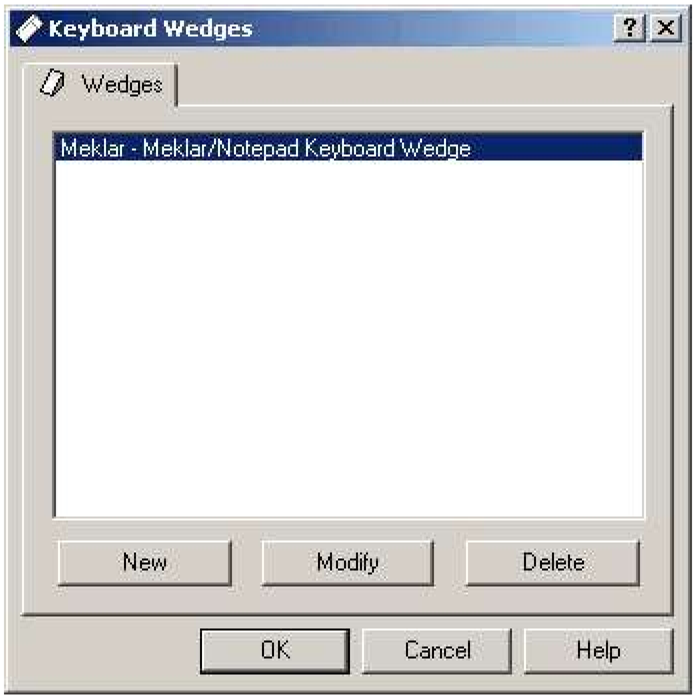
The Client Keyboard Wedge Icon

- Creating, Modifying or Deleting a Wedge
- The General Tab
- The Trigger Tab
- The Reset Tab
- The Events Tab

7.1.3.2.5.1 Creating a Keyboard Wedge

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

The first screen you see allows one to create, modify, or delete a wedge.



The Keyboard Wedge Selection Screen

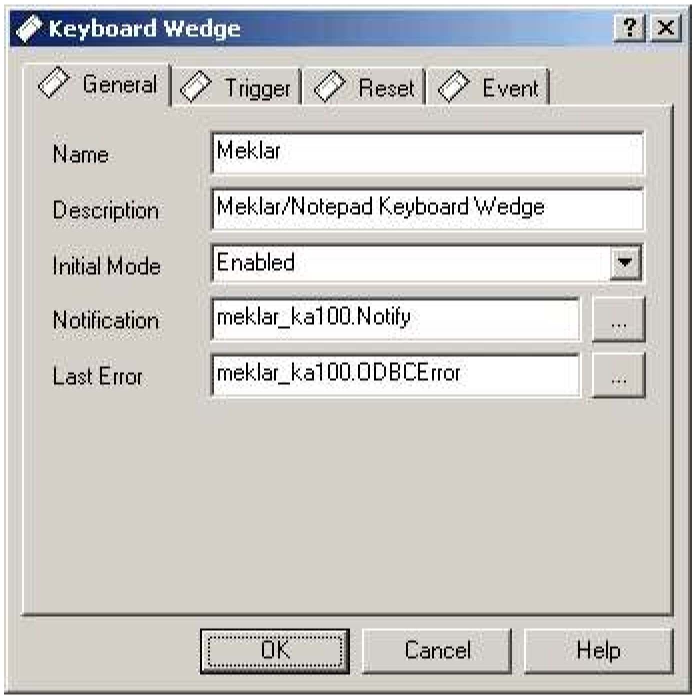
The selections are:

- **New**: Creates a new Keyboard Wedge. You will then be immediately taken the the **General** tab, where you can name your wedge.
- **Modify**: Opens up the selected wedge and allows you to begin changing it's properties. This option is not available if there is no wedge selected.
- **Delete**: Deletes the currently selected wedge. This option is not available if there is no wedge selected.

7.1.3.2.5.2 The General Tab

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

The General Tab is where you define the basic properties of the wedge.



Name: Meklar
Description: Meklar/Notepad Keyboard Wedge
Initial Mode: Enabled
Notification: meklar_ka100.Notify
Last Error: meklar_ka100.ODBCError

Keyboard Wedge - The General Tab

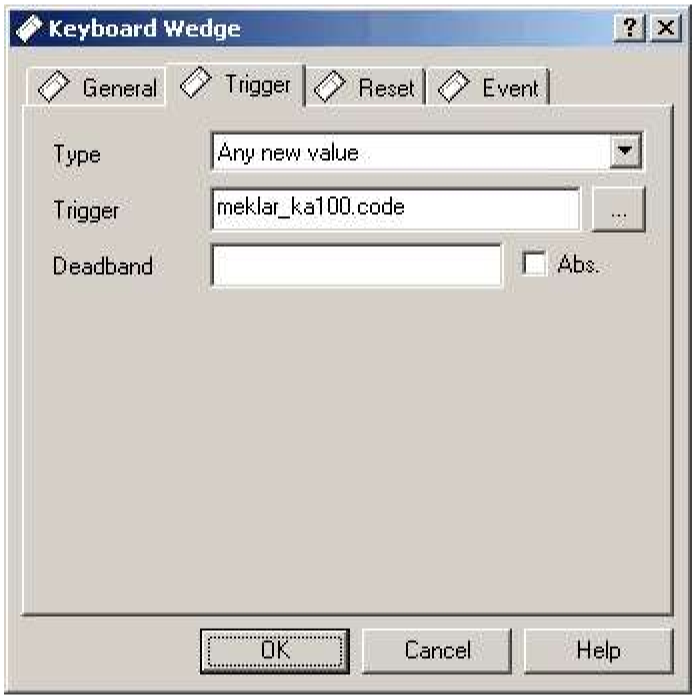
The options available are:

- **Name**: The name of the wedge.
- **Description**: A one-line description of the wedge. This is for documentation purposes only and has no effect on the operation of the wedge.
- **Initial Mode**: How the wedge starts up. There are two options available: **Enabled** activates the wedge upon startup of the server; **Disabled** disables all wedge activity for this wedge. The default is **Enabled**.
- **Notification**: The item used to notify the client that a wedge action has completed. To select an item, click on the **Browse** button to the right of the text box. If there is an item listed, the value of this item will be set to Logical One upon the successful completion of any action by this wedge (see the **Trigger** tab for a list of actions available).
- **Last Error**: The item used to notify the client of any errors encountered by an action of the wedge. To select an item, click on the **Browse** button to the right of the text box. If there is an item listed, any error encountered by the wedge will be recorded to this item.

7.1.3.2.5.3 The Trigger Tab

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

The **Trigger** tab is used to specify how the server is to perform a wedge action. How the action is triggered is based upon the **type** of action on a certain **item**.



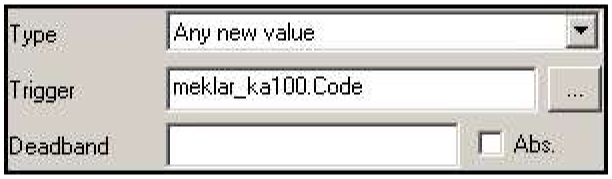
Type: Any new value
Trigger: meklar_ka100.code
Deadband:
Abs.

Keyboard Wedge - The Trigger Tab (Any New Value Type)

Depending upon the **Type** selected below, you will have available different options. Please click on the links below for the different types:

- **Type**: The type of trigger for this wedge. The type specifies the condition on which the trigger activates. The available types are:

  - Any New Value



Type: Any new value
Trigger: meklar_ka100.Code
Deadband:
Abs.

File Wedge: Trigger: Any New Value Selection

The options available are:

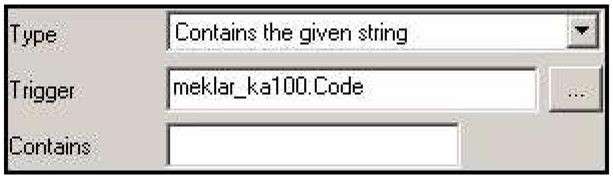
- **Type**: Causes the wedge to be triggered upon the receipt of any new value for the trigger. Not that this is not any value - if two like values are received consecutively, then only the first value will trigger the wedge.
- **Trigger**: The item the server polls for the trigger.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is readied again to be triggered.

  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

  The default for this field is **<blank>**, meaning that there is no deadband value and any new value will trigger the wedge.
- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

- Contains the Given String



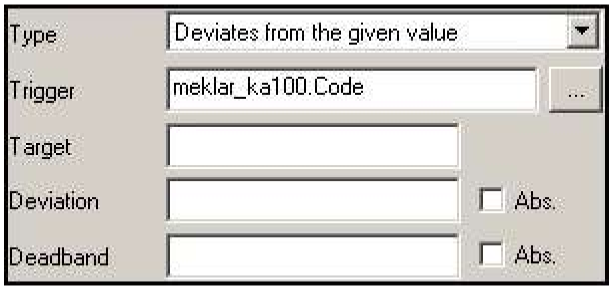
Type	Contains the given string
Trigger	meklar_ka100.Code
Contains	

File Wedge: Trigger: Contains Given String Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the trigger contains a given string.
- **Trigger**: The item the server polls for the trigger.
- **Contains**: The string that the trigger is to be checked against. The default for this field is **<blank>**, meaning that there is no string to check against. If you use this option, it's a good idea to place some type of data in this field.

- Deviates from the given value



Type	Deviates from the given value	
Trigger	meklar_ka100.Code	
Target		
Deviation		☐ Abs.
Deadband		☐ Abs.

File Wedge: Trigger: Deviates Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger deviates from a given value.
- **Trigger**: The item the server polls for the trigger.
- **Target**: The target value for the trigger. This is the base value from which the server computes the deviation.
- **Deviation**: The deviation specifies a percentage above and below the **Target** that the value of the trigger is compared to in order to determine if the wedge is to be

triggered. For example, if the target was "200", and the Deviation is set to "5" (meaning 5%), then for the wedge to trigger, the next value of the trigger must be less than "190" or greater than "210". The default for this field is **<blank>**, meaning that there is no deviation value and any new value will trigger the wedge.

- **Deviation Abs.**: The ABS flag tells the server that the number in the **Deviation** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deviation value is treated as the number "5", and not 5%. That means the value of the trigger must be less than "195" or greater than "205" to trigger the wedge (since the target is set for "200").
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is readied again to be triggered.

  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

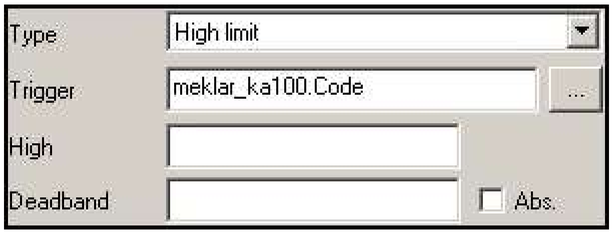
  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

  Once the wedge is re-activated for triggering, the deviation comes back into effect.

  The default for this field is **<blank>**, meaning that there is no deadband value and any new value will be compared against the target and the deviation as described above.

- **Deadband Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

- High Limit



Type	High limit	
Trigger	meklar_ka100.Code	...
High		
Deadband		☐ Abs.

File Wedge: Trigger: High Limit Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger moves above the given value.
- **Trigger**: The item the server polls for the trigger.
- **High**: The value on which the server compares the trigger value. If the value of the trigger moves above the value given in this box, the wedge will be triggered.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below before the wedge is readied again to be triggered.

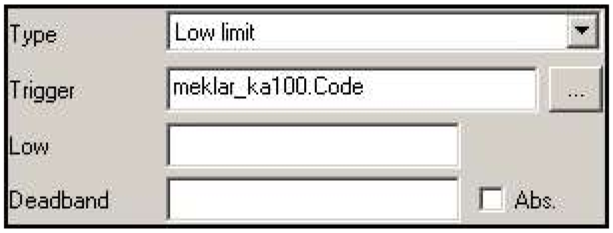
  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a value in which the value of any subsequent triggers must fall below in order to clear the wedge for another trigger. Any values falling above this value will not cause the wedge to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190". Any values above and equal to "190" will not cause a trigger.

The default for this field is **<blank>**, meaning that there is no deadband value, and any new value must fall below the target value before the wedge will be readied for triggering again.

- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

- Low Limit



Type: Low limit
Trigger: meklar_ka100.Code
Low:
Deadband:
☐ Abs.

File Wedge: Trigger: Low Limit Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger moves below the given value.
- **Trigger**: The item the server polls for the trigger.
- **Low**: The value on which the server compares the trigger value. If the value of the trigger moves below the value given in this box, the wedge will be triggered.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must move above before the wedge is readied again to be triggered.

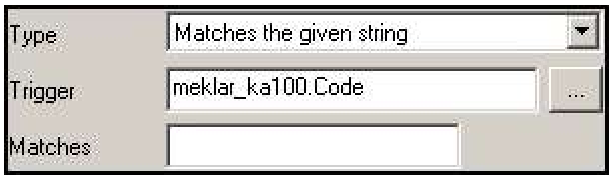
Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a value in which the value of any subsequent triggers must move above in order to clear the wedge for another trigger. Any values falling below this value will not cause the wedge to be triggered.

For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must move above "210". Any values below and equal to "210" will not cause a trigger.

The default for this field is **<blank>**, meaning that there is no deadband value, and any new value must be above the target value before the wedge will be readied for triggering again.

- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

- Matches the Given String


Type: Matches the given string
Trigger: meklar_ka100.Code
Matches:

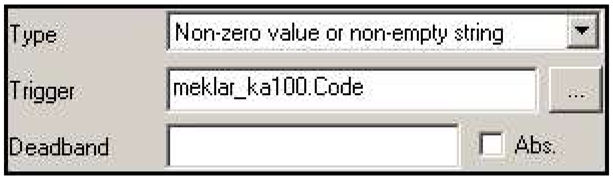
File Wedge: Trigger: Matches Given String Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger matches the string specified in the **Matches** box. This string is case sensitive and must match exactly.
- **Trigger**: The item the server polls for the trigger.

- **Matches**: The value on which the server compares the trigger value. If the value of the trigger exactly matches this value, then the wedge will be triggered.

- Non-zero Value or non-empty string



Type	Non-zero value or non-empty string
Trigger	meklar_ka100.Code
Deadband	

File Wedge: Trigger: Non-zero Value Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger contains a non-zero value or a non-empty string.
- **Trigger**: The item the server polls for the trigger.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is readied again to be triggered.

  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

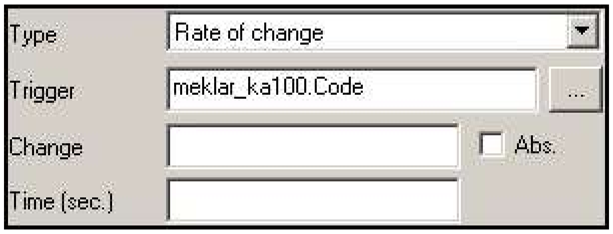
  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

  Please note that the deadband will not work properly with non-numeric strings.

  The default for this field is **<blank>**, meaning that there is no deadband value and any new value will trigger the wedge.

- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

- Rate of Change (Not Available at this Time)


Type: Rate of change
Trigger: meklar_ka100.Code
Change:
Abs.
Time (sec.):

File Wedge: Trigger: Rate of Change Selection

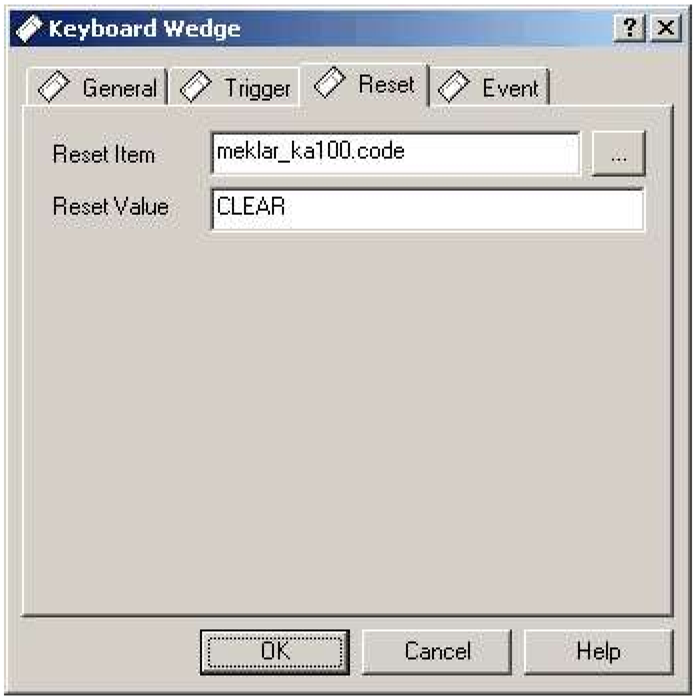
The options available are:

- **Type**: Causes the wedge to be triggered when the rate of change of the trigger over time exceeds the value in the **Change** box.
- **Trigger**: The item the server polls for the trigger.
- **Change**: The percentage value on which the server compares the trigger value in relationship to the **Time** value. If the value of the trigger has changed more than this value under or equal to the time value in **Time**, then the wedge will be triggered. For example, if the value of Change was "5", then the value of the trigger must have changed by 5% within the **Time** specified.
- **Abs.**: The ABS flag tells the server that the Change value is to be interpreted as a number and not a percentage.
- **Time (sec.)**: The amount of time in determining if the trigger value has changed.

7.1.3.2.5.4 The Reset Tab

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

The Reset Tab is where you tell the wedge what item to reset (and to what value) just before the action of the wedge is executed. One use for the Reset value is to reset the value of a trigger once the wedge has completed its action, thereby forcing the wedge to record all occurrences of the trigger and not just new values.



Reset Item: meklar_ka100.code

Reset Value: CLEAR

Keyboard Wedge - The Reset Tab

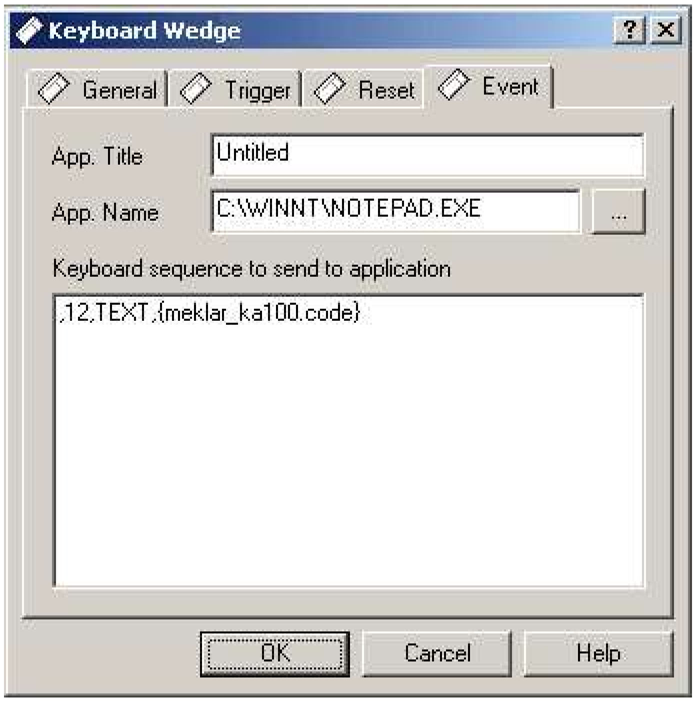
The options available are:

- **Reset Item**: The item that the **Reset Value** will be applied just before the wedge has completed its action. Use the **Browse** button to select your item.
- **Reset Value**: The value that will be inserted into the **Reset Item**.

7.1.3.2.5.5 The Event Tab

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

The Events Tab defines the application title, application name, and keyboard sequence to which the the wedge will send the trigger information.



App. Title: Untitled

App. Name: C:\WINNT\NOTEPAD.EXE

Keyboard sequence to send to application: ,12,TEXT,{meklar_ka100.code}

Keyboard Wedge - The Events Tab

The options available are:

- **App. Title**: The name of the application to which to send the data once the wedge has been triggered.
- **App. Name**: The executable filename of the application. If there is not an active window for this application, the wedge will launch the application specified here. Use the **Browse** button to the right of this box to select your application.
- **Keyboard sequence to send to application**: The data sent to the application once the wedge is triggered. For a complete explanation of how the data must be formatted, please see the section on Expressions.

7.1.3.2.5.6 Special Keystrokes

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

Within the keyboard wedge, special expressions can be used to create keystrokes not normally available with regular text (such as the ESCAPE key). While normal characters are represented by single keystrokes, many of these special keys must be represented by a sequence of characters that the wedge can understand.

How do you use the special keyboard expressions?

All special characters and sequences are surrounded by **braces** ({}). So if you wanted to send an **Escape** key, you would write in the text box this sequence: **{ESCAPE}**. This will tell the wedge not to send out these eight characters, but to send out one ESCAPE character.

What are the Special Characters?

The list below give you all the special characters supported by the wedge. Please note that if you do not see the character or keystroke listed, it's either not supported (such as the Print Screen key), or you can simply type in the correct character straight from the keyboard.

Special Reserved Characters
Use these sequences for the listed keys instead of typing them in directly from the keyboard:

Key	**Sequence**
"+" (Plus)	{+}

"^" (Caret)	{^}
"%" (Percent)	{%}
"~" (Tilde)	{~}
"[" (L. Bracket)	{[}
"]" (R. Bracket)	{]}

Special Keyboard Characters
Use these sequences for the listed keys to represent keys unavailable on the keyboard:

Key	**Sequence**
BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}
BRE	{BREAK}

AK	
CAPSLOCK	{CAPSLOCK}
CLEAR	{CLEAR}
DELETE or DEL	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}

ENTER	{ENTER} or ~
ESCAPE or ESC	{ESCAPE}
HELP	{HELP}
HOME	{HOME}
INSERT or INS	{INSERT} or {INS}
LEFT	{LEFT}

ARROW	
NUMLOCK	{NUMLOCK}
PAGEDOWN	{PGDN}
PAGEUP	{PGUP}
PRINTSCREEN	{PRTSC}
RI	{RIGH

GHTARROW	T}
SCROLLLOCK	{SCROLLLOCK}
TAB	{TAB}
UPARROW	{UP}
F1	{F1}
F2	{F1}
F3	{F3}
F4	{F4}

F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

Can I send a CTRL, ALT, or SHIFT?

To send a CTRL, ALT, or SHIFT in combination with regular keys, simply append one of the characters below before the keystroke:

Key	**Sequence**
SHIFT	+
CTRL	^
ALT	%

For example, to send a CTRL-A, use the sequence **^A**.

If you need to send a sequence of characters while a CTRL, SHIFT, or ALT key is held down, type in the CTRL, SHIFT, or ALT sequence, followed by the characters in parenthesis. For example:

- **^(qw)** - Sends a CTRL-Q and a CTRL-W.
- **^qw** - Sends a CTRL-Q and a lowercase "w".

7.1.3.2.6 File Plug-in / Wedge

[This section applies only to the OmniServer Professional Edition.]

The server's **File Wedge** is designed to output data from the server to any file accessible by the local computer. This includes files both on the local computer and files that are accessible via a network. The file wedge is activated by some change in a data value (called a **Trigger**) which causes some data to be sent to a pre-defined file.

For a video tutorial on working with the Text File Wedge, click here.

How it works:

Once polling is activated, the server will search through the listing of wedges and perform the following operations:

- The wedge is checked to see if it is enabled. If the wedge is disabled, then the processes that follow are not executed, and the server continues to the next wedge
- The **Trigger** information is then checked to find out if the data logging of the wedge is performed. If the trigger conditions are not satisfied, the wedge is not processed further and the server continues to the next wedge.
- The wedge will then look at the **Reset** section to see if any server item's value will be reset to a specific value.
- Finally, the wedge will look at the **Events** section and determine not only the file to which to send the information, but also the format of the data. Should any errors occur at this time, the error message will be logged to the item specified in the **General** section under the **Last Error** item.
- Finally, if there is an item in in the **Notification Item** box in the **General** section, the server will set the value of that item to Logical One.

How to use the File Wedge

You can find the file wedge in the **Clients** section of the server's Configuration program. Locate the File Wedge Icon (shown below) and double-click on it.

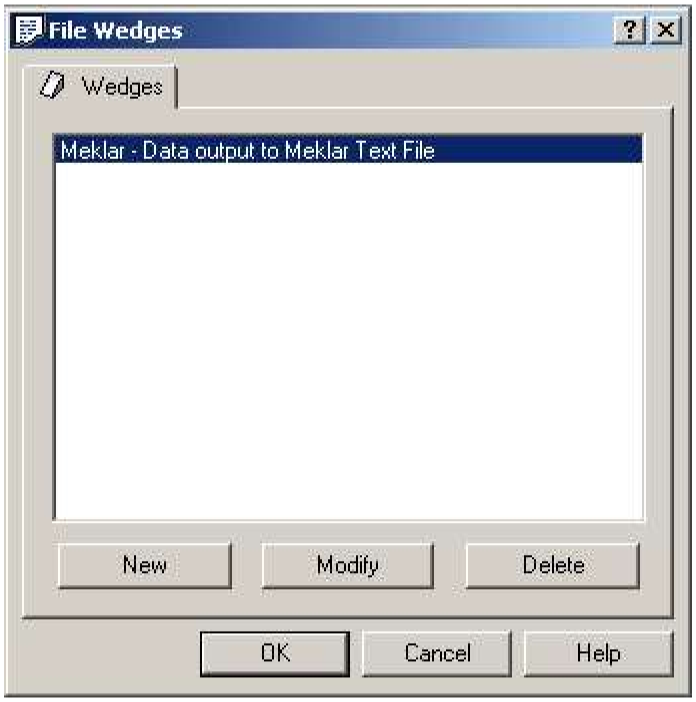
The Client File Wedge Icon

- Creating, Modifying or Deleting a Wedge
- The General Tab
- The Trigger Tab
- The Reset Tab
- The Events Tab

7.1.3.2.6.1 Creating a File Wedge

[This section applies only to the OmniServer Professional Edition.]

The first screen you see allows one to create, modify, or delete a wedge.



Meklar - Data output to Meklar Text File

The File Wedge Selection Screen

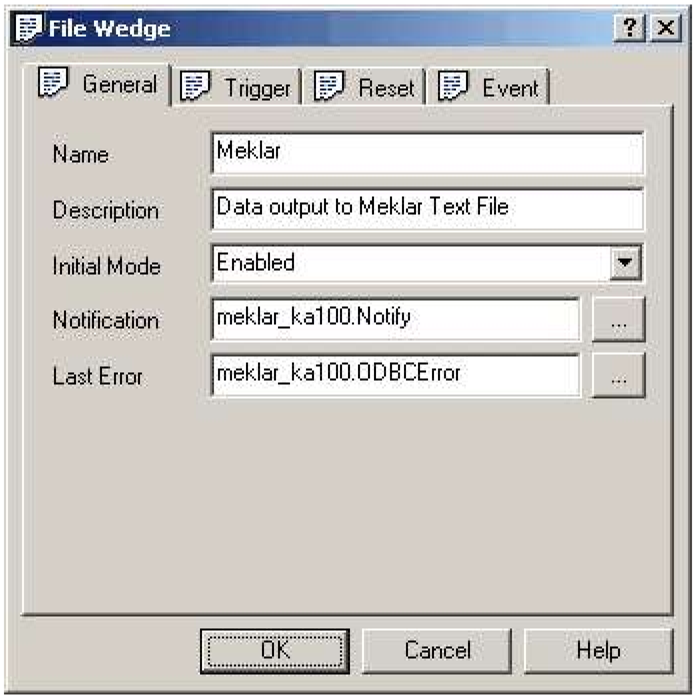
The selections are:

- **New**: Creates a new File Wedge. You will then be immediately taken the the **General** tab, where you can name your wedge.
- **Modify**: Opens up the selected wedge and allows you to begin changing it's properties. This option is not available if there is no wedge selected.
- **Delete**: Deletes the currently selected wedge. This option is not available if there is no wedge selected.

7.1.3.2.6.2 The General Tab

[This section applies only to the OmniServer Professional Edition.]

The General Tab is where you define the basic properties of the wedge.



Name: Meklar
Description: Data output to Meklar Text File
Initial Mode: Enabled
Notification: meklar_ka100.Notify
Last Error: meklar_ka100.ODBCError

File Wedge - The General Tab

The options available are:

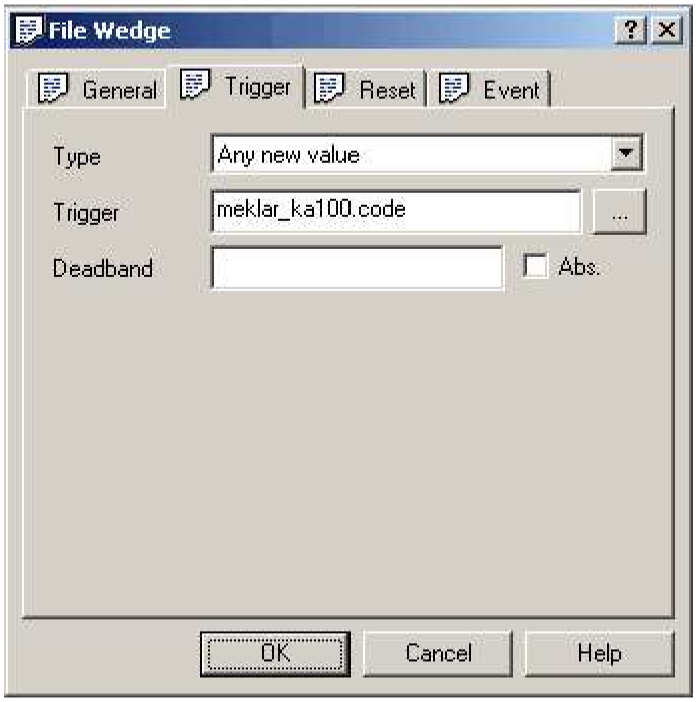
- **Name**: The name of the wedge.
- **Description**: A one-line description of the wedge. This is for documentation purposes only and has no effect on the operation of the wedge.

- **Initial Mode**: How the wedge starts up. There are two options available: **Enabled** activates the wedge upon startup of the server; **Disabled** disables all wedge activity for this wedge. The default is **Enabled**.
- **Notification**: The item used to notify the client that a wedge action has completed. To select an item, click on the **Browse** button to the right of the text box. If there is an item listed, the value of this item will be set to Logical One upon the successful completion of any action by this wedge (see the **Trigger** tab for a list of actions available).
- **Last Error**: The item used to notify the client of any errors encountered by an action of the wedge. To select an item, click on the **Browse** button to the right of the text box. If there is an item listed, any error encountered by the wedge will be recorded to this item.

7.1.3.2.6.3 The Trigger Tab

[This section applies only to the OmniServer Professional Edition.]

The **Trigger** tab is used to specify how the server is to perform a wedge action.



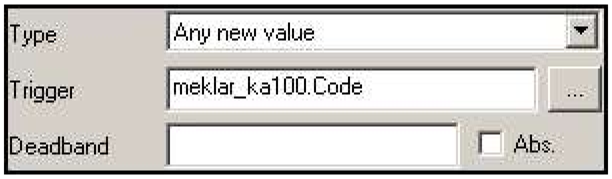
Type: Any new value
Trigger: meklar_ka100.code
Deadband:
Abs. [ ]

Trigger Wedge - The Trigger Tab (Any New Value Type)

Depending upon the **Type** selected below, you will have available different options. Please click on the links below for the different types:

- **Type**: The type of trigger for this wedge. The type specifies the condition on which the trigger activates. The available types are:

Any New Value


Type: Any new value
Trigger: meklar_ka100.Code
Deadband: ☐ Abs.

File Wedge: Trigger: Any New Value Selection

The options available are:

Type: Causes the wedge to be triggered upon the receipt of any new value for the trigger. Not that this is not any value - if two like values are received consecutively, then only the first value will trigger the wedge.

Trigger: The item the server polls for the trigger.

Deadband: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is readied again to be triggered.

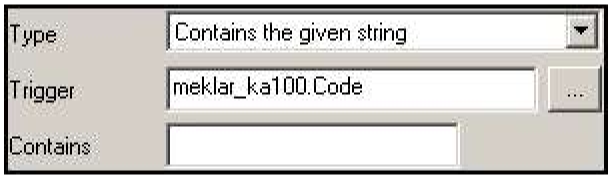
Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

The default for this field is **<blank>**, meaning that there is no deadband value and any new value will trigger the wedge.

Abs.: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

Contains the Given String



Type	Contains the given string
Trigger	meklar_ka100.Code
Contains	

File Wedge: Trigger: Contains Given String Selection

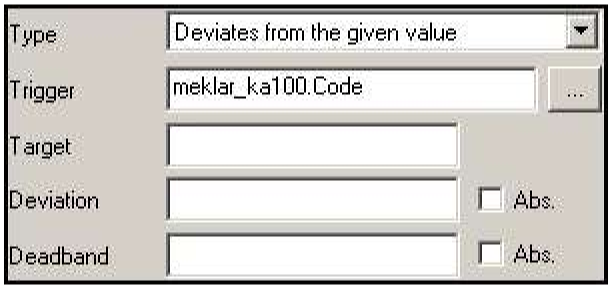
The options available are:

Type: Causes the wedge to be triggered when the trigger contains a given string.

Trigger: The item the server polls for the trigger.

Contains: The string that the trigger is to be checked against. The default for this field is **<blank>**, meaning that there is no string to check against. If you use this option, it's a good idea to place some type of data in this field.

Deviates from the given value



Type	Deviates from the given value	
Trigger	meklar_ka100.Code	
Target		
Deviation		☐ Abs.
Deadband		☐ Abs.

File Wedge: Trigger: Deviates Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger deviates from a given value.
- **Trigger**: The item the server polls for the trigger.
- **Target**: The target value for the trigger. This is the base value from which the server computes the deviation.
- **Deviation**: The deviation specifies a percentage above and below the **Target** that the value of the trigger is compared to in order to determine if the wedge is to be triggered. For example, if the target was "200", and the Deviation is set to "5" (meaning 5%), then for the wedge to trigger, the next value of the trigger must be less than "190" or greater than "210". The default for this field is **<blank>**, meaning that there is no deviation value and any new value will trigger the wedge.
- **Deviation Abs.**: The ABS flag tells the server that the number in the **Deviation** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deviation value is treated as the number "5", and not 5%. That means the value of the trigger must be less than "195" or greater than "205" to trigger the wedge (since the target is set for "200").
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is readied again to be triggered.

