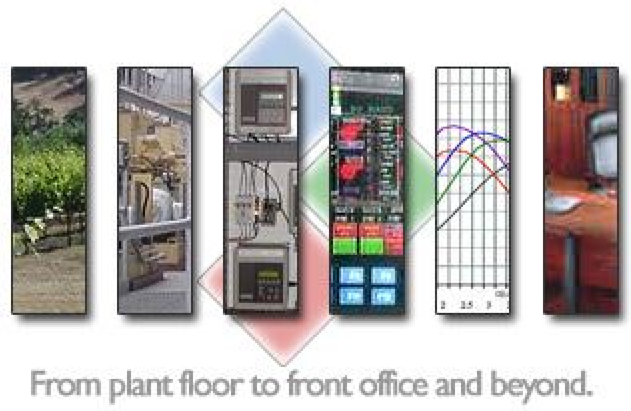
Welcome to FactorySQL



Welcome to FactorySQL!

FactorySQL is the fastest, easiest and most reliable way to link your PLC data to an SQL database. FactorySQL is an OPC client and database connector program that makes bi-directional transfer of data a snap.   It is the most powerful and affordable solution on the market today.

FactorySQL allows you to:

- Communicates with nearly any SQL database system, and any OPC compliant data server.
- Browse PLC addresses, drag them, drop them and start logging them.
- Set your own logging rules - OPC to DB, DB to OPC, or bi-directional.
- Create your own trigger rules - periodic, on value change and others.
- Use built-in handshaking to insure data integrity.
- Automatically create and maintain database tables.
- Call and retrieve values from stored procedures.
- Use built-in alarm server to log alarms and send email alerts based on rules you create.
- Keep an audit log to track changes made to PLC memory from database for accountability.
- Run hundreds of logging or transaction processes simultaneously.
- Log or transact into one or many different databases at the same time.
- Run as a Windows service for reliability.

Main Features of FactorySQL

Log from any PLC to any SQL database.

FactorySQL complies with the OPC and ODBC standards making it fully compatible with virtually all major PLCs and all major SQL databases. It supports native .NET database connectors as well, available for nearly all brands of database.

Write from any SQL database to any PLC

FactorySQL supports writing down to PLCs just as easily as reading up from them. This is very useful for batch downloads, recipe management and a host of other tasks.

Bi-directionally synchronize any SQL database to any PLC

FactorySQL can bi-directionally synchronize SQL databased and PLC data. A change on either end can trigger sending the changed data to the other side instantly. This establishes a whole new concept in SCADA system design. This is the concept of the SQL database as the "center of the world". SQL databases are fast, multi-user, secure and robust - perfectly suited as tag databases.

Trigger on value change, value range, value or time interval

FactorySQL allows you to set up triggers for when to transfer data. This can be based on value change, a value range, any value or time interval - or a combination of several of these. All this is easily configured in intuitive configuration windows.

Handshaking for data integrity and confirmation

Handshaking is provided for in several different ways. This feature can be used to guaurantee or acknowledge the actual transfer of data for data integrity, confirmation and timing purposes.

Browse PLC addresses, drag, drop and start logging

FactorySQL is configured with simple drag and drop configuration. Drag an entire PLC file from the OPC browse window and drop it into a logging group. Select a database connection and press "GO". Instantly your database table and fields corresponding to each PLC address are created and logging begins. Of course, you can customize these settings for your exact needs! Drag items from group to group. Drag groups between folders and lots more. The point is, you can set up data transfers and logging between an SQL database and multiple PLCs in just minutes!

Log thousands of tags efficiently with block groups.

Have a large number of tags that you want to mirror into the database? Block groups can efficently handle many thousands of tags easily and efficiently. Additionally, they support nearly all of the features of standard groups: bi-directional synchronization, triggers, handshakes, and indirect block addressing.

Leverage the power of stored procedures

The Store Procedure group can let you easily map OPC tags to and from stored procedures. Additionally, you can leverage the power of action items and include their values as input parameters. The stored procedure group supports the same trigger and handshake capabilities as the other groups.

Audit Log records changes written to PLC which initiated by a change in the database

You can elect to set up an audit log which will record to the SQL database anytime a new value, initiated from the database, has been written to the PLC. The date, time, old value and new value are stored as well as a field for the username of the person initiating the change. The audit log table and all its fields are created for you automatically when you elect to use audit logging.

Log history, store status

With FactorySQL, setting up historical logging and real-time status are equally easy. You can choose to insert data, update existing data, or update data based on a value from an expression, query, or OPC tag!

Hour meter mode turns database fields into timer fields

This powerful feature is very useful in downtime tracking applications - greatly reducing PLC programming. Accumulate downtime for multiple sections of a line and at the end of the hour or shift trigger the insertion of a new record - leaving the last record with the accumulated values for the last shift. From here you can read the database values from you web application and report on line efficiency, causes of downtime and more.

Event counter mode turns database fields into counter fields

Similar to the hour meter feature above, you can elect to turn any field being logged to into a counter. Each time the tag goes true, the value of the item will be incremented. Track product and production rates and lots more with this feature of FactorySQL.

Comprehensive alert server - alert logging and alerting on any PLC item

FactorySQL is a complete alarm server. Any logged item may also be configured to trigger alarm logging and email alerts. Any number of alarm states are supported. Once set for a single item you can copy your settings to other items. Additionally, the current alarm state is reflected in the alarm status table. These can then be reflected back down to the PLC eliminating a tremendous amounts of PLC programming for alarms. The intuitive and rapid development environment of FactorySQL can speed alarm development enormously.

Dynamic Alert Setpoints

Alert setpoints can be mapped from PLC or SQL database items. Therefore, setpoints can be changed from the plant floor or the front office.

SQL commands can run on any trigger event

SQL queries can be set up to run with the normal update interval of logging groups. In this way SQL database data can be merged with PLC item data in expressions and then be written back up to the database.

Action Items

Action items can be added to any group to evaluate expressions or queries you create every time a group is updated. These can combine numerous PLC items, SQL query results and the results of other Action items to yield values which can be written back to the database or an OPC item. This can save extensive amounts of PLC programming time.
Additionally, Action Items can be enhanced with your own programming with the drop-in function plugin architecture.

Runs as Windows service

FactorySQL runs as a Windows service. The FactorySQL frontend configuration client can connect to the service locally or remotely. One frontend can configure many FactorySQL service applications in an enterprise remotely. The FactorySQL service can be configured to be password protected. By running as a service, FactorySQL runs anytime the host PC is turned on.

Local Data Caching

Feel free to log data to a remote database- should the connection go down, FactorySQL will cache the data until it is back up. You won't lose any historical information.

Free download is fully functional. Only limitation is 2 hour runtime. Develop and don't purchase until deployed.

The FactorySQL service and frontend configuration client is free to download and use with the only limitation being that the logging groups will stop running after two hours. But they can be restarted every two hours to continue developing and testing with FactorySQL.

Entire projects can be developed without ever buying FactorySQL. When your project is finally deployed, it will be necessary to register FactorySQL to remove the two hour limit. Otherwise, FactorySQL is fully functional. Unlimited data points. Unlimited PLCs and SQL database connections. Any project you develop before you register FactorySQL will still be good after your register.

System Requirements

Required Hardware:

- Windows 2000/XP/2003
- 128 MB RAM or more
- 20 MB free disk space
- At least 800*600 screen resolution

Required Software:

- Any OPC server such as Keware's KEPServerEX, Rockwell's RSLinx (OEM or above), etc.
- A supported database: Microsoft SQL Server, Oracle, MySQL, Postgres, IBM DB2, EnterpriseDB, or any database with and ODBC driver.
- .NET framework v2.0

License Agreement

IMPORTANT-READ CAREFULLY: This FactorySQL End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Inductive Automation. By installing, copying, or otherwise using FactorySQL, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE.

1. **BY DOWNLOADING, INSTALLING AND/OR IMPLEMENTING THIS SOFTWARE YOU AGREE TO THE FOLLOWING LICENSE:**
2. **DEFINITIONS.** "You" and "Licensee" refers to the person, entity or organization which is using the Software known as "FactorySQL", and any successor or assignee of same. "Inductive Automation" refers to Hechtman Enterprises, Inc. dba Inductive Automation and its successors, or manufacturer and owner of this Software.
3. **AGREEMENT.** After reading this agreement carefully, if you ("Customer") do not agree to all of the terms of this agreement, you may not use this Software. Unless you have a different license agreement signed by Inductive Automation that covers this copy of the Software, your use of this Software indicates your acceptance of this license agreement and warranty. All updates to the Software shall be considered part of the Software and subject to the terms of this Agreement. Changes to this Agreement may accompany updates to the Software, in which case by installing such update Customer accepts the terms of the Agreement as changed. The Agreement is not otherwise subject to addition, amendment, modification, or exception unless in writing signed by an officer of both Customer and Inductive Automation. If you do not wish to agree to the terms of this Agreement, do not install or use this Software.
4. **GRANT OF LICENSE.**
    i. **Evaluation Copy.** You may use FactorySQL without charge on an evaluation basis. In the unregistered version you have these limitations: **two hour runtime.** You must pay the license fee and register your copy if you wish to use FactorySQL without any limitation.

    ii. **Redistribution of Evaluation Copy.** If you are using FactorySQL on an evaluation basis you may make copies of the evaluation FactorySQL as you wish; give exact copies of the original evaluation FactorySQL to anyone; and distribute the evaluation FactorySQL in its unmodified form via electronic means (Internet, BBS's, Shareware distribution libraries, CD-ROMs, etc.). You may not charge any fee for the copy or use of the evaluation FactorySQL itself, but you may charge a distribution fee that is reasonably related to any cost you incur distributing the evaluation FactorySQL (e.g. packaging). You must not represent in any way that you are selling FactorySQL itself. Your distribution of the evaluation FactorySQL will not entitle you to any compensation from Inductive Automation. You must distribute a copy of this EULA with any copy of FactorySQL and anyone to whom you distribute FactorySQL is subject to this EULA.

    iii. **Registered Copy.** After you have purchased the license for FactorySQL, and have received the serial number enabling the registered copy, you are licensed to copy FactorySQL only into the memory of the number of computers corresponding to the number of licenses purchased. Under no other circumstances may FactorySQL be operated at the same time on more than the number of computers for which you have paid a separate license fee. You may terminate this license at any time by destroying the original and all copies of FactorySQL in whatever form. You may permanently transfer all of your rights under this EULA provided you transfer all copies of FactorySQL(including copies of all prior versions if FactorySQL is an upgrade) and retain none, and the recipient agrees to the terms of this EULA.
5. **RESTRICTIONS.** You may not reverse engineer, de-compile, or disassemble FactorySQL, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation. You may not rent, lease, or lend the SOFTWARE. You may permanently transfer all of your rights under this EULA, provided the recipient agrees to the terms of this EULA. You may not publish or publicly distribute any serial numbers, access codes, unlock-codes, passwords, or other end-user-specific registration information that would allow a third party to activate FactorySQL without a valid license.
6. **OWNERSHIP OF SOFTWARE AND COPYRIGHTS.** The Software is copyrighted and protected by the laws of the United States and other countries, and international treaty provisions. You may not remove any copyright notices from the Software. Inductive Automation may make changes to the Software at any

time without notice, but is not obligated to support or update the Software. Except as otherwise expressly provided, Inductive Automation grants no express or implied right under Inductive Automation patents, copyrights, trademarks, or other intellectual property rights. You may transfer the Software only if the recipient agrees to be fully bound by these terms and if you retain no copies of the Software.

7. **GRANT OF LICENSE AND PROHIBITIONS.** Title to all copies of the Software remains with Inductive Automation. This Software is licensed to you. You are not obtaining title to the Software or any copyrights. You may not sublicense, rent, lease, convey, translate, decompile, or disassemble the Software for any purpose. The license may be transferred to another Licensee if you keep no copies of the Software. Permission must be obtained before mirroring or redistributing the evaluation copies of the Software. You may not convert this Software or its parts to a different computer language or environment, either manually, or using an automated conversion tool, such that this Software or any modification thereof will run under any language, software, or program other than implemented by Inductive Automation. You agree that any modifications made to this Software belong to Inductive Automation and are permitted for your exclusive use during the period of this License Agreement, and may not be transferred, sold or licensed to another entity.
8. **LIMITED WARRANTY.** THE SOFTWARE IS PROVIDED AS IS AND INDUCTIVE AUTOMATION DISCLAIMS ALL WARRANTIES RELATING TO THIS SOFTWARE, WHETHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE.
9. **LIMITATION ON CONSEQUENTIAL DAMAGES.** NEITHER INDUCTIVE AUTOMATION NOR ANYONE INVOLVED IN THE PRODUCTION, OR DELIVERY OF THIS SOFTWARE SHALL BE LIABLE FOR ANY INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH SOFTWARE EVEN IF INDUCTIVE AUTOMATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR CLAIMS. IN NO EVENT SHALL INDUCTIVE AUTOMATION LIABILITY FOR ANY DAMAGES EXCEED THE PRICE PAID FOR THE LICENSE TO USE THE SOFTWARE, REGARDLESS OF THE FORM OF CLAIM. THE PERSON USING THE SOFTWARE BEARS ALL RISK AS TO THE QUALITY AND PERFORMANCE OF THE SOFTWARE. IN NO EVENT WILL INDUCTIVE AUTOMATION BE LIABLE FOR ANY AMOUNT GREATER THAN WHAT YOU ACTUALLY PAID FOR THE SOFTWARE.
10. **TERMINATION.** This Agreement is effective until terminated. This Agreement terminates on the date of the first occurrence of either of the following events: (1) The expiration of one month from written notice of termination from Customer to Inductive Automation; or (2) At any time if you violate the terms of this Agreement. Upon termination you shall destroy all copies of the Software, including modified copies, if any. You agree that monetary damages alone is not an adequate and just relief resulting from any breach of this License, that a court order prohibiting any further breach of this License is necessary to prevent further damages, and that you will not oppose any reasonable request for a temporary restraining order, preliminary injunction, or other relief sought by Inductive Automation in the event of a breach of this License. Inductive Automation shall not be required to notify you of any breach, nor make any demand or claim against you resulting from any such breach, or for a demand to stop any use or distribution in violation of the terms of this License, and you agree that any breach of this License and damages resulting therefrom shall relate back to the first and earliest breach thereof. Failure of Inductive Automation to enforce its rights pursuant to this License shall not constitute a waiver of such rights, and shall not prejudice Inductive Automation in any later enforcement of its rights or rights to seek damages therefrom.
11. **UPGRADES.** If you acquired this Software as an upgrade of a previous version, this Agreement replaces and supercedes any prior Agreements. You may continue to use the previous version of the Software, provided that both the previous version and the upgrade are installed on the same computer at all times. You may not have a previous version and the related upgrade version installed on separate computers at any time.
12. **ENTIRE AGREEMENT.** This End-User Agreement is the entire agreement between you and Inductive Automation relating to the Licensed Software, and supercedes all prior written or oral statements, promises, representations and agreements.
13. **GOVERNING LAW.** The agreement shall be governed by the laws of the State of California. Any action or proceeding brought by either party against the other arising out of or related to this agreement shall be brought only in a state or federal court of competent jurisdiction located in Sacramento County, California. The parties hereby consent to the jurisdiction of such courts.

Quick Start Overview

A full installation of FactorySQL requires three components: A database, an OPC server, and of course, the FactorySQL service. This quick start will run you through an example installation of each component, and how to get up and running almost immediately with a quick data logging example.

For this example, we will use the Kepware OPC server, and the MySQL database system.

Kepware's KepserverEX is a popular and versatile OPC server, that supports a very large range of devices through pluggable drivers. There are many other popular OPC servers available, for almost every device in the field. They will vary in their exact features (for instance, whether they support address space browsing), but in general, they should be fairly similar in their configuration.

Similarly, there are many brands of databases available. Generally, different database systems function more or less the same, and will generally vary primarily in the alterior features they provide. We have chosen to highlight MySQL because it is a powerful, reliable system that is open source, easy to administer, and usually free to use.

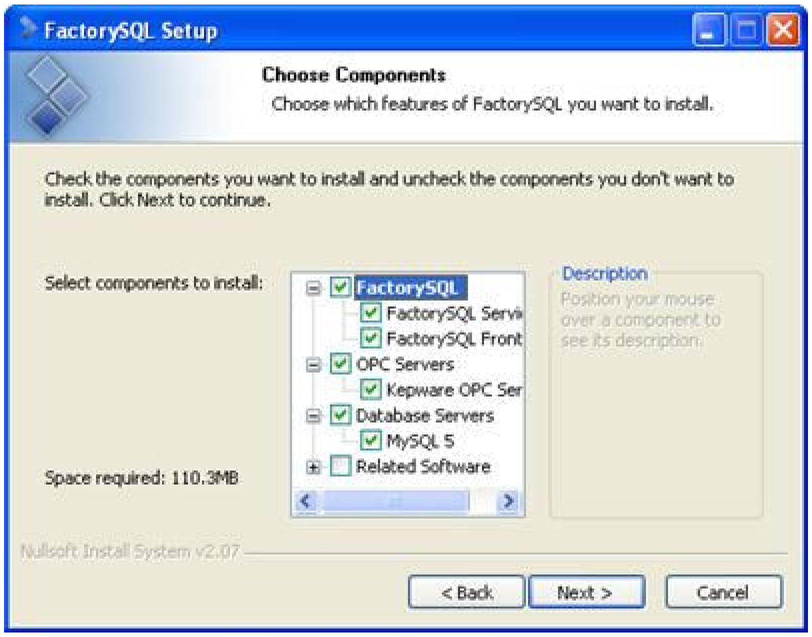
For more information on these two featured products, visit their respective websites at: www.kepware.com and www.mysql.com .