  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below

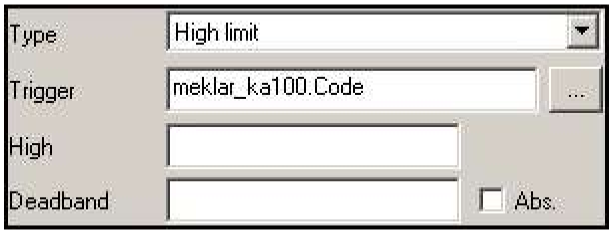
"190" or above "210". Any values within this range will not cause a trigger.

Once the wedge is re-activated for triggering, the deviation comes back into effect.

The default for this field is **<blank>**, meaning that there is no deadband value and any new value will be compared against the target and the deviation as described above.

- **Deadband Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

⊟ High Limit



Type	High limit
Trigger	meklar_ka100.Code
High	
Deadband	☐ Abs.

File Wedge: Trigger: High Limit Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger moves above the given value.
- **Trigger**: The item the server polls for the trigger.
- **High**: The value on which the server compares the trigger value. If the value of the trigger moves above the value given in this box, the wedge will be triggered.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below before the wedge is readied

again to be triggered.

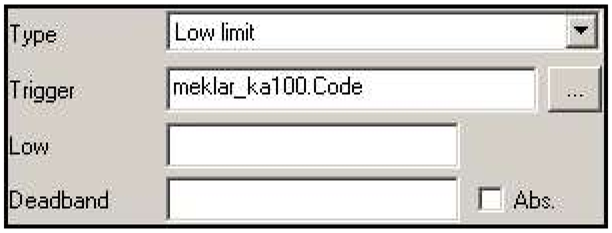
Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a value in which the value of any subsequent triggers must fall below in order to clear the wedge for another trigger. Any values falling above this value will not cause the wedge to be triggered.

For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190". Any values above and equal to "190" will not cause a trigger.

The default for this field is **<blank>**, meaning that there is no deadband value, and any new value must fall below the target value before the wedge will be readied for triggering again.

- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

⊟ Low Limit


Type: Low limit
Trigger: meklar_ka100.Code
Low:
Deadband:
Abs.: ☐

File Wedge: Trigger: Low Limit Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger moves below the given value.

- **Trigger**: The item the server polls for the trigger.
- **Low**: The value on which the server compares the trigger value. If the value of the trigger moves below the value given in this box, the wedge will be triggered.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must move above before the wedge is readied again to be triggered.

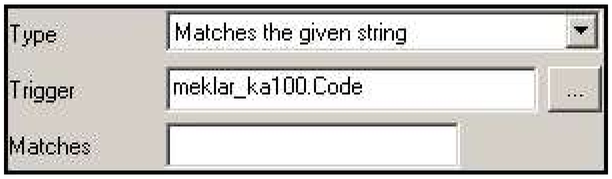
  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a value in which the value of any subsequent triggers must move above in order to clear the wedge for another trigger. Any values falling below this value will not cause the wedge to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must move above "210". Any values below and equal to "210" will not cause a trigger.

  The default for this field is **<blank>**, meaning that there is no deadband value, and any new value must be above the target value before the wedge will be readied for triggering again.
- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger

must be less than "195" or greater than "205" to trigger the wedge.

Matches the Given String

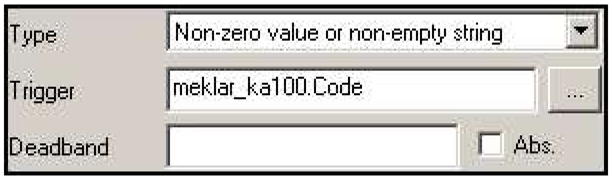

Type: Matches the given string
Trigger: meklar_ka100.Code
Matches:

File Wedge: Trigger: Matches Given String Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger matches the string specified in the **Matches** box. This string is case sensitive and must match exactly.
- **Trigger**: The item the server polls for the trigger.
- **Matches**: The value on which the server compares the trigger value. If the value of the trigger exactly matches this value, then the wedge will be triggered.

Non-zero Value or non-empty string


Type: Non-zero value or non-empty string
Trigger: meklar_ka100.Code
Deadband:

File Wedge: Trigger: Non-zero Value Selection

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger contains a non-zero value or a non-empty string.
- **Trigger**: The item the server polls for the trigger.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is

readied again to be triggered.

Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

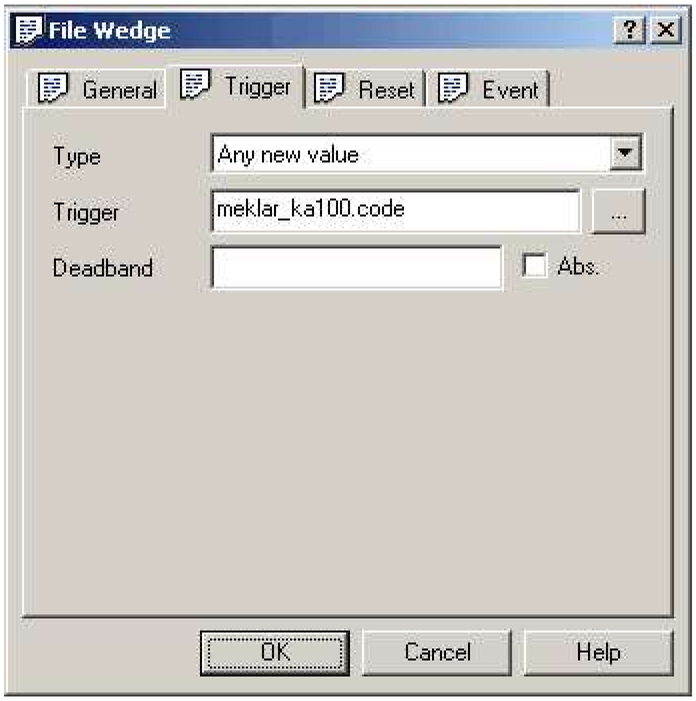
For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

Please note that the deadband will not work properly with non-numeric strings.

The default for this field is **<blank>**, meaning that there is no deadband value and any new value will trigger the wedge.

- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

⊟ Rate of Change (Not Available at this Time)



Type: Any new value
Trigger: meklar_ka100.code
Deadband:
☐ Abs.

File Wedge: Trigger: Rate of Change Selection

The options available are:

Type: Causes the wedge to be triggered when the rate of change of the trigger over time exceeds the value in the **Change** box.

Trigger: The item the server polls for the trigger.

Change: The percentage value on which the server compares the trigger value in relationship to the **Time** value. If the value of the trigger has changed more than this value under or equal to the time value in **Time**, then the wedge will be triggered. For example, if the value of Change was "5", then the value of the trigger must have changed by 5% within the **Time** specified.

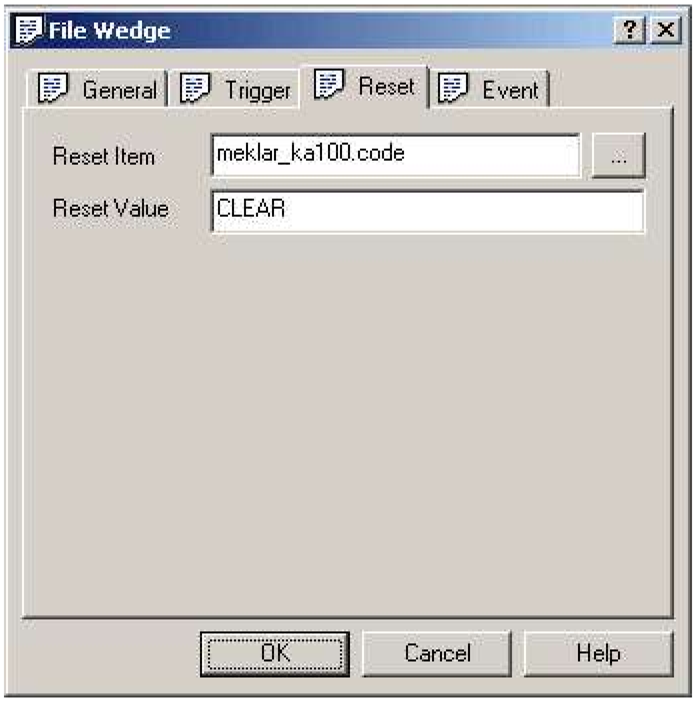
Abs.: The ABS flag tells the server that the Change value is to be interpreted as a number and not a percentage.

Time (sec.): The amount of time in determining if the trigger value has changed.

7.1.3.2.6.4 The Reset Tab

[This section applies only to the OmniServer Professional Edition.]

The Reset Tab is where you tell the wedge what item to reset (and to what value) just before the action of the wedge is executed. One use for the Reset value is to reset the value of a trigger once the wedge has completed its action, thereby forcing the wedge to record all occurrences of the trigger and not just new values.



Reset Item: meklar_ka100.code
Reset Value: CLEAR

File Wedge - The Reset Tab

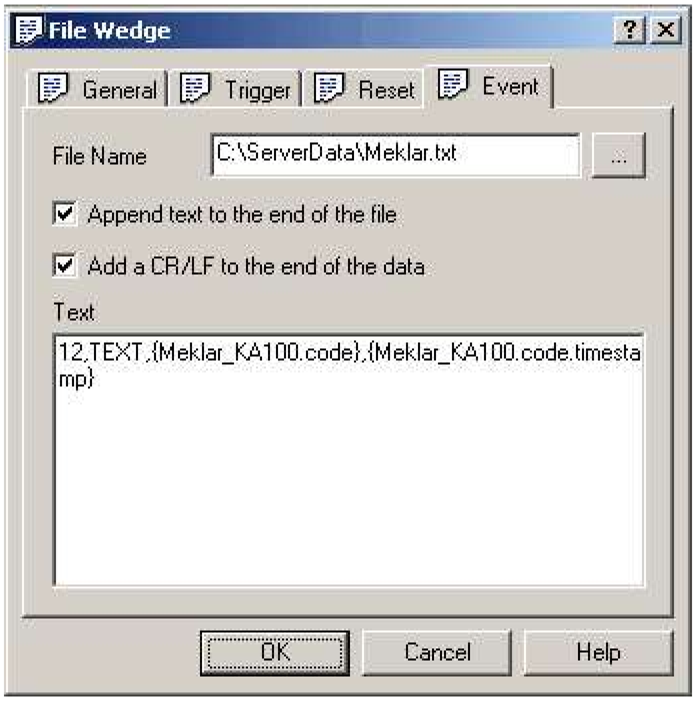
The options available are:

- **Reset Item**: The item that the **Reset Value** will be applied just before the wedge has completed its action. Use the **Browse** button to select your item.
- **Reset Value**: The value that will be inserted into the **Reset Item**.

7.1.3.2.6.5 The Event Tab

[This section applies only to the OmniServer Professional Edition.]

The Events Tab defines the file and data format to which the the server will send the trigger information.


File Name: C:\ServerData\Meklar.txt

☑ Append text to the end of the file

☑ Add a CR/LF to the end of the data

Text: 12,TEXT,{Meklar_KA100.code},{Meklar_KA100.code.timestamp}

File Wedge - The Events Tab

The options available are:

- **File Name**: The name of the file to which to send the data once the wedge has been triggered. The file will be created if it does not exist. However, all directories **must** exist, otherwise the wedge will return an error..
- **Append text to the end of the file**: If checked, the wedge will append the data to the end of the file. Otherwise, it will replace the file's contents with the data.
- **Add a CR/LF to the end of the data**: If checked, the wedge will append a Carriage Return / Line Feed combination to the end of the data before it is sent to the file. Otherwise, the data is sent as-is.
- **Text**: This text box contains the format of the data that will be sent to the file once the wedge is triggered. For a complete explanation of how the data must be formatted, please see the section on Expressions.

7.1.3.2.7 MQTT Client Plug-in / Wedge

[This section applies only to the OmniServer Professional Edition.]

The server's **MQTT Client Plug-in** is designed to publish data from the server to a MQTT Broker. The **MQTT Client Plug-in** is activated by some change in a data value (called a **Trigger**) which causes configured data to be published to a pre-defined MQTT Broker.

How it works:

Once the server's runtime engine is activated, the server will search through the listing of wedges and perform the following operations:

- The wedge is checked to see if it is enabled. If the wedge is disabled, then the processes that follow are not executed, and the server continues to the next wedge
- The **Trigger** information is then checked to find out if the wedge is executed. If the trigger conditions are not satisfied, the wedge is not processed further and the server continues to the next wedge.
- The wedge will then look at the **Reset** section to see if any server item's value will be reset to a specific value.
- The **Event** section is then used to determine: 1) The **MQTT Broker** to which the data will be published; 2) The **Topic Name** to publish data under; 3) The **Format** of broker expected payload; 4) The **Message Body** which includes the items that will be published to the broker.
- Finally, if there is an item in in the **Notification Item** box in the **General** section, the server will set the value of that item to Logical One.

How to use the File Wedge

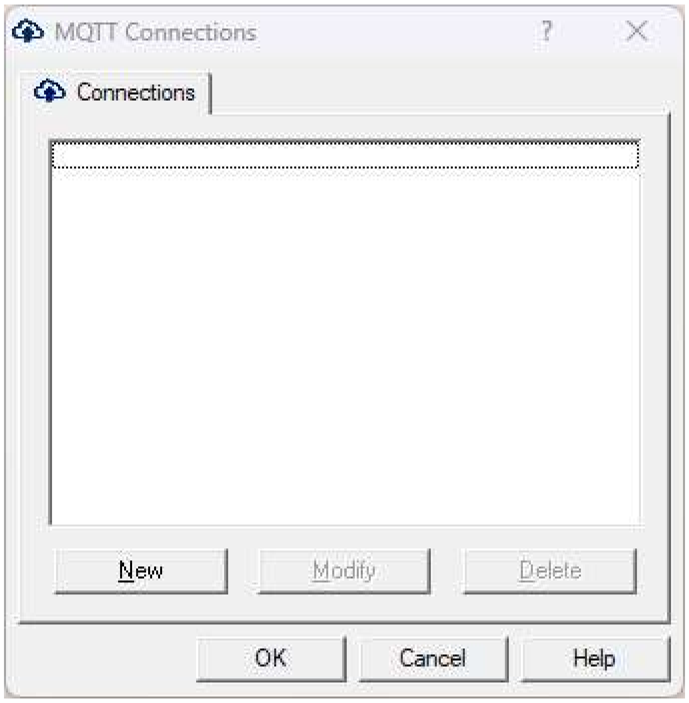
You can find the file wedge in the **Clients** section of the server's Configuration program.

- Creating, Modifying or Deleting a Wedge
- The General Tab
- The Trigger Tab
- The Reset Tab
- The Events Tab

7.1.3.2.7.1 Creating an MQTT Client connection

[This section applies only to the OmniServer Professional Edition.]

The first screen you see allows one to create, modify, or delete a wedge.



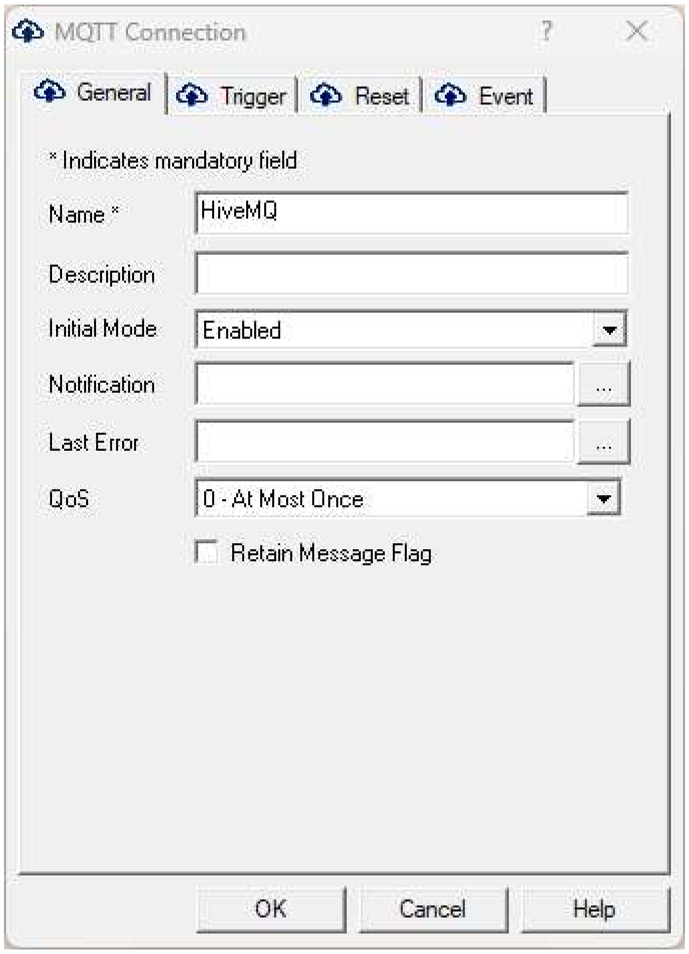
The selections are:

- **New**: Creates a new File Wedge. You will then be immediately taken the the **General** tab, where you can name your wedge.
- **Modify**: Opens up the selected wedge and allows you to begin changing it's properties. This option is not available if there is no wedge selected.
- **Delete**: Deletes the currently selected wedge. This option is not available if there is no wedge selected.

7.1.3.2.7.2 The General Tab

[This section applies only to the OmniServer Professional Edition.]

The General Tab is where you define the basic properties of the wedge.



* Indicates mandatory field

Name *	HiveMQ
Description	
Initial Mode	Enabled
Notification	
Last Error	
QoS	0 - At Most Once

☐ Retain Message Flag

NOTE: For all tabs, any field name followed by an asterisk (*) is mandatory for configuration.

The options available are:

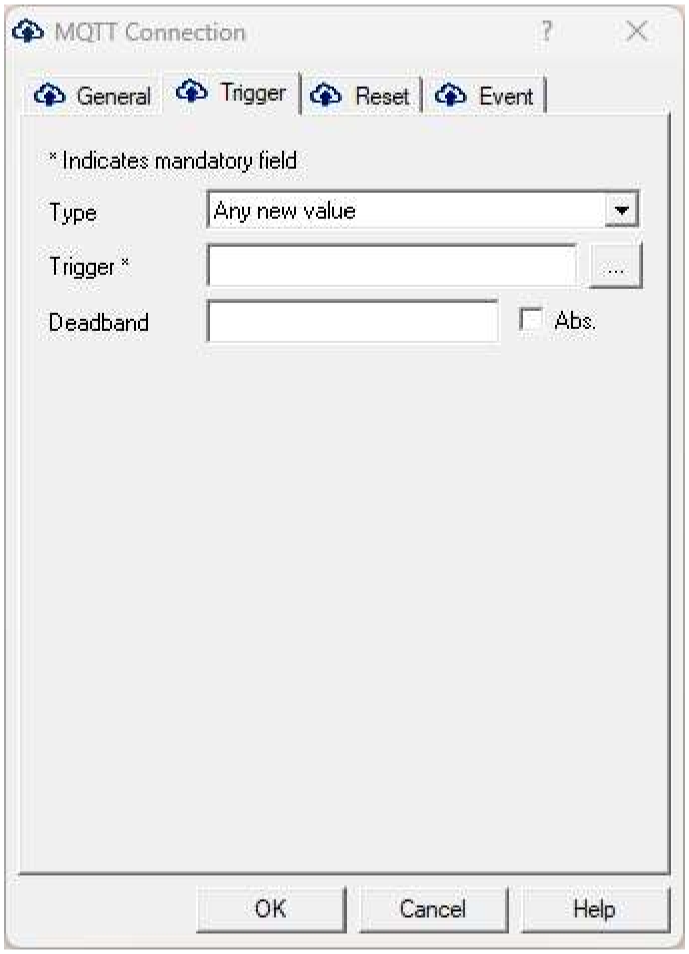
- **Name**: The name of the wedge.
- **Description**: A one-line description of the wedge. This is for documentation purposes only and has no effect on the operation of the wedge.

- **Initial Mode**: How the wedge starts up. There are two options available: **Enabled** activates the wedge upon startup of the server; **Disabled** disables all wedge activity for this wedge. The default is **Enabled**.
- **Notification**: The item used to notify the client that a wedge action has completed. To select an item, click on the **Browse** button to the right of the text box. If there is an item listed, the value of this item will be set to Logical One upon the successful completion of any action by this wedge (see the **Trigger** tab for a list of actions available).
- **Last Error**: The item used to notify the client of any errors encountered by an action of the wedge. To select an item, click on the **Browse** button to the right of the text box. If there is an item listed, any error encountered by the wedge will be recorded to this item.
- **QoS (Quality of Service):** MQTT supports 3 levels of quality of service
    - **At Most Once (0):** Every message will be delivered on a best-effort basis, similar to UDP. If the message is lost in transit for whatever reason, it is abandoned, the receive never receives it, and the sender does not know that it was lost. **This is the default.**
    - **At Least Once (1):** Every message will be delivered to a receiver, though sometimes the same message will be delivered two or more times. The receiver may be able to distinguish the duplicates, but perhaps not. The sender is not aware that the receiver received multiple copies of the message.
    - **Exactly Once (2):** Every message will be delivered exactly once to the receiver, and the send will be aware that it was received. Some MQTT brokers and services, such as Azure IoT Hub, Google IoT, and AWS IoT Core do not support this quality of service, and will simply disconnect when you attempt to send a topic update.
- **Retain Message Flag:** By default, an MQTT message that has no subscribers is simply discarded by the broker. This option tells the broker to keep the last message on this topic even if there are no subscribers.

7.1.3.2.7.3 The Trigger Tab

[This section applies only to the OmniServer Professional Edition.]

The **Trigger** tab is used to specify how the server is to perform a wedge action.



* Indicates mandatory field

Type: Any new value

Trigger *

Deadband

Abs.

<u>NOTE: For all tabs, any field name followed by an asterisk (*) is mandatory for configuration.</u>

- **Type**: The type of trigger for this wedge. The type specifies the condition on which the trigger activates. The available types are:

⊟ Any New Value

Trigger: Any New Value

The options available are:

Type: Causes the wedge to be triggered upon the receipt of any new value for the trigger. Not that this is not any value - if two like values are received consecutively, then only the first value will trigger the wedge.

Trigger: The item the server polls for the trigger.

Deadband: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is readied again to be triggered.

Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

The default for this field is **<blank>**, meaning that there is no deadband value and any new value will trigger the wedge.

Abs.: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

⊟ Contains the Given String

Trigger: Contains Given String

The options available are:

Type: Causes the wedge to be triggered when the trigger contains a given string.

Trigger: The item the server polls for the trigger.

Contains: The string that the trigger is to be checked against. The default for this field is **<blank>**, meaning that there is no string to check against. If you use this option, it's a good idea to place some type of data in this field.

⊟ Deviates from the given value

Trigger: Deviates From the Given Value

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger deviates from a given value.
- **Trigger**: The item the server polls for the trigger.
- **Target**: The target value for the trigger. This is the base value from which the server computes the deviation.
- **Deviation**: The deviation specifies a percentage above and below the **Target** that the value of the trigger is compared to in order to determine if the wedge is to be triggered. For example, if the target was "200", and the Deviation is set to "5" (meaning 5%), then for the wedge to trigger, the next value of the trigger must be less than "190" or greater than "210". The default for

this field is **<blank>**, meaning that there is no deviation value and any new value will trigger the wedge.

- **Deviation Abs.**: The ABS flag tells the server that the number in the **Deviation** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deviation value is treated as the number "5", and not 5%. That means the value of the trigger must be less than "195" or greater than "205" to trigger the wedge (since the target is set for "200").
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is readied again to be triggered.

  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

  Once the wedge is re-activated for triggering, the deviation comes back into effect.

  The default for this field is **<blank>**, meaning that there is no deadband value and any new value will be compared against the target and the deviation as described above.
- **Deadband Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not

5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

⊟ High Limit

Trigger: High Limit

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger moves above the given value.
- **Trigger**: The item the server polls for the trigger.
- **High**: The value on which the server compares the trigger value. If the value of the trigger moves above the value given in this box, the wedge will be triggered.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below before the wedge is readied again to be triggered.

  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a value in which the value of any subsequent triggers must fall below in order to clear the wedge for another trigger. Any values falling above this value will not cause the wedge to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190". Any values above and equal to "190" will not cause a trigger.

  The default for this field is **<blank>**, meaning that there is no deadband value, and any new value must fall below the target value before the wedge will be readied for triggering again.

- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

⊟ Low Limit

Trigger: Low Limit

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger moves below the given value.
- **Trigger**: The item the server polls for the trigger.
- **Low**: The value on which the server compares the trigger value. If the value of the trigger moves below the value given in this box, the wedge will be triggered.
- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must move above before the wedge is readied again to be triggered.

  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a value in which the value of any subsequent triggers must move above in order to clear the wedge for another trigger. Any values falling below this value will not cause the wedge to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must move above "210". Any values below and equal to "210" will not cause a trigger.

The default for this field is **<blank>**, meaning that there is no deadband value, and any new value must be above the target value before the wedge will be readied for triggering again.

- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

⊟ Matches the Given String

Trigger: Matches the Given String

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger matches the string specified in the **Matches** box. This string is case sensitive and must match exactly.
- **Trigger**: The item the server polls for the trigger.
- **Matches**: The value on which the server compares the trigger value. If the value of the trigger exactly matches this value, then the wedge will be triggered.

⊟ Non-zero Value or non-empty string

Trigger: Non-zero Value or Non-empty String

The options available are:

- **Type**: Causes the wedge to be triggered when the value of the trigger contains a non-zero value or a non-empty string.

- **Trigger**: The item the server polls for the trigger.

- **Deadband**: After the initial trigger, the deadband specifies a percentage that an incoming value of the trigger must drop below or rise above before the wedge is readied again to be triggered.

  Normally, a wedge is "cleared" and is made available for triggering once the current trigger is processed. The "Deadband" modifies this by specifying a range in which the value of any subsequent triggers must fall outside of in order to clear the wedge for another trigger. Any values falling within this range will not cause the wedge to be triggered.

  For example, if the trigger was "200", and the deadband is set to "5" (meaning 5%), then for the wedge to be triggered again, a value of the trigger must fall below "190" or above "210". Any values within this range will not cause a trigger.

  Please note that the deadband will not work properly with non-numeric strings.

  The default for this field is **<blank>**, meaning that there is no deadband value and any new value will trigger the wedge.

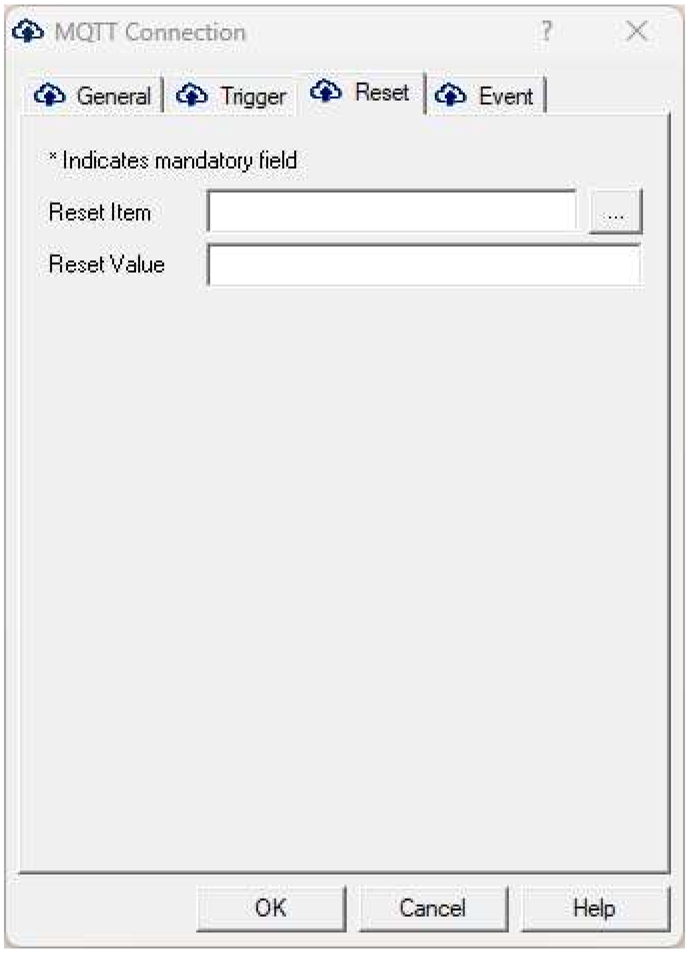
- **Abs.**: The ABS flag tells the server that the number in the **Deadband** text box is to be interpreted as an absolute value and not a percentage. Using the example above, if the ABS flag is set, then the Deadband value is

treated as the number "5", and not 5%. That means the next value of the trigger must be less than "195" or greater than "205" to trigger the wedge.

7.1.3.2.7.4 The Reset Tab

[This section applies only to the OmniServer Professional Edition.]

The Reset Tab is where you tell the wedge what item to reset (and to what value) just before the action of the wedge is executed. One use for the Reset value is to reset the value of a trigger once the wedge has completed its action, thereby forcing the wedge to record all occurrences of the trigger and not just new values.



MQTT Wedge - The Reset Tab

NOTE: For all tabs, any field name followed by an asterisk (*) is mandatory for configuration.

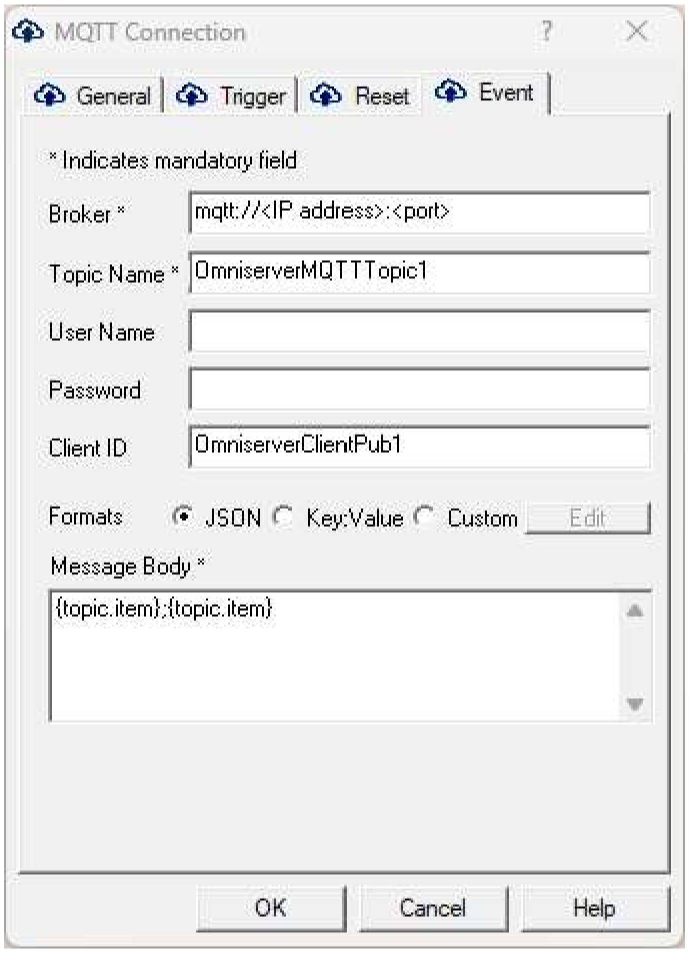
The options available are:

- **Reset Item**: The item that the **Reset Value** will be applied just before the wedge has completed its action. Use the **Browse** button to select your item.
- **Reset Value**: The value that will be inserted into the **Reset Item**.

7.1.3.2.7.5 The Event Tab

[This section applies only to the OmniServer Professional Edition.]

The Events Tab defines the Broker, Topic Name, Authentication Parameters, Client ID, Required Format and Message Body for publishing.



* Indicates mandatory field

Broker *	mqtt://<IP address>:<port>
Topic Name *	OmniserverMQTTTopic1
User Name	
Password	
Client ID	OmniserverClientPub1
Formats	[✓ JSON] Key:Value Custom
Message Body *	{topic.item};{topic.item}

MQTT Wedge - The Event Tab

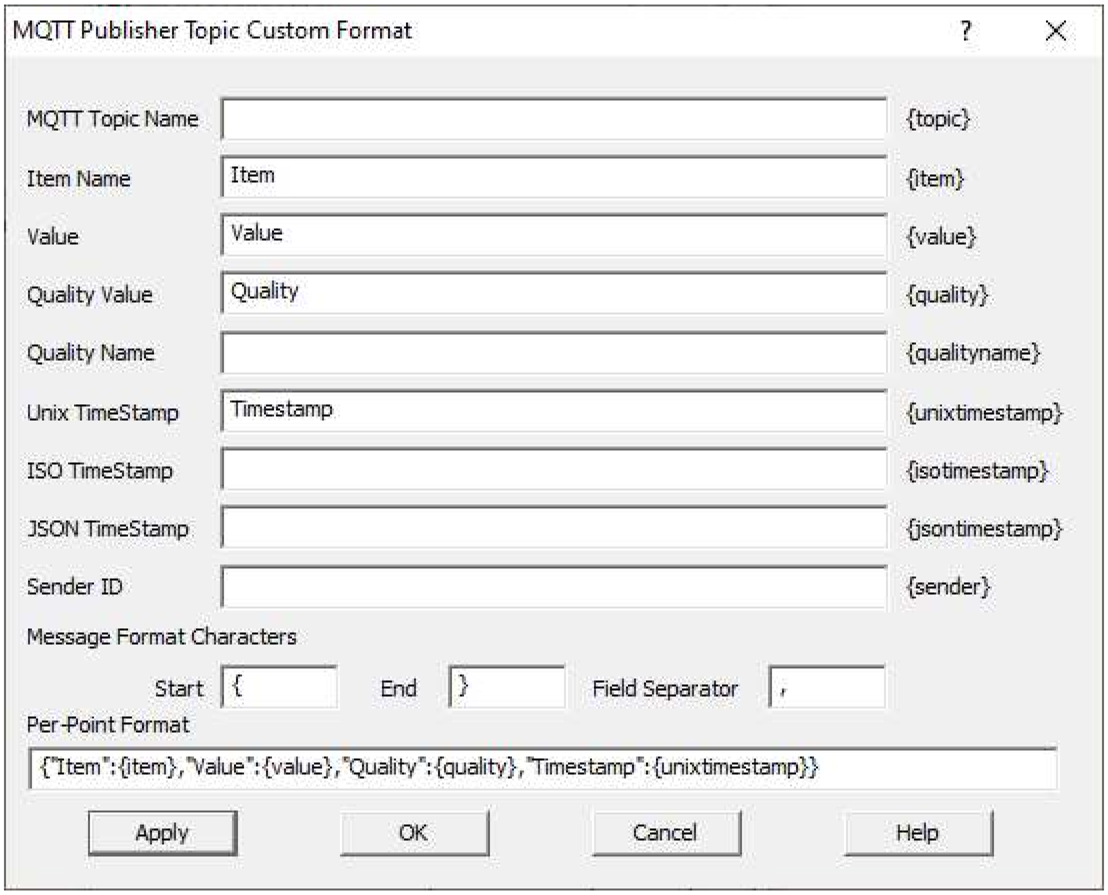
NOTE: For all tabs, any field name followed by an asterisk (*) is mandatory for configuration.

The options available are:

- **Broker:** MQTT Broker address - The address typically includes the protocol, hostname or IP address, and port number. Please refer to your MQTT Broker for this information.
- **Topic Name:** MQTT Topic for connection.
- **User Name:** valid user name configured within the MQTT broker that allows OmniServer to be recognized as an authorized client.
- **Password:** corresponding password configured within the MQTT broker for user name above.
- **Client ID:** user-configurable Client IDs allow you to assign meaningful identifiers to each MQTT client connection.
- **Formats:** payload format types to structure and transmit data
    - **JSON:** consists of key-value pairs enclosed within curly braces {}. Each key represents an identifier or attribute, and its associated value.
    - **Key Value:** simplifies data exchange with straightforward pairs of identifiers and values, ideal for straightforward, compact data transmissions.
    - **Custom:** user-configurable and allows flexibility by accommodating specific application needs, enabling tailored data structures and protocols. Help for configuring a custom format is located here.

[This section applies only to the OmniServer Professional Edition.]

The Custom Format Editor will allow you to create a custom payload to match the format of the broker you are connecting to. An identifier in each field can be entered to include and signify the listed value in the payload. Once those have been configured to the users preference and applied, the Per-Point Format will show the payload that will be sent to the MQTT broker for each point included in the message body on the Event Tab.


MQTT Publisher Topic Custom Format

MQTT Topic Name		{topic}
Item Name	Item	{item}
Value	Value	{value}
Quality Value	Quality	{quality}
Quality Name		{qualityname}
Unix TimeStamp	Timestamp	{unixtimestamp}
ISO TimeStamp		{isotimestamp}
JSON TimeStamp		{jsontimestamp}
Sender ID		{sender}

Message Format Characters

Start: { End: } Field Separator: ,

Per-Point Format

{"Item":{item},"Value":{value},"Quality":{quality},"Timestamp":{unixtimestamp}}

The JSON Path fields configured above matches the current entry seen configured under **Per-Point Format**. To make changes to this format, edit the message member names available above and Message Format Characters to match what is expected by the Broker.

1. Enter an identifying name for each required message member name to match the Broker specific format.

- **MQTT Topic Name**
- **Item Value**
- **Value**
- **Quality Value** (i.e. 192 = Good, 200 = Bad)
- **Quality Name** (i.e. Good, Bad, Unknown)
- **TimeStamp** (Unix, ISO, JSON)

- **Sender ID**

2. Enter required **Message Format Characters** to match Broker specific format requirement.

3. Once complete, click **"Apply".** The **Per-Point Format** will automatically populate.

7.1.4 Topics

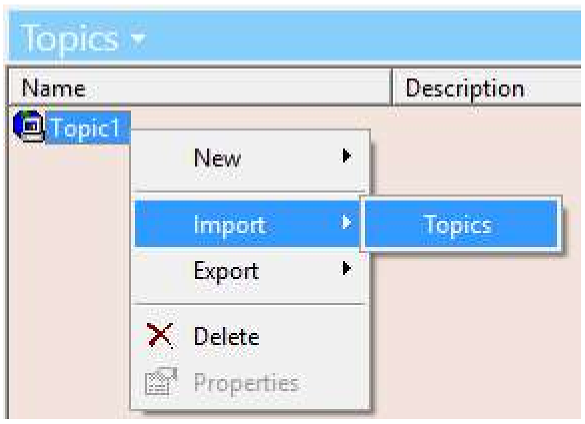
A Server Topic links a server protocol with a server device. Since protocols are independent of devices, the topic is needed to tell the server which protocol belongs to which device. Then, when the client requests data from the server, the client will specify the application, topic, and item from the server.

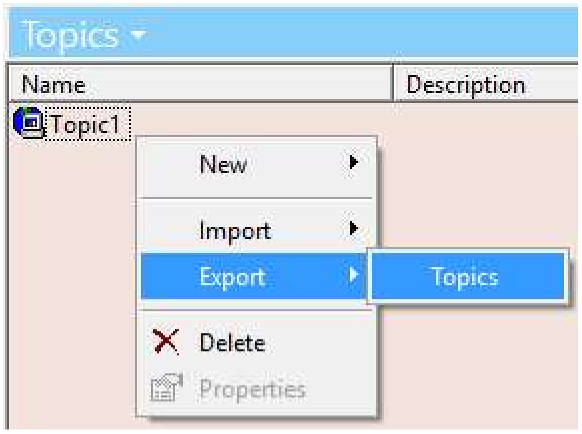
While you can have multiple topics assigned to one device, you can only assign one device per topic.

Server Topic Settings View

CSV Import / Export of Topics

Topics can be imported or exported for ease in merging configurations or for mass edits. You can access the import or export options from the File menu while in the Topics view, by right-clicking in the white space of the Topics view or by clicking the Import button or Export button while in the Topics view.

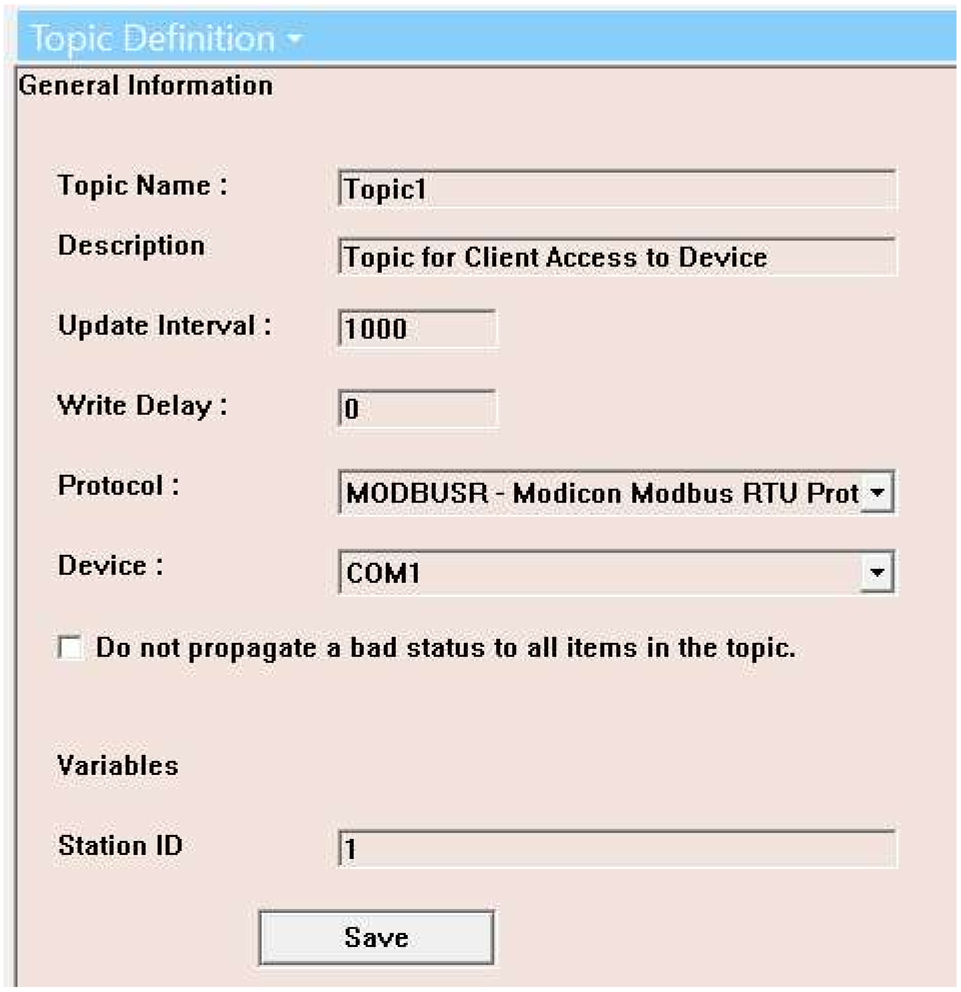




To ensure you have a properly formatted CSV file, it is recommended to configure one topic in OmniServer and then perform a CSV export of that topic.

7.1.4.1 Settings

The Topic Definition view is used to set up topics or edit existing topics. A Topic is used by the server to associate a protocol with a device. The client program, in turn, uses the topic name to help the server determine the location of the requested item within the protocol.



General Information

Topic Name :	Topic1
Description	Topic for Client Access to Device
Update Interval :	1000
Write Delay :	0
Protocol :	MODBUSR - Modicon Modbus RTU Prot
Device :	COM1

☐ Do not propagate a bad status to all items in the topic.

Variables

Station ID	1

The Topic Definition view

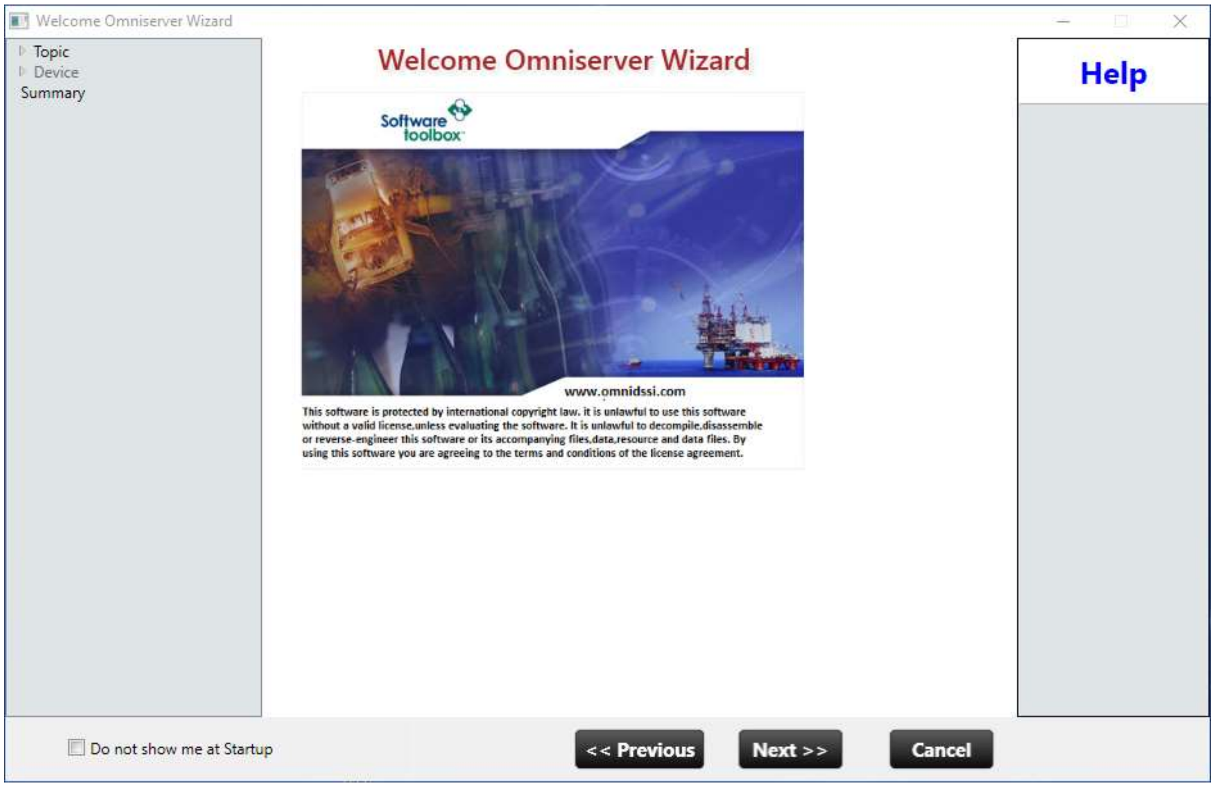
The different options are:

- Topic Name: The name of the topic. This name must be unique and is used by your client application to request communication with a specific device.
- Description: The description of the topic. This is used for documentation purposes only and has no other function within the server.
- Update Interval: The number of milliseconds the server waits in between reading a particular item. The server will attempt to read in all the items required to be read in within the time specified in the Update Interval. However, please be aware that external circumstances affect the server's performance, so no guarantees can be made for accuracy.

- Write Delay: The number of milliseconds the server will wait between each operation that writes data to the device.
- Protocol: The protocol used for communication by this topic.
- Device: The specific server device (Serial, Ethernet, LPT) used for communication by this topic.
- Do not propagate a bad status to all items in the topic: By default, once the Status goes bad, it will propagate that status to all the items in the topic. Checking this box will force the server not to do that, meaning that while the topic status is bad, the items themselves will retain both their last known value and status.
- Variables: If your selected protocol contains Topic Variables, they will be listed here and you can enter the desired values.

7.1.4.2 Topic Wizard

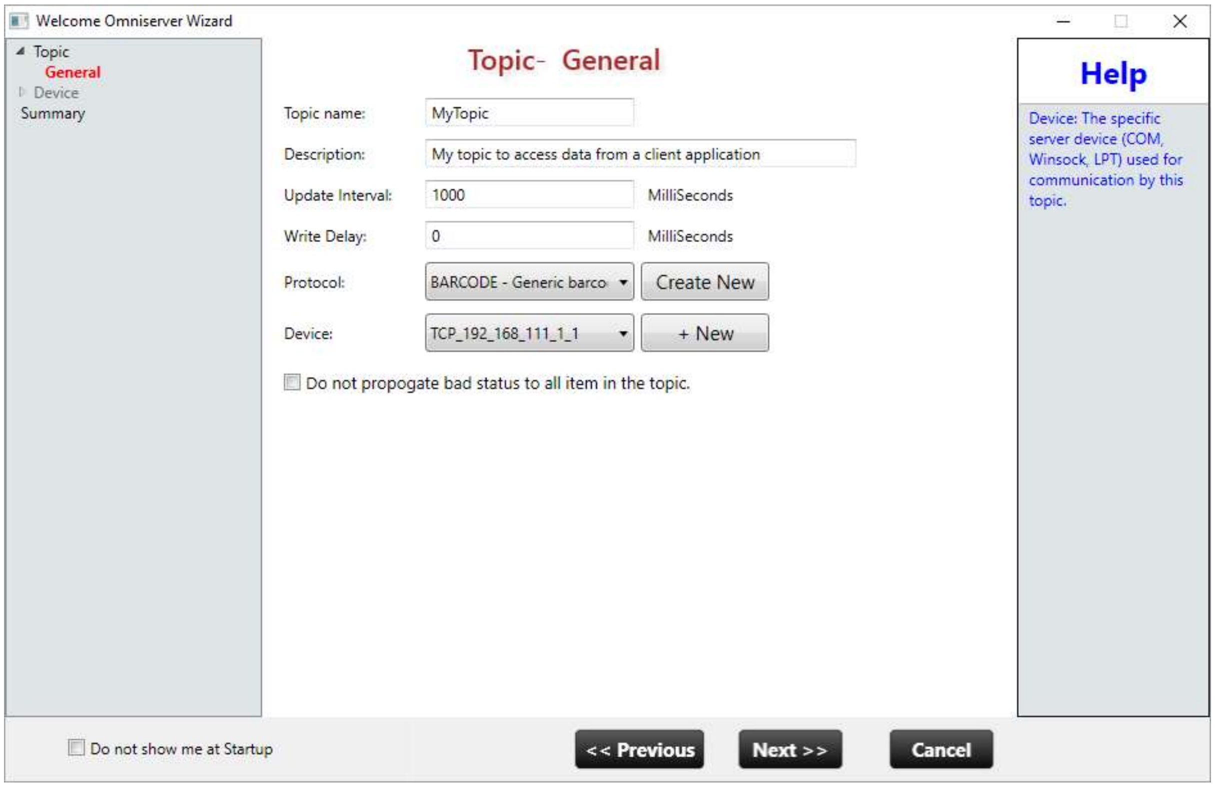
The **Topic Configuration Wizard** opens by default when the OmniServer Configuration is launched. It walks you through the process of configuring an OmniServer topic step-by-step including either selecting a new or existing protocol and new or existing device.



This software is protected by international copyright law. it is unlawful to use this software without a valid license,unless evaluating the software. It is unlawful to decompile,disassemble or reverse-engineer this software or its accompanying files,data,resource and data files. By using this software you are agreeing to the terms and conditions of the license agreement.

Selecting the "Do not show me at Startup" option will prevent the wizard from launching automatically when starting the OmniServer Configuration.

Simply click the **Next** button to get started.



Topic name: MyTopic

Description: My topic to access data from a client application

Update Interval: 1000 MilliSeconds

Write Delay: 0 MilliSeconds

Protocol: BARCODE - Generic barco

Device: TCP_192_168_111_1_1

Do not propogate bad status to all item in the topic.

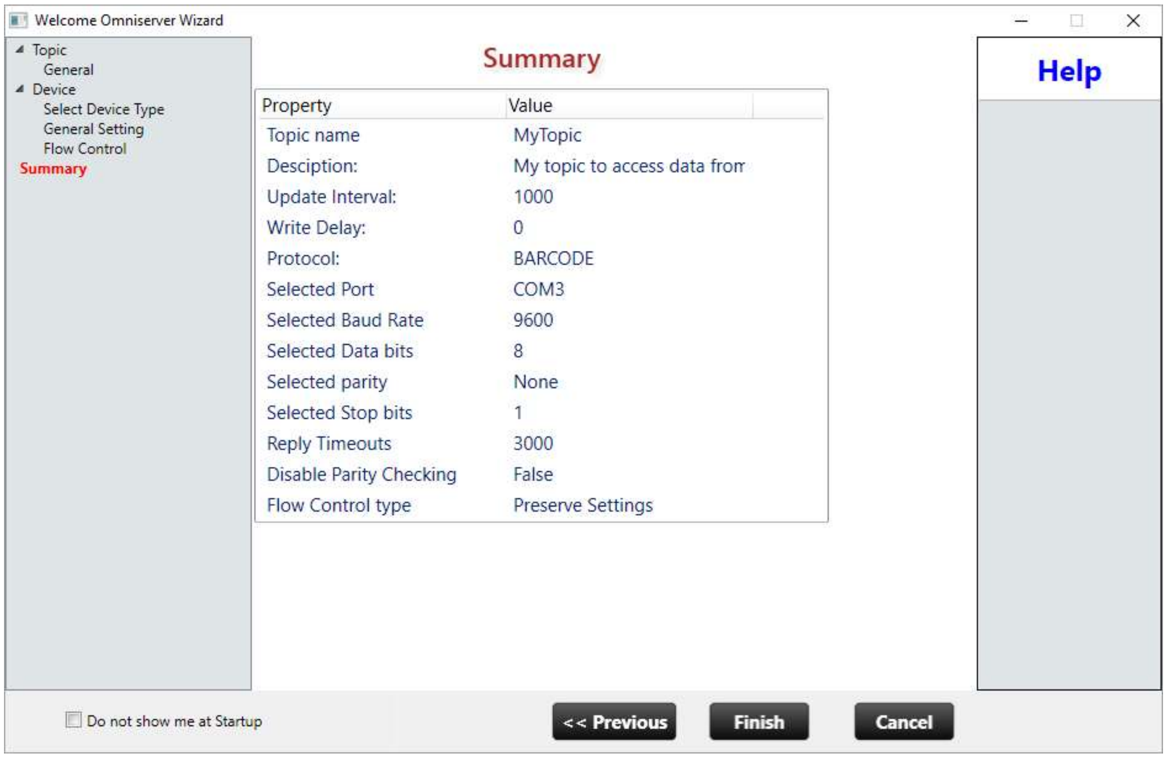
Device: The specific server device (COM, Winsock, LPT) used for communication by this topic.

The different options are:

- Topic Name: The name of the topic. This name must be unique and is used by your client application to request communication with a specific device.
- Description: The description of the topic. This is used for documentation purposes only and has no other function within the server.
- Update Interval: The number of milliseconds the server waits in between reading a particular item. The server will attempt to read in all the items required to be read in within the time specified in the Update Interval. However, please be aware that external circumstances affect the server's performance, so no guarantees can be made for accuracy.
- Write Delay: The number of milliseconds the server will wait between each operation that writes data to the device.

- Protocol: The protocol used for communication by this topic. Either select an existing protocol or click the **Create New** button to define the name for your new protocol (you will configure the new protocol later in the protocol editor).
- Device: The specific server device (Serial, Ethernet, LPT) used for communication by this topic. Either select an existing device or click the +New button to configure a new device. Click here for details on Device types and settings.
- Do not propagate a bad status to all items in the topic: By default, once the Status goes bad, it will propagate that status to all the items in the topic. Checking this box will force the server not to do that, meaning that while the topic status is bad, the items themselves will retain both their last known value and status.
- Topic Variable: If your selected protocol contains Topic Variables, they will be listed here and you can enter the desired values.

When finished, click **Next** to get the Topic Summary.



Summary

Property	Value
Topic name	MyTopic
Desciption:	My topic to access data from
Update Interval:	1000
Write Delay:	0
Protocol:	BARCODE
Selected Port	COM3
Selected Baud Rate	9600
Selected Data bits	8
Selected parity	None
Selected Stop bits	1
Reply Timeouts	3000
Disable Parity Checking	False
Flow Control type	Preserve Settings

Just click the **Finish** button to complete configuration of your new topic.

7.1.5 Using Expressions

An expression is any sequence of items, constants, and/or operators that, when evaluated, produces a single result.

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

Please note: Expressions can only be used in conjunction with the Wedge components of the server. Expressions cannot be used within Protocol Messages.

Elements of an Expression

There are three main parts, or elements, of an expression:

- Constants: Constants are simple numeric or string expressions, like **12** or **"Hello World"**. They are called "Constants" because their value (in this case, **12** and **"Hello World"**) never changes. Please note that string constants must be enclosed by quotes.
- Items: Items are made available by the server. An item is expressed using its **topic.item** specification. For example, to use the item **Data** in the topic **Meklar**, use **Meklar.Data**.

  Please note: The exception to this rule is when using expressions in Protocol message building. In this case, only the item name is used.
- Operators: An operator performs functions on items and/or constants. Arithmetic, Comparison, and Logical operators are included in this group. For example, the operation **Meklar.Data+12** adds the item **Meklar.Data** to the constant **12**. The plus sign (+), meaning addition, is the operator in this expression.

Specifying Expressions.

The goal of an expression is to evaluate and interpret the data used within an expression, perform any operations, then return a single value. This can be done in one of three ways:

- Constant: Specify a simple constant (like above) to always return the constant.

- Item: Specify an item (as explained above) to always return the value of that item.
- Complex Expression: Specify a legal combination of constants, items, and operators to where the calculations performed will return a single value.

Since expressing a constant or an item is as simple as typing in that constant or item as above, the rest of this page will concentrate on the construction of complex expressions.

Complex Expressions

An expression is formatted in the same way that an equation is written by hand. For example, suppose that the item **Meklar.Data** returns the number of seconds since Midnight.

In the real world, If you wanted to find out the number of minutes past midnight, you would write and equation like this:

Number of Minutes = Meklar.Data / 60

If you wanted to write an expression instead, you would use the format to the right of the equality sign, like this:

Meklar.Data / 60

Once the expression is parsed, the value returned will be the number of minutes past midnight.

Again, for another example, suppose three values, **topic.hours**, **topic.minutes**, and **topic.seconds**, are needed to be returned in the format **h:m:s**. To to this, this will be the expression:

topic.hours & ":" & topic.minutes & ":" & topic.seconds

In this case, the three items are "concatenated" together with the colon (":") sign using the concatenation operator (&).

Finally, suppose you need to return an average based upon an item (called **topic.er**) that is multiplied by a constant (**9**), and that product is divided by another item (called **topic.ip**). The expression will be:

(topic.er * 9) / topic.ip

Notice the use of parenthesis in the above example. All expressions within parenthesis will be evaluated before the rest of the expression.

Operators

Operators are used to perform "operations" on expressions. Addition (+), Subtraction (-), Multiplication, (*), etc., are all operators.

A complete discussion of operators and there precedence can be found here.

7.1.5.1 Operators

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

As their name implies, **operators** perform functions on items, constants, or expressions to produce a computed result.

There are three types of operators:

- **Arithmetic**. These operators perform arithmetic operations upon the items, constants, and expressions. Included in this group: **Negation (-)**; **Exponentiation (^)**; **Multiplication and Division (*,/)**; **Integer Division (\)**; **Modulus Arithmetic (Mod)**; **Addition and Subtraction (+,-)**; and **String Concatenation (&)**.

  For a detail explanation of arithmetic operators, please see this page.

- **Comparison**. These operators compare the values of two items, constants, or expressions, and returns a value of **True** or **False** based upon the comparison. Included in this group: **Equality (=)**; **Inequality(<>)**; **Less Than (<)**; **Greater Than (>)**; **Less Than or Equal To (<=)**; and **Greater Than or Equal To (>=)**.

  For a detail explanation of arithmetic operators, please see this page.

- **Logical**. These operators perform "truth-table" calculations on two items, constants, or expressions, and returns a value of **True** or **False** based upon the value of those items, constants, or expressions. Included in this group: **Not**; **And**; **Or**; **Xor**; **Eqv**; and **Imp**.

  For a detail explanation of arithmetic operators, please see this page.

Operator Precedence.

In any expression, the operators are evaluated in a predetermined order. This order is called **operator precedence.** This means that, for example, if the expression **2 + 3 * 6** appears, the multiplication will take place first, then the addition (see below for a table). This example gives a value of **20**.

However, you can override the operator precedence by using **parenthesis "()"**. Any expression inside parenthesis is evaluated before the result of the expression. So, for example, if the expression **(2 + 3) * 6** appears, the parenthesis is evaluated first (in this case, the addition of two and three) before the multiplication occurs. The final value in this case is **30**.

If arithmetic, comparison, and logical expressions appear in the same expression, arithmetic operators are evaluated first, then comparison operators, and finally logical operators.

Operator Preference

Arithmetic	**Comparison**	**Logical**
Negation (-)	Equality (=)	Not
Exponentiation (^)	Inequality (<>)	And
Multiplication and Division (*,/)	Less than (<)	Or
Integer Division (\)	Greater than (>)	Xor
Modulus Arithmetic (Mod)	Less than or Equal to (<=)	Eqv
Addition and Subtraction (+,-)	Greater than or Equal to (>=)	Imp
String Concatenation (&)		

7.1.5.1.1 Arithmetic Operators

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

Nine types of arithmetic operators are supported. The order they are evaluated is the order they are listed below.

Operators **Negation**, **Exponentiation**, **Multiplication**, **Division**, **Integer Division**, **Modulus Arithmetic**, and **Subtraction** must have numeric expressions. Any **string** or **No Value** will be converted to it's numeric equivalent. It that cannot be done, an error will be generated.

Operator **Addition** will take both numeric and string data and will execute different operations based upon the types of expressions given. A **No Value** will be interpreted as a zero (0) if the accompanying expression is numeric, and an empty string ("") if the accompanying expression is a string.

Operation **Concatenation** will interpret all expressions as **strings**. A **No Value** will be interpreted as an empty string ("").

For Boolean expressions, a **True** value will be treated as the number **1**, while a value of **False** will be treated as the number **0**.

Any overflows, underflows, and division by zero will be reported as an error.

The arithmetic operators are:

⊟ Negation (-)

Negation is a **Subtraction** sign at the beginning of an expression and is used to specify the negative value of that expression. For example:

-4
-topic.item

Although negation uses the same sign as subtraction (see **Subtraction**), it is the placement of the sign that signifies a negation operation or a subtraction operation.

⊟ Exponentiation (^)

Raises a number to the power of an exponent. For example:

12 ^ 2
topic.item ^ 3
topic.item ^ topic.data

⊟ Multiplication (*)

Multiplies two expressions. For example:

12 * 5
topic.item * 5
topic.item * topic.data

⊟ Division (/)

Divides two numbers and returns the result. For example:

12 / 4
topic.item / 5
topic.item / topic.data

If the divisor (**topic.data** in the above example) evaluates to a zero, an error will be returned.

⊟ Integer Division (\)

Integer Division will round each expression to the nearest integer, the division will be made, then the final expression will also be rounded to the nearest integer. For example:

12 \ 4
12.2 \ 6.1 (computed as **12 \ 6**)
topic.item \ 6
topic.item \ topic.data

If the divisor (**topic.data** in the above example) evaluates to a zero, an error will be returned.

⊟ Modulus Arithmetic (Mod)

Modulus Arithmetic returns the remainder of an integer division operation between two expressions. For example:

22 Mod 5 yields **2** (12 divided by 5 equals 4 with a remainder of 2).
topic.item Mod 5
topic.item Mod topic.data

All expressions are converted to integers before the modulus arithmetic takes place.

If the expression after the MOD operator (**topic.data** in the above example) evaluates to a zero, an error will be returned.

Addition (+)

Sums two numbers together or concatenates two strings. For example:

2 + 2
topic.item + 4
topic.item + topic.data + 6

You can also use the addition operator to concatenate two strings together. However, you should use the **Concatenation Operator (&)** to do this to eliminate confusion and self-document your expression.

How the addition is performed is based upon the type of data being added together. For example, if two numbers are involved, then those numbers are added together. If two strings are involved, they are concatenated. If a number and a string is used, the string will be interpreted as a number and the addition made. If the string cannot be converted, then an error is generated.

Subtraction (-)

Subtracts to expressions and returns the difference. For example:

12 - 4
topic.item - 4
topic.item - topic.data

If the number appears at the beginning of the expression, such as:

-4
-topic.item

...then the expression is "negated", and the negative value of the expression is returned (see **Negation (-)**).

String Concatenation (&)

Appends one string to another. All values are treated as strings, even if all they are explicitly declared to be a number. For example:

"Hello" & "World" yields **"HelloWorld"**
"Hello" & 12 yields **"Hello12"**
"ATDT" & topic.item yields **ATDT5551212**, if the value of **topic.item** equals **"5551212"**
12 & 34 yields **"1234"**

7.1.5.1.2 Comparison Operators

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

Six types of comparisons are supported. A comparison operator will test two expressions for a certain condition (given by the comparison sign) and will return either a "true" or "false" based upon that comparison. For example:

12 = 4 returns **False**
12 = 12 returns **True**

A **No Value** expression is treated as a zero when using numeric comparisons, and an empty string ("") when using string comparisons.

The order they are evaluated is the order they are listed below.

The comparison operators are:

⊟ Equality (=)

Tests two expressions to determine if both are the same (or equal).

Example: **topic.data = topic.item**

Returns **true** if both expressions are equal to each other (either numeric or string). Returns **false** if the two expressions do not compare to each other.

⊟ Inequality (<>)

Tests two expression to determine if they are different (or in-equal)

Example: **topic.data <> topic.item**

Returns **true** if both expressions are not equal to each other (either numeric or string). Returns **false** if the two expressions are equal to each other.

⊟ Less Than (<)

Tests to determine if the expression to the left of the sign is less-than the expression on the right of the sign.

Example: **topic.data < topic.item**

For numeric expressions, the expressions are compared based upon the value of the expressions. For string expressions, the comparison is made on a byte-by-byte basis using the ASCII value of each byte.

Returns **true** if the expression on the left is less-than the expression on the right. Otherwise, **false** is returned.

⊟ Greater Than (>)

Tests to determine if the expression to the left of the sign is greater-than the expression on the right of the sign.

Example: **topic.data > topic.item**

For numeric expressions, the expressions are compared based upon the value of the expressions. For string expressions, the comparison is made on a byte-by-byte basis using the ASCII value of each byte.

Returns **true** if the expression on the left is greater-than the expression on the right. Otherwise, **false** is returned.

⊟ Less Than or Equal To (<=)

Tests to determine if the expression to the left of the sign is less-than or equals the expression on the right of the sign.

Example: **topic.data <= topic.item**

For numeric expressions, the expressions are compared based upon the value of the expressions. For string expressions, the comparison is made on a byte-by-byte basis using the ASCII value of each byte.

Returns **true** if the expression on the left is less-than or equals the expression on the right. Otherwise, **false** is returned.

⊟ Greater Than or Equal To (>=)

Tests to determine if the expression to the left of the sign is greater-than or equals the expression on the right of the sign.

Example: **topic.data >= topic.item**

For numeric expressions, the expressions are compared based upon the value of the expressions. For string expressions, the comparison is made on a byte-by-byte basis using the ASCII value of each byte.

Returns **true** if the expression on the left is greater-than or equals the expression on the right. Otherwise, **false** is returned.

7.1.5.1.3 Logical Operators

[This section applies only to the OmniWedge and the OmniServer Professional Edition.]

Six types of logical operators are supported. A logical operator performs a "truth-table" comparison of two expressions. Each expression will evaluate to a **True** or **False** value, then the logical operator will perform it's operation to evaluate to a final true or false. For example:

1 AND 0 returns **False**
1 OR 0 returns **True**

If any expression has **No Value**, that expression is evaluated as zero.

How different data types are treated.

Numbers

For numeric expressions, logical operators perform bit-wise operations. What this means is that each operation is performed upon the internal binary representation of the number. Real numbers are converted to integers before the operation is done.

Strings

Strings are converted to their numeric representations. If they cannot be converted, an error will be generated.