Also used in many examples is the database frontend DBManager Professional, by DBTools Software. This program provides an easy to use frontend that is compatible with a number of popular database systems. Using a database frontend, you can view and alter raw data, adminster tables and users, and perform general maintainence. For more information on DBTools, visit http://www.dbtools.com.br

Please Note: Inductive Automation is not affiliate with, nor can offer support for, any of the third-party products mentioned in this document.

Install FactorySQL

Install FactorySQL by double-clicking the setup.exe installer file on the cd, or the file that you downloaded. After agreeing to the licensing terms this screen will appear. From this screen you can choose to install the FactorySQL Frontend, Service, and to launch the installers for Kepware and MySQL as well (covered in the following pages). If this is a fresh installation, with no exisiting database or OPC server, we recommend that you simply click 'Next'.


FactorySQL Setup
Choose Components
Choose which features of FactorySQL you want to install.
Description
Position your mouse over a component to see its description.
Nullsoft Install System v2.07
< Back
Next >
Cancel

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

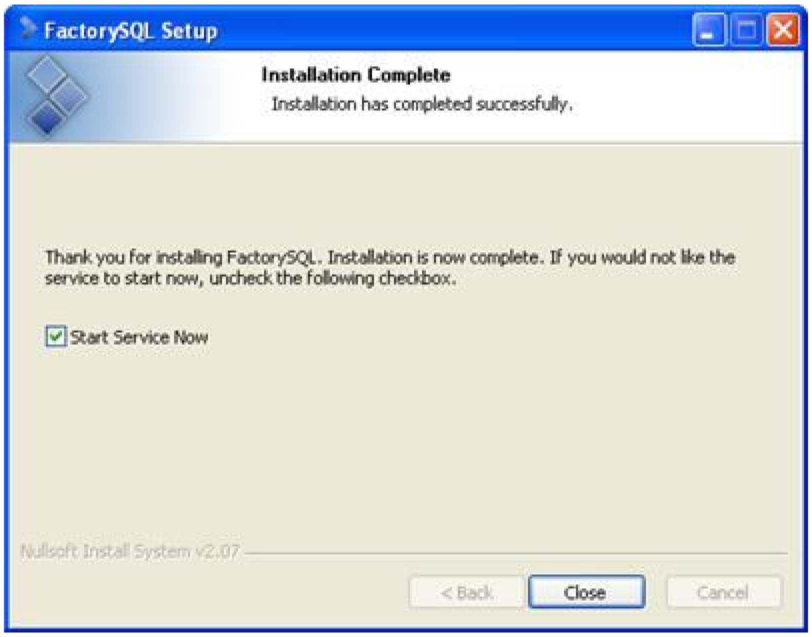
Select components to install:

- ✓ FactorySQL
  - ✓ FactorySQL Servi
  - ✓ FactorySQL Front
- ✓ OPC Servers
  - ✓ Kepware OPC Ser
- ✓ Database Servers
  - ✓ MySQL 5
- Related Software

Space required: 110.3MB

The next few screens will ask you the normal installation questions. The actual installation will then take place, and if selected, the Kepware & MySQL installers will be launched.

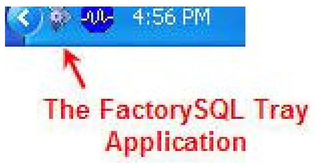
Upon completion, you will be asked if you want to start the service. Click 'Next' to continue.


FactorySQL Setup
Installation Complete
Installation has completed successfully.
Nullsoft Install System v2.07
< Back
Close
Cancel

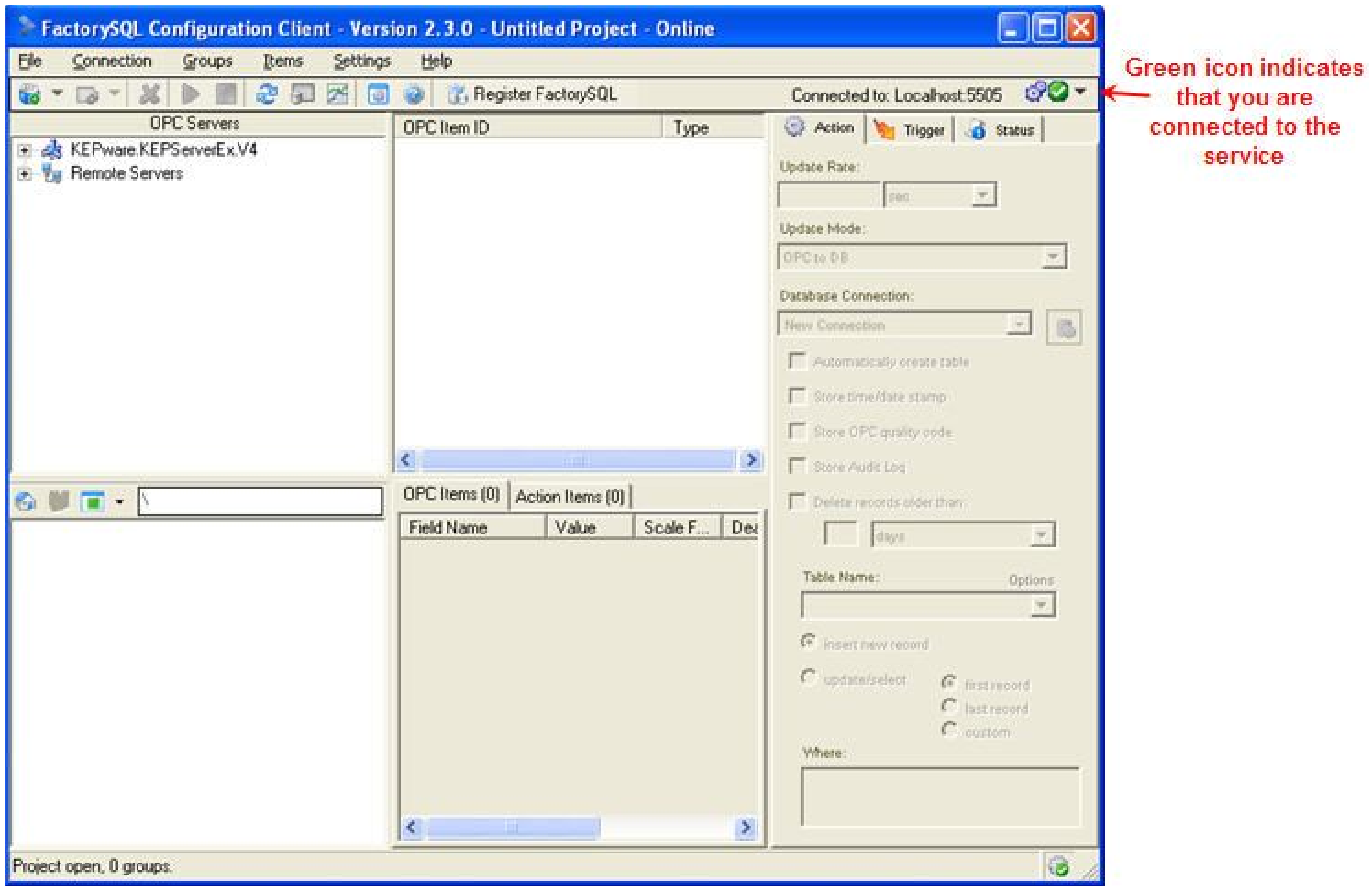
Thank you for installing FactorySQL. Installation is now complete. If you would not like the service to start now, uncheck the following checkbox.

✓ Start Service Now

When the service runs this icon will appear in the application tray. Double clicking it will bring up a status window for the service. To open the configuration frontend, right-click on the icon and choose 'Open Frontend' (or you may also locate it in the Start Menu).



When the frontend starts, it should look more or less like the following (assuming you've installed Kepware):



At this point you are ready to create a simple application. Click Here to see how easy it is!

Install OPC Server

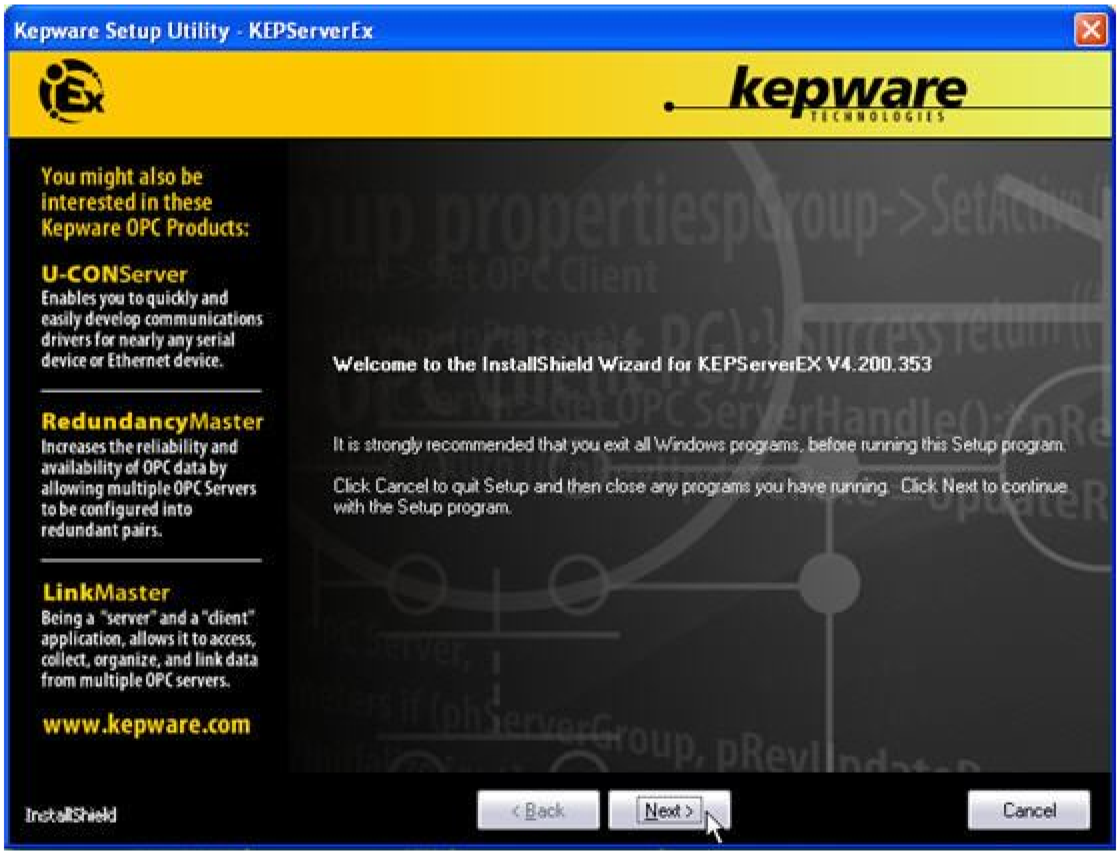
The OPC server is what you will use to interface between your propriatary PLC network and the OPC standard that FactorySQL recognizes. There are OPC servers made for nearly every brand of PLC and manufacturers normally have one available for their own brand of PLC. OPC servers can also be obtained through third party companies such as Matrikon or Kepware.

OPC servers vary depending on the one you use. Some run as Windows services (recommended) and some don't. Some support PLC address space browsing (highly recommended) and some don't. No matter which OPC server you use, once it is installed it will show up automatically in the FactorySQL OPC Topic pane. If you've configured communications to your PLC, and your OPC server supports address space browsing, you can then drill down into the PLC memory structure.

We've chosen to include Kepware along with FactorySQL, as it provides support for a wide range of PLC brands. In our experience it has also proven to be extremely efficient and stable. Conveniently, a standard Kepserver installation includes a data simulator, which will allow you to get up and running quickly with FactorySQL.

Installing the KEPServerEx OPC Server

Kepserver may be installed by choosing it from the FactorySQL installer, or by running KepserverEX.exe directly. You can find the installer on the FactorySQL cd, or at www.kepware.com. The installer is fairly straight forward, but we'll walk through the major screens:


Kepware Setup Utility - KEPServerEx

You might also be interested in these Kepware OPC Products:

U-CONServer
Enables you to quickly and easily develop communications drivers for nearly any serial device or Ethernet device.

RedundancyMaster
Increases the reliability and availability of OPC data by allowing multiple OPC Servers to be configured into redundant pairs.

LinkMaster
Being a "server" and a "client" application, allows it to access, collect, organize, and link data from multiple OPC servers.

www.kepware.com

Welcome to the InstallShield Wizard for KEPServerEX V4.200.353

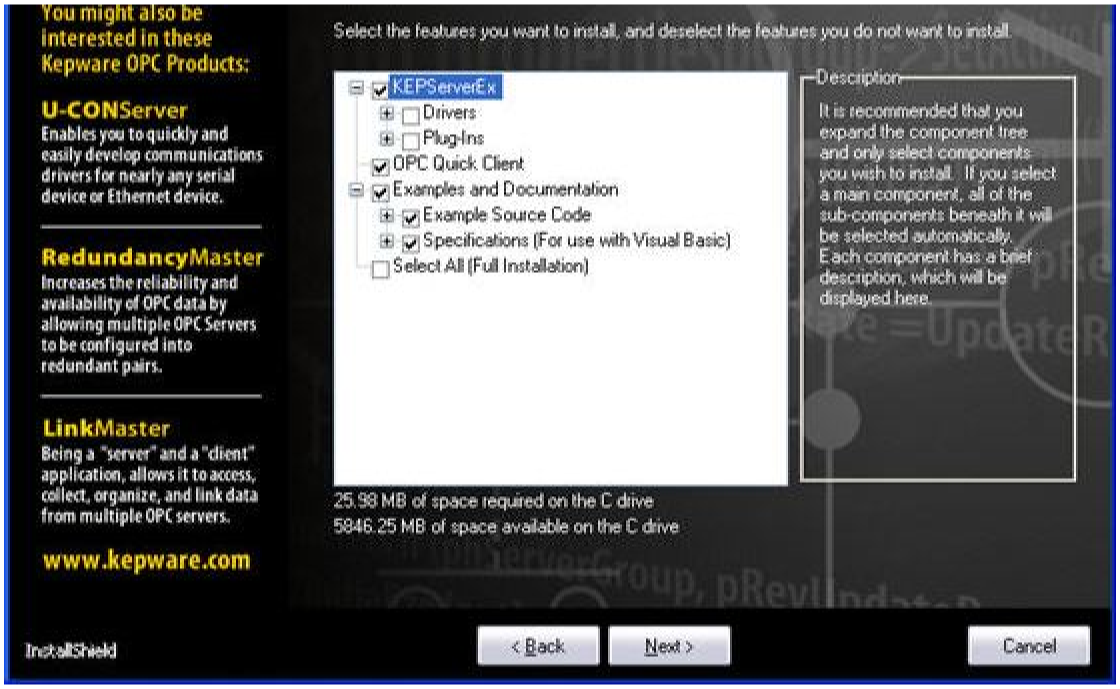
It is strongly recommended that you exit all Windows programs, before running this Setup program.

Click Cancel to quit Setup and then close any programs you have running. Click Next to continue with the Setup program.

Initial Welcome Screen

Kepware Setup Utility - KEPServerEx

Select Features
Select the features setup will install.


You might also be interested in these Kepware OPC Products:

U-CONServer
Enables you to quickly and easily develop communications drivers for nearly any serial device or Ethernet device.

RedundancyMaster
Increases the reliability and availability of OPC data by allowing multiple OPC Servers to be configured into redundant pairs.

LinkMaster
Being a "server" and a "client" application, allows it to access, collect, organize, and link data from multiple OPC servers.

www.kepware.com

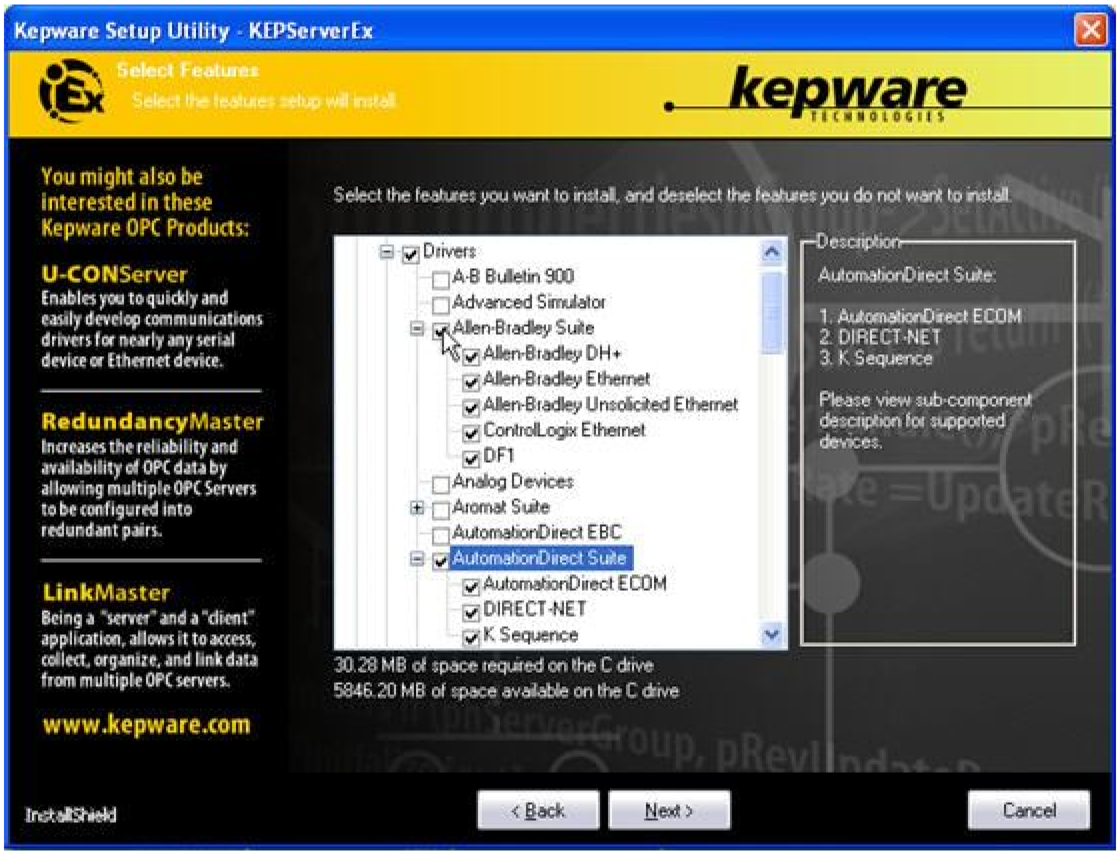
Select the features you want to install, and deselect the features you do not want to install.