Booleans (True/False)

If the operation contains two Boolean expressions (ex: **BOOL and BOOL**), then the normal operations is performed, resulting in a **True** or **False**. If a Boolean expression is being used with a numeric expression, the Boolean value is treated as either all bits set (**True**) or all bits not set (**False**), with the value returned as a numeric expression.. For example, **14 AND TRUE** results in **14**, while **14 AND FALSE** results in **0**.

Operator Precedence and Listing

The order they are evaluated is the order they are listed below.

The logical operators are:

⊟ NOT

Performs a logical negation of an expression. Unlike the other logical operators, the NOT appears with only one expression -

that expression being placed just after the NOT operator. The result is a negation of the current state of the expression.

Example: **NOT topic.item**

The truth-table for a NOT expression is as follows:

If the Expression is:	The Result is:
True	False
False	True

⊟ AND

Performs a logical conjunction between two expressions. The result will be **true** if both expressions are **true**, otherwise, the result will be **false**.

Example: **topic.item AND topic.data**

The truth-table for an AND expression is as follows:

If Expression 1 is:	And Expression 2 is:	The Result is:
True	True	True
True	False	False
False	True	False
False	False	False

⊟ OR

Performs a logical disjunction on two expressions. The result is **true** unless both expressions are **false**.

Example: **topic.item OR topic.data**

The truth-table for an OR expression is as follows:

If Expression 1 is:	And Expression 2 is:	The Result is:
True	True	True

True	False	True
False	True	True
False	False	False

XOR (Exclusive-OR)

Performs a logical exclusion on two expressions. The result is **true** if either expression is **true**, but **false** if both expressions are **true** or **false**.

Example: **topic.item XOR topic.data**

The truth-table for an XOR expression is as follows:

If Expression 1 is:	And Expression 2 is:	The Result is:
True	True	False
True	False	True
False	True	True
False	False	False

EQV

Performs a logical equivalence on two expressions. If both expressions are equal to each other, then the result is **true**.

Example: **topic.item EQV topic.data**

The truth-table for an EQV expression is as follows:

If Expression 1 is:	And Expression 2 is:	The Result is:
True	True	True
True	False	False
False	True	False
False	False	True

IMP

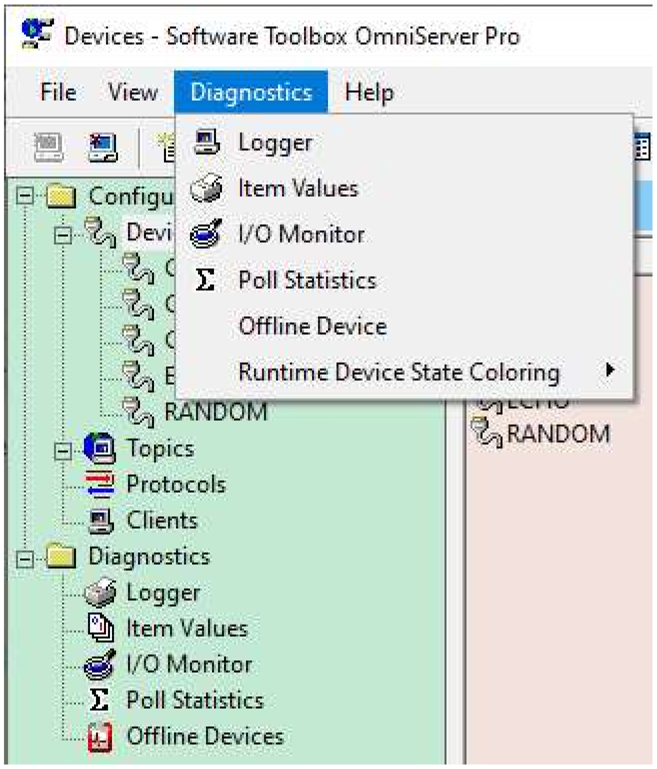
Performs a logical implication on two expressions. The result is **true** unless the first expression is **true** and the second expression is **false**.

Example: **topic.item IMP topic.data**

The truth-table for an IMP expression is as follows:

If Expression 1 is:	And Expression 2 is:	The Result is:
True	True	True
True	False	False
False	True	True
False	False	True

7.2 Diagnostic Tools



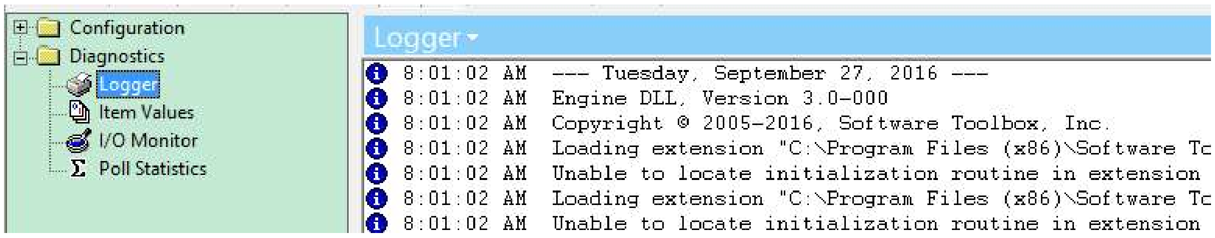
See also online - Exploring OmniServer Diagnostic Tools

To help you debug your protocol and/or troubleshooting any issues, the following tools are available:

- I/O Monitor: This diagnostic window gives you a window on the all the data flowing through the server.
- Item Values: This window show the current value of all the items being requested by the client or items automatically activated by the protocol.
- Local Logger: This window contains all the error and event information for the server. Additionally, it can show all the data going into and coming out of the server.
- Poll Statistics: This diagnostic window gives you statistics for each active server device and server topic.
- Offline Device: This diagnostic window displays any devices that currently have errors associated with them.
- Runtime Device State Coloring: When enabled, this tool provides a visual green or red indication in the device list of any devices that are currently experiencing errors.

7.2.1 The Local Logger

The Local Logger is a text-based audit tool that displays internal server information.



```
8:01:02 AM  --- Tuesday, September 27, 2016 ---
8:01:02 AM  Engine DLL, Version 3.0-000
8:01:02 AM  Copyright © 2005-2016, Software Toolbox, Inc.
8:01:02 AM  Loading extension "C:\Program Files (x86)\Software Tc
8:01:02 AM  Unable to locate initialization routine in extension
8:01:02 AM  Loading extension "C:\Program Files (x86)\Software Tc
8:01:02 AM  Unable to locate initialization routine in extension
```

The Local Logger Window

There are three sections to the Local Logger:

- **Line Type:** Displays a graphic informing the user what type of information is contained on the line.

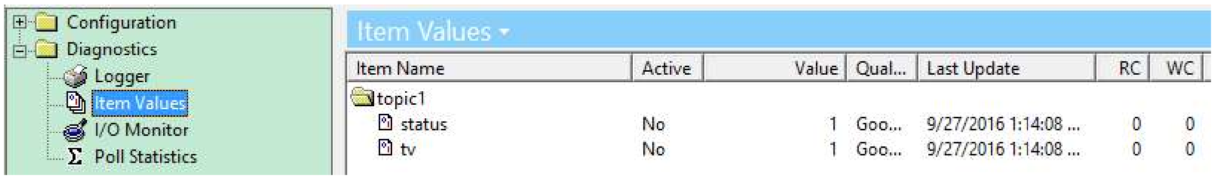
List of Line Types

- **Error:** The line contains an Error message. This type is only available when the **View Errors** flag is set.
- **Event:** The line contains a Server Event. This type is only available when the **View Events** flag is set.
- **Failure:** The message from the device matched what was expected by the server, but failed the Error Detection Code check.
- **Ignore:** The message from the device did not match what was expected by the server.
- **Informational:** Server messages for information only.
- **Receive:** The incoming message matched what was expected by the server.
- **Send:** Data sent to the device by the server.

- **Date/Time Stamp:** Date and Time stamp for the line.
- **Information:** The text description of the event.

7.2.2 The Item Values Window

The Item Values Window displays the values of the items being requested by the client or items that have been automatically activated by the protocol.



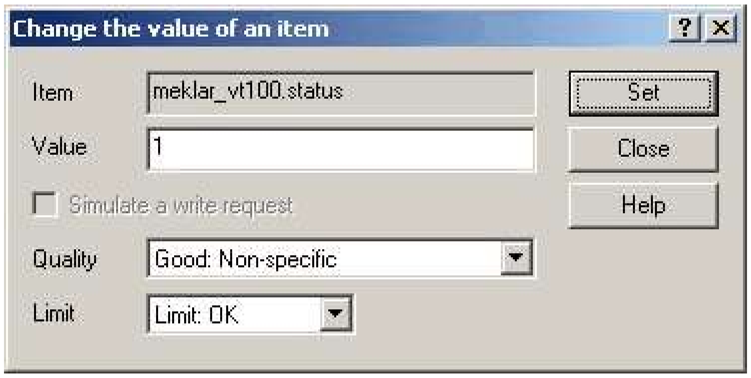
Item Name	Active	Value	Qual...	Last Update	RC	WC
topic1						
status	No	1	Goo...	9/27/2016 1:14:08 ...	0	0
tv	No	1	Goo...	9/27/2016 1:14:08 ...	0	0

The Item Values Window

The items are group by the server topic for which they are associated. The descriptions of each of the columns in the listing are:

- Item Name: The name of the item. If you double-click on this item, you change the value and status of the item.

The Change Item Dialog Box



Item: meklar_vt100.status
Value: 1
☐ Simulate a write request
Quality: Good: Non-specific
Limit: Limit: OK

The Change Item Dialog Box

To set an item, change the Value and click on the Set button. The different options available for this item are:

- Item: The item name. This cannot be changed.
- Value: The new value for the item.
- Simulate a Write Request: Once you change the value, the server will use the value to send out a write request. By default, this option is not checked, and is not available if no write message is available for this item.
- Quality: The Quality of the item.
- Limit: The limit for the item.

- Active: Set to "Yes" if the item has been requested by the client. Set to "No" if the item has been automatically activated by the protocol but not requested by the client.
- Value: The value of the item. This column will say "Not Set" if the item has not been initialized by the device, the client, or the server.

  Some client transports such as DDE and SuiteLink do no support the concept of having no value. Thus, until an item has an actual value, nothing will be sent to the client. Even if the quality of the item is known, nothing will be sent. One way to work around this problem is set an initial value for the item. This is a limitation of the client transport.
- Quality: The quality of the item point. Used in OPC and SuiteLink Communications (See Note 1 Below).

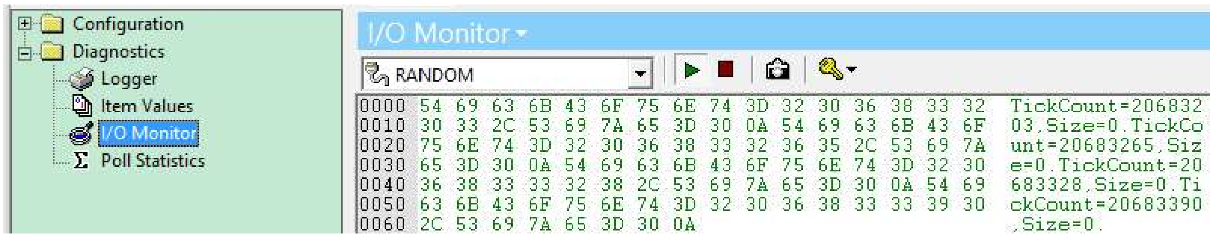
- Last Update: The last update time is the last time the value was modified OR polled. However, some client transports such as OPC might not send the last update time to the client if it has changed and the actual value has not (see Note 1 Below).
- RC: Number of pending Read operations for this item.
- WC: Number of pending Write operations for this item.

Note: An optional setting is available in the General Options for collapsing the topics shown in the Item Values window, hiding the items for that topic from view.

Note 1: OPC and SuiteLink communications are available only in the OmniServer Server Edition and the OmniServer Professional Edition.

7.2.3 The I/O Monitor

The I/O Monitor display all the data that passes through the server - both generated by the server and coming in from the device.



```
0000 54 69 63 6B 43 6F 75 6E 74 3D 32 30 36 38 33 32  TickCount=206832
0010 30 33 2C 53 69 7A 65 3D 30 0A 54 69 63 6B 43 6F  03,Size=0.TickCo
0020 75 6E 74 3D 32 30 36 38 33 32 36 35 2C 53 69 7A  unt=20683265,Siz
0030 65 3D 30 0A 54 69 63 6B 43 6F 75 6E 74 3D 32 30  e=0.TickCount=20
0040 36 38 33 33 32 38 2C 53 69 7A 65 3D 30 0A 54 69  683328,Size=0.Ti
0050 63 6B 43 6F 75 6E 74 3D 32 30 36 38 33 33 39 30  ckCount=20683390
0060 2C 53 69 7A 65 3D 30 0A                          ,Size=0.
```

The I/O Monitor

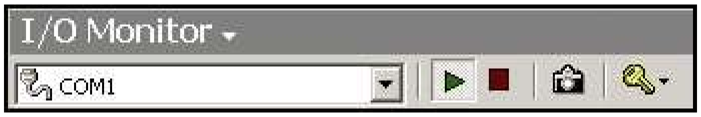
Each line of the I/O monitor displays sixteen characters, and each line is divided into three parts:

- **Reference Numbers:** These are the numbers that appear in black on the far left side of the I/O Monitor. They are used as a "line counter" and are for reference only. They do not represent any data.
- **Hexadecimal Data:** These color-coded numbers (see below for the meaning of the colors) represents the hexadecimal equivalent of the displayed data. Each byte will be shown in a range of "00" to "FF", depending upon the value of the byte (0 - 255).
- **ASCII Data:** These color-coded characters (see below for the meaning of the colors) represents the ASCII equivalent of the displayed data. This is the same view you would see if you had printed the data on a printer. Unprintable characters, such as the

Line-Feed, Carriage-Return, or Start-of-Text character, as represented as periods.

One thing to note is that the data is displayed in a sequential sequence, that is, each line is a continuation of the line before it.

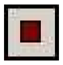
The I/O Monitor Tool Bar

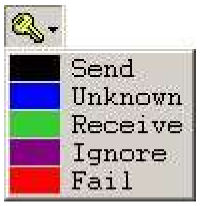


The I/O Monitor Tool Bar

You can change the view of the I/O Monitor by using the I/O Monitor Tool Bar. The options are:

-

  I/O Stream Selector Allows you to select a server device or server topic for viewing.
-
  Resume Listing Will resume the listing once it was halted by the Halt Listing button below.
- 
  Halt Listing Will halt all output to the I/O Monitor (but not to the device nor the server). This listing can be resumed by using the Resume Listing button above.
-
  Snapshot Takes an HTML snapshot of the I/O Monitor. The file will be called "io_monitor.html" and will be located in your default server directory.
-
  Key to Colors Will display a key explaining the colors used in the I/O Monitor.



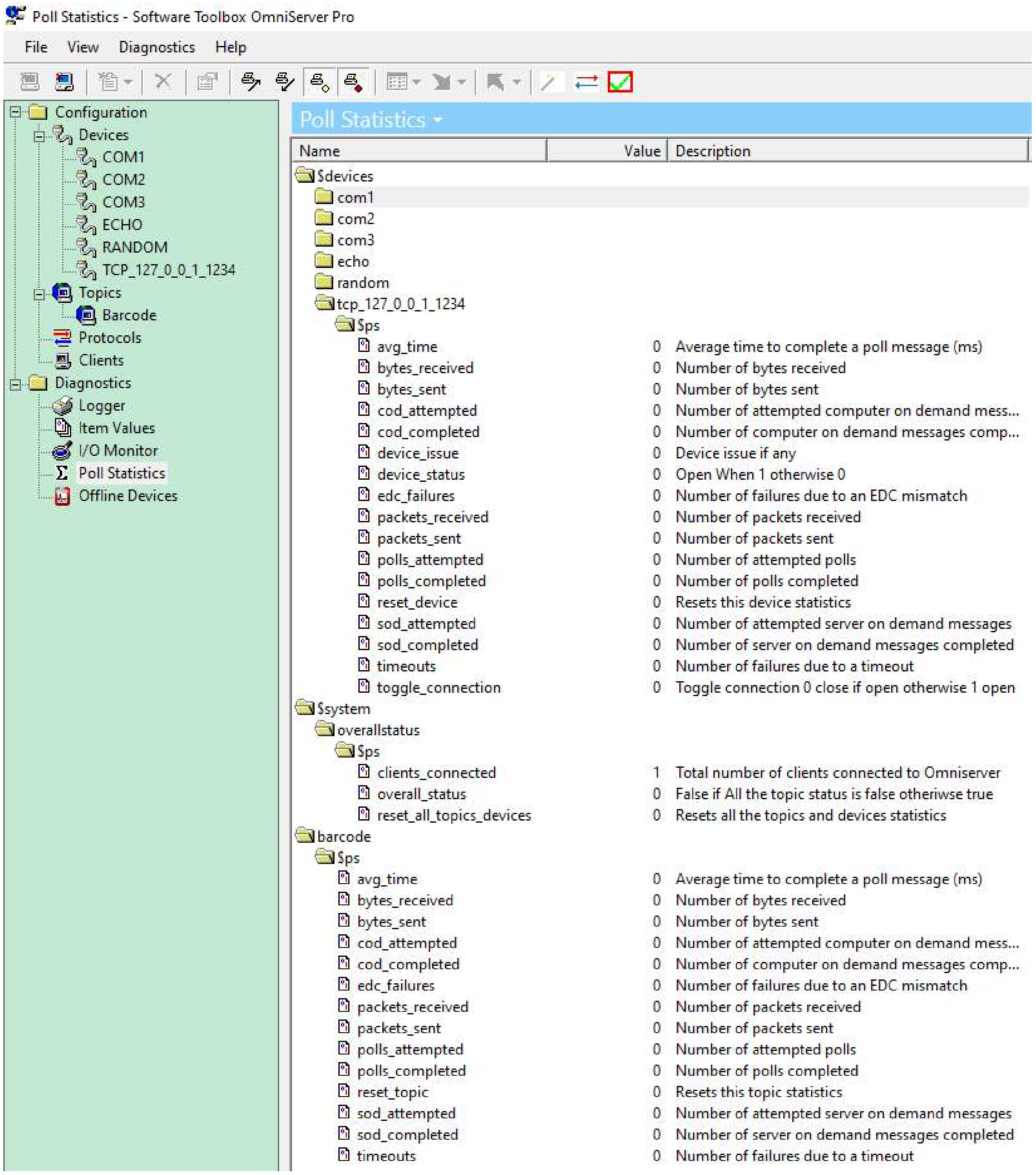
Sample Key Codes

The meanings of the keys are:

- **Send:** Data sent by the server to the device.
- **Unknown:** Data send by the device to the server that has not been processed.
- **Receive:** Data sent by the device to the server that has been processed and matched to a message.
- **Ignore:** Data sent by the device to the server that has been processed, but which did not match any message.
- **Fail:** Data sent by the device to the server that failed an Error Detection Code (EDC) scheme.

7.2.4 The Poll Statistics Window

The Poll Statistics Window displays statistics for each of the server topics and server devices available.



Name	Value	Description
$devices		
com1		
com2		
com3		
echo		
random		
tcp_127_0_0_1_1234		
$ps		
avg_time	0	Average time to complete a poll message (ms)
bytes_received	0	Number of bytes received
bytes_sent	0	Number of bytes sent
cod_attempted	0	Number of attempted computer on demand mess...
cod_completed	0	Number of computer on demand messages comp...
device_issue	0	Device issue if any
device_status	0	Open When 1 otherwise 0
edc_failures	0	Number of failures due to an EDC mismatch
packets_received	0	Number of packets received
packets_sent	0	Number of packets sent
polls_attempted	0	Number of attempted polls
polls_completed	0	Number of polls completed
reset_device	0	Resets this device statistics
sod_attempted	0	Number of attempted server on demand messages
sod_completed	0	Number of server on demand messages completed
timeouts	0	Number of failures due to a timeout
toggle_connection	0	Toggle connection 0 close if open otherwise 1 open
$system		
overallstatus		
$ps		
clients_connected	1	Total number of clients connected to Omniserver
overall_status	0	False if All the topic status is false otheriwse true
reset_all_topics_devices	0	Resets all the topics and devices statistics
barcode		
$ps		
avg_time	0	Average time to complete a poll message (ms)
bytes_received	0	Number of bytes received
bytes_sent	0	Number of bytes sent
cod_attempted	0	Number of attempted computer on demand mess...
cod_completed	0	Number of computer on demand messages comp...
edc_failures	0	Number of failures due to an EDC mismatch
packets_received	0	Number of packets received
packets_sent	0	Number of packets sent
polls_attempted	0	Number of attempted polls
polls_completed	0	Number of polls completed
reset_topic	0	Resets this topic statistics
sod_attempted	0	Number of attempted server on demand messages
sod_completed	0	Number of server on demand messages completed
timeouts	0	Number of failures due to a timeout

The Poll Statistics Window

The items are grouped by both the server devices and the server topics. You can access this information by creating items within your client (see Working with the Client).

To access each statistic item, the address is the name of the containing folder (as seen in the Poll Statistics Window) and the name of the statistic item, with a separating period.

For example, suppose you want to access the Average Time in the screen shot above. The tag address will look like this:

$devices.com1.$ps.avg_time

In addition to statistics, you will also find several other useful informational and configuration items in the Poll Statistics window:

- **device_issue** - (Available for each device) - This item maintains a value of 0/False as long as the device communicating normally and transitions to 1/True if there are any errors occurring for the device.
- **device_status** - (Available for each device) - This item specifically reflects the current status of the socket connection to the device. Maintains a value of 1/True as long as the connection to the device is active/connected and transitions to 0/False if the connection is discontinued.
- **toggle_connection** - (Available for each device) - This item allows the user to force a connect/disconnect of the socket connection to the device while also reflecting the current status of the connection. A value of 1/True indicates the connection is active/connected - writing a value of 0/False to the item will force a disconnect. A value of 0/False indicates the connection is not currently active - wirting a value of 1/True to the item will force a connection.
- **reset_device** - (Available for each device) - This item allows the user to reset all current statistics counters for the corresponding device to a value of 0 by simply writing a value of 1 to the item from a client application.
- **reset_topic** - (Available for each topic) - This item allows the user to reset all current statistics counters for the corresponding topic to a value of 0 by simply writing a value of 1 to the item from a client application.
- **reset_all_topics_devices** - (Available under each device) - These items allow the user to reset all current statistics counters for all topics and devices in OmniServer to a value of 0 by simply writing a value of 1 to the items from a client application. **The function is the same regardless of which instance of this tag is written to - ALL statistics for ALL topics and devices will be reset.**

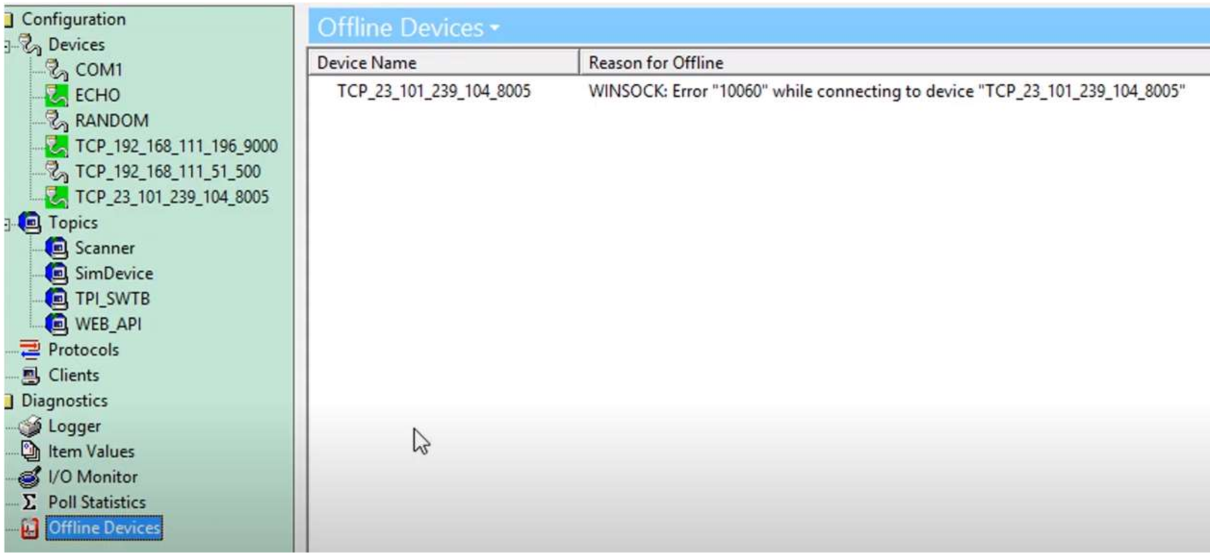
$system.overallstatus.$ps

There is also a system group with several useful system level items:

- **clients_connected** - This items displays the total number of clients currently connected to OmniServer.
- **overall_status** - This item is a composite item of all individual topic-level Status items that will show a state of "True" (or 1) when any topic status is "False" (or 0) meaning the topic is having communication issues. It can be monitored from client applications as an overall server-level indicator that there may be an underlying communications issue to diagnose.
- **reset_all_topics_devices** - This item can be toggled to reset all values for the statistics items for all topics and devices in the server (as opposed to toggling the "reset_topic" or "reset_device" for each individual topic/device).

7.2.5 The Offline Devices Window

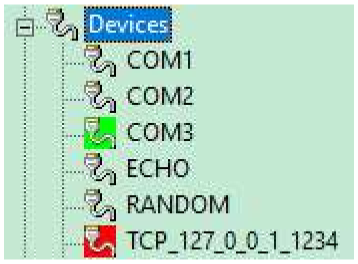
The Offline Devices window displays any devices having communications issues (i.e. devices in red, if you have Runtime Device State Coloring enabled) along with the error causing the communication issue. This error corresponds to the the error displayed in the OmniServer Logger for the associated device.



Device Name	Reason for Offline
TCP_23_101_239_104_8005	WINSOCK: Error "10060" while connecting to device "TCP_23_101_239_104_8005"

7.2.6 Runtime Device State Coloring

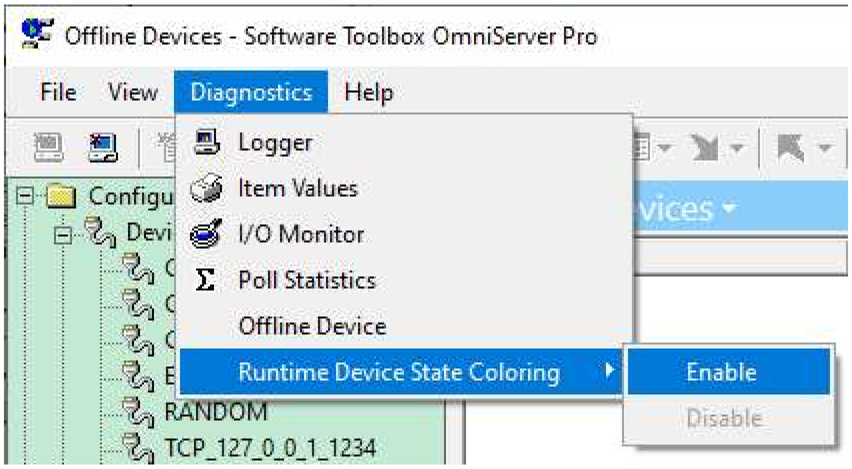
Runtime Device State Coloring in the OmniServer user interface works in conjunction with an individual topic's status, enhancing visual troubleshooting.



When the topic's Status is "True", communications are normal and the associated device will be green in the Devices view in OmniServer.

When the topic's Status is "False", there is a communications fault and the associated device will be red in the Devices view in OmniServer. You can determine the cause of the issue by going to either the OmniServer Logger or Offline Devices windows.

To utilize this functionality, Runtime Device State Coloring must be enable in the Diagnostics menu under Runtime Device State Coloring.



7.3 User Interface

Of course, if the heart of the server is the runtime engine, the hands is the Graphical User Interface, or GUI for short. Using the GUI, you can create, change, and delete just about any option or object available.

Here are the four main parts of the server's GUI:

- The Menu Bar: First off, the Menu Bar is used to access all the different functions of the server.
- The Tool Bar: Next, the Tool Bar is used to quickly access commonly used functions, such as New, Delete, and Properties.

- The Tree View: The Tree View, located on the left side of the configuration window, contains an expandable/collapsible hierarchy allowing for quick navigation through the various configuration and diagnostic windows.
- The View Window: Finally, the View Window is what displays all the information for each of the windows.

7.3.1 The Menu Bar

If tool bars do not strike your fancy, all functions for the server can be accessed via the **Menu Bar**.

The menu bar looks like this:

The Menu Bar

Placing your mouse pointer over one of the words and clicking the left mouse button will reveal a "pull-down" menu, which gives you an even greater choice in options.

To discover what each of these menu selections do, click on the links below.

- The FILE Option
- The VIEW Option
- The DIAGNOSTICS Option
- The HELP Option

7.3.1.1 File

The first menu option available on the server is **File**. Here you will find the majority of the options not associated with viewing or diagnostics.

Some options, like "Connect" to the left, are grayed out and unavailable whenever that particular option cannot be selected due to the current server state. For example, when an item, such as a topic or a protocol, is selected, the "Delete" and "Properties" options become available.

Also, the menu options may change, depending upon if you are looking at Configuration Windows or Diagnostic Windows.

File Menu Options:

- **Connect** - Connects the OmniServer Configuration to the OmniServer Poller (Runtime)
- **Disconnect** - Disconnect the OmniServer Configuration from the OmniServer Poller (Runtime) - You will receive a warning prompt indicating the disconnect is occurring.
- **New** - Configure new server objects. Different sub-menus will be displayed depending on which view has the current focus (Views with sub-menus are Devices, Topics and Protocols).
- **Import** - Import Devices (CSV file), Topics (CSV file) or Protocols (DPD file) depending on which view has the current focus.
- **Export** - Export Devices (CSV file) or Topics (CSV file) depending on which view has the current focus. Devices are exported based on type (COM, WINSOCK or LPT) to individual CSV files.
- **Delete** - Deletes the currently selected server object (if permitted).
- **Properties** - Opens the property window of the currently selected server object (where applicable)
- **Print** - Prints a list of selected configuration objects and their settings. Options are Device Configurations, Topic Definitions, Protocols, Clients, Logger Contents and Snapshot File Contents.
- **Print Setup...** - Launches a configuration window for configured printer settings.
- **Build by Wizard** - Launches the configuration wizard for building a new topic (including selection of new or existing protocols and new or existing devices). Click Here for Details.
- **Snapshot** - Generates a file named snapshot.txt in the OmniServer installation directory that contains snapshot data of all currently selected logging options under View | Logging. The default selections are Errors and Events only, but Sends and Receives may also be selected.
- **Import CFG** - Opens and applies a selected omnisrv.cfg configuration file (which contains Topic Definitions and Device Configurations).

- **Start Poller** - Starts the OmniServer Poller (runtime). Only available when the OmniServer Poller is currently stopped.
- **Stop Poller** - Stops the OmniServer Poller (runtime). Only available when the OmniServer Poller is currently started. This is the equivalent of right-clicking on the OmniServer system tray icon and selecting "Exit", and is required to restart the demo period and to complete registration.
- **Exit** - Closes the OmniServer Configuration window.

Selecting the **File | Connect** option from the menu connects the server configuration program (the one you are currently working with) to the server polling program (the one that actually polls the device).

This selection is active when there is no polling program connected to the configuration program.

This selection is inactive then there is an existing connection between the configuration program and a polling program.

The **File | Disconnect** option is used to disconnect the server configuration program (the program in which you are currently working) with the server polling program (the program that polls the device). All configurations are saved immediately when you exit out of a configuration dialog for devices and topics. In the case of protocols, the protocol is not automatically saved, but the protocol window still remains connected to the server. Once the protocol is saved and the window closed, then all connections will be closed.

This selection is active when there is a polling program connected to the configuration program.

This selection is inactive when there is no polling program connected to the configuration program.

The **New** option creates new instances of whatever configuration is currently on the screen.

This is a sample File | New Menu. In this case, the options you see are the ones that appear when the **Devices** configuration window is active. If the **Topics** window was active, the sub-menu will show the "Topic" menu, while the **Protocols** window will show the "Protocol" menu.

This selection is active when a polling program is connected to the configuration program, and one of the Configuration Windows are being displayed.

This selection is inactive when a polling program is not connected to the configuration program, or when one of the Diagnostic Windows are being displayed.

The **Delete** option will delete the currently selected object in the configuration window. If no object is selected, then the Delete option will be grayed out.

Upon selecting Delete, you will get a confirmation dialog box. Click on "Yes" to complete the deletion, and "No" to cancel.

Warning: There is no "Undo" function. Once the object has been deleted, it is gone for good. Make sure you mean "Yes" when you click on "Yes".

This selection is active when a Configuration Window is displayed and one object is selected.

This selection is inactive when a polling program is not connected to the configuration program, when any Diagnostic Window is displayed, or when an object is not selected. If an object is selected and cannot be deleted, then this option is inactive.

The **Properties** option displays a dialog box that allows for the changing of any parameters associated with the object selected. For example, one of three menus will appear if you have selected one of the following:

- Topic. The Topic Definitions dialog box will appear. You can now change any Topic data.
- Device. The Device Configuration dialog box will appear. This is called by one of three names: **COM Port Configuration** for Serial Devices; **LPT Configuration** for Parallel Devices; **Winsock Device Configuration** for TCP, UDP, and TELNET Devices.
- Protocol. The Protocol Definition dialog box will appear, giving you a chance to change your protocol.

This selection is active if a polling program is connected to the configuration program, one of the Configuration Windows is displayed, and one object is selected in that window.

This selection is inactive if a polling program is not connected to the configuration program, one of the Diagnostic Windows is displayed, or an object on a displayed Configuration Window is not selected. If an object is selected and cannot display properties, then this option is inactive.

Selecting the **Print** option takes you to the Print Dialog Box, which contains all the different printing possibilities for the server.

The options are:

- Print Device Configurations: Prints out the setup for each server device.
- Print Topic Definitions: Prints out the details of each server topic defined.
- Print Protocols: Prints out the configuration of each protocol in the server. By default, all the protocols will be printed. However, you can select individual protocols or groups of protocols by clicking on the Select button next to the option.