- [✓] KEPServerEx
  - [ ] Drivers
  - [ ] Plug-Ins
- [✓] OPC Quick Client
- [✓] Examples and Documentation
  - [✓] Example Source Code
  - [✓] Specifications (For use with Visual Basic)
- [ ] Select All (Full Installation)

Description
It is recommended that you expand the component tree and only select components you wish to install. If you select a main component, all of the sub-components beneath it will be selected automatically. Each component has a brief description, which will be displayed here.

25.98 MB of space required on the C drive
5846.25 MB of space available on the C drive

Choose install options. These are the default, and will suffice if you do not have any PLCs that you wish to connect to at the current time.


Kepware Setup Utility - KEPServerEx

Select Features
Select the features setup will install.

You might also be interested in these Kepware OPC Products:

U-CONServer
Enables you to quickly and easily develop communications drivers for nearly any serial device or Ethernet device.

RedundancyMaster
Increases the reliability and availability of OPC data by allowing multiple OPC Servers to be configured into redundant pairs.

LinkMaster
Being a "server" and a "client" application, allows it to access, collect, organize, and link data from multiple OPC servers.

www.kepware.com

Select the features you want to install, and deselect the features you do not want to install.

- [✓] Drivers
  - [ ] A-B Bulletin 900
  - [ ] Advanced Simulator
  - [✓] Allen-Bradley Suite
    - [✓] Allen-Bradley DH+
    - [✓] Allen-Bradley Ethernet
    - [✓] Allen-Bradley Unsolicited Ethernet
    - [✓] ControlLogix Ethernet
    - [✓] DF1
  - [ ] Analog Devices
  - [ ] Aromat Suite
  - [ ] AutomationDirect EBC
  - [✓] AutomationDirect Suite
    - [✓] AutomationDirect ECOM
    - [✓] DIRECT-NET
    - [✓] K Sequence

Description
AutomationDirect Suite:

1. AutomationDirect ECOM
2. DIRECT-NET
3. K Sequence

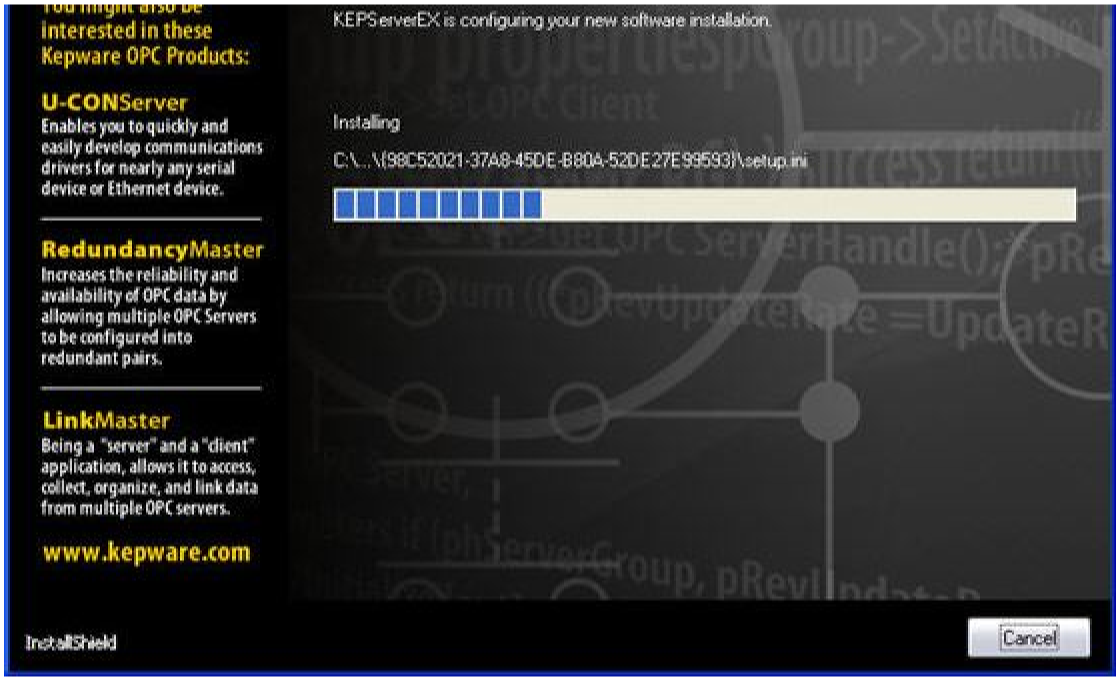
Please view sub-component description for supported devices.

30.28 MB of space required on the C drive
5846.20 MB of space available on the C drive

Optional: Choose Drivers to Install. By expanding the "Drivers" section, you can choose various communication drivers to install. This will allow you to connect to your PLCs. However, even if you do not install a driver, you can still evaluate KEPServer and FactorySQL with the simulator that is installed by default.

Kepware Setup Utility - KEPServerEx

Setup Status

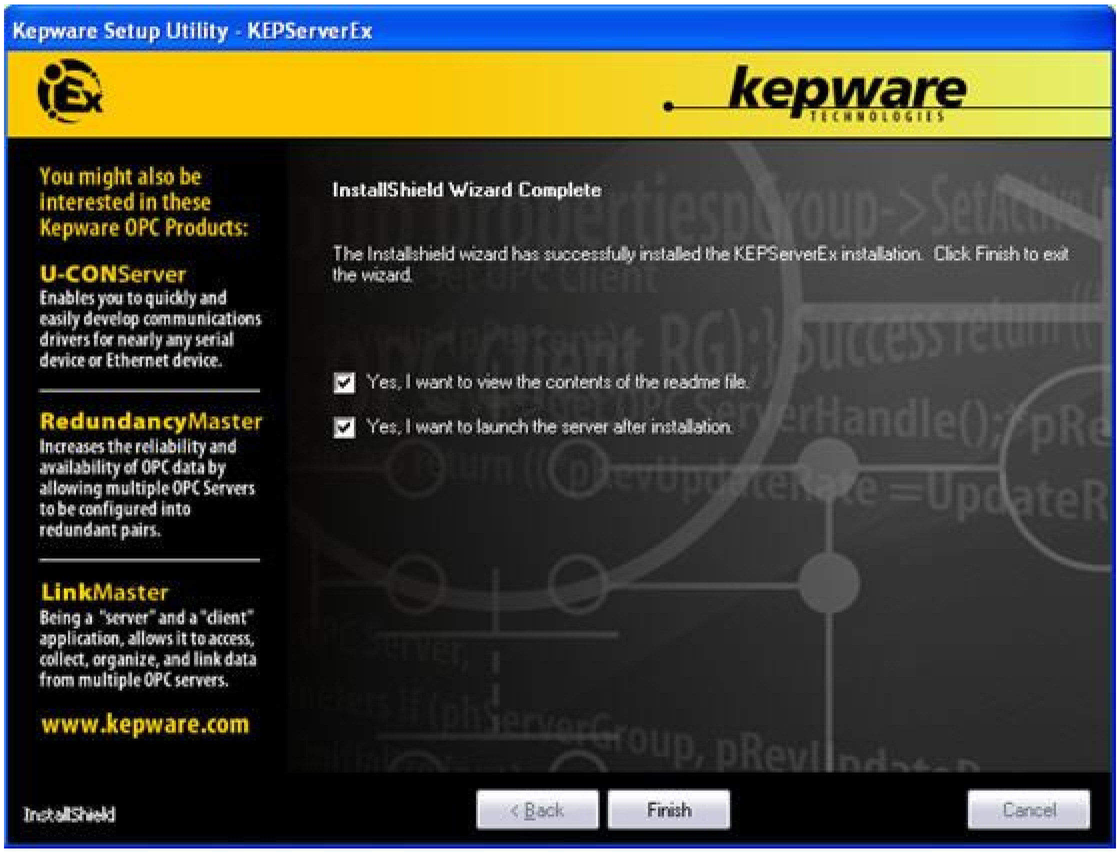

You might also be interested in these Kepware OPC Products:
U-CONServer
Enables you to quickly and easily develop communications drivers for nearly any serial device or Ethernet device.
RedundancyMaster
Increases the reliability and availability of OPC data by allowing multiple OPC Servers to be configured into redundant pairs.
LinkMaster
Being a "server" and a "client" application, allows it to access, collect, organize, and link data from multiple OPC servers.
www.kepware.com
InstallShield
Cancel

KEPServerEX is configuring your new software installation.

Installing

C:\...\{98C52021-37A8-45DE-B80A-52DE27E99593}\setup.ini

The installation will perform.


Kepware Setup Utility - KEPServerEx
kepware TECHNOLOGIES
You might also be interested in these Kepware OPC Products:
U-CONServer
Enables you to quickly and easily develop communications drivers for nearly any serial device or Ethernet device.
RedundancyMaster
Increases the reliability and availability of OPC data by allowing multiple OPC Servers to be configured into redundant pairs.
LinkMaster
Being a "server" and a "client" application, allows it to access, collect, organize, and link data from multiple OPC servers.
www.kepware.com
InstallShield
< Back
Finish
Cancel

InstallShield Wizard Complete

The Installshield wizard has successfully installed the KEPServerEx installation. Click Finish to exit the wizard.

☑ Yes, I want to view the contents of the readme file.

☑ Yes, I want to launch the server after installation.

Once complete, you will be able to finish the install, and launch the server.

At this point the KEPServer application may launch. If you simply want to use the simulator, there is nothing else you need to do. Otherwise, you may configure your device communications now. Refer to the KEPSever help file for more information on driver configuration.

Install the Database Server

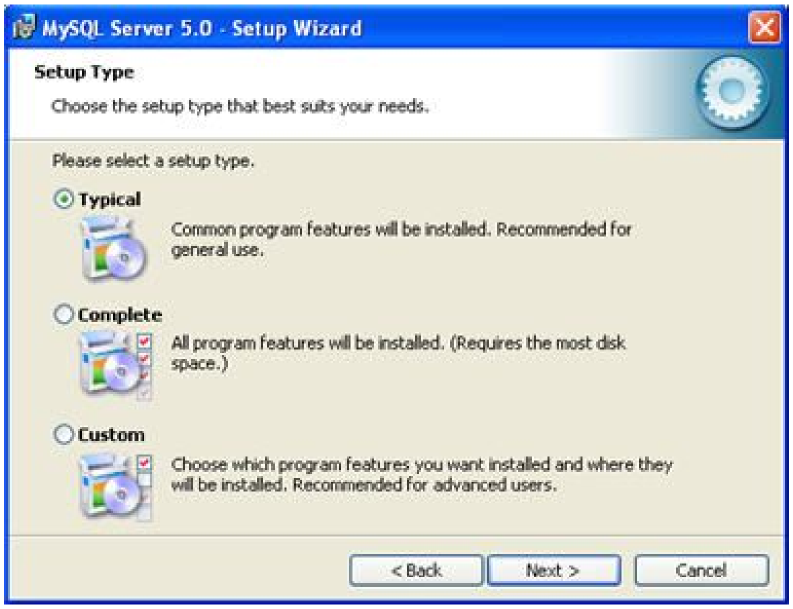
FactorySQL comes ready to talk to Microsoft SQL Server, MySQL, PostgreSQL, Oracle, and Microsoft Access databases. It can talk to other systems through ODBC, though some systems may require specialized definition files in order to reconcile differences in their support of SQL.

For purposes of our quick start tutorial here we will use MySQL, a powerful and free database sytem. It is included in the installer, or may be downloaded from www.mysql.com. MySQL does not automatically install a frontend- you must install the MySQL Query Browser seperately, or use a third party program such as DBManager Pro, available at www.dbtools.com.br.

The following are the steps for installing and using MySQL:

Step 1: Install MySQL

The MySQL installer will be launched if selected during FactorySQL installation. Otherwise, you may download the installer and run the executable manually.



MySQL Server 5.0 - Setup Wizard

Setup Type

Choose the setup type that best suits your needs.

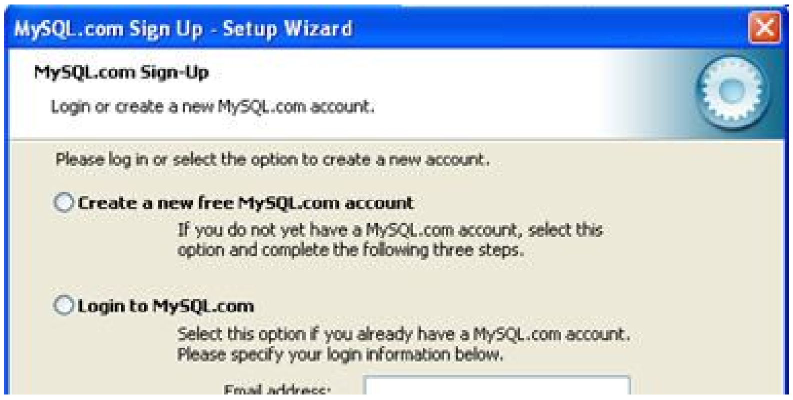
Please select a setup type.

[✓ Typical] Common program features will be installed. Recommended for general use.

Complete: All program features will be installed. (Requires the most disk space.)

Custom: Choose which program features you want installed and where they will be installed. Recommended for advanced users.

The installation process is very simple if you choose 'Typical'. This option should suffice for most installations. For more control, you may want to choose one of the other options.



MySQL.com Sign Up - Setup Wizard

MySQL.com Sign-Up

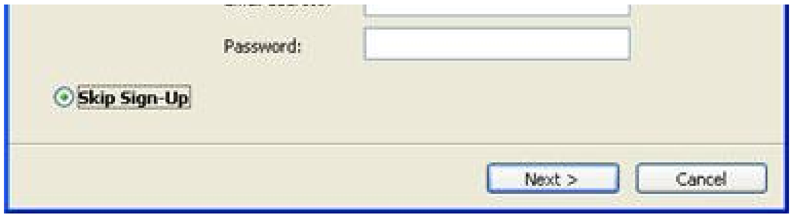
Login or create a new MySQL.com account.

Please log in or select the option to create a new account.

Create a new free MySQL.com account: If you do not yet have a MySQL.com account, select this option and complete the following three steps.

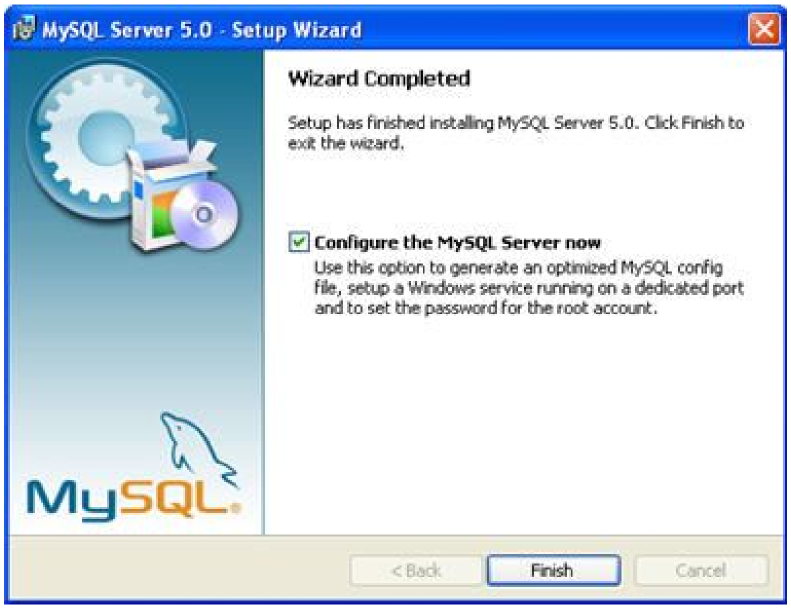
Login to MySQL.com: Select this option if you already have a MySQL.com account. Please specify your login information below.

Email address:


Password:

[✓ Skip Sign-Up]

Once installation is complete, you will be asked to create a MySQL.com account. This is completely optional and not necessary. Click Next to continue.