  You can select the protocols to include in the printout from this dialog box.
- Print Logger Contents: This will print out the contents of the Local Logger.
- Print Clients: This will print out client information.
- Print Snapshot File Contents: Prints out the SNAPSHOT.TXT file. This file is created when you select **File | Snapshot** from the server menu. It is usually located in the default server directory, and contains the contents of all the diagnostic windows.

This selection is active whenever a polling program is connected to the configuration program.

This selection is inactive whenever a polling program is not connected to the configuration program.

Print Setup allows access to the Setup routine for your default printer. Here you can change the appearance of the printing for anything within the server.

This selection is always active.

The **Snapshot** option produces a file called **SNAPSHOT.TXT**, which is normally located in the directory in which your server's was installed. This snapshot file contains the contents of all the diagnostic windows available with the server. The snapshot file can be printed using the **File | Print** option in the server, viewed with any text editor on your computer, or sent via email to a service technician.

This file is primarily used as a diagnostic tool for trouble shooting protocols. Used in conjunction with the protocol file, service technicians can diagnose most protocol problems.

This selection is always active.

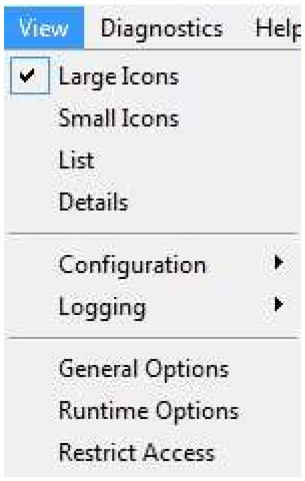
The **Exit** option will exit the Configuration program.

It is important to note that this option will **NOT** exit the server's Polling program, only the server's Configuration program. If you need to exit the polling program, right-click on the Polling program's icon in the icon tray and select the Exit option.

If the polling program is running as a service, there will **not** be an icon on the desktop. The polling program must be stopped (or started) from the Services window in the control panel.

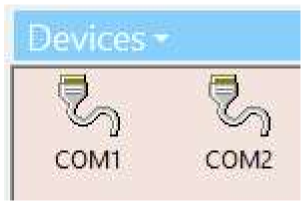
This selection is always active.

7.3.1.2 View



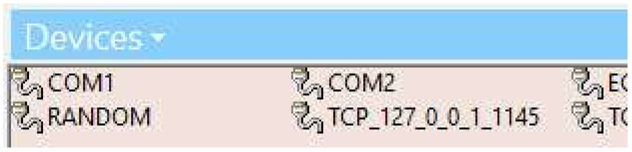
With the **View** menu, you will find the options to change the looks of your server software.

Selecting the **View | Large Icon** option from the menu displays each server object using a larger icon.



Sample view of a Large Icon Screen

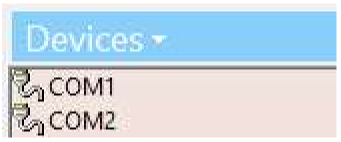
Selecting the **View | Small Icon** option from the menu displays each server object using a smaller icon.



Sample view of a Small Icon Screen

The main difference between this view and the **List** view is that the Small Icon view will list the objects along multiple columns, while the List view merely lists all the server objects along one column.

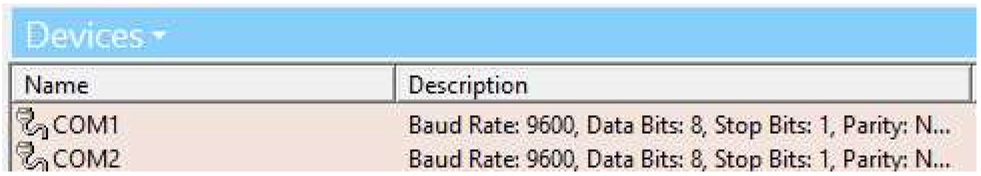
Selecting the **View | List** option from the menu displays each server object in a list form.



Sample view of a List Screen

The main difference between this view and the **Small Icon** view is that the List view merely lists all the server objects along one column, while the Small Icon View will list the objects along multiple columns.

Selecting the **View | Detail** option from the menu displays each server object in two columns. Column #1 is the name of the object, while Column #2 is the Description. (This is the Default View)

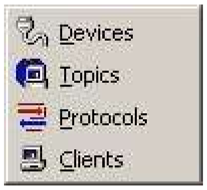


Name	Description
COM1	Baud Rate: 9600, Data Bits: 8, Stop Bits: 1, Parity: N...
COM2	Baud Rate: 9600, Data Bits: 8, Stop Bits: 1, Parity: N...

Sample view of a Detail Screen

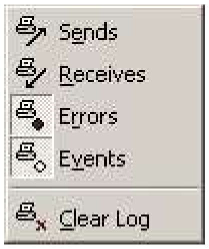
Please note that some objects - specifically Server Devices - do not have descriptions.

The **View Configuration** menu selection allows you to select one of the four server configuration objects available.



The View Configuration Menu

The **View Logging** menu selection allows you to select one of the four server logging options available. Any option selected here will appear in the Local Logger diagnostic window.



The View Logging Menu

The options are:

- Show Sends: Setting this option records all data going from the server to the device. The default is **off**.
- Show Received: Setting this option records all data coming into the server from the instrumnet. The default is **off**.
- Show Errors: Setting this option records all errors encountered by the server. The default is **on**.
- Show Events: Setting this option records all events generated by the server. The default is **on**.
- Clear Log: Select this option to clear out the contents of the Local Logger diagnostic window.

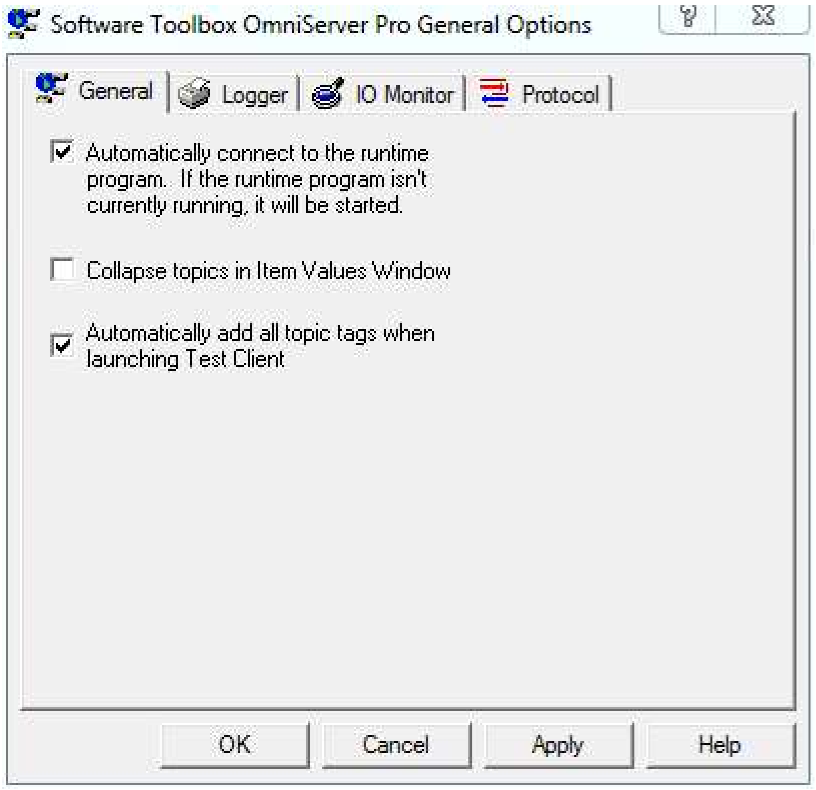
7.3.1.2.1 Options

General configuration for how the server "looks" and boots up can be set up using the View option from the main menu.

Here are the three options available:

- General Options: Change how the different diagnostic windows appear, plus connection parameters for the server runtime program.
- Logger Options: Change how the Local Logger diagnostic window appears.
- I/O Monitor Options: Change how the I/O Monitor diagnostic window appears.
- Protocol Options: Change how the Protocol Editor window appears.

7.3.1.2.1.1 General



☑ Automatically connect to the runtime program. If the runtime program isn't currently running, it will be started.

☐ Collapse topics in Item Values Window

☑ Automatically add all topic tags when launching Test Client

View Options General is the repository for the "global" parameters used by the server. Currently, those options are:

- **Automatically Connect to the Runtime Program -** Enabling this option will connect the OmniServer Configuration program to the server runtime program whenever the configuration program is started. If the runtime program is not running, the configuration program will launch it.

By not checking the box, you will have to manually start the runtime program or manually connect to an existing running one (see the **Connect** option) before you can edit any devices, topics, or protocols.

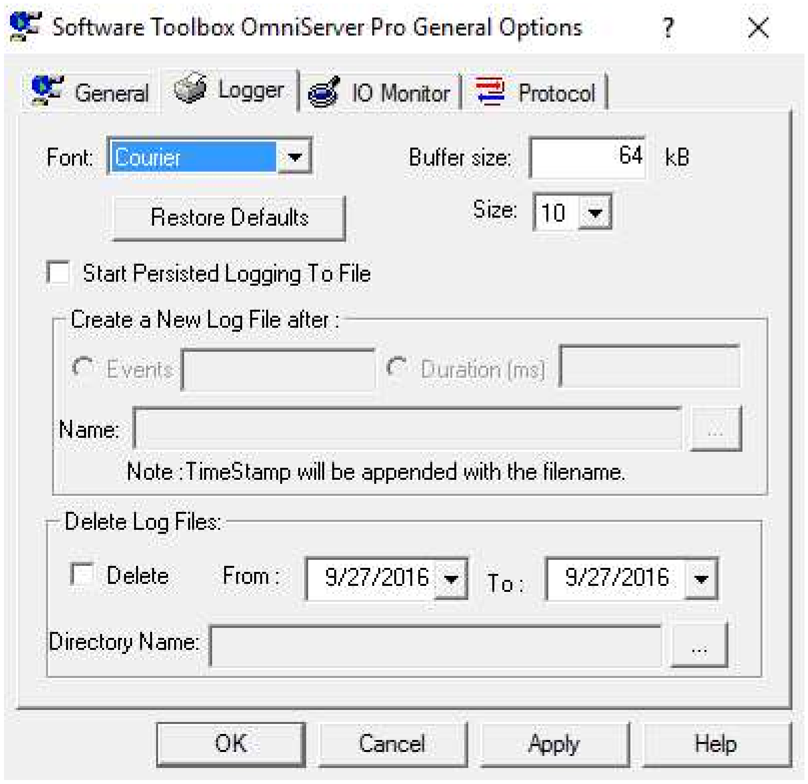
- **Collapse topics in Item Values Window** - Enabling this setting allows topics in the Item Values diagnostic window to be collapsed, hiding any items for a topic that are being accessed by a client application. This allows the user to view items for only the desired topics, making the view much more user friendly.

  Note: Upon enabling or disabling this setting, it will only take effect after restarting the OmniServer Poller.

- **Automatically add all topic tags when launching Test Client** - Enabled by default, this setting controls whether or not the Software Toolbox OPC Test Client that installs with OmniServer will auto-populate all of your protocol tags for each configured topic when launched from the OmniServer Configuration toolbar.

  Disabling this setting means the test client will launch with a blank configuration and you will need to manually configure a connection to OmniServer and subscribe to the desired tags for your topics.

7.3.1.2.1.2 Logger



Font: Courier

Buffer size: 64 kB

Size: 10

☐ Start Persisted Logging To File

Create a New Log File after :

Events | Duration (ms)

Name:

Note :TimeStamp will be appended with the filename.

Delete Log Files:

☐ Delete From : 9/27/2016 To : 9/27/2016

Directory Name:

The View Options Logger allows you to change the font displayed on the Local Logger diagnostic window. (Please note that not all fonts are available - only those that are of fixed-width characters.)

You can also specify the size of the buffer reserved for the Logger. The maximum size is 999 KB, while the minimum is 64 KB.

Persisted Logging to File - OmniServer allows logging the event log details to an extended log file for longer term troubleshooting.

1. Enable persisted logging by checking "Start Persisted Logging To File" setting.
   a. Once enabled, you will have the following options:
      i. **Create a New Log File after** - two options are provided that determines when file spanning will create a new log file.
      ii. **Events** - selecting this option and entering the desired number of events allows file spanning to create a new log file after the specified number of events has been logged to the current log file. Valid range is 1,000 events up to a maximum of 10,000 events.
      iii. **Duration (ms)** - selecting this option and entering the desired time in milliseconds allows file spanning to create a new log file after the specified amount of time has elapsed. Valid range is 10,000 ms up to a maximum of 2,147,483,647 ms.

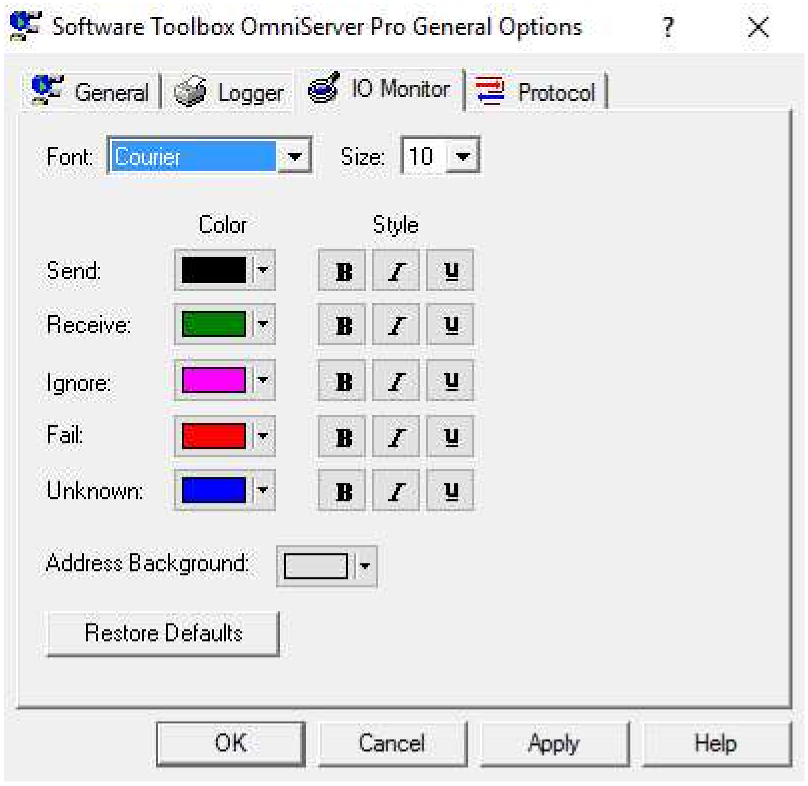
b. **Name** - field specifies the file location and root file name that OmniServer should use for creating log files. Current timestamp will be appended to the root file name upon the file being created.

By default, a maximum of 10 log files will be maintained until manually deleted. As additional log files are created, the oldest log file will be removed (First In, First Out).

To easily delete multiple old log files within a date range, the following settings are provided:

1. Enable old file deletion by checking the "Delete" setting under the "Delete Log Files" heading.
   a. **Note:** Directory Name is automatically filled based on the location where log files are currently stored.
2. Once enabled, you will need to specify a "To" date and "From" date that defines the range of old files that you would like to delete from the specified directory.

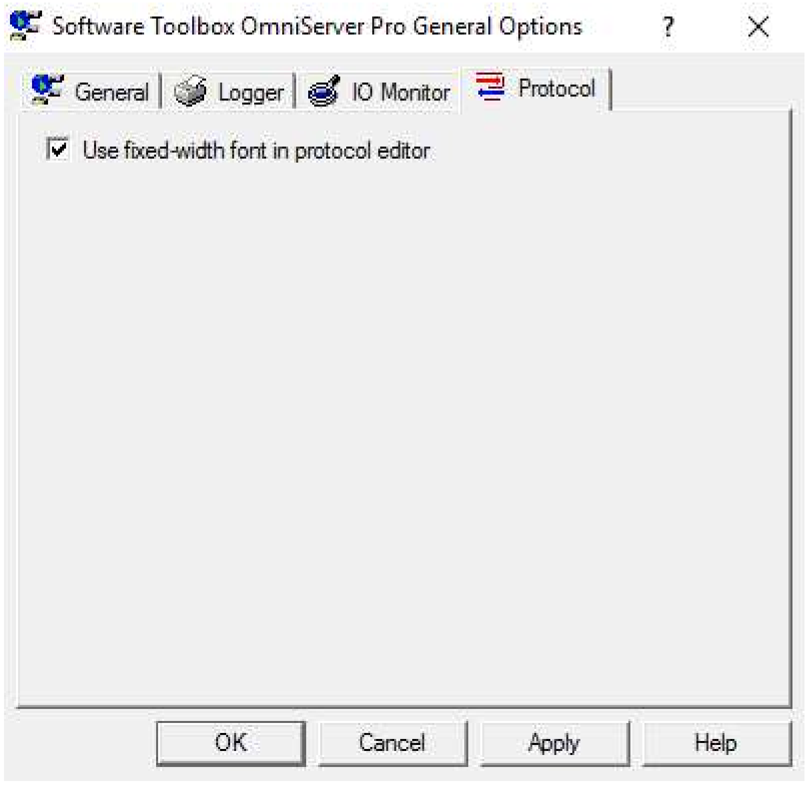
7.3.1.2.1.3 I/O Monitor



Font: Courier
Size: 10
Send:
Receive:
Ignore:
Fail:
Unknown:
Address Background:

The View Options I/O Monitor tab allows you to change the appearance of the I/O Monitor. You can change the color or the style of each of the five display components of the I/O Monitor, or change the font for the entire screen. Please note that not all fonts are available - only those of fixed-character width are included.

7.3.1.2.1.4 Protocol

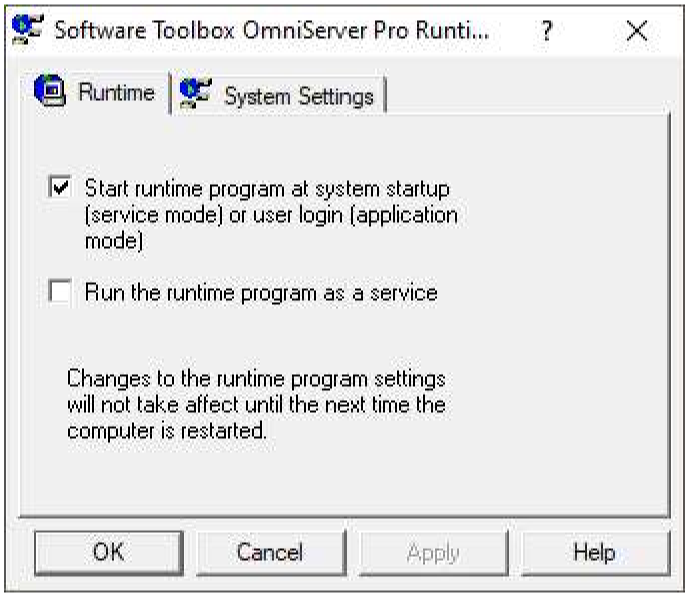


[✓] Use fixed-width font in protocol editor

View Options Protocol is where you will change global parameters associated with the Protocol Editor. Currently, those options are:

- Use fixed-width font in protocol editor. Checking this box will display a fixed-width font in the Protocol Editor instead of the proportional font as used in earlier versions of the server.

7.3.1.2.1.5 Runtime Options



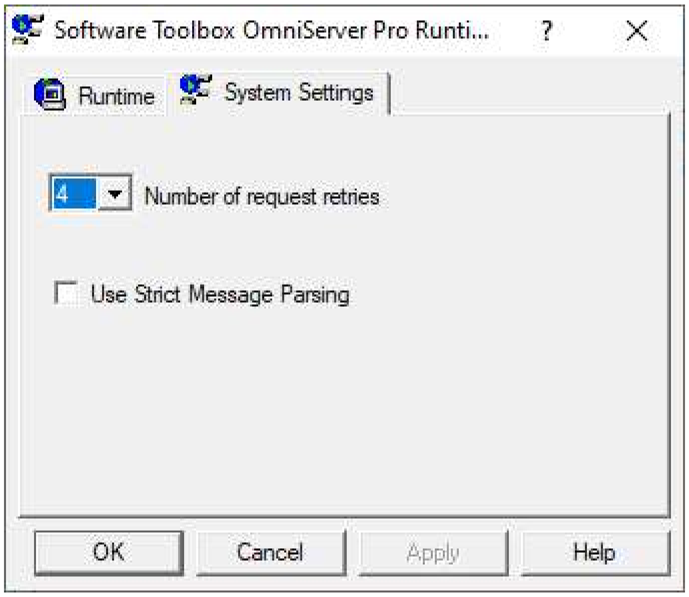
- [✓] Start runtime program at system startup (service mode) or user login (application mode)
- [ ] Run the runtime program as a service

Changes to the runtime program settings will not take affect until the next time the computer is restarted.

The **View -> Runtime Options** section contains configuration information for the server runtime program. The Runtime section options are:

- **Start Runtime Program at System Startup of User Login** - Set this option if you wish to runtime program to begin once the system starts or a user logs in.
- **Run the Runtime Program as a Server** - Set this option to have the server start up as a Windows system service (i.e. allows the OmniServer runtime to continue operating without a user logged in).

The System Settings section options are:



4 Number of request retries

☐ Use Strict Message Parsing

- **Number of request retries** - This setting corresponds to the number of times OmniServer will resend a Request for which OmniServer did not receive a response within the Reply Timeout setting in the Device Properties. This only affects Command/Request (Host) Messages with a Response configured. The default setting is 2 retries, meaning a Request will be sent a total of 3 times.

- **Use Strict Message Parsing** - This setting (disabled by default) increases the degree that the OmniServer parsing engine expects an exact, byte-for-byte response from your devices compared to the response-type messages configured in your protocol.

  When this setting is enabled, OmniServer will no longer consider responses that are returned in the middle of a larger message body as valid responses. OmniServer will expect every response from the device to match the configured response in the protocol exactly.

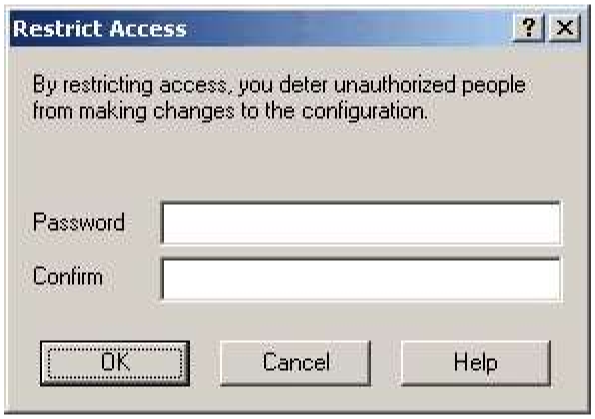
  This setting can be useful when data from the device does not begin with a leading character.

  By default, when this setting is disable, the OmniServer parsing engine is more flexible with handling partially matching responses (such as when the valid response is contained within other data that is undefined in the OmniServer protocol, so this setting should remain disabled for most users.

7.3.1.2.1.6 Restrict Access

The Restrict Access option allows you to restrict the access of this server to anyone who does not know a password set up previousl.

By default, the server is open and available to all for changes. To restrict the access, select this option. You will then be given this screen:


Restrict Access

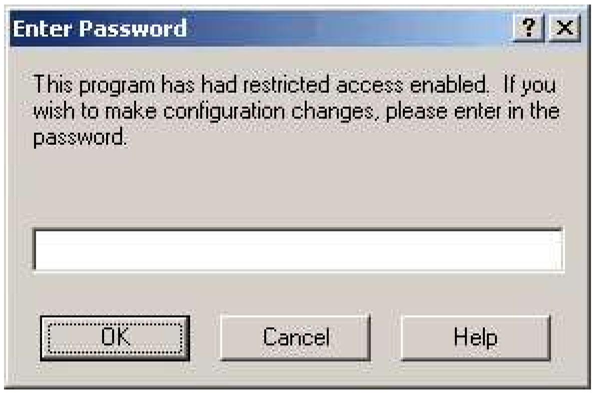
By restricting access, you deter unauthorized people from making changes to the configuration.

Password

Confirm

Set up a New Restriction

Type in a password and a confirmation password, then click OK. Once you save and close the server, any one attempting to access it will get this dialog box:


Enter Password

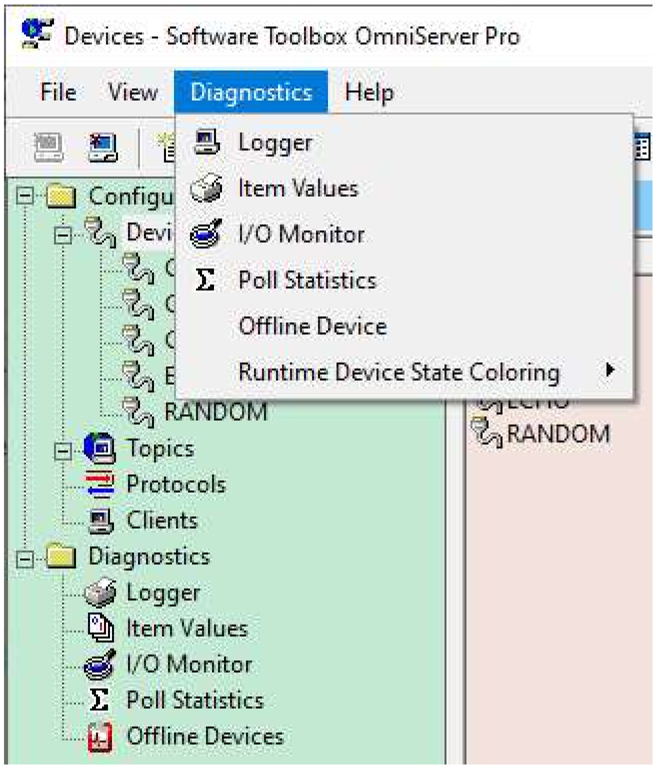
This program has had restricted access enabled. If you wish to make configuration changes, please enter in the password.

Confirming Access

No one will be able to change anything in the server until the correct password is entered.

To remove the restriction, select **View | Restrict Access** again, type in the confirming password, and you will receive a message that the restriction has been taken off.

7.3.1.3 Diagnostics



See also online - Exploring OmniServer Diagnostic Tools

To help you debug your protocol and/or troubleshooting any issues, the following tools are available:

- I/O Monitor: This diagnostic window gives you a window on the all the data flowing through the server.
- Item Values: This window show the current value of all the items being requested by the client or items automatically activated by the protocol.
- Local Logger: This window contains all the error and event information for the server. Additionally, it can show all the data going into and coming out of the server.
- Poll Statistics: This diagnostic window gives you statistics for each active server device and server topic.
- Offline Device: This diagnostic window displays any devices that currently have errors associated with them.

- Runtime Device State Coloring: When enabled, this tool provides a visual green or red indication in the device list of any devices that are currently experiencing errors.

7.3.1.4 Help

For Help on any of these menu options, place your mouse over the options and click.

The **Help** menu gives you help on this server, plus links to support and website addresses for the manufacturers of this server.

This option is always available.

The **Help | Index** option displays this help screen.

This option is always active.

Selecting the **Help | Registration Code** option from the menu provides options for licensing your OmniServer, either via Software License or Hardware License. If neither a software or hardware license is present, OmniServer will run as a demonstration version. This demonstration version will run for 2 hours before all functions are shut down (you can always restart the Omniserver runtime to reset this demo period).

This option is always available.

The **Help | Email Support** option will use your Email program to begin composing a mail message that will be sent to the support personnel for this server. Simply compose the message and send it out.

This selection is always active.

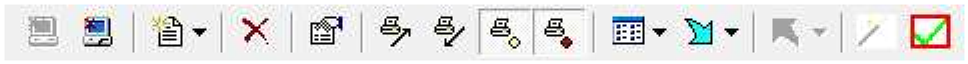
The **Help | On The Web** option opens up your default web browser program and sends you to the web site for this server's manufacturers.

This selection is always active.

The **Help | About** option displays a box that gives general information about the server and the manufacturers of the server.

7.3.2 The Tool Bar

You can access all the functions available in configuring or diagnosing your server objects through the Tool Bar.


The Server Tool Bar

There are 14 tool bar selections. They are:

-

  Connect

- Connects the configuration program to a server runtime program. This option is only available when there is no current connection to a runtime program.

-

  Disconnect

- Disconnects the configuration program to a server runtime program. Please note that this does **not** stop the runtime program, only the ability to configure objects. This option is only available when there is an active connection to a runtime program.

-

  New

- Create a new server object. This option is only available when a Configuration window is the current window.

-

  Delete

- Delete a selected server object. This option is only available when a server object is selected (such as a topic, device or protocol).
-

Properties

- Display the properties for the current object. This option is only available if an object has been selected, and a Configuration window is active. Clicking this option has the same effect as double-clicking on the object itself.

-

Show Sends

- Logs all data generated by the server and sent to the device. The data is logged into the Local Logger diagnostic window. The default for this option is **off**.

-

Show Receives

- Logs all data generated by the device coming into the server. The data is logged into the Local Logger diagnostic window. The default for this option is **off**.

-

Show Events

- Logs all events generated by the server. The data is logged into the Local Logger diagnostic window. The default for this option is **on**.

-

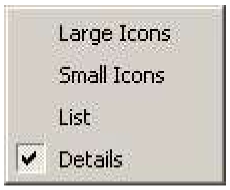
  Show Errors

- Logs all errors generated by the server. The data is logged into the Local Logger diagnostic window. The default for this option is **on**.

-

  View

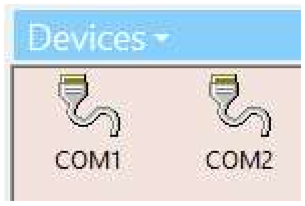
- Changes the current format of the server's View Window.



The View Options

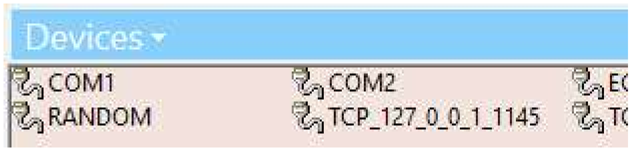
The four options are:

- Large Icon: Displays each server object using a larger icon.



Sample view of a Large Icon Screen

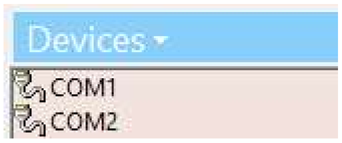
- Small Icon: Displays each server object using a smaller icon.



Sample view of a Small Icon Screen

The main difference between this view and the **List** view is that the Small Icon view will list the objects along multiple columns, while the List view merely lists all the server objects along one column.

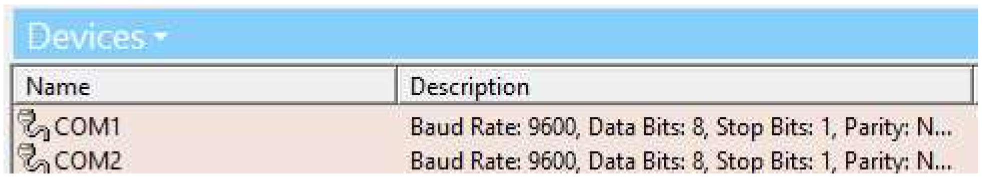
- List: Displays each server object in a list form.



Sample view of a List Screen

The main difference between this view and the **Small Icon** view is that the List view merely lists all the server objects along one column, while the Small Icon View will list the objects along multiple columns.

- Detail: Displays each server object in two columns. Column #1 is the name of the object, while Column #2 is the Description.



Name	Description
COM1	Baud Rate: 9600, Data Bits: 8, Stop Bits: 1, Parity: N...
COM2	Baud Rate: 9600, Data Bits: 8, Stop Bits: 1, Parity: N...

Sample view of a Detail Screen

Please note that some objects may not have descriptions.

-

Import

- Imports server objects based on the current view (Devices or Topics from CSV file) / (Protocols from DPD file).

-

Export

- Exports server objects to CSV file based on the current view (Devices or Topics).

-

  OmniServer Wizard

- Launches the Topic Configuration Wizard.

-

  Test Connectivity

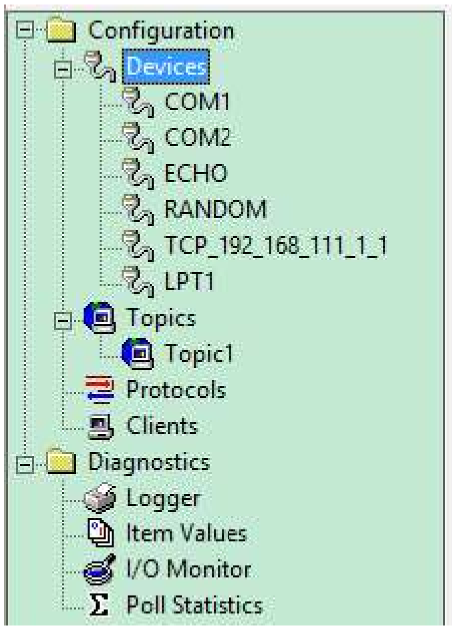
- Launches the Software Toolbox OPC Test Client.

7.3.3 The Tree View

Server Details : The User Interface

Tree View

The **Server Tree View** allows you to quickly navigate between the different Server Configuration components and Server Diagnostic tools.

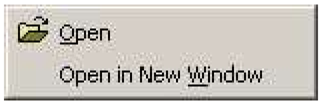


The Configuration section of the Tree View displays the possible selections available for configuring the server. Expanding or highlighting any of these sections will change the View window to the appropriate list of objects for that selection.

For Devices and Topics, expanding the Tree View will display the list of Devices and Topics configured, respectively.

Highlighting a configured device or topic then displays the settings in the right pane for editing.

You can open the window not only in the current view window, but also in a new window separate from the current one. To do this, right-click on an item. You will get this popup menu:

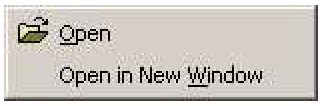


The Right-Click Popup Menu

Open will open the object's window in the current window, while **Open in New Window** will open the object's window in a new, separate window.

The Diagnostics section of the Tree View displays the possible selections available for diagnosing the current condition of the server and the runtime engine. Selecting any of these options will change the View window to the corresponding diagnostic tool.

You can open the window not only in the current view window, but also in a new window separate from the current one. To do this, right-click on an item. You will get this popup menu:



The Right-Click Popup Menu

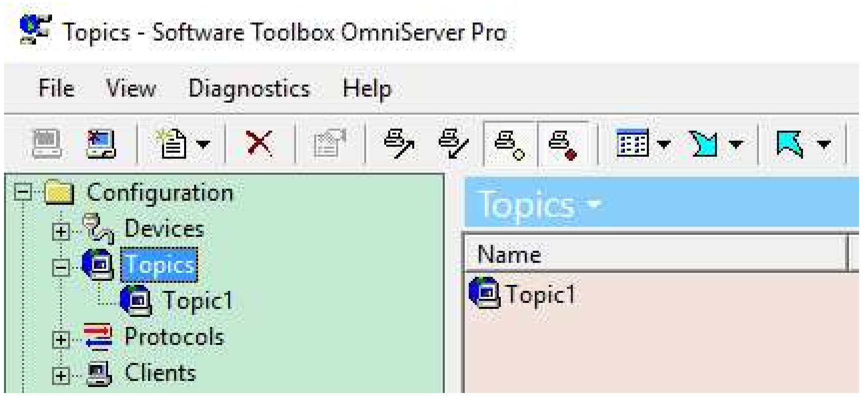
Open will open the object's window in the current window, while **Open in New Window** will open the object's window in a new, separate window.

7.3.4 The View Window

Server Details : The User Interface

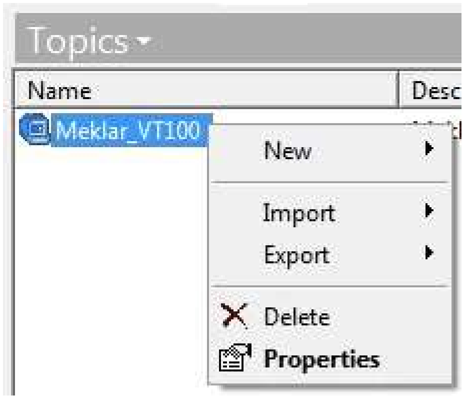
The View Window

The View Window holds all the detailed information for the any of the Configuration Windows or the Diagnostics Windows. When you select any one of those windows (via the menu, tree view, or view drop down menu), the detailed information for that window will appear here.



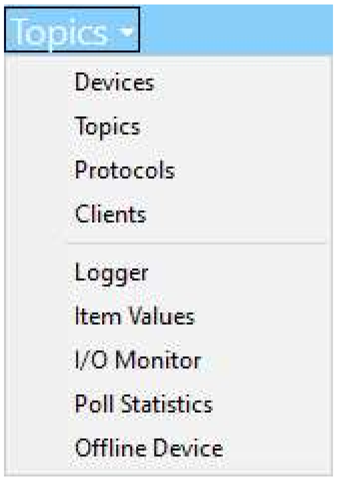
Sample VIEW Window - In this case, the Topics List Window

You can also use the right mouse button to bring up a quick pop-up menu for adding new objects, deleting objects, or setting the properties of an object.



Sample Pop-up menu

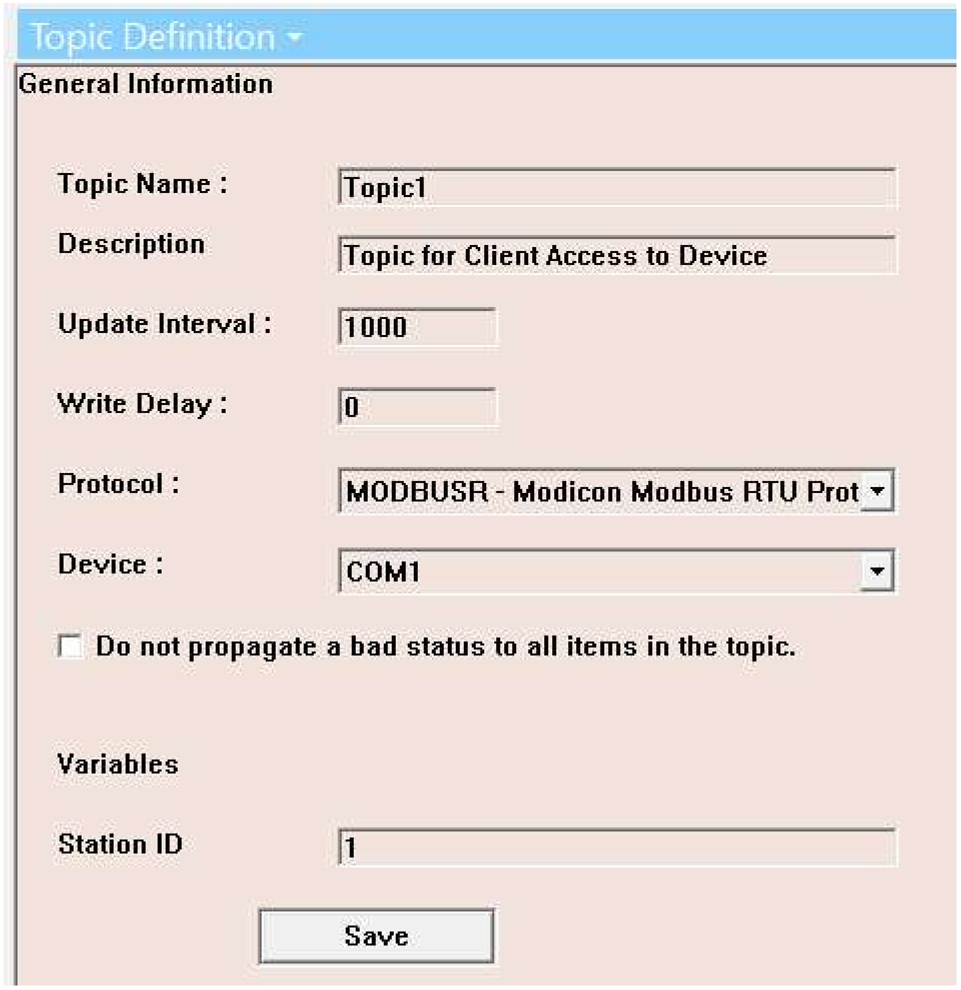
You can use the View Selection button at the top of the view window to quickly change the view. Do this by clicking on the Down Arrow located next to the name of the window.



The View Selection Menu

Selecting a configured Topic or Device from the Tree View will display the available settings in the View Window.

With a configured Topic selected in the Tree View:


Topic Definition

General Information

Topic Name : Topic1

Description Topic for Client Access to Device

Update Interval : 1000

Write Delay : 0

Protocol : MODBUSR - Modicon Modbus RTU Prot

Device : COM1

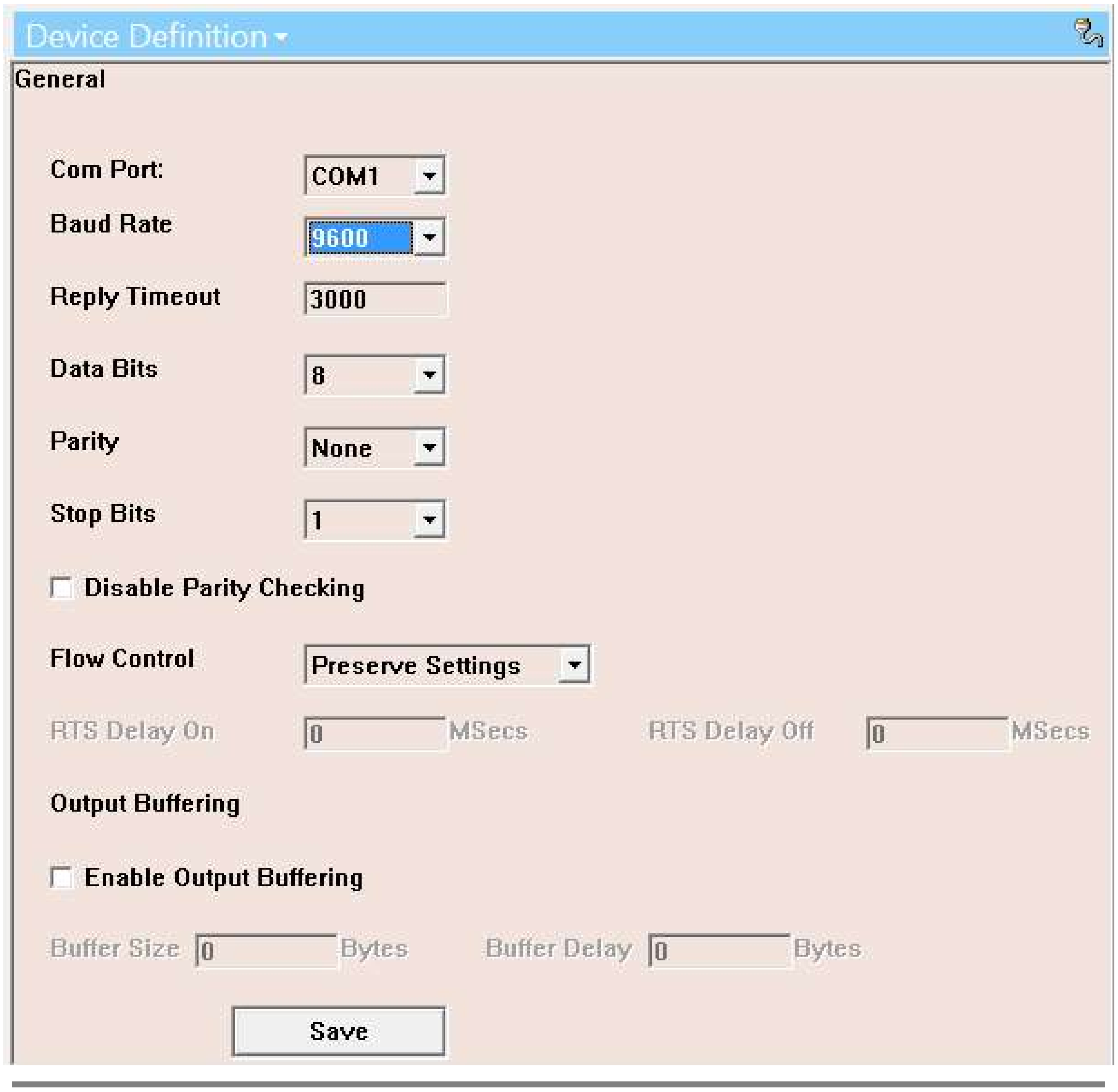
☐ Do not propagate a bad status to all items in the topic.

Variables

Station ID 1

Save

With a configured Device selected in the Tree View:



Com Port: COM1

Baud Rate 9600

Reply Timeout 3000

Data Bits 8

Parity None

Stop Bits 1

☐ Disable Parity Checking

Flow Control Preserve Settings

RTS Delay On 0 MSecs RTS Delay Off 0 MSecs

Output Buffering

☐ Enable Output Buffering

Buffer Size 0 Bytes Buffer Delay 0 Bytes

Sample Tutorials

8 Sample Tutorials

Sample Tutorials

This help file contains a number of step-by-step tutorials on how to implement the protocols of certain types of devices.

See also OmniServer Online Training Videos

Although you will not find your specific device here (in most cases), you should be able to locate a general description of you device, and from it's tutorial determine how to build your own protocol.

You can choose from one of following tutorials:

- Barcode Scanner Tutorials
- Weigh Scale Tutorials
- "Marquee"-type device Tutorials
- Measurement Temperature, Pressure, etc.) Tutorials
- PLC Tutorials
- Response to Events (Alarms, Push Buttons) Tutorials
- Communicating with devices using Modems

Or, you can choose your tutorial based upon the server function you wish to learn:

- Using Unsolicited Messages:
  Barcode Scanner Tutorials
  Response to Events (Alarms, Push Buttons) Tutorials
- Using Read Command/Request (Host) Messages:
  Measurement Temperature, Pressure, etc.) Tutorials
  PLC Tutorials
- Using Write Command/Request (Host) Messages:
  "Marquee"-type device Tutorials
  Communicating with devices using Modems
- Using Registers:
  PLC Tutorials

- Using Topic Variables:
  PLC Tutorials
- Using Chained Message:
  Communicating with devices using Modems
- Using Message Triggers:
  "Marquee"-type device Tutorials
  Communicating with devices using Modems
  Response to Events (Push Buttons Tutorial)
- Using Error Detection Codes:
  Measurement Temperature, Pressure, etc.) Tutorials
  PLC Tutorials

As always, if you require assistance with your particular protocol, you can always click here for contact information.

8.1 Barcode

Tutorial: Barcodes

These Barcode Tutorials take you through the steps in building barcode-like protocols. These are useful for most barcode and laser scanner devices.

- RS-232 Tutorial - Learn how to setup a protocol when the device is on an RS-232 port.
- Terminal Server - If you have multiple scanners on a terminal server, use this tutorial.

8.1.1 Barcode on a Serial Port

Tutorial: Barcode - RS232 - Simple String

This tutorial takes you through implementing a simple barcode protocol.

Device Specifications

This device is a barcode reader that scans custom–made barcodes and reports that data through a COM port on the computer. The data returned is rather simple: the actual string of characters printed on the barcode followed by a carriage return.

Technical Specifications

- Server Device: COM3
- Data Stream from device: A random number of characters, followed by a carriage return (CR).
- Data Needed by Client: One string of characters, which will be transfer under the name ***Code***

Server Concepts Covered

Since the barcode can send data at any time, we will use an unsolicited message to capture the data. An unsolicited message describes data that the device sends without prompting, or polling, from the Server. This is opposed to the Command/Request (Host) Message, in which the Server requests data from the device.

Implementation

To implement your protocol, do the following:

1. **Create a New Protocol**
   First off, we will need to create the protocol. This protocol will define exactly how the server will interact with both the device (via the server device) and the client (via the items).
   - Open up the Server Configuration Program
   - Select the Protocol Configuration Window.
   - Select **New | Protocol** via the Menu, Toolbar, or Popup Menu.
   - Click on **Protocol Settings** in the tree menu on the left side of the configuration window. Select **Properties** either from the Menu, the Toolbar, or the Popup Menu. You can also double-click on the **Protocol Settings** name.
   - Enter in a name for the protocol. Use **BARCODE1** in this example.
   - Enter in a description and a copyright notice, if desired. Keep all other options at their default.

- Click OK.

2. **Create New Items**
   Now we need to create Items, which is what the server uses to transfer data to and from the client. Items are also used to build data streams (messages) that are sent to the device, and to interpret those data streams that come back to the server from the device.

   - Select **New | Item** via the Menu, Toolbar, or Popup Menu.
   - Type in the item name. For this example, use **Code**.
   - Type in a description, if so desired.
   - Change the data type from **Integer** to **String**.
   - Leave all other options at their default.
   - Click OK.

3. **Create an Unsolicited Message**
   An Unsolicited Message is used because of the very nature of the barcode device. The server never knows when the device will send a barcode across the line, so it must open up the server device and wait until the barcode appears. This means the data is "unsolicited".

   By contrast, "solicited" data is data received from the device once the server sends out a request to the device for that data. All solicited messages are designed in **Command/Request (Host) Messages**.

   - Select **New | Unsolicited Msg** via the Menu, Toolbar, or Popup Menu.
   - Click on the **General** Tab
   - Type in the name of the message. For this example, use **Get_Barcode** (although any name here is appropriate. The name of the message does not matter to the server or to the client, except that there needs to be a name, and it must be unique).
   - Type in a description, if desired.
   - Leave all other options at their default.

- Click on the **Received** Tab.
  In the **Received Message Box**, type in the following exactly (or use the **Sequence Builder** to build the message):

  {Code}{$CR}

  When using the Sequence Builder, click on the **Change Sequence** button in the **Single Assignment** section. "Code" is on the Item Tab, while "$CR" is on the Control Tab.

- Click on OK.

4. **Save and Close the Protocol File**
   Finally, you need to Save the protocol file and close it in order to go to the next steps.

   - Select **File | Save** from the server menu to save the protocol file.
   - Select **File | Close** to close the protocol configuration window.

5. **Create/Change a Server Device**
   The server device connects the physical device to the server. By default, the serial ports COM1 and COM2 are created for you, so if you are going to use these two devices, just skip the first step. Since our test device is on COM3, we'll do the entire process.

   - Select the **Configuration Window / Devices** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
   - Select **New | COM Device** via one of the options.
   - Change the COM port name to **COM3**.
   - Keep all other options at their default.
   - Click OK.

6. **Create/Change a Server Topic**
   A server topic associates a server device with a protocol. The topic name is used by both the server and the client to help identify items. This allows for multiple topics to refer to the same protocol, and thereby the same item name. Only the different topic names keep the data separate.

- Select the **Configuration Window / Topics** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
- Select **New | Topics** via one of the options.
- Enter in a new topic name. For our example, we will use **BARCODE_COM3**. Please note that the name is not case-sensitive. Also, it's a good idea to use a description name for the topic name. In this case, I can look at the topic name and know that the "Barcode" protocol is used on "COM3" for this topic. It's also a good idea to use the names of the devices. For example, if I was using a Krieger RS-90 Barcode Scanner, I might name the topic **KRGR_RS90_COM3**.
- Select **BARCODE1** from the **Protocol** pull-down box.
- Select **COM3** from the **Device** pull-down box.
- Keep all other options at their default.
- Click OK.

That's it! You have completed the setup for a simple Barcode reader.

Of course, if you have an actual device, you will need to test the protocol. To do that, refer to the Trouble Shooting section of this help manual.

8.1.2 Barcode on a Winsock Port

Tutorial: Barcode - Winsock

This tutorial takes you through implementing a simple barcode protocol using a Terminal Server.

Device Specifications

This device is a barcode reader that scans custom–made barcodes and reports that data through a port on a Terminal Server. The terminal server then sends the data back down a TCP/IP line using a TELNET protocol to the host computer. The data returned is rather simple: the actual string of characters printed on the barcode followed by a carriage return.

Technical Specifications

- Server Device: TCP/IP Connection, TELNET protocol
- Data Stream from device: A random number of characters, followed by a carriage return (CR).
- Data Needed by Client: One string of characters, which will be transfer under the name ***Code***

Server Concepts Covered

Since the barcode can send data at any time, we will use an unsolicited message to capture the data. An unsolicited message describes data that the device sends without prompting, or polling, from the Server. This is opposed to the Command/Request (Host) Message, in which the Server requests data from the device.

Implementation

To implement your protocol, do the following:

1. **Create a New Protocol**
   First off, we will need to create the protocol. This protocol will define exactly how the server will interact with both the device (via the server device) and the client (via the items).
   - Open up the Server Configuration Program
   - Select the Protocol Configuration Window.
   - Select **New | Protocol** via the Menu, Toolbar, or Popup Menu.
   - Click on **Protocol Settings** in the tree menu on the left side of the configuration window. Select **Properties** either from the Menu, the Toolbar, or the Popup Menu. You can also double-click on the **Protocol Settings** name.
   - Enter in a name for the protocol. Use **BARCODE1** in this example.
   - Enter in a description and a copyright notice, if desired. Keep all other options at their default.
   - Click OK.

2. **Create New Items**
   Now we need to create Items, which is what the server uses to transfer data to and from the client. Items are also used to build data streams (messages) that are sent to the device, and to interpret those data streams that come back to the server from the device.

   - Select **New | Item** via the Menu, Toolbar, or Popup Menu.
   - Type in the item name. For this example, use **Code**.
   - Type in a description, if so desired.
   - Change the data type from **Integer** to **String**.
   - Leave all other options at their default.
   - Click OK.

3. **Create an Unsolicited Message**
   An Unsolicited Message is used because of the very nature of the barcode device. The server never knows when the device will send a barcode across the line, so it must open up the server device and wait until the barcode appears. This means the data is "unsolicited".

   By contrast, "solicited" data is data received from the device once the server sends out a request to the device for that data. All solicited messages are designed in **Command/Request (Host) Messages**.

   - Select **New | Unsolicited Msg** via the Menu, Toolbar, or Popup Menu.
   - Click on the **General** Tab
   - Type in the name of the message. For this example, use **Get_Barcode** (although any name here is appropriate. The name of the message does not matter to the server or to the client, except that there needs to be a name, and it must be unique).
   - Type in a description, if desired.
   - Leave all other options at their default.

- Click on the **Received** Tab.
  In the **Received Message Box**, type in the following exactly (or use the **Sequence Builder** to build the message):

  {Code}{$CR}

  When using the Sequence Builder, click on the **Change Sequence** button in the **Single Assignment** section. "Code" is on the Item Tab, while "$CR" is on the Control Tab.

- Click on OK.

4. Save and Close the Protocol File
Finally, you need to Save the protocol file and close it in order to go to the next steps.

- Select **File | Save** from the server menu to save the protocol file.

- Select **File | Close** to close the protocol configuration window.

5. Create/Change a Server Device
The server device connects the physical device to the server. Our test device is on an Ethernet port.

- Select the **Configuration Window / Devices** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.

- Select **New | Ethernet Device** via one of the options.

- For **Type**, select **TELNET**.

- In the **Address** box, type in the numeric or name address for the device. For example, you can use an address like **192.168.0.1**, or a name like **www.yourdevice.com**. See you system's administrator for directions.

- In the **Port**, type in the Port number in which your device is connected. This is normally setup by the administrator of the terminal server. For example, many terminal servers begin port addressing at 2000, so if your device was in Port #1, type in **2000** for the port number.

- Keep all other options at their default.

- Click OK.

6. **Create/Change a Server Topic**
   A server topic associates a server device with a protocol. The topic name is used by both the server and the client to help identify items. This allows for multiple topics to refer to the same protocol, and thereby the same item name. Only the different topic names keep the data separate.

   - Select the **Configuration Window / Topics** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
   - Select **New | Topics** via one of the options.
   - Enter in a new topic name. For our example, we will use **BARCODE_TS_2000**. Please note that the name is not case-sensitive. Also, it's a good idea to use a description name for the topic name. In this case, I can look at the topic name and know that the "Barcode" protocol is used on a terminal server with port "2000" for this topic. It's also a good idea to use the names of the devices. For example, if I was using a Krieger RS-90 Barcode Scanner, I might name the topic **KRGR_RS90_TS2000**.
   - Select **BARCODE1** from the **Protocol** pull-down box.
   - Select your TELNET port in the **Device** pull-down box. Using the example above, this will be **TELNET_192_168_0_1:2000**
   - Keep all other options at their default.
   - Click OK.

That's it! You have completed the setup for a simple Barcode reader.

Of course, if you have an actual device, you will need to test the protocol. To do that, refer to the Trouble Shooting section of this help manual.

8.2 Weigh Scales

Tutorial: Weigh Scales

Since Weight Scales produce data much like Barcode Scanner-type devices, please refer to the Barcode Tutorials for additional information.

Barcode Tutorials

8.3 Marque Instruments

Tutorial: Marquees

These Marquee Tutorials take you through the steps in building protocols which handle devices like Marques on a factory floor. These are useful for clients that need to control the text that appears on Marquee-type systems. The client will be responsible for the message that appears on the device, while the server will actually send the message to the device.

- Marquee - COM - Learn how to setup a protocol that talks to a Marquee device on a Serial port.
- Marquee - Ethernet - Learn how to setup a protocol that talks to a Marquee device on an Ethernet port.

8.3.1 Marquee on a Serial Port

Tutorial: Marquee - COM

This tutorial takes you through implementing a protocol that sends messages to a Marquee-type device on a Serial port.

Device Specifications

This device takes a string from the server and outputs it to the Marquee. The client will activate the message by changing the value of a tag on the client's screen, which in turn loads the contents of the tag into the server, which finally sends the message to the device. The data sent is a simple ASCII string.

Technical Specifications

- Server Device: COM3
- Data Stream to device: A Start-of-Text character, a command code, the actual message, an End-of-Text character, and a Longitudinal Redundancy Check (LRC) on all characters except the Start-Of-Text character.
- Data Needed by Client: None.

- Data Needed by Server: A change of the condition of the item ***Message*.**

Server Concepts Covered

In this tutorial, we will use the "Write Command/Request (Host) Message", which is a Command/Request (Host) Message set up in such a way in that it will only be sent out if a value of an item in the **Request** part of the message changes from the client.

Also, we introduce the Error Detection Code **LRC8**, which does an 8-bit Exclusive-OR error check on the data within a certain range.

Implementation

In this protocol example, we will tell the server to send out the message contained in an item called MESSAGE to the device. To implement your protocol, do the following:

1. **Create a New Protocol**
   First off, we will need to create the protocol. This protocol will define exactly how the server will interact with both the device (via the server device) and the client (via the items).

   - Open up the Server Configuration Program
   - Select the Protocol Configuration Window.
   - Select **New | Protocol** via the Menu, Toolbar, or Popup Menu.
   - Click on **Protocol Settings** in the tree menu on the left side of the configuration window. Select **Properties** either from the Menu, the Toolbar, or the Popup Menu. You can also double-click on the **Protocol Settings** name.
   - Enter in a name for the protocol. Use **MARQUEE** in this example.
   - Enter in a description and a copyright notice, if desired. Keep all other options at their default.
   - Click OK.

2. **Create New Items**
   Now we need to create Items, which is what the server uses to transfer data to and from the client. Items are also used to build

data streams (messages) that are sent to the device, and to interpret those data streams that come back to the server from the device.

- Select **New | Item** via the Menu, Toolbar, or Popup Menu.
- Type in the item name. For this example, use **Message**.
- Type in a description, if so desired.
- Change the data type from **Integer** to **String**.
- Leave all other options at their default.
- Click OK.

3. **Create a Command/Request (Host) Message**
   An Command/Request (Host) Message is used because of the ability of the server to send out a Command/Request (Host) Message whenever an item located in the Request part of the Command/Request (Host) Message changes. This is called a **Write Command/Request (Host) Message.**

   - Select **New | Command/Request (Host) Message** via the Menu, Toolbar, or Popup Menu.
   - Click on the **General** Tab
   - Type in the name of the message. For this example, use **Send_Message** (although any name here is appropriate. The name of the message does not matter to the server or to the client, except that there needs to be a name, and it must be unique).
   - Type in a description, if desired.
   - Change the **Type** from **Read** to **Write**.
   - Leave all other options at their default.
   - Click on the **Request** Tab.
     The Request tab is the message that is sent to the device when an item located in this sections changes. In the **Received Message Box**, type in the following exactly (or use the **Sequence Builder** to build the message):

     {$STX}[P{Message}{$ETX}]{$LRC8:1UB}

When using the Sequence Builder, click on the **Change Sequence** button in the **Single Assignment** section to select the Start-of-Text Character ($STX, located on the Control Tab), the MESSAGE item (located on the Item Tab), the End-of-Text Character ($ETX< located on the Control Tab), and the Error Detection Code ($LRC8, located on the Others Tab).

For all other characters, type them in as is.

The Error Detection Code LRC8: This tutorial uses an Error Detection Code. Please see the end of this article for a complete explanation of what was done in this message. In this example, we want to output the error detection code as a one-byte Binary Unsigned number (giving us a number between 0 and 255). To do this, select **LRC8** from the Others Tab, and click OK. Then, select the **Formatting** tab and select **Integer - Unsigned** from the Format Style section, then select the **Binary** checkbox. Finally, type in a width of **1** in the Format Width box, and click OK.

- Click on OK.

4. **Save and Close the Protocol File**
   Finally, you need to Save the protocol file and close it in order to go to the next steps.

   - Select **File | Save** from the server menu to save the protocol file.
   - Select **File | Close** to close the protocol configuration window.

5. **Create/Change a Server Device**
   The server device connects the physical device to the server. By default, the serial ports COM1 and COM2 are created for you, so if you are going to use these two devices, just skip the first step. Since our test device is on COM3, we'll do the entire process.

   - Select the **Configuration Window / Devices** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
   - Select **New | COM Device** via one of the options.
   - Change the COM port name to **COM3**.
   - Keep all other options at their default.

- Click OK.

6. **Create/Change a Server Topic**
   A server topic associates a server device with a protocol. The topic name is used by both the server and the client to help identify items. This allows for multiple topics to refer to the same protocol, and thereby the same item name. Only the different topic names keep the data separate.

   - Select the **Configuration Window / Topics** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
   - Select **New | Topics** via one of the options.
   - Enter in a new topic name. For our example, we will use **MARQUEE_COM3**. Please note that the name is not case-sensitive. Also, it's a good idea to use a description name for the topic name. In this case, I can look at the topic name and know that the "Marquee" protocol is used on "COM3" for this topic. It's also a good idea to use the names of the devices. For example, if I was using a Krieger RS-90 Marquee, I might name the topic **KRGR_RS90_COM3**.
   - Select **MARQUEE** from the **Protocol** pull-down box.
   - Select **COM3** from the **Device** pull-down box.
   - Keep all other options at their default.
   - Click OK.

That's it! You have completed the setup for a simple protocol that accepts a push button change from the client.

On the client side, create a tag associated with the item **Message**. Then, whenever the tag gets changed to some text (any text), the Command/Request (Host) Message **Send_Message** will be sent to the device. This is because the item **Message** appears in the Request part of the Command/Request (Host) Message, and the Command/Request (Host) Message is a **Write** message.

How to Use Error Detection Codes: The server has the ability to calculate error detection codes on the data in a message. To use one of these codes, there are three steps involved:

- **Select the correct code**. Look over the descriptions in the Help File to determine the correct code to use for this message.
- **Select the correct format**. Next, format the output using the **Formatting** and **Advanced Formatting** tabs on the Sequence Builder.
- **Select the correct characters**. Finally, you need to tell the server what characters are included in the error detection code calculation. To do this, you will insert square brackets ("[" and "]") around the characters in the message that must be used in the calculation. For example, take a look at the example above:

  {$STX}[P{Message}{$ETX}]{$LRC8:1UB}

  In this sample notice the right square bracket ("[") just after the the Start-of-Text character ("{$STX}"), and the left square bracket ("]") just after the End-of-Text character ("{$ETX}"). The server will use only those characters between the brackets, so in this case, all characters except the Start-of-Text character is used in the calculation.

8.3.2 Marquee on a Winsock Port

Tutorial: Marquee - Ethernet

This tutorial takes you through implementing a protocol that sends messages to a Marquee-type device on a Ethernet port.

Device Specifications

This device takes a string from the server and outputs it to the Marquee. The client will activate the message by changing the value of a tag on the client's screen, which in turn loads the contents of the tag into the server, which finally sends the message to the device. The data sent is a simple ASCII string.

Technical Specifications

- Server Device: Ethernet port, using TCP.

- Data Stream to device: A Start-of-Text character, a command code, the actual message, an End-of-Text character, and a Longitudinal Redundancy Check (LRC) on all characters except the Start-Of-Text character.
- Data Needed by Client: None.
- Data Needed by Server: A change of the condition of the item ***Message*.**

Server Concepts Covered

In this tutorial, we will use the "Write Command/Request (Host) Message", which is a Command/Request (Host) Message set up in such a way in that it will only be sent out if a value of an item in the **Request** part of the message changes from the client.

Also, we introduce the Error Detection Code **LRC8**, which does an 8-bit Exclusive-OR error check on the data within a certain range.

Implementation

In this protocol example, we will tell the server to send out the message contained in an item called MESSAGE to the device. To implement your protocol, do the following:

1. **Create a New Protocol**
   First off, we will need to create the protocol. This protocol will define exactly how the server will interact with both the device (via the server device) and the client (via the items).
   - Open up the Server Configuration Program
   - Select the Protocol Configuration Window.
   - Select **New | Protocol** via the Menu, Toolbar, or Popup Menu.
   - Click on **Protocol Settings** in the tree menu on the left side of the configuration window. Select **Properties** either from the Menu, the Toolbar, or the Popup Menu. You can also double-click on the **Protocol Settings** name.
   - Enter in a name for the protocol. Use **MARQUEE** in this example.

- Enter in a description and a copyright notice, if desired. Keep all other options at their default.
- Click OK.

2. **Create New Items**
   Now we need to create Items, which is what the server uses to transfer data to and from the client. Items are also used to build data streams (messages) that are sent to the device, and to interpret those data streams that come back to the server from the device.

   - Select **New | Item** via the Menu, Toolbar, or Popup Menu.
   - Type in the item name. For this example, use **Message**.
   - Type in a description, if so desired.
   - Change the data type from **Integer** to **String**.
   - Leave all other options at their default.
   - Click OK.

3. **Create a Command/Request (Host) Message**
   An Command/Request (Host) Message is used because of the ability of the server to send out a Command/Request (Host) Message whenever an item located in the Request part of the Command/Request (Host) Message changes. This is called a **Write Command/Request (Host) Message.**

   - Select **New | Command/Request (Host) Message** via the Menu, Toolbar, or Popup Menu.
   - Click on the **General** Tab
   - Type in the name of the message. For this example, use **Send_Message** (although any name here is appropriate. The name of the message does not matter to the server or to the client, except that there needs to be a name, and it must be unique).
   - Type in a description, if desired.
   - Change the **Type** from **Read** to **Write**.
   - Leave all other options at their default.

- Click on the **Request** Tab.
  The Request tab is the message that is sent to the device when an item located in this sections changes. In the **Received Message Box**, type in the following exactly (or use the **Sequence Builder** to build the message):

  {$STX}[P{Message}{$ETX}]{$LRC8:1UB}

  When using the Sequence Builder, click on the **Change Sequence** button in the **Single Assignment** section to select the Start-of-Text Character ($STX, located on the Control Tab), the MESSAGE item (located on the Item Tab), the End-of-Text Character ($ETX< located on the Control Tab), and the Error Detection Code ($LRC8, located on the Others Tab).

  For all other characters, type them in as is.

  The Error Detection Code LRC8: This tutorial uses an Error Detection Code. Please see the end of this article for a complete explanation of what was done in this message. In this example, we want to output the error detection code as a one-byte Binary Unsigned number (giving us a number between 0 and 255). To do this, select **LRC8** from the Others Tab, and click OK. Then, select the **Formatting** tab and select **Integer - Unsigned** from the Format Style section, then select the **Binary** checkbox. Finally, type in a width of **1** in the Format Width box, and click OK.

- Click on OK.

4. **Save and Close the Protocol File**
   Finally, you need to Save the protocol file and close it in order to go to the next steps.

   - Select **File | Save** from the server menu to save the protocol file.

   - Select **File | Close** to close the protocol configuration window.