MySQL Server 5.0 - Setup Wizard

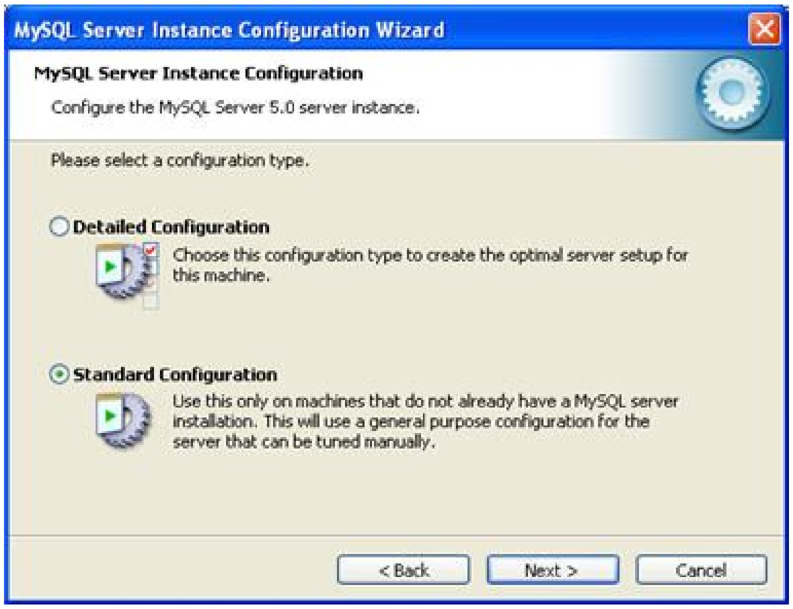
Wizard Completed

Setup has finished installing MySQL Server 5.0. Click Finish to exit the wizard.

[✓ Configure the MySQL Server now]
Use this option to generate an optimized MySQL config file, setup a Windows service running on a dedicated port and to set the password for the root account.

The final screen will ask if you want to configure MySQL now. Make sure this option is selected, and click "Finish". Note: If you accidently de-select it and click 'Finish', you can launch the configuration wizard later from the start menu.

Step 2: Configuring MySQL


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

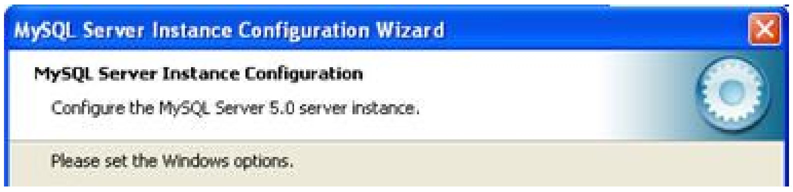
Configure the MySQL Server 5.0 server instance.

Please select a configuration type.

Detailed Configuration
Choose this configuration type to create the optimal server setup for this machine.

[✓ Standard Configuration]
Use this only on machines that do not already have a MySQL server installation. This will use a general purpose configuration for the server that can be tuned manually.

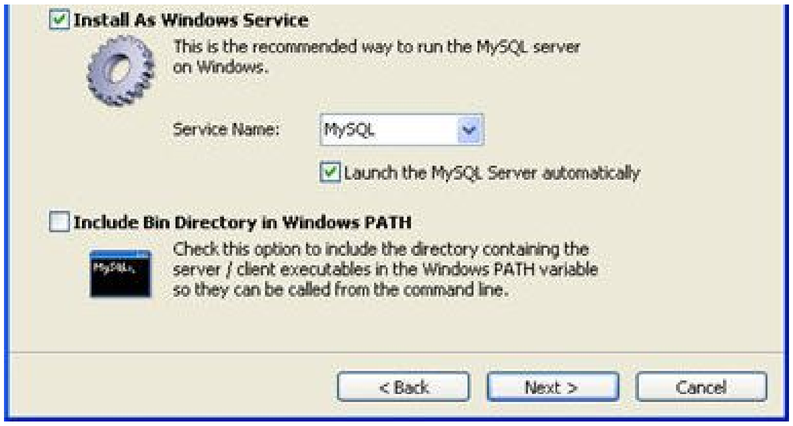
In most cases, you can simply choose 'Standard Configuration' and continue.


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Please set the Windows options.


☑ Install As Windows Service

This is the recommended way to run the MySQL server on Windows.

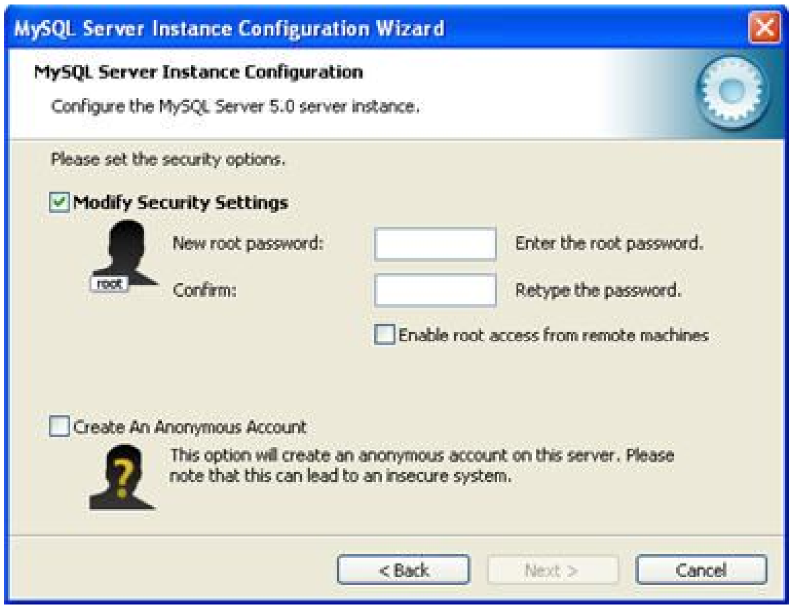
Service Name: MySQL

☑ Launch the MySQL Server automatically

☐ Include Bin Directory in Windows PATH

Check this option to include the directory containing the server / client executables in the Windows PATH variable so they can be called from the command line.

Leave this screen as default. Select "Next."



MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Please set the security options.

☑ Modify Security Settings

New root password: Enter the root password.

Confirm: Retype the password.

☐ Enable root access from remote machines

☐ Create An Anonymous Account

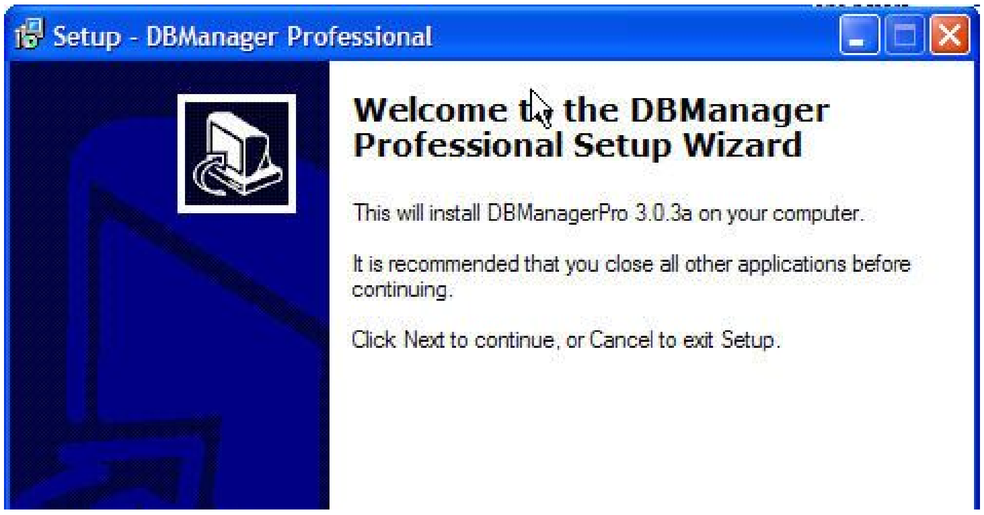
This option will create an anonymous account on this server. Please note that this can lead to an insecure system.

On this screen you can must set the password for the *root* account. The root account is the only account created by the installer, and is the highest level of authority. This password should be secure and easy to remember.

On the final screen, hit "Execute" to run the configuration. If everything is successful, your server should be up and running, and ready to connect to.

Step 3: Install DBManager Professional

After downloading DBManager, run the executable to begin installation.

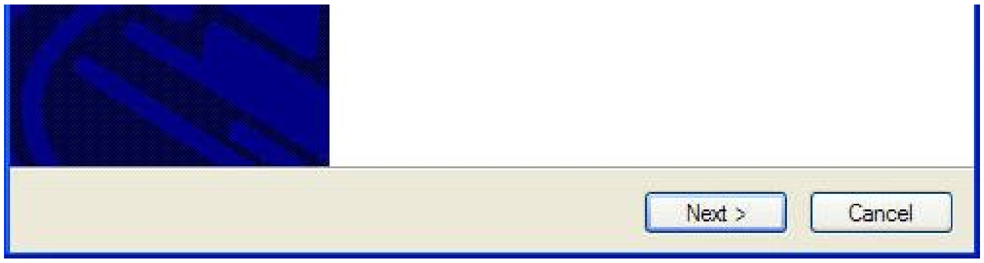


Welcome to the DBManager Professional Setup Wizard

This will install DBManagerPro 3.0.3a on your computer.

It is recommended that you close all other applications before continuing.

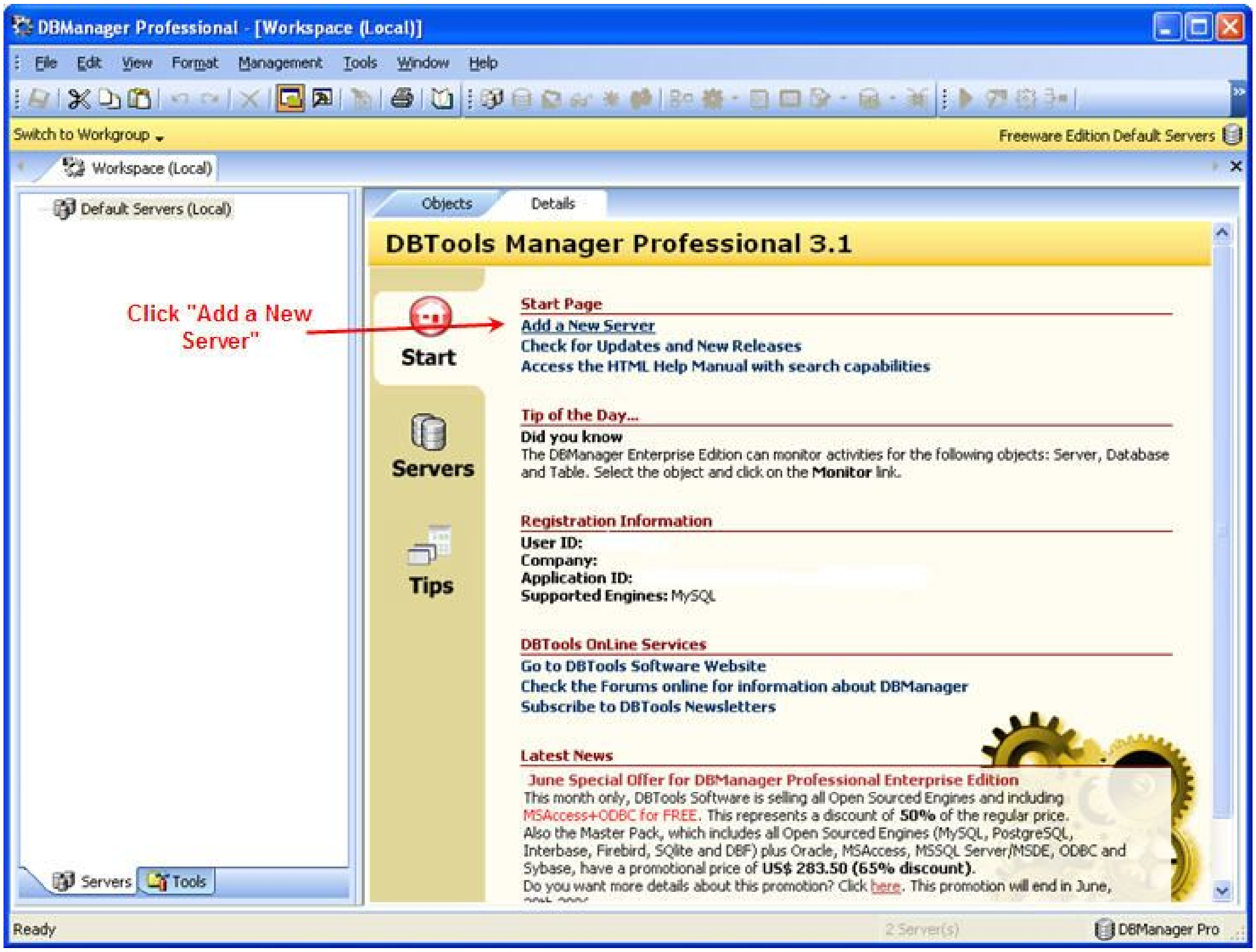
Click Next to continue, or Cancel to exit Setup.



Select "Next" and complete the installation process.

Step 4: Connecting and Verifying Server

Open DBManager Professional. It should look as follows. Add the MySQL server by clicking the link "Add a new server." Conversely, you can use the menu bar option "Tools" -> "Server" -> "Server Manager."



DBTools Manager Professional 3.1

Start Page
Add a New Server
Check for Updates and New Releases
Access the HTML Help Manual with search capabilities

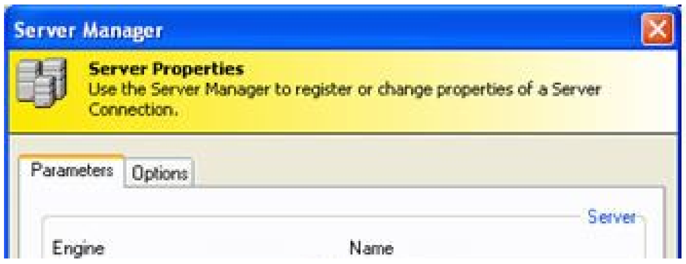
Tip of the Day...
Did you know
The DBManager Enterprise Edition can monitor activities for the following objects: Server, Database and Table. Select the object and click on the **Monitor** link.

Registration Information
User ID:
Company:
Application ID:
Supported Engines: MySQL

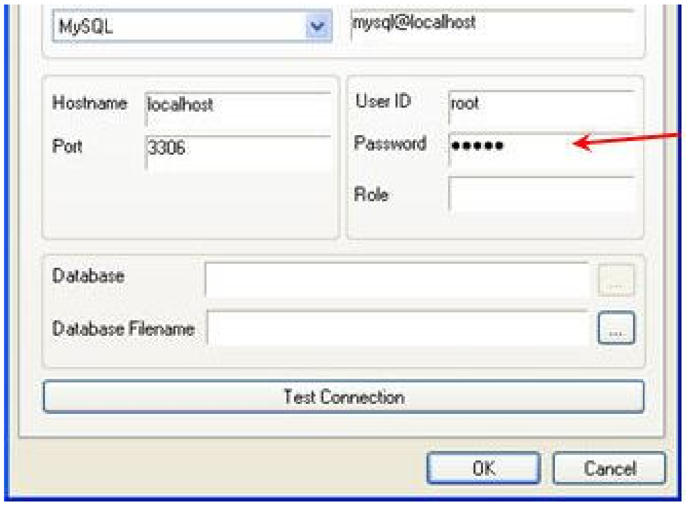
DBTools OnLine Services
Go to DBTools Software Website
Check the Forums online for information about DBManager
Subscribe to DBTools Newsletters

Latest News
June Special Offer for DBManager Professional Enterprise Edition
This month only, DBTools Software is selling all Open Sourced Engines and including MSAccess+ODBC for FREE. This represents a discount of **50%** of the regular price.
Also the Master Pack, which includes all Open Sourced Engines (MySQL, PostgreSQL, Interbase, Firebird, SQlite and DBF) plus Oracle, MSAccess, MSSQL Server/MSDE, ODBC and Sybase, have a promotional price of **US$ 283.50 (65% discount)**.
Do you want more details about this promotion? Click here. This promotion will end in June,

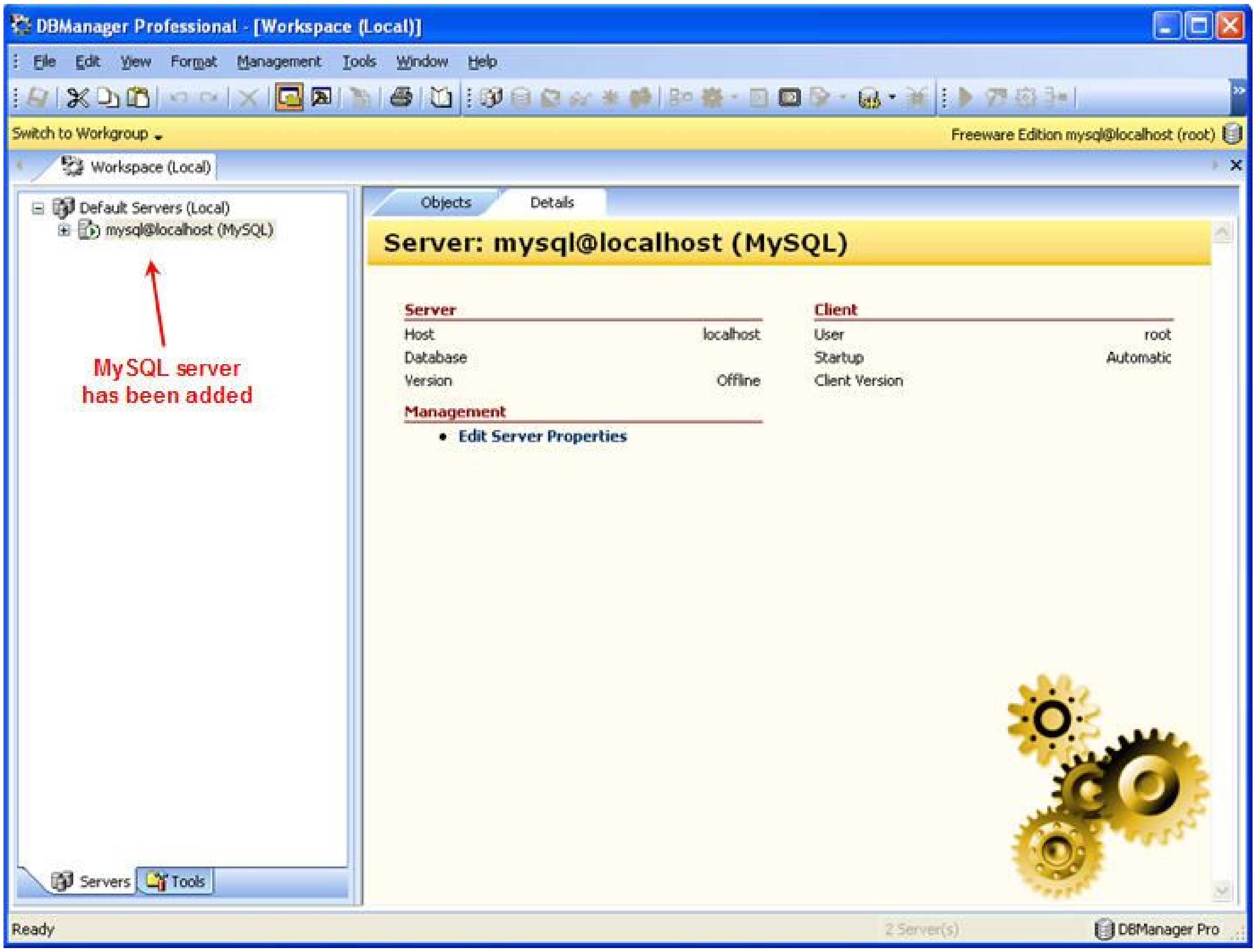
Now type in the password that you created for the root account during the MySQL installation process. Leave all the other settings as default. Click "Test connection." You should see a message that says "Connection successful."