5. **Create/Change a Server Device**
   The server device connects the physical device to the server. Our test device is on an Ethernet port.

   - Select the **Configuration Window / Devices** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.

- Select **New | Ethernet Device** via one of the options.
- For **Type**, select **TCP**.
- In the **Address** box, type in the numeric or name address for the device. For example, you can use an address like **192.168.0.1**, or a name like **www.yourdevice.com**. See you system's administrator for directions.
- In the **Port**, type in the Port number in which your device is connected. This is normally setup by the administrator of the terminal server. In this example, we will use 2000, so type in **2000** for the port number.
- Keep all other options at their default.
- Click OK.

6. **Create/Change a Server Topic**
   A server topic associates a server device with a protocol. The topic name is used by both the server and the client to help identify items. This allows for multiple topics to refer to the same protocol, and thereby the same item name. Only the different topic names keep the data separate.

   - Select the **Configuration Window / Topics** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
   - Select **New | Topics** via one of the options.
   - Enter in a new topic name. For our example, we will use **MARQUEE_2000**. Please note that the name is not case-sensitive. Also, it's a good idea to use a description name for the topic name. In this case, I can look at the topic name and know that the "Marquee" protocol is used on port "2000" for this topic. It's also a good idea to use the names of the devices. For example, if I was using a Krieger RS-90 Marquee, I might name the topic **KRGR_RS90_2000**.
   - Select **MARQUEE** from the **Protocol** pull-down box.
   - Select **TCP_192_168_0_1_2000** from the **Device** pull-down box.
   - Keep all other options at their default.
   - Click OK.

That's it! You have completed the setup for a simple protocol that accepts a push button change from the client.

On the client side, create a tag associated with the item **Message**. Then, whenever the tag gets changed to some text (any text), the Command/Request (Host) Message **Send_Message** will be sent to the device. This is because the item **Message** appears in the Request part of the Command/Request (Host) Message, and the Command/Request (Host) Message is a **Write** message.

How to Use Error Detection Codes: The server has the ability to calculate error detection codes on the data in a message. To use one of these codes, there are three steps involved:

- **Select the correct code**. Look over the descriptions in the Help File to determine the correct code to use for this message.
- **Select the correct format**. Next, format the output using the **Formatting** and **Advanced Formatting** tabs on the Sequence Builder.
- **Select the correct characters**. Finally, you need to tell the server what characters are included in the error detection code calculation. To do this, you will insert square brackets ("[" and "]") around the characters in the message that must be used in the calculation. For example, take a look at the example above:

  {$STX}[P{Message}{$ETX}]{$LRC8:1UB}

  In this sample notice the right square bracket ("[") just after the the Start-of-Text character ("{$STX}"), and the left square bracket ("]") just after the End-of-Text character ("{$ETX}"). The server will use only those characters between the brackets, so in this case, all characters except the Start-of-Text character is used in the calculation.

8.4 Meaurement Instruments

Tutorial: Measurement Devices

These Measurement Tutorials take you through the steps in building protocols which handle devices that need to be polled for specific information, such as boilers, micrometers, and the like. These are useful for clients that need to monitor system information on various devices.

- Measurement - COM - Learn how to setup a protocol that talks to a Measurement device on a Serial port.

8.4.1 Measurement on a Serial Port

Tutorial: Measurement - COM

This tutorial takes you through implementing a protocol that reads in the Present Value from a monitoring device on a Serial port.

Device Specifications

This device measures the flow pressure in a duct. The server will send out a message to the device once every second that tells the device to send the measurement back to the server. The server then sends the data back to the client.

Technical Specifications

- Server Device: COM3
- Data Stream to device: A Start-of-Text character, a command code, a question mark, an End-of-Text character, and a Longitudinal Redundancy Check (LRC) on all characters except the Start-Of-Text character.
- Data Stream to Server: An Acknowledgement character, a Start-of-Text character, the command code, an equals sign, the actual data value itself, an End-of-Text character, and a Longitudinal Redundancy Check (LRC) on all characters except the Start-Of-Text character.
- Data Needed by Client: The Present Value of the Device.
- Data Needed by Server: A change of the condition of the item ***PV*.**

Server Concepts Covered

In this tutorial, we will use the "Read Command/Request (Host) Message", which is a Command/Request (Host) Message set up in such a way in that it will be sent out on a regular basis (known as the Update Interval).

A Read Command/Request (Host) Message is triggered when a client requests the value of an item which appears in the **Response** part of the Command/Request (Host) Message. This Command/Request (Host) Message is sent out once every **Update Interval**, a value set when the server topic is created (see below). The default operation of a Read Message can be over-ridden by the **Chains and Trigger's Trigger point**.

Also, we introduce the Error Detection Code **LRC8**, which does an 8-bit Exclusive-OR error check on the data within a certain range.

Implementation

In this protocol example, we will tell the server to send out the message contained in an item called MESSAGE to the device. To implement your protocol, do the following:

1. **Create a New Protocol**
   First off, we will need to create the protocol. This protocol will define exactly how the server will interact with both the device (via the server device) and the client (via the items).

   - Open up the Server Configuration Program
   - Select the Protocol Configuration Window.
   - Select **New | Protocol** via the Menu, Toolbar, or Popup Menu.
   - Click on **Protocol Settings** in the tree menu on the left side of the configuration window. Select **Properties** either from the Menu, the Toolbar, or the Popup Menu. You can also double-click on the **Protocol Settings** name.
   - Enter in a name for the protocol. Use **FLOWMETER** in this example.
   - Enter in a description and a copyright notice, if desired. Keep all other options at their default.
   - Click OK.

2. **Create New Items**
   Now we need to create Items, which is what the server uses to transfer data to and from the client. Items are also used to build data streams (messages) that are sent to the device, and to interpret those data streams that come back to the server from the device.

   - Select **New | Item** via the Menu, Toolbar, or Popup Menu.
   - Type in the item name. For this example, use **PV**.
   - Type in a description, if so desired.
   - Change the data type from **Integer** to **Real**.
   - Leave all other options at their default.
   - Click OK.

3. **Create a Command/Request (Host) Message**
   An Command/Request (Host) Message is used because of the ability of the server to send out a Command/Request (Host) Message whenever an item located in the Response part of the Command/Request (Host) Message is requested by the client. This is called a **Read Command/Request (Host) Message.**

   - Select **New | Command/Request (Host) Message** via the Menu, Toolbar, or Popup Menu.
   - Click on the **General** Tab
   - Type in the name of the message. For this example, use **Get_PV** (although any name here is appropriate. The name of the message does not matter to the server or to the client, except that there needs to be a name, and it must be unique).
   - Type in a description, if desired.
   - Insure that the **Type** is **Read**.
   - Leave all other options at their default.
   - Click on the **Request** Tab.
     The Request tab is the message that is sent to the device whenever the server needs to update the value of an item that appears in the **Response** part of the Command/Request

(Host) Message. In the **Request Message Box**, type in the following exactly (or use the **Sequence Builder** to build the message):

{$STX}[PV?{$ETX}]{$LRC8:1UB}

When using the Sequence Builder, click on the **Change Sequence** button in the **Single Assignment** section to select the Start-of-Text Character ($STX, located on the Control Tab), the End-of-Text Character ($ETX, located on the Control Tab), and the Error Detection Code ($LRC8, located on the Others Tab).

For all other characters, type them in as is.

The Error Detection Code LRC8: This tutorial uses an Error Detection Code. Please see the end of this article for a complete explanation of what was done in this message. In this example, we want to output the error detection code as a one-byte Binary Unsigned number (giving us a number between 0 and 255). To do this, select **LRC8** from the Others Tab, and click OK. Then, select the **Formatting** tab and select **Integer - Unsigned** from the Format Style section, then select the **Binary** checkbox. Finally, type in a width of **1** in the Format Width box, and click OK.

- Click on the **Response** Tab.
  The Response part of the Command/Request (Host) Message describes to the server exactly how the data will look like from the device. Within the Response, you will normally find the item that the client is requesting. In the **Response Message Box**, type in the following exactly (or use the **Sequence Builder** to build the message):

  {$ACK}{$STX}[PV={PV}{$ETX}]{$LRC8:1UB}

  When using the Sequence Builder, click on the **Change Sequence** button in the **Single Assignment** section to select the Acknowledgment Character ($ACK, located on the Control Tab), the Start-of-Text Character ($STX, located on the Control Tab), the item PV (located on the Item Tab), the End-of-Text Character ($ETX, located on the Control Tab), and the Error Detection Code ($LRC8, located on the Others Tab).

  For all other characters, type them in as is.

- Click on OK.

4. **Save and Close the Protocol File**
   Finally, you need to Save the protocol file and close it in order to go to the next steps.

   - Select **File | Save** from the server menu to save the protocol file.
   - Select **File | Close** to close the protocol configuration window.

5. **Create/Change a Server Device**
   The server device connects the physical device to the server. By default, the serial ports COM1 and COM2 are created for you, so if you are going to use these two devices, just skip the first step. Since our test device is on COM3, we'll do the entire process.

   - Select the **Configuration Window / Devices** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
   - Select **New | COM Device** via one of the options.
   - Change the COM port name to **COM3**.
   - Keep all other options at their default.
   - Click OK.

6. **Create/Change a Server Topic**
   A server topic associates a server device with a protocol. The topic name is used by both the server and the client to help identify items. This allows for multiple topics to refer to the same protocol, and thereby the same item name. Only the different topic names keep the data separate.

   - Select the **Configuration Window / Topics** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
   - Select **New | Topics** via one of the options.
   - Enter in a new topic name. For our example, we will use **FLOWMETER_COM3**. Please note that the name is not case-sensitive. Also, it's a good idea to use a description name for the topic name. In this case, I can look at the topic name and know that the "Marquee" protocol is used on "COM3" for this topic. It's also a good idea to use the names of the devices.

For example, if I was using a Krieger RS-90 Flow Meter, I might name the topic **KRGR_RS90_COM3**.

- Select **FLOWMETER** from the **Protocol** pull-down box.
- Select **COM3** from the **Device** pull-down box.
- Make sure that the **Update Interval** is set to **1000**. The Update Interval tells the server how long to wait between attempts to send out a Command/Request (Host) Message. For example, setting the value to 1000 means that the server will make it's best attempt to get all data from the device once every 1000 milliseconds, or once per second.
- Keep all other options at their default.
- Click OK.

That's it! You have completed the setup for a simple protocol that reads the present value of a flow meter and returns that information to the client.

On the client side, create a tag associated with the item **PV**. Then, once per second (1000 milliseconds), the Command/Request (Host) Message **Get_PV** will be sent to the device. This is because the item **PV** appears in the Response part of the Command/Request (Host) Message, and the Command/Request (Host) Message is a **Read** message.

How to Use Error Detection Codes: The server has the ability to calculate error detection codes on the data in a message. To use one of these codes, there are three steps involved:

- **Select the correct code**. Look over the descriptions in the Help File to determine the correct code to use for this message.
- **Select the correct format**. Next, format the output using the **Formatting** and **Advanced Formatting** tabs on the Sequence Builder.
- **Select the correct characters**. Finally, you need to tell the server what characters are included in the error detection code calculation. To do this, you will insert square brackets ("[" and "]") around the

characters in the message that must be used in the calculation. For example, take a look at the example above:

{$STX}[PV?{$ETX}]{$LRC8:1UB}

In this sample notice the right square bracket ("[") just after the the Start-of-Text character ("{$STX}"), and the left square bracket ("]") just after the End-of-Text character ("{$ETX}"). The server will use only those characters between the brackets, so in this case, all characters except the Start-of-Text character is used in the calculation.

8.5 PLC

Tutorial: PLC

This PLC Tutorial take you through the steps in building protocols which handle the addressing schemes common to many PLCs.

- PLC - COM - Learn how to setup a protocol that talks to a PLC device with multiple connected to a Modem on a Serial port.

8.5.1 PLC on a Serial Port

Tutorial: PLC - COM

This tutorial takes you through implementing a protocol that reads in addresses from a PLC connected to a on a Serial port.

Device Specifications

This device returns values in various registers in a PLC. The server will communicate with multiple devices, and then use a Command/Request (Host) Message to retrieve values from individual addresses.

Technical Specifications

- Server Device: COM3, using an RS-485 connection.
- Data Stream to device: A Start-of-Text character, a device address, a command code, a point address, a question mark, an End-of-Text character, and a Longitudinal Redundancy Check (LRC) on all characters except the Start-Of-Text character.

- Data Stream to Server: An Acknowledgement character, a Start-of-Text character, the device address, the command code, the point address, an equals sign, the actual data value itself, an End-of-Text character, and a Longitudinal Redundancy Check (LRC) on all characters except the Start-Of-Text character.
- Data Needed by Client: The Present Value of the Device.
- Data Needed by Server: A change of the condition of the item ***DATAxx***, where "xx" is an address number between 0 and 99.

Server Concepts Covered

In this tutorial, we will use the a combination of the Topic Variable and Register Number.

The Topic Variable will be used by the server to differentiate between different device addresses. It is set up within the protocol, and the values are set when the topic is created.

The Register Number will be used to address the different points in the PLC. Just like the Topic Variable, these are defined in the protocol, but uses a value in the data stream as its current value in the message.

Also, we introduce the Error Detection Code **LRC8**, which does an 8-bit Exclusive-OR error check on the data within a certain range.

Implementation

In this protocol example, the server will send out a message that will contain the contents of the **Topic Variable** and **Register Number.**

1. **Create a New Protocol**
   First off, we will need to create the protocol. This protocol will define exactly how the server will interact with both the device (via the server device) and the client (via the items).

   - Open up the Server Configuration Program
   - Select the Protocol Configuration Window.
   - Select **New | Protocol** via the Menu, Toolbar, or Popup Menu.
   - Click on **Protocol Settings** in the tree menu on the left side of the configuration window. Select **Properties** either from the Menu, the Toolbar, or the Popup Menu. You can also double-click on the **Protocol Settings** name.

- Enter in a name for the protocol. Use **PLC** in this example.
- Enter in a description and a copyright notice, if desired. Keep all other options at their default.
- Click OK.

2. **Create New Registers**
   A register is a special field that, when appended to an item, will use a number added to the physical name of that item as the number for the register.

   In this example, we will have a register called **Addr**, which will have a range of numbers between 0 and 99. This register will then be appended to the item **Data** (see below). When the client makes a request for the item **Data**, it will need to append a number between 0 and 99 to the name, so the client will actually request items like **Data1, Data2, Data45,** etc. The server will then strip off the word **Data** and use the number at the end for the value of **Addr** throughout the mesage:

   - Select **New | Register** via the Menu, Toolbar, or Popup Menu.
   - Type in the item name. For this example, use **Addr**.
   - Type in a description, if so desired.
   - Enter a minimum range of **0** and a maximum range of **99.**
   - Click OK.

3. **Create New Topic Variables**
   Topic Variables are used primarily as addressing schemes for devices. Since you have one topic per device, you can use the Topic Variable (which is defined here but has a value assigned to it when the Topic is created) to differentiate betwwen different devices. This allows you to use the same protocol for multiple devices (even into the hundreds):

   - Select **New | Topic Variable** via the Menu, Toolbar, or Popup Menu.
   - Type in the item name. For this example, use **DevAddr**.
   - Type in a description, if so desired.

- Click OK.

4. **Create New Items**
   Now we need to create Items, which is what the server uses to transfer data to and from the client. Items are also used to build data streams (messages) that are sent to the device, and to interpret those data streams that come back to the server from the device. For each of the items listed below, do the following:

   - Select **New | Item** via the Menu, Toolbar, or Popup Menu.
   - Type in the item name. For this example, use **DATA{Addr}**. We use the **{Addr}** at the end so that when the client requests the point **DATA13** (or any number from 0 to 99 after DATA), the server will use the number **13** as the value of the register **Addr** for the duration of the current message.
   - Type in a description, if so desired.
   - Change the data type to **Integer**.
   - Leave all other options at their default.
   - Click OK.

5. **Create Command/Request (Host) Messages**
   An Command/Request (Host) Message is used because of the ability of the server to send out a Command/Request (Host) Message whenever an item located in the Response part of the Command/Request (Host) Message is requested by the client. This is called a **Read Command/Request (Host) Message.**

   - Select **New | Command/Request (Host) Message** via the Menu, Toolbar, or Popup Menu.
   - Click on the **General** Tab
   - Type in the name of the message. For this example, use **Get_Data** (although any name here is appropriate. The name of the message does not matter to the server or to the client, except that there needs to be a name, and it must be unique).
   - Type in a description, if desired.
   - Insure that the **Type** is **Read**.

- Leave all other options at their default.
- Click on the **Request** Tab.
  The Request tab is the message that is sent to the device whenever the server needs to update the value of an item that appears in the **Response** part of the Command/Request (Host) Message. In the **Request Message Box**, type in the following exactly (or use the **Sequence Builder** to build the message):

  {$STX}[{DevAddr:2I}PV{Addr:2I}?{$ETX}]{$LRC8:1UB}

  When using the Sequence Builder, click on the **Change Sequence** button in the **Single Assignment** section to select the Start-of-Text Character ($STX, located on the Control Tab), the End-of-Text Character ($ETX, located on the Control Tab), the Register Number (ADDR, located on the Register Numbers tab), the Error Detection Code ($LRC8, located on the Others Tab) and Topic Variable (DEVADDR, located on the Others Tab).

  For all other characters, type them in as is.

  The Error Detection Code LRC8: This tutorial uses an Error Detection Code. Please see the end of this article for a complete explanation of what was done in this message. In this example, we want to output the error detection code as a one-byte Binary Unsigned number (giving us a number between 0 and 255). To do this, select **LRC8** from the Others Tab, and click OK. Then, select the **Formatting** tab and select **Integer - Unsigned** from the Format Style section, then select the **Binary** checkbox. Finally, type in a width of **1** in the Format Width box, and click OK.
- Click on the **Response** Tab.
  The Response part of the Command/Request (Host) Message describes to the server exactly how the data will look like from the device. Within the Response, you will normally find the item that the client is requesting. In the **Response Message Box**, type in the following exactly (or use the **Sequence Builder** to build the message):

  {$ACK}{$STX}[{DevAddr:2I}PV{Addr:2I}={Data{Addr}}
  {$ETX}]{$LRC8:1UB}

When using the Sequence Builder, click on the **Change Sequence** button in the **Single Assignment** section to select the Acknowledgment Character ($ACK, located on the Control Tab), the Start-of-Text Character ($STX, located on the Control Tab), the item PV (located on the Item Tab), the End-of-Text Character ($ETX, located on the Control Tab), the Register Number (ADDR, located on the Register Numbers tab), the Topic Variable (DEVADDR, located on the Others Tab), and the Error Detection Code ($LRC8, located on the Others Tab).

For all other characters, type them in as is.

- Click on OK.

6. **Save and Close the Protocol File**
   Finally, you need to Save the protocol file and close it in order to go to the next steps.

   - Select **File | Save** from the server menu to save the protocol file.
   - Select **File | Close** to close the protocol configuration window.

7. **Create/Change a Server Device**
   The server device connects the physical device to the server. By default, the serial ports COM1 and COM2 are created for you, so if you are going to use these two devices, just skip the first step. Since our test device is on COM3, we'll do the entire process.

   - Select the **Configuration Window / Devices** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
   - Select **New | COM Device** via one of the options.
   - Change the COM port name to **COM3**.
   - Keep all other options at their default.
   - Click OK.

8. **Create/Change a Server Topic**
   A server topic associates a server device with a protocol. The topic name is used by both the server and the client to help identify items. This allows for multiple topics to refer to the same protocol,

and thereby the same item name. Only the different topic names keep the data separate.

- Select the **Configuration Window / Topics** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
- Select **New | Topics** via one of the options.
- Enter in a new topic name. For our example, we will use **PLC_COM3**. Please note that the name is not case-sensitive. Also, it's a good idea to use a description name for the topic name. In this case, I can look at the topic name and know that the "Marquee" protocol is used on "COM3" for this topic. It's also a good idea to use the names of the devices. For example, if I was using a Krieger RS-90 PLC, I might name the topic **KRGR_RS90_COM3**.
- Select **PLC** from the **Protocol** pull-down box.
- Select **COM3** from the **Device** pull-down box.
- Click on the **Variables** tab.
- Click to the right of the topic variable name **DevAddr** and enter in the device number. For this example, use **1**.
- Keep all other options at their default.
- Click OK.

That's it! You have completed the setup for a simple protocol that reads in multiple data points for multiple devices on a PLC.

On the client side, create a tag associated with the item **DATAxx**, where "xx" is a number between 0 and 99 (do not use a leading zero). Once you activate the client, the server will place the value of the topic variable **DevAddr** and **Addr** into the message stream, thereby retrieving the correct point number for the correct device.

How to communicate with more than one device: Simple create addition Topics referencing the same device and protocol. Just make sure that each one has a different address value for the topic variable **DevAddr**.

How to get values for more than one point: Have the client create different tag names. The important thing here is that you place a number between 0 and 99 at the end of each tag. In this example, you can create 100 different tags names in your client, each one called **DATA** with a number between 0 and 99 appended to the end.

How to Use Error Detection Codes: The server has the ability to calculate error detection codes on the data in a message. To use one of these codes, there are three steps involved:

- **Select the correct code**. Look over the descriptions in the Help File to determine the correct code to use for this message.
- **Select the correct format**. Next, format the output using the **Formatting** and **Advanced Formatting** tabs on the Sequence Builder.
- **Select the correct characters**. Finally, you need to tell the server what characters are included in the error detection code calculation. To do this, you will insert square brackets ("[" and "]") around the characters in the message that must be used in the calculation. For example, take a look at the example above:

  {$STX}[PV?{$ETX}]{$LRC8:1UB}

  In this sample notice the right square bracket ("[") just after the the Start-of-Text character ("{$STX}"), and the left square bracket ("]") just after the End-of-Text character ("{$ETX}"). The server will use only those characters between the brackets, so in this case, all characters except the Start-of-Text character is used in the calculation.

8.6 Event Responder

Tutorial: Events

These Event Tutorials take you through the steps in building protocols which handle events like Alarms and push-buttons. These are useful for clients that must respond to alarms from the device or send specific information to the device based upon an action by the operator.

- Alarm Tutorial - Learn how to setup a protocol that responds to alarms from the device.
- Push Button Tutorial - A tutorial that takes you through the steps of writing a protocol that responds to "push buttons" from the client.

8.6.1 Alarm

Tutorial: Events - Designing for Alarms

This tutorial takes you through implementing a simple protocol that responds to an alarm status from the device.

Device Specifications

This device is a boiler that sends out an alarm once a pressure reading goes above a preset setpoint. The data returned is in ASCII format, with an "AL" prefix, and alarm number, and a carriage return.

Technical Specifications

- Server Device: COM3
- Data Stream from device: A prefix ("AL"), an alarm number, ending with a carriage return (CR).
- Data Needed by Client: The alarm number, which will be transferred under the name ***Alarmcode***

Server Concepts Covered

Since the boiler can send data at any time, we will use an unsolicited message to capture the data. An unsolicited message describes data that the device sends without prompting, or polling, from the Server. This is opposed to the Command/Request (Host) Message, in which the Server requests data from the device.

Implementation

To implement your protocol, do the following:

1. **Create a New Protocol**
   First off, we will need to create the protocol. This protocol will define exactly how the server will interact with both the device (via the server device) and the client (via the items).

    - Open up the Server Configuration Program

- Select the Protocol Configuration Window.
- Select **New | Protocol** via the Menu, Toolbar, or Popup Menu.
- Click on **Protocol Settings** in the tree menu on the left side of the configuration window. Select **Properties** either from the Menu, the Toolbar, or the Popup Menu. You can also double-click on the **Protocol Settings** name.
- Enter in a name for the protocol. Use **BOILER** in this example.
- Enter in a description and a copyright notice, if desired. Keep all other options at their default.
- Click OK.

2. **Create New Items**
   Now we need to create Items, which is what the server uses to transfer data to and from the client. Items are also used to build data streams (messages) that are sent to the device, and to interpret those data streams that come back to the server from the device.

   - Select **New | Item** via the Menu, Toolbar, or Popup Menu.
   - Type in the item name. For this example, use **Alarmcode**.
   - Type in a description, if so desired.
   - Insure that the data type is **Integer**.
   - Leave all other options at their default.
   - Click OK.

3. **Create an Unsolicited Message**
   An Unsolicited Message is used because of the very nature of the alarm condition sent by the boiler. The server never knows when the device will send an alarm across the line, so it must open up the server device and wait until an alarm appears. This means the data is "unsolicited".

   By contrast, "solicited" data is data received from the device once the server sends out a request to the device for that data. All solicited messages are designed in **Command/Request (Host) Messages**.

- Select **New | Unsolicited Msg** via the Menu, Toolbar, or Popup Menu.
- Click on the **General** Tab
- Type in the name of the message. For this example, use **Get_Alarm** (although any name here is appropriate. The name of the message does not matter to the server or to the client, except that there needs to be a name, and it must be unique).
- Type in a description, if desired.
- Leave all other options at their default.
- Click on the **Received** Tab.
  In the **Received Message Box**, type in the following exactly (or use the **Sequence Builder** to build the message):

  AL{Alarmcode}{$CR}

  When using the Sequence Builder, click on the **Change Sequence** button in the **Single Assignment** section. "Alarmcode" is on the Item Tab, while "$CR" is on the Control Tab.
- Click on OK.

4. **Save and Close the Protocol File**
   Finally, you need to Save the protocol file and close it in order to go to the next steps.

   - Select **File | Save** from the server menu to save the protocol file.
   - Select **File | Close** to close the protocol configuration window.

5. **Create/Change a Server Device**
   The server device connects the physical device to the server. By default, the serial ports COM1 and COM2 are created for you, so if you are going to use these two devices, just skip the first step. Since our test device is on COM3, we'll do the entire process.

- Select the **Configuration Window / Devices** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
- Select **New | COM Device** via one of the options.
- Change the COM port name to **COM3**.
- Keep all other options at their default.
- Click OK.

6. **Create/Change a Server Topic**
   A server topic associates a server device with a protocol. The topic name is used by both the server and the client to help identify items. This allows for multiple topics to refer to the same protocol, and thereby the same item name. Only the different topic names keep the data separate.

   - Select the **Configuration Window / Topics** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
   - Select **New | Topics** via one of the options.
   - Enter in a new topic name. For our example, we will use **BOILER_COM3**. Please note that the name is not case-sensitive. Also, it's a good idea to use a description name for the topic name. In this case, I can look at the topic name and know that the "Boiler" protocol is used on "COM3" for this topic. It's also a good idea to use the names of the devices. For example, if I was using a Krieger RS-90 Boiler, I might name the topic **KRGR_RS90_COM3**.
   - Select **BOILER** from the **Protocol** pull-down box.
   - Select **COM3** from the **Device** pull-down box.
   - Keep all other options at their default.
   - Click OK.

That's it! You have completed the setup for a simple protocol that responds to an Alarm from a boiler.

To use this protocol, you will create a tag to the item **Alarmcode**, setting the default to 0 (or any number that is not a valid Alarm code). Once that tag goes to any number other than the default, then the **Alarmcode** message has picked up an alarm. You will then process the alarm and reset the value of the **Alarmcode** tag to the default.

Of course, if you have an actual device, you will need to test the protocol. To do that, refer to the Trouble Shooting section of this help manual.

8.6.2 Push Button - Client

Tutorial: Events - Responding to a Push Button on the Client

This tutorial takes you through implementing a protocol that responds to a push button event from the client program.

Device Specifications

This device needs a manually start and/or stop command. The client will activate a push button on the screen which will send a command to the device to start it up, or to stop operations. The data sent is a simple ASCII string, either the command "START" or the command "STOP", and a carriage return.

Technical Specifications

- Server Device: COM3
- Data Stream to device: An ASCII command, followed by a carriage return (CR).
- Data Needed by Client: None.
- Data Needed by Server: A change of the condition of the item ***Command*.**

Server Concepts Covered

In this tutorial, we will use what is known as the Trigger Point, which automatically sends out a Command/Request (Host) Message to the device once the trigger item is set to a Logical One. This overrides the default operation of the Command/Request (Host) Message, which will either automatically be sent out at regular intervals (a "read" message), or sent out when the contents of an item has changed (a "write" message).

Implementation

In this protocol example, we will tell the server to send out the message "START" to the device. To implement your protocol, do the following:

1. **Create a New Protocol**
   First off, we will need to create the protocol. This protocol will define exactly how the server will interact with both the device (via the server device) and the client (via the items).

   - Open up the Server Configuration Program
   - Select the Protocol Configuration Window.
   - Select **New | Protocol** via the Menu, Toolbar, or Popup Menu.
   - Click on **Protocol Settings** in the tree menu on the left side of the configuration window. Select **Properties** either from the Menu, the Toolbar, or the Popup Menu. You can also double-click on the **Protocol Settings** name.
   - Enter in a name for the protocol. Use **PUSHBUTTON** in this example.
   - Enter in a description and a copyright notice, if desired. Keep all other options at their default.
   - Click OK.

2. **Create New Items**
   Now we need to create Items, which is what the server uses to transfer data to and from the client. Items are also used to build data streams (messages) that are sent to the device, and to interpret those data streams that come back to the server from the device.

   In the case of this protocol, items are also used as **Trigger Points** that, when set to Logical One, will automatically send out a Command/Request (Host) Message.

   - Select **New | Item** via the Menu, Toolbar, or Popup Menu.
   - Type in the item name. For this example, use **Start**.
   - Type in a description, if so desired.

- Change the data type from **Integer** to **Discrete**.
- Leave all other options at their default.
- Click OK.

3. **Create a Command/Request (Host) Message**
   An Command/Request (Host) Message is used because of the ability of the server to send out a Command/Request (Host) Message based upon the setting of the Trigger Point. This Trigger Point is an item that, when set to Logical One, will override the default setting of the Command/Request (Host) Message and immediately queue the message for delivery to the device.

   - Select **New | Command/Request (Host) Message** via the Menu, Toolbar, or Popup Menu.
   - Click on the **General** Tab
   - Type in the name of the message. For this example, use **Start** (although any name here is appropriate. The name of the message does not matter to the server or to the client, except that there needs to be a name, and it must be unique).
   - Type in a description, if desired.
   - Leave all other options at their default.
   - Click on the **Request** Tab.
     The Request tab is the message that is sent to the device when the Command/Request (Host) Message is activated. In the **Received Message Box**, type in the following exactly (or use the **Sequence Builder** to build the message):

     START{$CR}

     When using the Sequence Builder, first type in the characters **START** (since any characters outside of the "{}" characters are interpreted as-is), then click on the **Change Sequence** button in the **Single Assignment** section. "$CR" is on the Control Tab.
   - Next, click on the **Chains and Triggers** tab.
     Here, we will change the **Trigger Point** of the Command/Request (Host) Message to inform the server that this message will be sent out once the item listed is set to Logical One.

Change **"{None}"** to **Start**.

- Click on OK.

4. **Save and Close the Protocol File**
   Finally, you need to Save the protocol file and close it in order to go to the next steps.

   - Select **File | Save** from the server menu to save the protocol file.
   - Select **File | Close** to close the protocol configuration window.

5. **Create/Change a Server Device**
   The server device connects the physical device to the server. By default, the serial ports COM1 and COM2 are created for you, so if you are going to use these two devices, just skip the first step. Since our test device is on COM3, we'll do the entire process.

   - Select the **Configuration Window / Devices** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
   - Select **New | COM Device** via one of the options.
   - Change the COM port name to **COM3**.
   - Keep all other options at their default.
   - Click OK.

6. **Create/Change a Server Topic**
   A server topic associates a server device with a protocol. The topic name is used by both the server and the client to help identify items. This allows for multiple topics to refer to the same protocol, and thereby the same item name. Only the different topic names keep the data separate.

   - Select the **Configuration Window / Topics** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
   - Select **New | Topics** via one of the options.
   - Enter in a new topic name. For our example, we will use **PB_COM3**. Please note that the name is not case-sensitive. Also, it's a good idea to use a description name for the topic

name. In this case, I can look at the topic name and know that the "Barcode" protocol is used on "COM3" for this topic. It's also a good idea to use the names of the devices. For example, if I was using a Krieger RS-90 Barcode Scanner, I might name the topic **KRGR_RS90_COM3**.

- Select **PUSHBUTTON** from the **Protocol** pull-down box.
- Select **COM3** from the **Device** pull-down box.
- Keep all other options at their default.
- Click OK.

That's it! You have completed the setup for a simple protocol that accepts a push button change from the client.

On the client side, create a tag associated with the item **Start**. Then, whenever the tag gets set to Logical One, the Command/Request (Host) Message **Start** will be sent to the device. This is because the item **Start** is a Trigger Point for the Command/Request (Host) Message **Start**.

What about issuing the **Stop** command? Just create another item called **Stop**, and another Command/Request (Host) Message called **Stop**, just like the ones above. Then you can create a tag to the item **Stop** and issue the "Stop" command to the device.

Is there another way to do this? Yes! You can also create a "Write" Command/Request (Host) Message and have a String item called **Command** that accepts the actual command from the client and places it in the Request part of the Command/Request (Host) Message. But we will leave that exercise to you.

8.7 Using Modems in Protocols

Tutorial: Modems

This Modem Tutorial take you through the steps in building protocols which handle the unique communication concerns of modems. These are useful for clients that need to monitor remote system information on various devices.

- Modems - COM - Learn how to setup a protocol that talks to a Measurement device connected to a Modem on a Serial port.

8.7.1 Modems on a Serial Port

Tutorial: Modems - COM

This tutorial takes you through implementing a protocol that reads in the Present Value from a monitoring device connected to a on a Serial port.

Device Specifications

This device measures the flow pressure in a duct. The server will first initiate communications with a modem, and once communicating, will send out a message to the device to ask the device to send the measurement back to the server. The server then sends the data back to the client and hangs-up the modem.

Technical Specifications

- Server Device: COM3
- Data Stream to device: A Start-of-Text character, a command code, a question mark, an End-of-Text character, and a Longitudinal Redundancy Check (LRC) on all characters except the Start-Of-Text character.
- Data Stream to Server: An Acknowledgement character, a Start-of-Text character, the command code, an equals sign, the actual data value itself, an End-of-Text character, and a Longitudinal Redundancy Check (LRC) on all characters except the Start-Of-Text character.
- Data Needed by Client: The Present Value of the Device.
- Data Needed by Server: A change of the condition of the item ***PV.***

Server Concepts Covered

In this tutorial, we will use the Chained Command/Request (Host) Messages, which is a way that the server can establish a "conversation" with a device in which a series of requests and responses control the path the server takes in determining how to talk to the device.

This is done via the "Chains and Triggers" function. Particularly, we are using the "On Success" chain.

Also, we introduce the Error Detection Code **LRC8**, which does an 8-bit Exclusive-OR error check on the data within a certain range.

Implementation

In this protocol example, we will tell the server to establish communications with the modem, then send out the message contained in an item called MESSAGE to the device. To implement your protocol, do the following:

1. **Create a New Protocol**
   First off, we will need to create the protocol. This protocol will define exactly how the server will interact with both the device (via the server device) and the client (via the items).

   - Open up the Server Configuration Program
   - Select the Protocol Configuration Window.
   - Select **New | Protocol** via the Menu, Toolbar, or Popup Menu.
   - Click on **Protocol Settings** in the tree menu on the left side of the configuration window. Select **Properties** either from the Menu, the Toolbar, or the Popup Menu. You can also double-click on the **Protocol Settings** name.
   - Enter in a name for the protocol. Use **MODEM** in this example.
   - Enter in a description and a copyright notice, if desired. Keep all other options at their default.
   - Click OK.

2. **Create New Items**
   Now we need to create Items, which is what the server uses to transfer data to and from the client. Items are also used to build data streams (messages) that are sent to the device, and to interpret those data streams that come back to the server from the device. For each of the items listed below, do the following:

   - Select **New | Item** via the Menu, Toolbar, or Popup Menu.

- Type in the item name from the list below.
- Type in a description, if so desired.
- Change the data type to the correct type from the list below.
- Leave all other options at their default.
- Click OK.

The list for the items:

- Item Name: **Start**, Type: **Discrete**
- Item Name: **PV**, Type: **Real**

3. **Create Command/Request (Host) Messages**
   An Command/Request (Host) Message is used because of the ability of the server to send out a Command/Request (Host) message whenever the item in **Trigger** set set to Logical One. Also, you can chain Command/Request (Host) messages together via the **On Success** chain.

   For each of the Command/Request (Host) messages below, do the following:

   - Select **New | Command/Request (Host) message** via the Menu, Toolbar, or Popup Menu.
   - Click on the **General** Tab
   - Type in the name of the message from the list below (although any name here is appropriate. The name of the message does not matter to the server or to the client, except that there needs to be a name, and it must be unique).
   - Type in a description, if desired.
   - Type in the **Type** from the list below.
   - Leave all other options at their default.
   - Click on the **Request** Tab.
     The Request tab is the message that is sent to the device whenever the server executes the Command/Request (Host) message. This is done either through the **On Success** chain or the **Trigger** point.

When using the Sequence Builder (double-click in the message box to bring it up), click on the **Change Sequence** button in the **Single Assignment** section to select the characters form the various tabs. For example, the Start-of-Text Character ($STX) is located on the Control Tab, the End-of-Text Character ($ETX) is located on the Control Tab, and the Error Detection Code ($LRC8,) is located on the Others Tab.

For all other characters, type them in as is.

The Error Detection Code LRC8: This tutorial uses an Error Detection Code. Please see the end of this article for a complete explanation of what was done in this message. In this example, we want to output the error detection code as a one-byte Binary Unsigned number (giving us a number between 0 and 255). To do this, select **LRC8** from the Others Tab, and click OK. Then, select the **Formatting** tab and select **Integer - Unsigned** from the Format Style section, then select the **Binary** checkbox. Finally, type in a width of **1** in the Format Width box, and click OK.

- Click on the **Response** Tab.
  The Response part of the Command/Request (Host) message describes to the server exactly how the data will look like from the device. Within the Response, you will normally find the item that the client is requesting. In the **Response Message Box**, type in the Response as listed in the list below.

  When using the Sequence Builder (double-click in the message box to bring it up), click on the **Change Sequence** button in the **Single Assignment** section to select the characters form the various tabs. For example, the Start-of-Text Character ($STX) is located on the Control Tab, the End-of-Text Character ($ETX) is located on the Control Tab, and the Error Detection Code ($LRC8,) is located on the Others Tab.

  For all other characters, type them in as is.

- Click on the **Chains and Triggers** Tab.
  The Chains and Triggers tab controls the execution sequence of the server. For example, the **Trigger Point**, if set, will send

out the message once the item associated with the Trigger Point is set to Logical One.

The "On Success" chain will send the server to another Command/Request (Host) message if the proper Response was received.

The "On Failure" chain will send the server to another Command/Request (Host) message if the Response was not fulfilled.

For each of the Command/Request (Host) messages in the list below, change the **Trigger Point** and the **On Success** chain as need be.

- Click on OK.

THE Command/Request (Host) message LIST:

- Message #1
  Name: **Stop_Modem**
  Type: **Write**
  Request: **+++ATH{$CR}**
  Chains and Triggers:
  Only activate message via trigger or message chain: **Checked**

- Message #2
  Name: **Get_PV**
  Type: **Read**
  Request: **{$STX}[PV?{$ETX}]{$LRC8:1UB}**
  Response: **{$ACK}{$STX}[PV={PV}{$ETX}]{$LRC8:1UB}**
  Chains and Triggers:
  On Success: **STOP_MODEM**
  Only activate message via trigger or message chain: **Checked**

- Message #3
  Name: **Start_Modem_Msg2**
  Type: **Write**
  Request: **ATDT5551212{$CR}**
  Response: **CONNECT**
  Chains and Triggers:
  On Success: **GET_PV**

Only activate message via trigger or message chain: **Checked**

- Message #4
  Name: **Start_Modem_Msg1**
  Type: **Write**
  Request: **+++ATE1ATS2{$CR}**
  Response: **OK**
  Chains and Triggers:
  Trigger point: **START**
  On Success: **START_MODEM_MSG2**
  Only activate message via trigger or message chain: **Checked**

4. **Save and Close the Protocol File**
   Finally, you need to Save the protocol file and close it in order to go to the next steps.

   - Select **File | Save** from the server menu to save the protocol file.

   - Select **File | Close** to close the protocol configuration window.

5. **Create/Change a Server Device**
   The server device connects the physical device to the server. By default, the serial ports COM1 and COM2 are created for you, so if you are going to use these two devices, just skip the first step. Since our test device is on COM3, we'll do the entire process.

   - Select the **Configuration Window / Devices** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.

   - Select **New | COM Device** via one of the options.

   - Change the COM port name to **COM3**.

   - Keep all other options at their default.

   - Click OK.

6. **Create/Change a Server Topic**
   A server topic associates a server device with a protocol. The topic name is used by both the server and the client to help identify items. This allows for multiple topics to refer to the same protocol,

and thereby the same item name. Only the different topic names keep the data separate.

- Select the **Configuration Window / Topics** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
- Select **New | Topics** via one of the options.
- Enter in a new topic name. For our example, we will use **MODEM_COM3**. Please note that the name is not case-sensitive. Also, it's a good idea to use a description name for the topic name. In this case, I can look at the topic name and know that the "Marquee" protocol is used on "COM3" for this topic. It's also a good idea to use the names of the devices. For example, if I was using a Krieger RS-90 Modem, I might name the topic **KRGR_RS90_COM3**.
- Select **MODEM** from the **Protocol** pull-down box.
- Select **COM3** from the **Device** pull-down box.
- Keep all other options at their default.
- Click OK.

That's it! You have completed the setup for a simple protocol that initializes a modem, reads a present value of a flow meter, returns that information to the client, and finally shuts off the modem

On the client side, create a tag associated with the item **PV** and one to **START**. Then, set the value of **START** to Logical One. The server will send the message **START_MODEM_MSG1**, wait for a response, then send **START_MODEM_MSG2**, wait for a response, then send **GET_PV**, wait the for the present value to be sent by the device, then finally send out **STOP_MODEM** to shut down the modem.

Of course, the actual modem commands and the phone number may not be correct. Your implementation may be different (which means, do not use this tutorial as a primer for modem operations!). Also, no provisions for error messages from the modem (such as **BUSY**). In that case, you will use the **On Failure** chain to perform any type of shutdown or client notification that something has gone wrong.

How to Use Error Detection Codes: The server has the ability to calculate error detection codes on the data in a message. To use one of these codes, there are three steps involved:

- **Select the correct code**. Look over the descriptions in the Help File to determine the correct code to use for this message.
- **Select the correct format**. Next, format the output using the **Formatting** and **Advanced Formatting** tabs on the Sequence Builder.
- **Select the correct characters**. Finally, you need to tell the server what characters are included in the error detection code calculation. To do this, you will insert square brackets ("[" and "]") around the characters in the message that must be used in the calculation. For example, take a look at the example above:

  {$STX}[PV?{$ETX}]{$LRC8:1UB}

  In this sample notice the right square bracket ("[") just after the the Start-of-Text character ("{$STX}"), and the left square bracket ("]") just after the End-of-Text character ("{$ETX}"). The server will use only those characters between the brackets, so in this case, all characters except the Start-of-Text character is used in the calculation.

8.8 Mettler Toledo SICS

Tutorial: Mettler Toledo Scales via SICS Protocol

This tutorial takes you through the sample commands implemented in the MT_SICS_SCALE sample protocol installed with OmniServer. The device type used (COM or WINSOCK) depends on the specific Mettler Toledo scale model you are using and which physical interface is provided. Consult the owner's manual for your specific model to determine the physical connection settings required.

Device Specifications

These scales measure weights (tare weight, net weight, etc). There are many available commands in the SICS protocol. The sample provided implements only three specific commands to give you an example of how the SICS protocol commands must be formatted in OmniServer.

In general, with this sample, OmniServer will send out a message to the device once the client application has written to a specific trigger item that initiates a specific command telling the device to send the weight measurement back to the server. The server then sends the weight data back to the client.

Technical Specifications

In this tutorial, we will use "Command/Request (Host) messages" (Command/Request Message) setup to only be initiated by a trigger item, which is a Command/Request (Host) message set up in such that way that a specific item is configured as the trigger point. The Request portion of the Command/Request (Host) message will only be transmitted upon a write by a client application to that trigger item.

- Command: Stable Weight (S)
- Data Stream to device (Request Message): ASCII letter capital "S", a Carriage Return (Hex 0D), and a Line Feed (Hex 0A).
  - S{$CR}{$LF}
- Data Stream to Server (Response Message): ASCII letter capital "S". a Space (Hex 20), ASCII letter capital "S", a Space (Hex 20), Floating Point Stable Weight value, a Space (Hex 20), Units, a Carriage Return (Hex 0D), and a Line Feed (Hex 0A).
  - S{$SP}S{$SP}{StableWeight:RQ}{$SP}{:S}{$CR}{$LF}
- Data Needed by Client: The Stable Weight Value of the current weighed object.
- Trigger Item: StableWeigh (Boolean value - Client writes 1/True to this item to trigger the request)
- Data Needed by Server: A change of the value of the item ***StableWeight*.**

- Command: Tare Weight (T)
- Data Stream to device (Request Message): ASCII letter capital "T", a Carriage Return (Hex 0D), and a Line Feed (Hex 0A).

    - T{$CR}{$LF}

- Data Stream to Server (Response Message): ASCII letter capital "T". a Space (Hex 20), ASCII letter capital "S", a Space (Hex 20), Floating Point Tare Weight value, a Space (Hex 20), Units, a Carriage Return (Hex 0D), and a Line Feed (Hex 0A).
    - T{$SP}S{$SP}{Weight:RQ}{$SP}{:S}{$CR}{$LF}
- Data Needed by Client: The Tare Weight Value of the current weighed object.
- Trigger Item: Tare (Boolean value - Client writes 1/True to this item to trigger the request)
- Data Needed by Server: A change of the value of the item ***Weight*.**

- Command: Weight Now (SI)
- Data Stream to device (Request Message): ASCII letter capital "SI", a Carriage Return (Hex 0D), and a Line Feed (Hex 0A).
    - T{$CR}{$LF}
- Data Stream to Server (Two Possible Response Formats handled via Two Unsolicited Messages):
    - WeightNow_Stable Unsolicited Message Data Stream: ASCII letter capital "S". a Space (Hex 20), ASCII letter capital "S", a Space (Hex 20), Floating Point Weight value, a Space (Hex 20), Units, a Carriage Return (Hex 0D), and a Line Feed (Hex 0A).
        - S{$SP}S{$SP}{Weight:RQ}{$SP}{:S}{$CR}{$LF}
    - WeightNow_Unstable Unsolicited Message Data Stream: ASCII letter capital "S". a Space (Hex 20), ASCII letter capital "D", a Space (Hex 20), Floating Point Weight value, a Space (Hex 20), Units, a Carriage Return (Hex 0D), and a Line Feed (Hex 0A).
        - S{$SP}D{$SP}{Weight:RQ}{$SP}{:S}{$CR}{$LF}
- Data Needed by Client: The Weight Value of the current weighed object (regardless of stability).

- Trigger Item: Weigh (Boolean value - Client writes 1/True to this item to trigger the request)
- Data Needed by Server: A change of the value of the item ***Weight*.**

Implementation

To implement the protocol:

- Create/Change a Server Device
  The server device connects the physical device to the server.
  - Serial Connection?
    - If your Mettler Toledo scale has a serial COM port, you will configure a COM device to match in OmniServer. By default, the serial ports COM1 and COM2 are created for you, so if you are going to use one of those two COM ports, just edit the COM parameters to match your scale's COM parameters. Otherwise, you will need to do the following:
      - Select the **Configuration Window / Devices** via the Menu, Toolbar, Drop-Down Menu or Shortcut bar.
      - Select **New | COM Device** via one of those options.
      - Change the COM port name to the appropriate COM ID for the COM port on your machine (if you're unsure, consult the Windows Device Manager to confirm the proper COM ID).
      - Consult your owner's manual for the scale to confirm the other required settings (Baud, Parity, Stop Bits, etc.).
      - Click OK.
  - Ethernet Connection?
    - If your Mettler Toledo scale has an Ethernet port, you will configure a Winsock device to match the IP Address and Port of the scale in OmniServer

        - Select the Configuration Windows / Devices via the Menu, Toolbar, Drop-Down Menu or Shortcut bar.
        - Select New | Winsock Device via one of those options.
        - Define the Type as TCP.
        - Enter the scale's assigned IP Address.
        - Enter the scale's assigned Port number.
        - Leave the other settings at the defaults and click OK.

- Create/Change a Server Topic
  A server topic associates a server device with a protocol. The topic name is used by both the server and the client to help identify items. This allows for multiple topics to refer to the same protocol, and thereby the same item name. Only the different topic names keep the data separate.
    - Select the **Configuration Window / Topics** via the Menu, Toolbar, Drop-Down Menu or Short Cut bar.
    - Select **New | Topics** via one of the options.
    - Enter in a new topic name. For our example, we will use MT_SCALE. Please note that the name is not case-sensitive. Also, it's a good idea to use a descriptive name for the topic name.
        - It can also be a good idea to use the names or instances of devices in the Topic name.
    - Select **MT_SICS_SCALE** from the **Protocol** pull-down box.
    - Select the appropriate device that you configured above from the **Device** pull-down box.
    - Keep all other options at their default.
    - Click OK.

That's it! You have completed the setup for a simple protocol that reads the present value of a flow meter and returns that information to the client.

On the client side, create tags associated with the data items and trigger items referred to in the Technical Specifications section above. Configure your client such that you can easily write to the trigger items to initiate sending the commands to the scale.

Server Support

9 Server Support

Support Items for the Server

The design of the server is such that it is a "run and forget" server, but to get it to that point, some development and testing must take place.

However, there are questions that may pop up that concerns installation, migration, or trouble shooting.

For that, we present the following items:

- Upgrading from V1.x
- Upgrading from V2.0
- Upgrading from V2.5
- Upgrading from V2.6 (or higher)
- Upgrading from OmniWedge to OmniServer Professional Edition
- Upgrading from OmniServer Server Edition to OmniServer Professional Edition
- Using and Configuring Services
- Configuring DCOM for the server
- The Trouble Shooting Wizard
- If you need additional help

9.1 Upgrading from V2.6 (or higher)

Upgrading from V2.6 (or Higher)

Please note: This version will not recognize your existing license. To upgrade, y**ou will need to re-register OmniServer after installation, if you are eligible for a free upgrade. Going forward, V3.0 and newer recognizes existing licenses eligible for free upgrades and this will no longer be necessary.**

Please contact us for information on eligibility for upgrading your existing license and for resetting your existing license.

To upgrade from Version 2.6 (or Higher):

- IMPORTANT: Please make a backup of your existing installation directory. Although nothing in this procedure will destroy the current installation, it's always best to have backups, since your configuration and protocols will be relocated on newer operating systems.
- If you are running the existing version as a service, stop the service. To do this, first launch the server configuration program and select **View | Runtime options** from the menu. Uncheck the "Run as a service" option, then exit. Finally, check the Services section in your system's Administrative tools and insure that the server has been stopped and is disabled.
- If you are not running as a service, stop the OmniServer Poller by right-clicking on the icon in the icon tray and selecting **Exit**. If you do not see the icon, then OmniServer Poller is not running.
- Install your copy of the current version. This will automatically upgrade your existing version, moving your previous configuration and protocols files as necessary for the operating system.

That's it! The last thing to do is register your server. Please see Registration for more information.

Questions?

Please visit our OmniServer FAQ page for more information.

9.2 Upgrading from V2.5

Upgrading from V2.5

Please note: Versions starting with V2.6 use a new licensing scheme completely different from the older scheme. You must have your new license number available before beginning this procedure**.** Please contact us for information on how you can obtain your new license number.

To upgrade from Version 2.5:

- IMPORTANT: Please make a backup of your V2.5 installation directory. Although nothing in this procedure will destroy the V2.5 installation, it's always best to have backups.

- If you are running V2.5 as a service, stop the service. To do this, first launch the server configuration program and select View | Runtime options from the menu. Uncheck the "Run as a service" option, then exit. Finally, check the Services section in your system's Administrative tools and insure that the server has been stopped and is disabled.
- If you are not running V2.5 as a service, stop the polling for V2.5 by right-clicking on the icon in the icon tray and selecting Exit. If you do not see the icon, then V2.5 is not running.
- Install your copy of the current version. This will automatically upgrade your Version 2.5 and copy any existing configuration and protocols to the new version.

That's it! The last thing to do is register your server. Please see Registration for more information.

Questions?

Please visit our OmniServer FAQ page for more information.

9.3 Upgrading from V2.0

Upgrading from V2.0

All versions 2.6 or newer are treated as a complete and separate installation from Version 2.0, so this is technically not an upgrade. However, if you follow the steps below, you can move your V2.0 configuration files and protocols so that the migration from V2.0 to the current version is seamless.

Please note: Versions 2.6 or newer use a new licensing scheme completely different from the older scheme. **You must have your new license number available before beginning this procedure.** Please contact us for information on how you can obtain your new license number.

To upgrade from Version 2.0:

- IMPORTANT: Please make a backup of your V2.0 installation directory. Although nothing in this procedure will destroy the V2.0 installation, it's always best to have backups.

- If you are running V2.0 as a service, stop the service. To do this, first launch the OmniServer configuration program and select **View | Runtime options** from the OmniServer menu. Uncheck the "Run as a service" option, then exit. Finally, check the Services section in your system's Administrative tools and insure that the server has been stopped and is disabled.
- If you are not running V2.0 as a service, stop the polling for V2.0 by right-clicking on the icon in the icon tray and selecting **Exit**. If you do not see the icon, then V2.0 is not running.
- Install your copy of the current version. Do not launch the configuration program when prompted. **Please note:** It is **very important** that you **DO NOT** install the current version in the same directory as V2.0. Doing so will not only destroy your current copy of V2.0, but will cause the current version not to function.

At this point, the installation program will automatically copy over your V2.0 protocols and configuration files. However, should you ever need to copy them manually, here is what you do:

- Go to your V2.0 directory.
- Inside the V2.0 directory you will find a file with the extension **.CFG**. Copy this file to the current version application files directory (click here for file location details).
- Also in that directory you may find all your protocols. Each protocol will have the extension **.DPD**. Move those files over to the current version application files directory (click here for file location details).
- Finally, you may find a **Samples** directory (this directory would have been installed if you specifically wanted it during the original V2.0 installation). This directory contains all the sample protocols that came with V2.0. If you have changed any of these protocols, they will also have to be moved to the **Samples** directory in the current version application files directory.
- Start up OmniServer. You will see your configuration and your protocols.
- If you were running V2.0 as a service and you wish to run the current version as a service, please see our directions here: Running OmniServer as a service.

Client Configuration Changes:

- OPC DA ProgID: Changes from "DSSI.OmniServer.2" to "SWToolbox.OmniServer" (Old ProgID does still work)

And that's all there is to it! Your next step will be to test the current installation with your configuration and protocols just to make sure that everything works.

Finally, the last thing to do is register your server. Please see Registration for more information.

Questions?

Please visit our OmniServer FAQ page for more information.

9.4 Upgrading from V1.x

Upgrading from V1.x

All versions 2.6 or newer are treated as a complete and separate installation from Version 1.x, so this is technically not an upgrade. However, if you follow the steps below, you can move your V1.x configuration files and protocols so that the migration from V1.x to the current version is seamless.

Please note: Versions 2.6 or newer use a new licensing scheme completely different from the older scheme. **You must have your new license number available before beginning this procedure.** Please contact us for information on how you can obtain your new license number.

To upgrade from Version 1.x:

- IMPORTANT: Please make a backup of your V1.0 installation directory. Although nothing in this procedure will destroy the V2.0 installation, it's always best to have backups.
- Install your copy of the current version. Do not launch the configuration program when prompted. **Please note:** It is **very important** that you **DO NOT** install the current version in the same directory as V1.x. Doing so will not only destroy your current copy of V1.x, but will cause the current version not to function.

- Go to your V1.x directory.
- Inside the V1.x directory you will find a file with the extension **.CFG**. Copy this file to your current version application files directory (click here for file location details).
- Also in that directory you may find all your protocols. Each protocol will have the extension **.DPD**. Move those files over to the current version application files directory (click here for file location details).
- If you are running V1.x as a service, shut down that service. Then disable the V1 service. If you wish to run the current version as a service, please see our directions here: Running OmniServer as a service.
- Start up OmniServer. You will see your configuration and your protocols.

Client Configuration Changes:

- OPC DA ProgID: Changes from "DSSI.OmniOPCServer.1" to "SWToolbox.OmniServer"
- DDE Application Name: Changes from "omnisrv" to "osrvdde"
- Wonderware Suitelink Application Name: Changes from "omnisrv" to "osrvpoll"

And that's all there is to it! Your next step will be to test the current installation with your configuration and protocols just to make sure that everything works.

Finally, the last thing to do is register your server. Please see Registration for more information.

Questions?

Please visit our OmniServer FAQ page for more information.

9.5 Upgrading from OmniWedge to OmniServer Professional

Upgrading from OmniWedge to OmniServer Professional Edition

To upgrade from OmniWedge to OmniServer Professional Edition:

- IMPORTANT: Please make a backup of your OmniWedge installation directory. Although nothing in this procedure will destroy the OmniWedge installation, it's always best to have backups.
- If you are running OmniWedge as a service, stop the service. To do this, first launch the OmniServer configuration program and select **View | Runtime options** from the OmniServer menu. Uncheck the "Run as a service" option, then exit. Finally, check the Services section in your system's Administrative tools and insure that the server has been stopped and is disabled.
- If you are not running OmniWedge as a service, stop the polling for OmniWedge by right-clicking on the icon in the icon tray and selecting **Exit**. If you do not see the icon, then OmniWedge is not running.
- Install your copy of OmniServer Professional Edition. This will over-write the OmniWedge program but **will not** touch your current configuration.
- Start up OmniServer Professional Edition. You will see your configuration and your protocols.
- If you were running OmniWedge as a service and you wish to run OmniServer Professional Edition as a service, please see our directions here: Running OmniServer as a service.

Questions?

Please visit our OmniServer FAQ page for more information.

9.6 Upgrading from OmniServer Server edition to OmniServer Professional Edition

Upgrading from OmniServer Server Edition to OmniServer Professional Edition

To upgrade from OmniServer Server Edition to OmniServer Professional Edition:

- IMPORTANT: Please make a backup of your OmniServer Server Edition installation directory. Although nothing in this procedure will destroy the OmniServer Server Edition installation, it's always best to have backups.
- If you are running OmniServer Server Edition as a service, stop the service. To do this, first launch the OmniServer configuration program and select **View | Runtime options** from the OmniServer menu. Uncheck the "Run as a service" option, then exit. Finally, check the Services section in your system's Administrative tools and insure that the server has been stopped and is disabled.
- If you are not running OmniServer Server Edition as a service, stop the polling for OmniServer Server Edition by right-clicking on the icon in the icon tray and selecting **Exit**. If you do not see the icon, then OmniServer Server Edition is not running.
- Install your copy of OmniServer Professional Edition. This will over-write the OmniServer Server Edition program but **will not** touch your current configuration.
- Start up OmniServer Professional Edition. You will see your configuration and your protocols.
- If you were running OmniServer Server Edition as a service and you wish to run OmniServer Professional Edition as a service, please see our directions here: Running OmniServer as a service.

Questions?

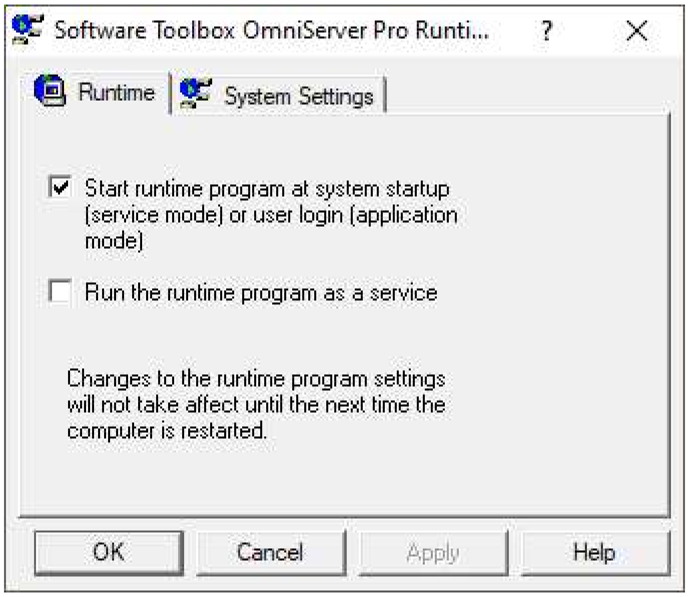
Please visit our OmniServer FAQ page for more information.

9.7 Using Services

Using Services

For All Systems

To start the server as a service, bring up the server's configuration program. Then select **View | Runtime Options** from the server's Menu. The dialog box that appears will look like this:



☑ Start runtime program at system startup (service mode) or user login (application mode)

☐ Run the runtime program as a service

Changes to the runtime program settings will not take affect until the next time the computer is restarted.

To start the server as a service, simple check the **Run the runtime program as a service** option in the dialog box above. Then click the OK button to save the settings.

Please note: You will have to shutdown the server's Poller program and start the server's service before these settings can take effect. This can be done simply by rebooting the computer. If that is not an option, you can also do it manually. Those steps are:

- Locate the server's Poller icon in the icon tray.
- Right-click on the icon and select "Exit" from the menu. If you cannot find the icon, then the poller has already been stopped, and you can proceede to the next step.
- Open up your Services program. This is normally found by going to the Control Panel and selecting the Adminitrative Tools menu. You will find the Services icon there.

- Find the server's normalized name in the list on the right. Right-click on the item and select "Start" from the pop-up menu. The server's Poller program will start up as a service at this point.

Step 1: Setting up the server to run as a Service

Follow the directions in the section above titled **All Systems** to setup the server as a service. After you shutdown the server's Poller program, go to the next step.

Step 2: Start the Server

Now you can start the server. To do this, go to the server's normalized name entry in the Services list, right-click on the entry, and select **Start**. The server's service will start and make itself available to outside clients.

Configuring DCOM

Finally, if you are still not able to communicate to your server, you may need to change your DCOM settings in order to grant access rights to the server. If you are receiving an "Access Denied" error when you try to access the OmniServer poller, you will need to do the following:

1. Go to Control Panel -> Administrative Tools -> Component Services.
2. Expand Component Services -> Computers -> My Computer -> DCOM Config.
3. Find "OmniServer by Software Toolbox" in the list, right-click and select "Properties".
4. Under Security, select "Customize" for "Access Permissions" and click the Edit button.
5. Click the Add button and either add any users who will be logging onto the OmniServer machine or add the "Everyone" user with at least "Local Access".

If you still have problems, more information on DCOM can be found here. DCOM Tutorial

Should you need Help: You can contact us in a number of ways. Click here for customer support information.

9.8 Using DCOM

Using DCOM with the Server

DCOM (Distributed COM), is what gives processes within a computer the ability to launch, access, and configure each other, or other processes on remote computers.

NOTE: DCOM is considered a legacy technology and users needing to connect remote OPC clients to OmniServer using OPC DA over DCOM should instead consider using the OmniServer's OPC UA interface if their client supports OPC UA. Alternatively, Software Toolbox offers OPC DA tunneling solutions that replace DCOM. Due to Microsoft security hardening on DCOM, we are limited in how much support we can provide on DCOM troubleshooting and suggest you start with our DCOM configuration guide.

Why this is important?

Any server needs to make itself available to the outside world, and this server is no exception. However, many times the computer on which the server is being installed will default to a very secure and closed system. This is due to security demands of the operating system - you wouldn't want any installed program to have complete (or even partial) access to any other program on your computer, much less a remote computer.

What can be done about it?

Luckily, setting the permissions of access to and from the server can be done through a program called **DCOMCNFG**. This program will configure the access rights that the server will grant for other processes on your machine or processes on remote machines.

How do I run DCOMCNFG?

We have a tutorial that covers using the program:

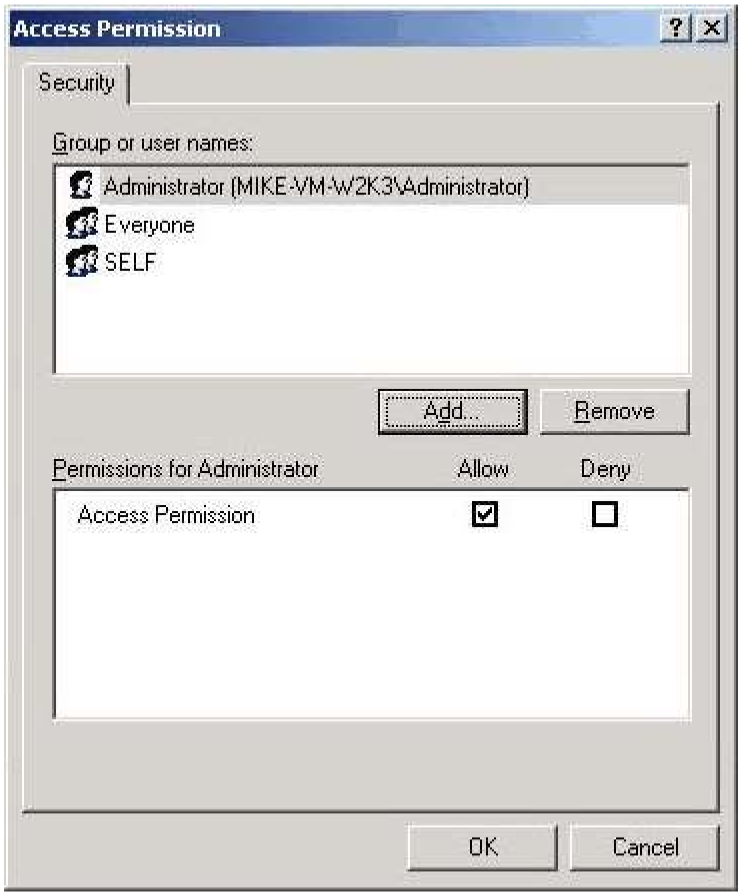
Please see our separate help file, "DCOMTutorial.chm", for the DCOM tutorial.

What if that still doesn't work?

In some cases, you may have to add individual users to your DCOM configuration instead of the groups in which those users were located.

To add individual users, here is what you need to do:

- Go back into the DCOMCNFG program (following the directions in the tutorial above).
- Find the server's DCOM entry, right-click on the entry, and select **Properties**.
- Click on the **Security** tab.
- Go to the "Access" permissions section, select "Customize", and click on the **Edit** button.



Administrator (MIKE-VM-W2K3\Administrator)
Everyone
SELF

Permissions for Administrator	Allow	Deny
Access Permission	✓	

- Next, click on the **Add** button and add in the user you wish to have access to the server.
- Finally, click **OK** twice to exit the screen.

Please note: You will need to restart your server for the changes to take effect.

Finally, if you are still not able to access the server:

Please see our support pages for any additional support or contact information. Consider OPC DA tunneling solutions that replace DCOM.

Due to Microsoft security hardening on DCOM, we are limited in how much support we can provide on DCOM troubleshooting and suggest you start with our DCOM configuration guide.

9.9 If You Need Additional Help

If you need Help

Support for OmniServer is available from:

Your Local Distributor or System Integrator

Your local distributor you bought the product from, or the system integrator that installed and configured OmniServer for you is your first line of support

Online

We have an extensive online support knowledgebase for OmniServer. We suggest you go there first to see if there are already answers to your questions.

Also check our troubleshooting resources list

Our Contact Information

Note that support services provided are described here and preference and priority is given to users with active support agreements. We reserve the right to limit the scope and amount of support we provide for those on on a support agreement. Paid services are also available.

Email

Open a Support Ticket

Phone

Phone: +1 704-849-2773

Mailing Address

Software Toolbox, Inc.
148A East Charles Street
Matthews NC 28105
USA

9.10 The Troubleshooting Wizard

Troubleshooting Flowchart Wizard

This wizard will help with solving communications problems between the device, the server, and the client.

Before you begin, there are some things that are assumed:

- You have created a server protocol.
- You have created a server device.
- You have created a server topic.
- You have configured your client interface.

If any of these steps have not been done, please complete these steps before continuing.

Ready? Access the Troubleshooting Wizard

See also other troubleshooting resources