Server Properties
Use the Server Manager to register or change properties of a Server Connection.


MySQL
mysql@localhost
Hostname localhost
Port 3306
User ID root
Password •••••
Role
Database
Database Filename

Now you can see the server you just added.


DBManager Professional - [Workspace (Local)]
File Edit View Format Management Tools Window Help
Switch to Workgroup
Freeware Edition mysql@localhost (root)
Workspace (Local)
Default Servers (Local)
mysql@localhost (MySQL)
MySQL server has been added
Objects
Details
Servers
Tools
Ready
2 Server(s)
DBManager Pro

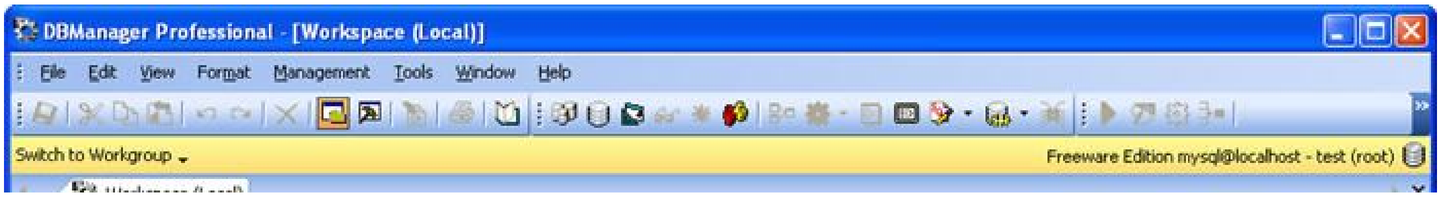
Server: mysql@localhost (MySQL)

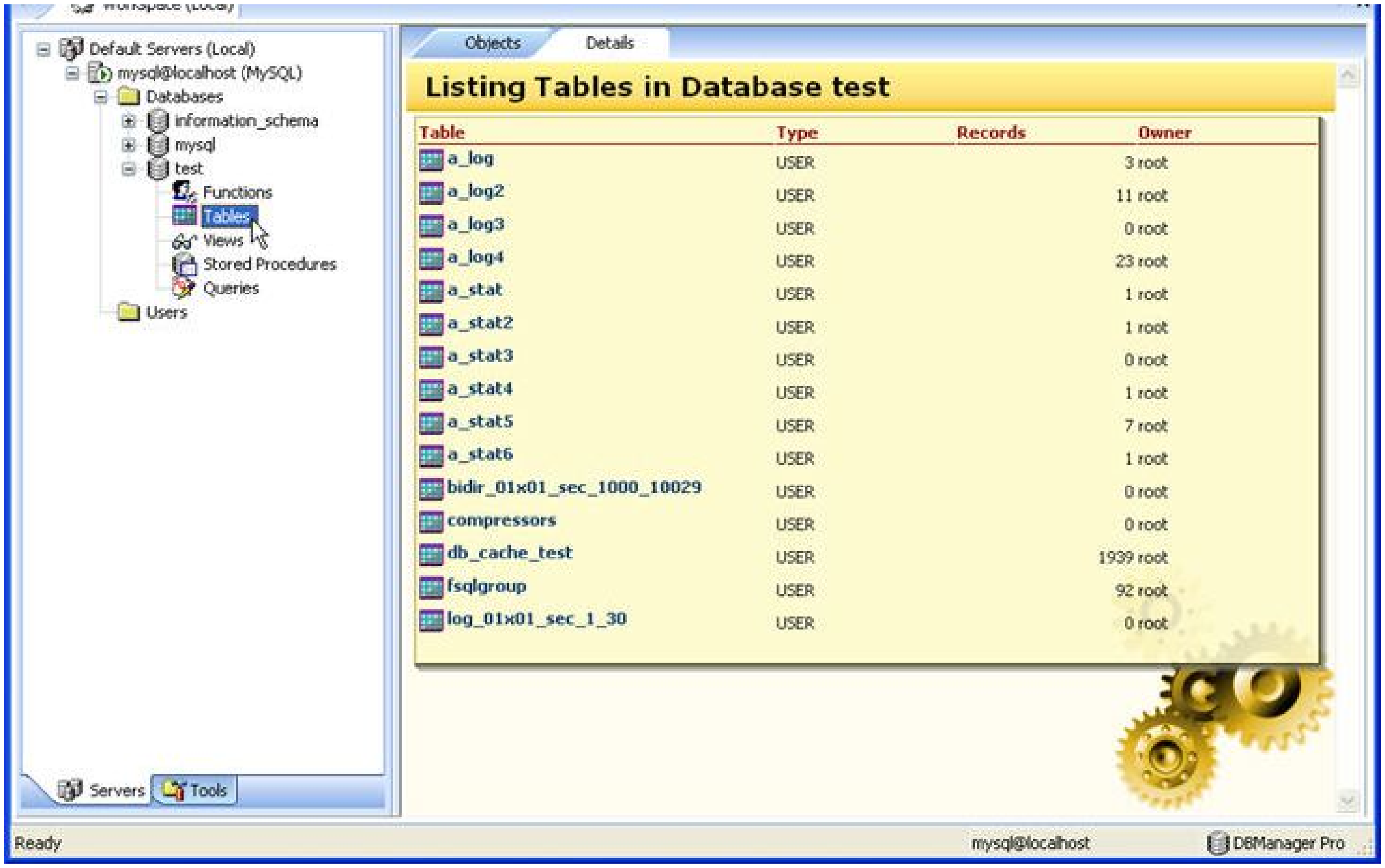
Server
Host localhost
Database
Version Offline

Client
User root
Startup Automatic
Client Version

Management
- Edit Server Properties

Now you can explore the MySQL database using DBManager Professional. Simply expand the tree and drill down as shown. You can add multiple databases to the server by right clicking on the databases folder. Then tables can be added to these databases, however, FactorySQL will do this for you automatically. It will create the the datatable and its fields as you name them in FactorySQL.

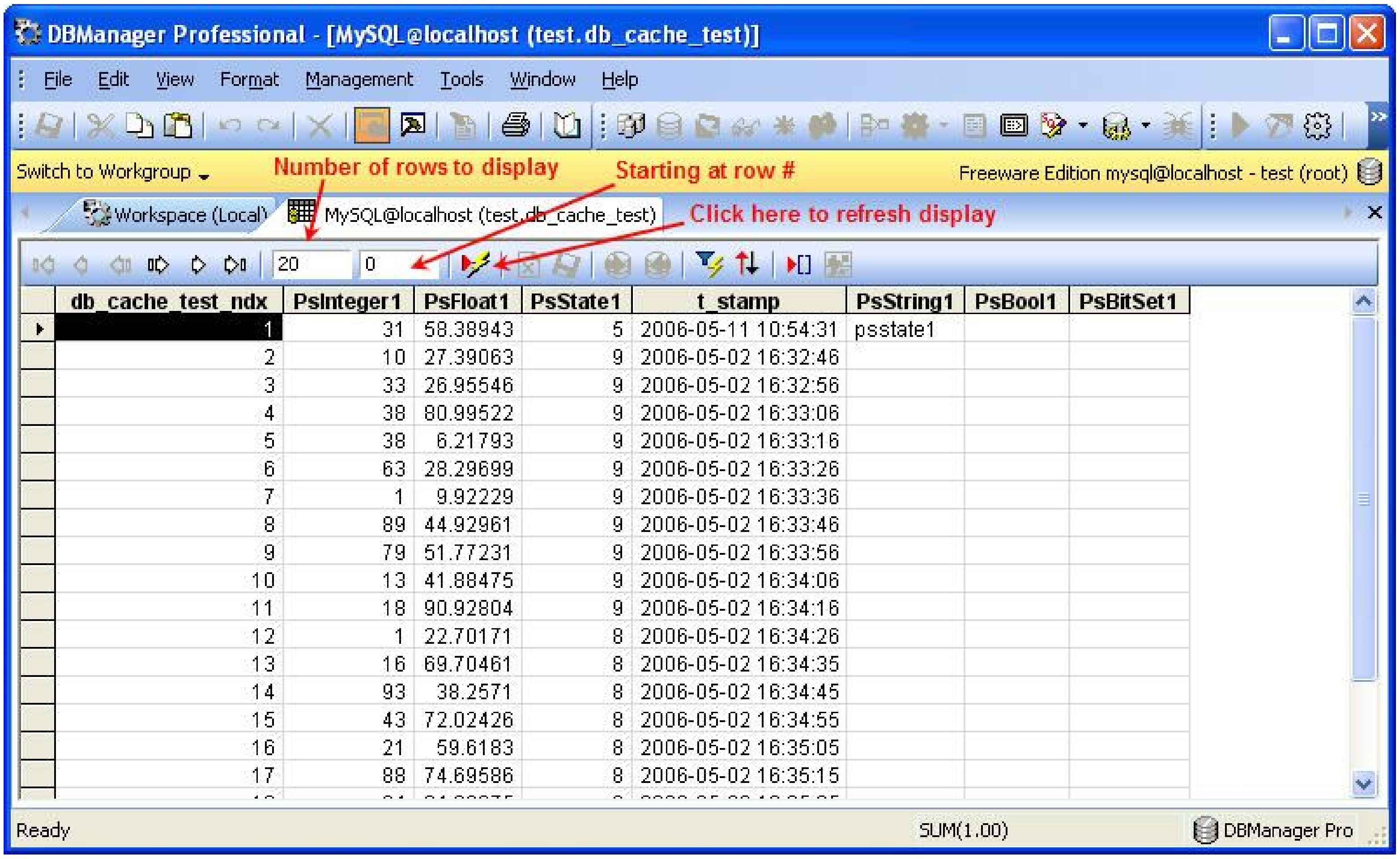

DBManager Professional - [Workspace (Local)]
File Edit View Format Management Tools Window Help
Switch to Workgroup
Freeware Edition mysql@localhost - test (root)



Listing Tables in Database test

Table	Type	Records	Owner
a_log	USER	3	root
a_log2	USER	11	root
a_log3	USER	0	root
a_log4	USER	23	root
a_stat	USER	1	root
a_stat2	USER	1	root
a_stat3	USER	0	root
a_stat4	USER	1	root
a_stat5	USER	7	root
a_stat6	USER	1	root
bidir_01x01_sec_1000_10029	USER	0	root
compressors	USER	0	root
db_cache_test	USER	1939	root
fsqlgroup	USER	92	root
log_01x01_sec_1_30	USER	0	root

Double clicking on the table name in the objects tab will give you a table viewer with the tables contects. You can modify the table contents from this viewer.


DBManager Professional - [MySQL@localhost (test.db_cache_test)]
File
Edit
View
Format
Management
Tools
Window
Help
Switch to Workgroup
Number of rows to display
Starting at row #
Freeware Edition mysql@localhost - test (root)
Workspace (Local)
MySQL@localhost (test.db_cache_test)
Click here to refresh display
20
0
Ready
SUM(1.00)
DBManager Pro

db_cache_test_ndx	PsInteger1	PsFloat1	PsState1	t_stamp	PsString1	PsBool1	PsBitSet1
1	31	58.38943	5	2006-05-11 10:54:31	psstate1		
2	10	27.39063	9	2006-05-02 16:32:46			
3	33	26.95546	9	2006-05-02 16:32:56			
4	38	80.99522	9	2006-05-02 16:33:06			
5	38	6.21793	9	2006-05-02 16:33:16			
6	63	28.29699	9	2006-05-02 16:33:26			
7	1	9.92229	9	2006-05-02 16:33:36			
8	89	44.92961	9	2006-05-02 16:33:46			
9	79	51.77231	9	2006-05-02 16:33:56			
10	13	41.88475	9	2006-05-02 16:34:06			
11	18	90.92804	9	2006-05-02 16:34:16			
12	1	22.70171	8	2006-05-02 16:34:26			
13	16	69.70461	8	2006-05-02 16:34:35			
14	93	38.2571	8	2006-05-02 16:34:45			
15	43	72.02426	8	2006-05-02 16:34:55			
16	21	59.6183	8	2006-05-02 16:35:05			
17	88	74.69586	8	2006-05-02 16:35:15			

This completes the database installation. The next step is to install FactorySQL!

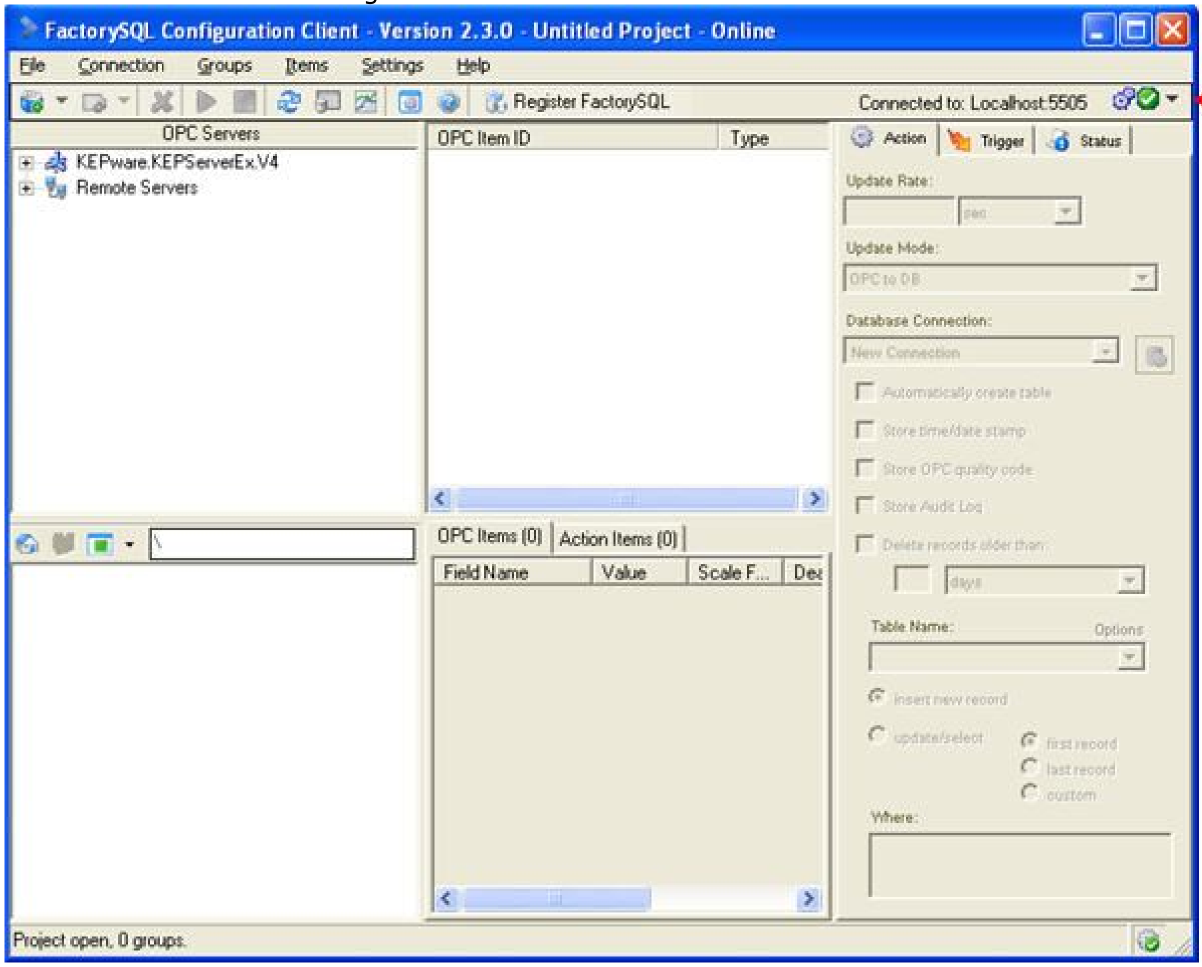
Start Simple Logging

FactorySQL is very simple to configure. What follows is a simple logging application which will take just a couple of minutes to setup. In this case we will use the previous steps of Quick Start and build on them.

It is assumed you have already installed MySQL, the DBManager frontend, KEPServerEX, and FactorySQL.   This example will use the simulated data that Kepware provides by default, but if you've establish communication with a device already, feel free to use it.

Step 1: Open the Frontend, Connect to Service

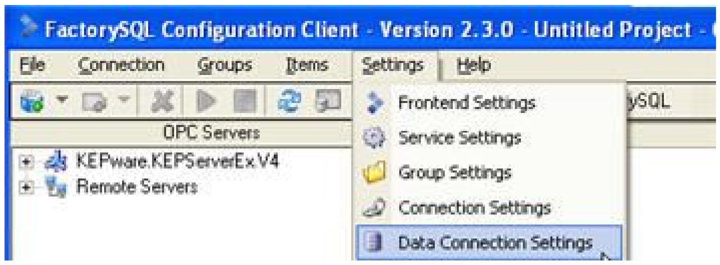
Open the FactorySQL Frontend from the start menu, or by right-clicking on the tray application. When it opens, it should look like the following screen shot:



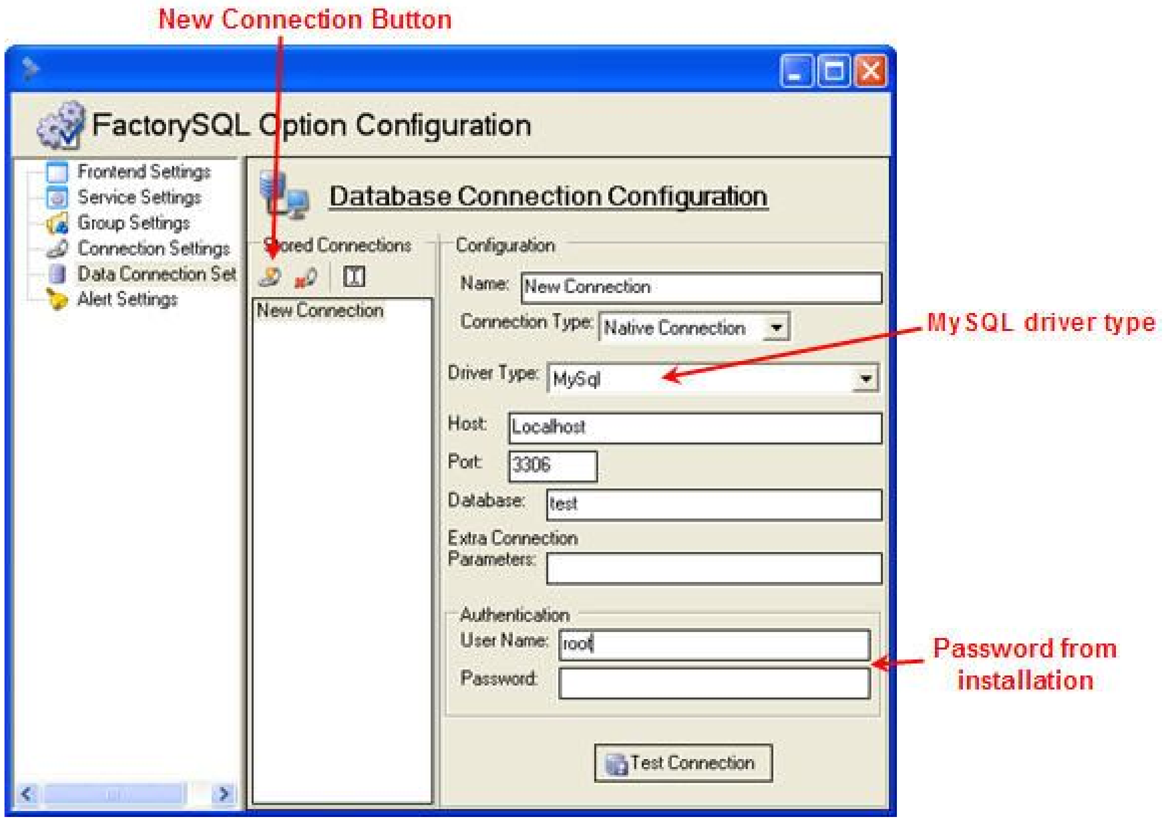
The frontend and the service are seperate applications. The frontend is capable of talking to any service, so it is necessary to first connect to the one you wish to configure. By default, a connection is made to the local service automatically, indicated by a green icon in the upper right.

Step 2: Setup Database Connection

This step will only need to be performed 1 time. You must define a "Data Connection" that FactorySQL can use. To do this, click on "Data Settings" from the main Settings menu:

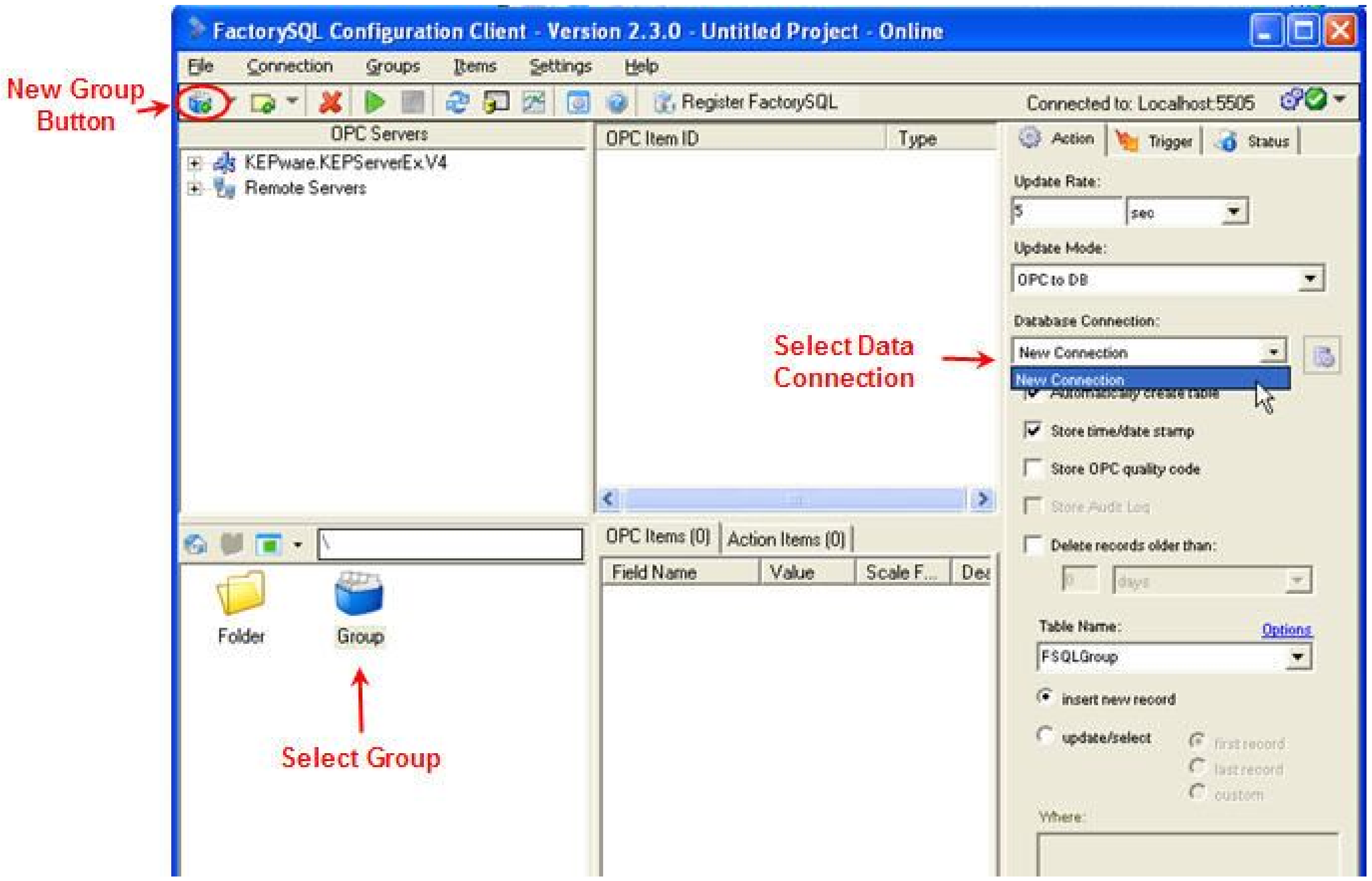


Click on the "New Connection" button, choose the MySql driver type, and fill out the details like you did to connect in DBTools. Once you've tested successfully, close the window.

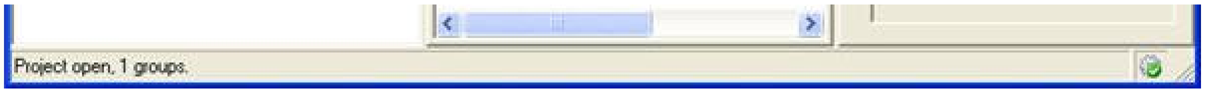


Name: New Connection
Connection Type: Native Connection
Driver Type: MySql
Host: Localhost
Port: 3306
Database: test
Extra Connection Parameters:
Authentication
User Name: root
Password:

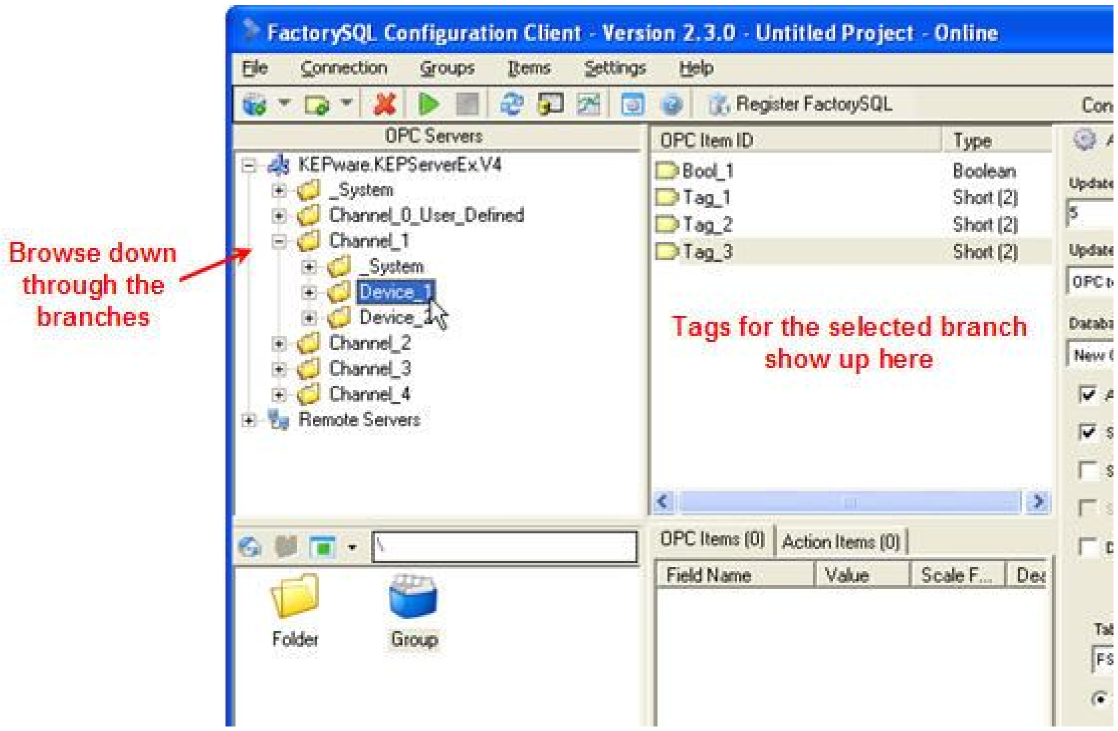
Now, select the Main group folder and click the "New Group" button on the toolbar. This will create a group in the main folder.

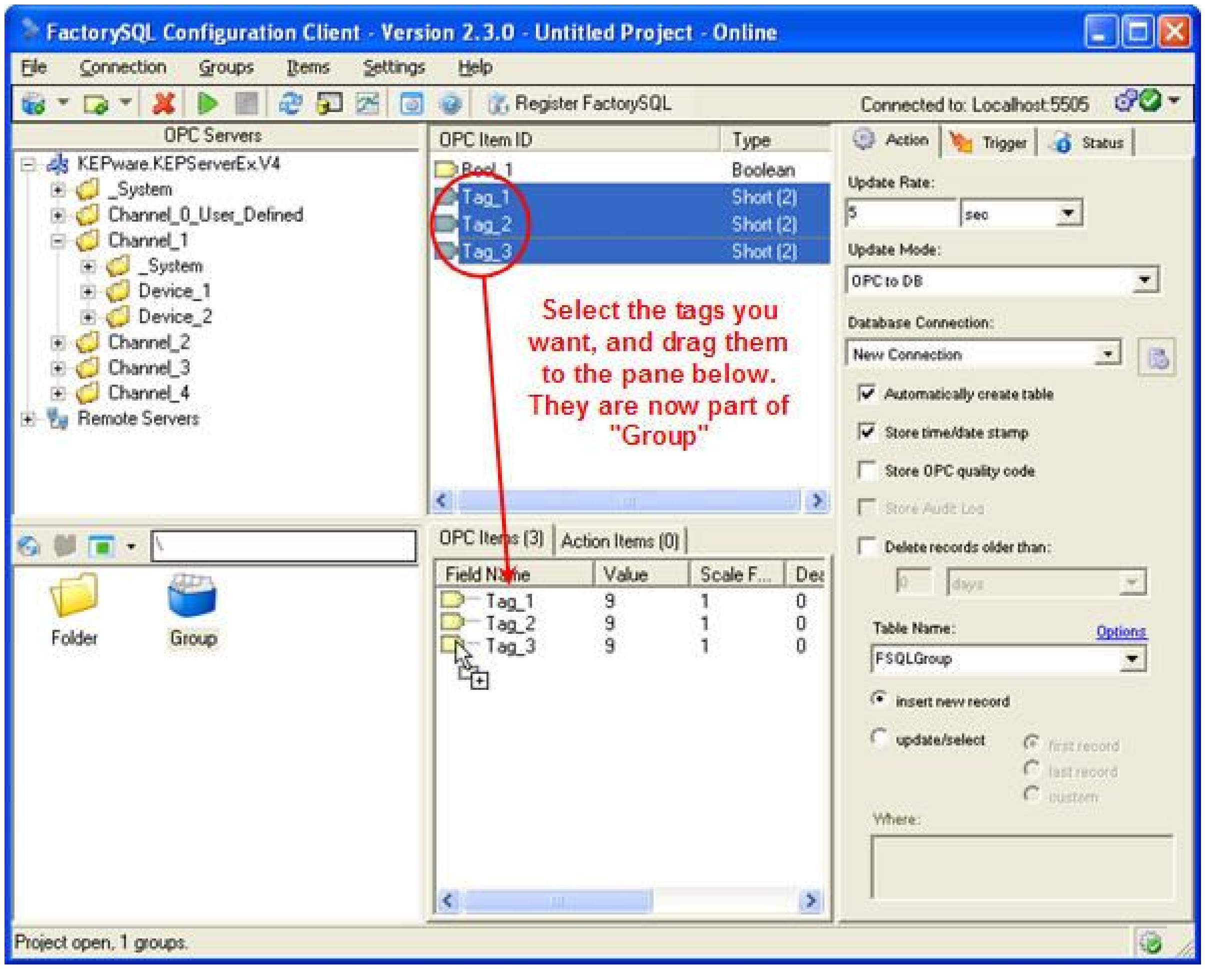


Update Rate: 5 sec
Update Mode: OPC to DB
Database Connection: New Connection
New Connection
Automatically create table
Store time/date stamp
Store OPC quality code
Store Audit Log
Delete records older than: 0 days
Table Name: FSQLGroup
insert new record
update/select
first record
last record
custom
Where:



You can now add tags by browsing through the OPC pane, selecting them, and dragging them into the Group Item Pane in the lower right:




FactorySQL Configuration Client - Version 2.3.0 - Untitled Project - Online
File Connection Groups Items Settings Help
Register FactorySQL
Connected to: Localhost:5505
OPC Servers
KEPware.KEPServerEx.V4
_System
Channel_0_User_Defined
Channel_1
_System
Device_1
Device_2
Channel_2
Channel_3
Channel_4
Remote Servers
Folder
Group
Select the tags you want, and drag them to the pane below. They are now part of "Group"
Action
Trigger
Status
Options
Project open, 1 groups.

OPC Item ID	Type
Bool_1	Boolean
Tag_1	Short (2)
Tag_2	Short (2)
Tag_3	Short (2)

OPC Items (3) | Action Items (0)

Field Name	Value	Scale F...	Dec
Tag_1	9	1	0
Tag_2	9	1	0
Tag_3	9	1	0

Update Rate: 5 sec

Update Mode: OPC to DB

Database Connection: New Connection

[✓] Automatically create table

[✓] Store time/date stamp

[ ] Store OPC quality code

[ ] Store Audit Log

[ ] Delete records older than: 0 days

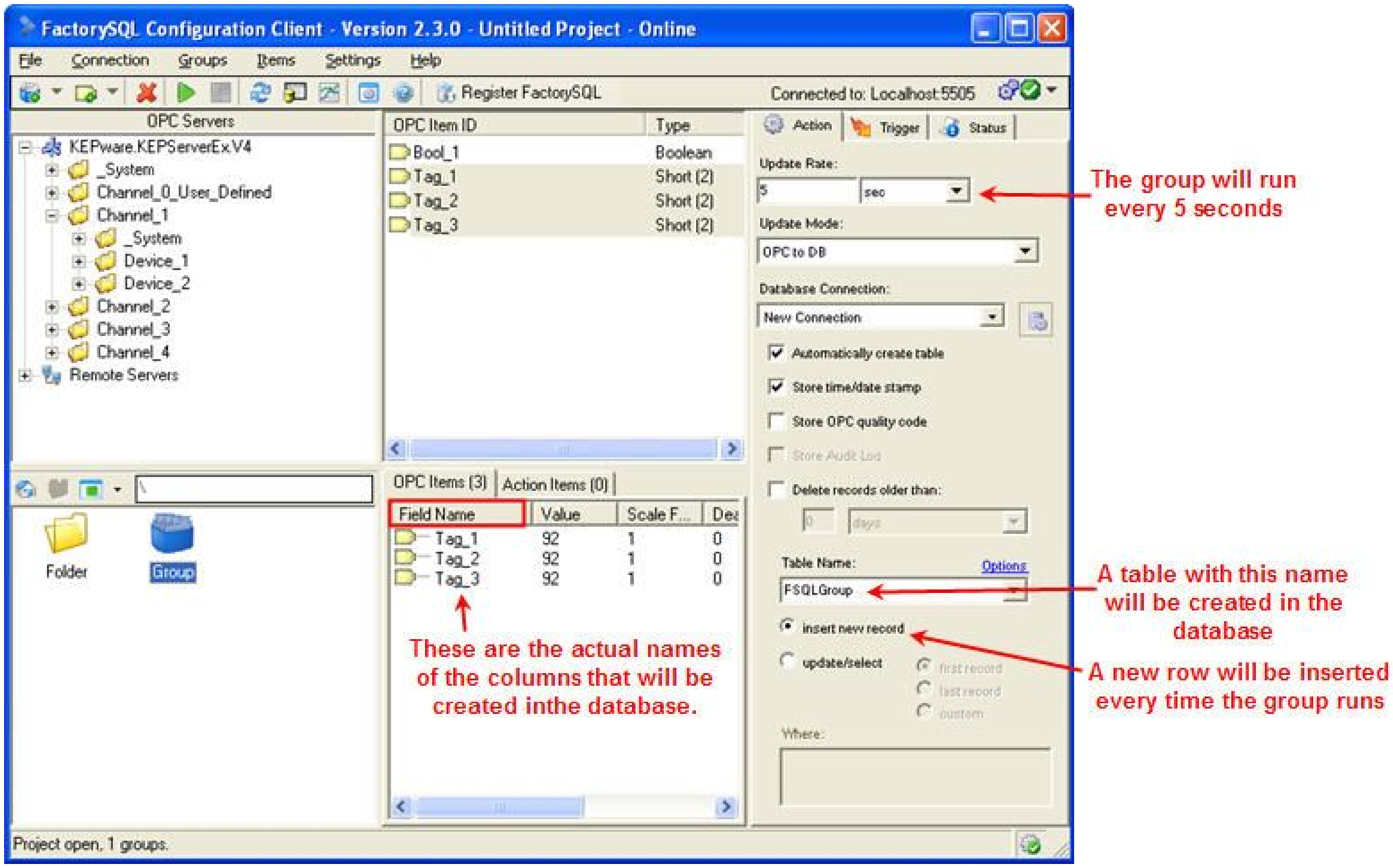
Table Name: FSQLGroup

(●) insert new record

( ) update/select — (●) first record ( ) last record ( ) custom

Where:

These items are now part of the group you created. At this point, your group is ready to execute. Before you do so though, take a second to look at all of the settings and realize what will happen:


FactorySQL Configuration Client - Version 2.3.0 - Untitled Project - Online
File Connection Groups Items Settings Help
Register FactorySQL
Connected to: Localhost:5505
OPC Servers
KEPware.KEPServerEx.V4
_System
Channel_0_User_Defined
Channel_1
_System
Device_1
Device_2
Channel_2
Channel_3
Channel_4
Remote Servers
Folder
Group
Project open, 1 groups.
The group will run every 5 seconds
These are the actual names of the columns that will be created inthe database.
A table with this name will be created in the database
A new row will be inserted every time the group runs

OPC Item ID	Type
Bool_1	Boolean
Tag_1	Short (2)
Tag_2	Short (2)
Tag_3	Short (2)

OPC Items (3) | Action Items (0)

Field Name	Value	Scale F...	Dea
Tag_1	92	1	0
Tag_2	92	1	0
Tag_3	92	1	0

Action | Trigger | Status

Update Rate: 5 sec

Update Mode: OPC to DB

Database Connection: New Connection

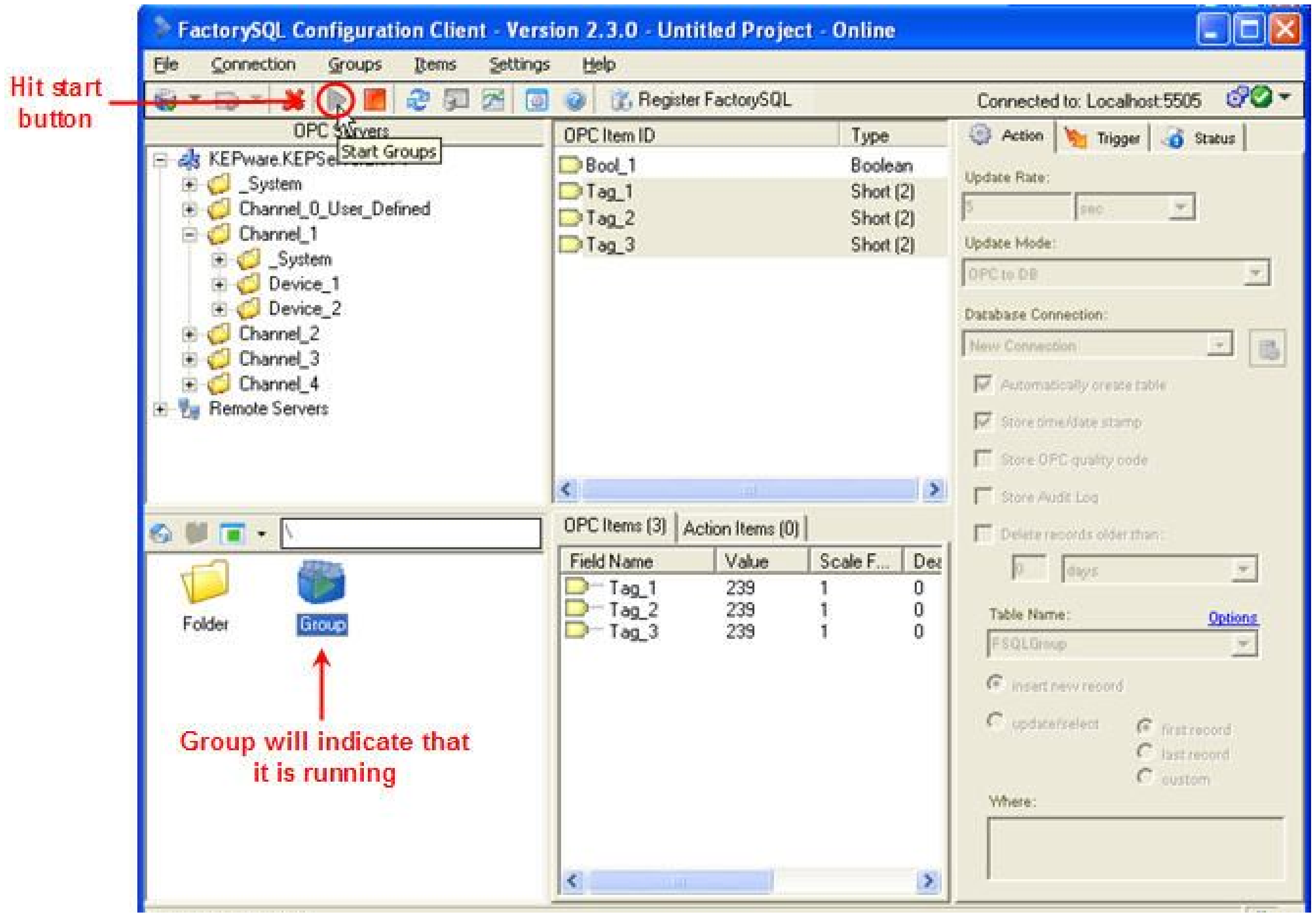
- [✓] Automatically create table
- [✓] Store time/date stamp
- [ ] Store OPC quality code
- [ ] Store Audit Log
- [ ] Delete records older than: 0 days

Table Name: FSQLGroup (Options)

(●) insert new record
( ) update/select — (●) first record ( ) last record ( ) custom

Where:

If you would like to change any aspect of group, such as the name of the table it will create, or the names of the columns, feel free to do so now. You may edit the column names by selecting the item and hitting "enter", or by double-clicking the item. When you are ready, hit the "Start" button. The group will begin to execute:



OPC Item ID	Type
Bool_1	Boolean
Tag_1	Short (2)
Tag_2	Short (2)
Tag_3	Short (2)

OPC Items (3) | Action Items (0)

Field Name	Value	Scale F...	Dea
Tag_1	239	1	0
Tag_2	239	1	0
Tag_3	239	1	0

Action | Trigger | Status

Update Rate: 5 sec

Update Mode: OPC to DB

Database Connection: New Connection

- [✓] Automatically create table
- [✓] Store time/date stamp
- [ ] Store OPC quality code
- [ ] Store Audit Log
- [ ] Delete records older than: 0 days

Table Name: FSQLGroup (Options)

(●) insert new record
( ) update/select — (●) first record ( ) last record ( ) custom

Where:

Project open, 1 groups.

Now that we are successfully logging PLC values to our database let's go to DBManager and look at the results. Browse down to the correct database, and you will see your new table to the right hand side. Double click this tablename to see the actual table. (Be sure are on the "objects" - not "details" - tab for this to work.

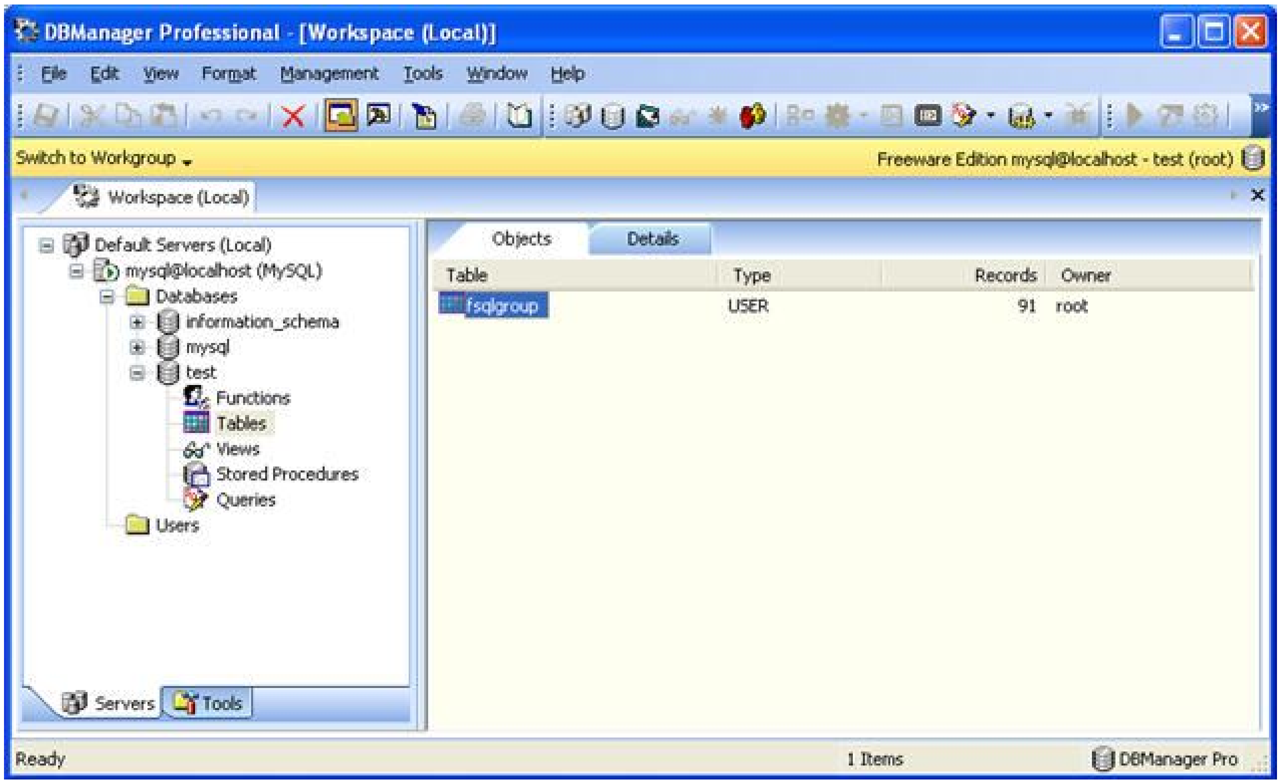
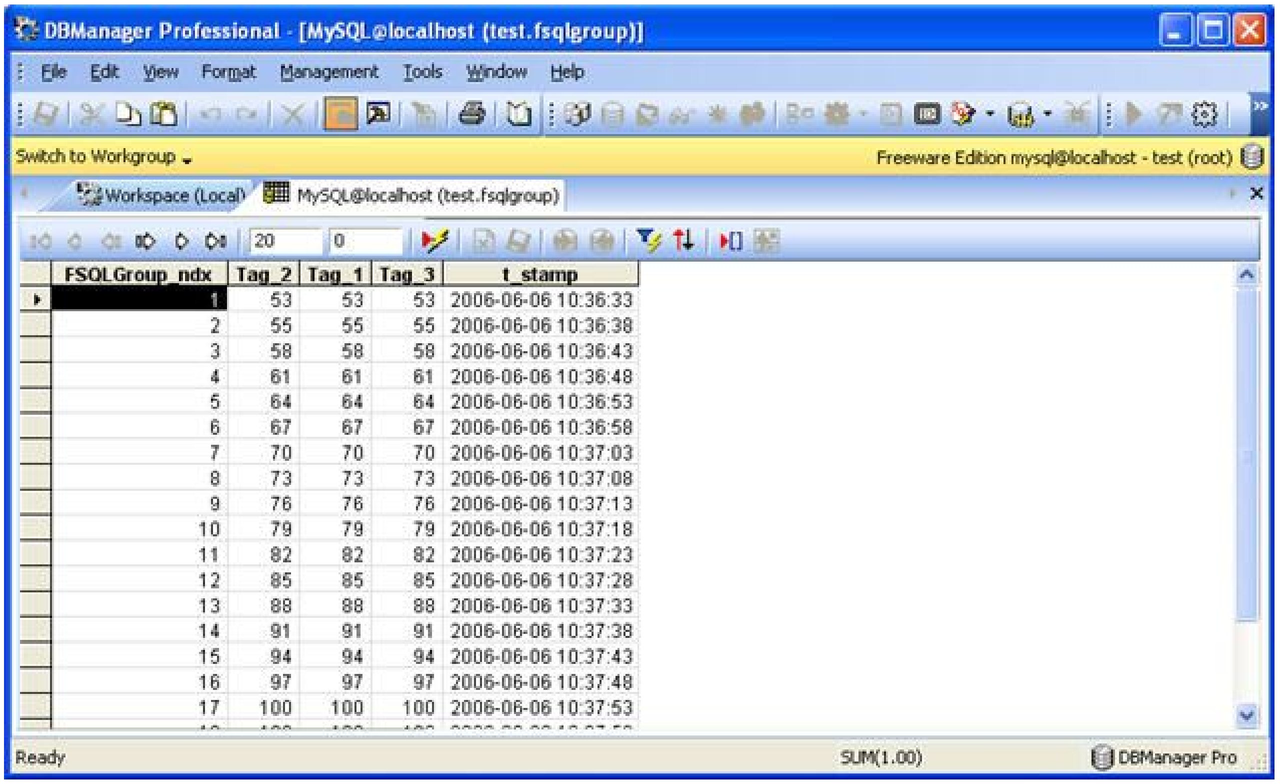


Table	Type	Records	Owner
fsqlgroup	USER	91	root

Here is the resultant table. This table can now be read by any application that interfaces with standard databases, such as various web technologies for presentation in web pages, Crystal Reports, or FactoryPMI. Once your data is in an SQL database, a whole new world of possibilities await you.


DBManager Professional - [MySQL@localhost (test.fsqlgroup)]
File Edit View Format Management Tools Window Help
Switch to Workgroup
Freeware Edition mysql@localhost - test (root)
Workspace (Local)
MySQL@localhost (test.fsqlgroup)
20
0
Ready
SUM(1.00)
DBManager Pro

FSQLGroup_ndx	Tag_2	Tag_1	Tag_3	t_stamp
1	53	53	53	2006-06-06 10:36:33
2	55	55	55	2006-06-06 10:36:38
3	58	58	58	2006-06-06 10:36:43
4	61	61	61	2006-06-06 10:36:48
5	64	64	64	2006-06-06 10:36:53
6	67	67	67	2006-06-06 10:36:58
7	70	70	70	2006-06-06 10:37:03
8	73	73	73	2006-06-06 10:37:08
9	76	76	76	2006-06-06 10:37:13
10	79	79	79	2006-06-06 10:37:18
11	82	82	82	2006-06-06 10:37:23
12	85	85	85	2006-06-06 10:37:28
13	88	88	88	2006-06-06 10:37:33
14	91	91	91	2006-06-06 10:37:38
15	94	94	94	2006-06-06 10:37:43
16	97	97	97	2006-06-06 10:37:48
17	100	100	100	2006-06-06 10:37:53

Concepts - Overview

This section attempts to break FactorySQL down into it's core components. It is organized in a logical progression, from an overview of the program, to core features, and then to other features that would be used in more advanced projects, in special cases, or that require little configuration.

Overview of Sections

User Interface Elements - Introduces the main areas of the application. Discusses some usage briefly, as it relates to the interface, but most discussion of particular features is avoided.

Frontend/Service Architecture - Discusses the split architecture of FactorySQL, what the FactorySQL service is, and how to connect to them.

Project Components - Discusses what a FactorySQL project is, and what it consists of.

Transaction Groups - Discusses everything concerning Transaction Groups, which are the core of a normal FactorySQL project.

Alerting - Introduces and explains FactorySQL's alerting system.

Scheduled Groups - Discusses another type of FactorySQL group, the Scheduled Group. These types of groups can provide useful functionality, but are not as commonly used in tipical projects.

Application Settings - This section goes over the various application level settings. In general, it will not be necessary to alter these settings to get up and running with FactorySQL.

Auditing - Discusses the FactorySQL audit log, what auditing is, and how to set it up.

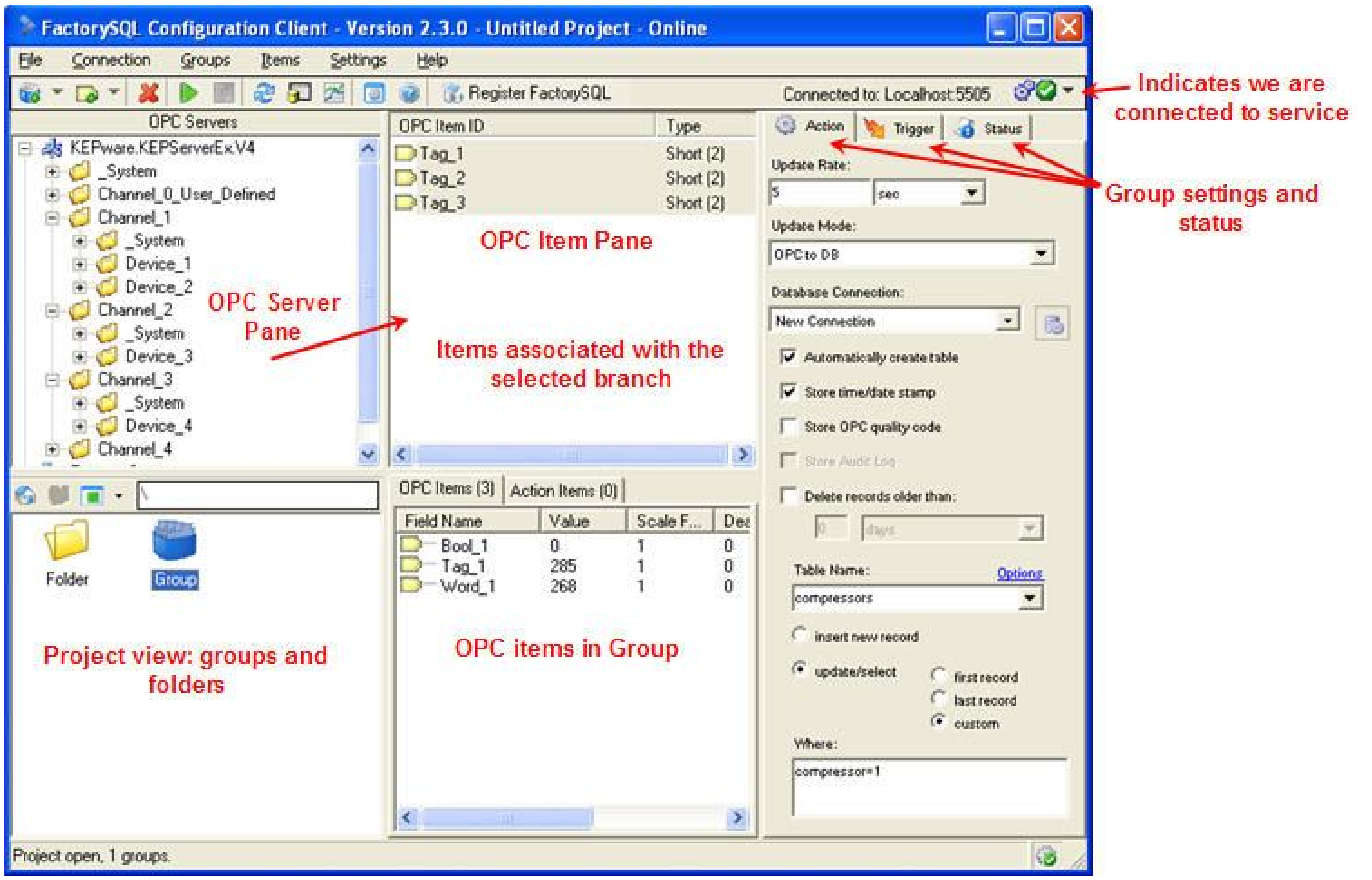
Program Status Table - Discusses the FactorySQL program status table- an optional table that can provide basic information about the state of FactorySQL.

Redundancy - Introduces and describes FactorySQL's redundancy system.

SQLTags - Describes the SQLTags system, and the options associated with it.

User Interface Overview

We'll start with a quick overview of the entire frontend. Looking at it from a high level, we see 5 main areas (the 4 windows and the right side bar). Addionally, we see the menu on top and the toolbar.



OPC Item ID	Type
Tag_1	Short (2)
Tag_2	Short (2)
Tag_3	Short (2)

Field Name	Value	Scale F...	Dea
Bool_1	0	1	0
Tag_1	285	1	0
Word_1	268	1	0

Update Rate: 5 sec

Update Mode: OPC to DB

Database Connection: New Connection

[✓] Automatically create table

[✓] Store time/date stamp

[ ] Store OPC quality code

[ ] Store Audit Log

[ ] Delete records older than: 0 days

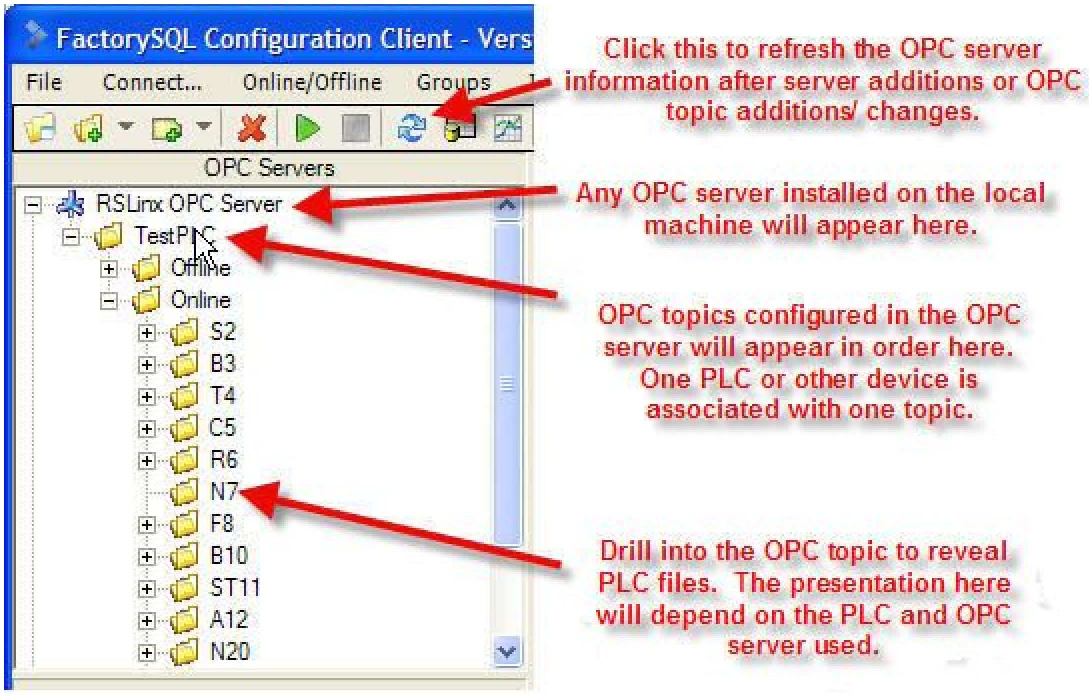
Table Name: compressors Options

( ) insert new record

(●) update/select ( ) first record ( ) last record (●) custom

Where: compressor=1

Starting in the upper right, we have the **OPC Server pane**. This window shows you all of the installed OPC servers, and allows you to browse them, if supported. Also provides you with the ability to browse servers on remote machines (security settings permitting).


FactorySQL Configuration Client - Vers
File
Connect...
Online/Offline
Groups
OPC Servers
RSLinx OPC Server
TestPLC
Offline
Online
S2
B3
T4
C5
R6
N7
F8
B10
ST11
A12
N20
Click this to refresh the OPC server information after server additions or OPC topic additions/ changes.
Any OPC server installed on the local machine will appear here.
OPC topics configured in the OPC server will appear in order here. One PLC or other device is associated with one topic.
Drill into the OPC topic to reveal PLC files. The presentation here will depend on the PLC and OPC server used.

Next, to the right, is the **OPC Item Browse pane**. This pane is used during browsing to show the items in an OPC branch. You can configure groups by dragging items directly from this window down, into the pane below...

The **OPC Group Item pane**. This pane shows all of the items that are held in the currently selected Transaction Group. A transaction group is nothing more than a group of items and specific logging instructions. All of the Transaction Groups in your project can be found directly to the left in the...

Project view pane. This area shows the elements that make up a project. Projects consist of *groups* (normal and scheduled) organized into *folders*. Folders have no properties of their own, but using the Start and Stop buttons you can affect all of the groups at and below a certain level.

There are ways to view the project view pane: Large items, small items, and treeview. The view button lets you switch between them. The home button will bring you to the top level, or you may click the Folder up button, or use the address bar, to navigate the folder structure.

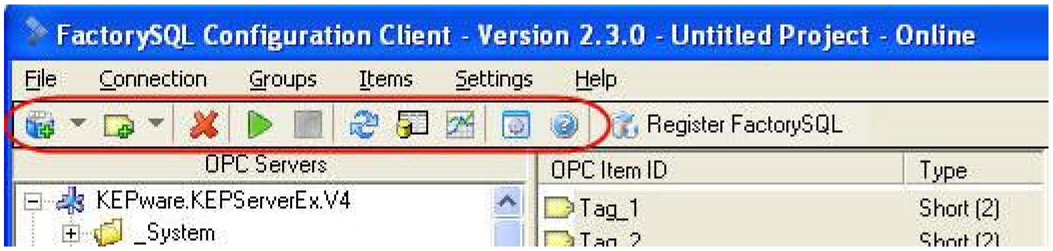
The **toolbar** provides quick access to important tasks, such as starting and stopping groups, and added new items.

On the right side of the toolbar, you see an icon indicating whether the frontend is connected to a service. If it is connected, the IP address and Port number of the service will be displayed next to it.

On the right side of the screen is the **Group Settings Tabs**, which let you configure exactly how the group will log, and view important information about its execution.

Toolbar

The **Toolbar** provides quick access to the most common features of FactorySQL.



New Group

Allows you to create a new Transaction Group, Scheduled Group, or Group Folder.

New Item

Allows you to create a new OPC, Action, or Scheduled Item (depending on context of selected group).

Delete

Deletes an element of the project, depending on where the selection focus is. Can delete **Groups**, **Group Folders**, and **Items**.

Start Logging

Allows you to create a new OPC, Action, or Scheduled Item (depending on context of selected group).

Stop Logging

Allows you to create a new OPC, Action, or Scheduled Item (depending on context of selected group).

Refresh Servers

Refreshes the OPC servers in the Server Browser Pane.

DB-To-OPC Mapper

Opens a tool that lets you quickly re-assign database fields to different OPC addresses. For more information, click Here.

Live Values

Toggles live values on & off in the Group Item Pane.

Settings

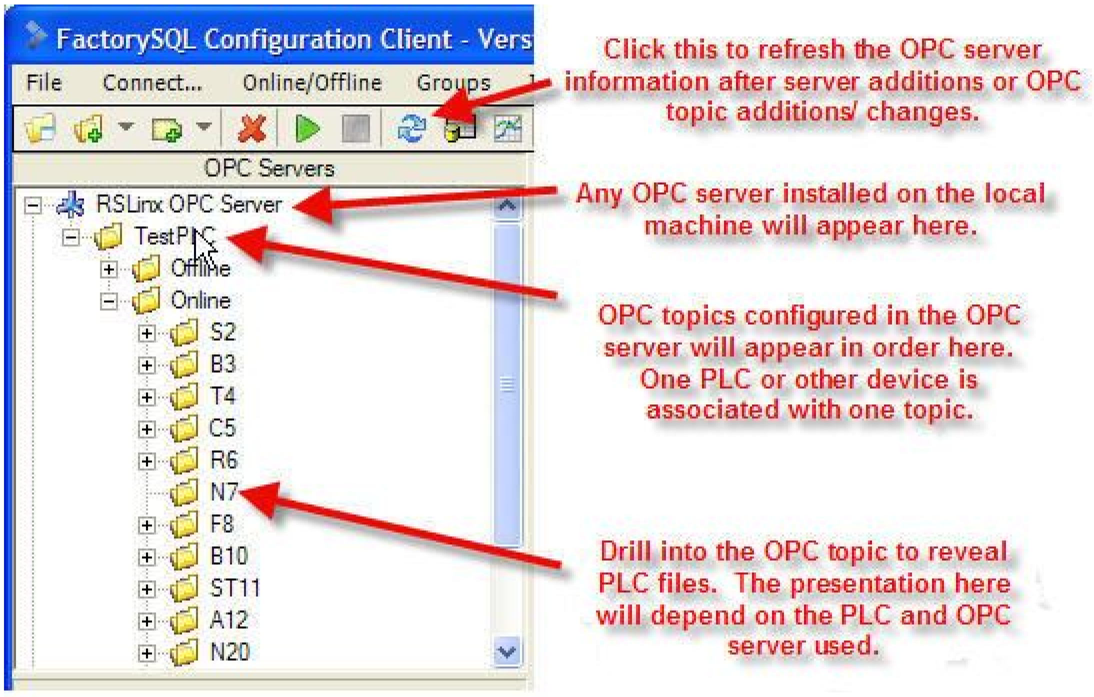
Opens the Application Settings window.

Help File

Opens this help file.

OPC Server Pane

The **OPC Server Pane** is an area on the main FactorySQL screen that allows browsing. Expanding the tree allows the user to "drill down" PLC memory. When you select a PLC file, its registers will appear in the **Browsing Pane**.

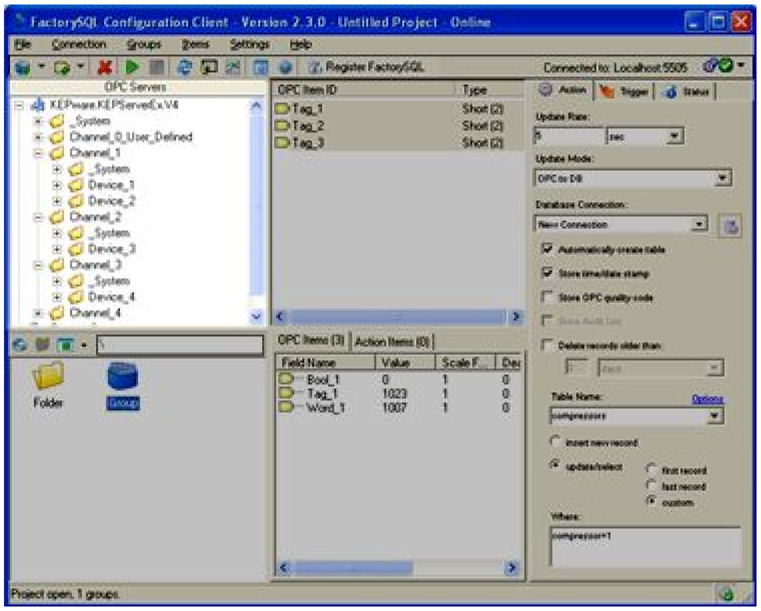


Click this to refresh the OPC server information after server additions or OPC topic additions/ changes.

Any OPC server installed on the local machine will appear here.

OPC topics configured in the OPC server will appear in order here. One PLC or other device is associated with one topic.

Drill into the OPC topic to reveal PLC files. The presentation here will depend on the PLC and OPC server used.



OPC Item ID	Type
Tag_1	Short (2)
Tag_2	Short (2)
Tag_3	Short (2)

Field Name	Value	Scale F...	Dea
Bool_1	0	1	0
Tag_1	1023	1	0
Word_1	1007	1	0

Update Rate: 5 sec

Update Mode: OPC to DB

Database Connection: New Connection

- [x] Automatically create table
- [x] Store time/date stamp
- [ ] Store OPC quality code
- [ ] Store fault log
- [ ] Delete records older than

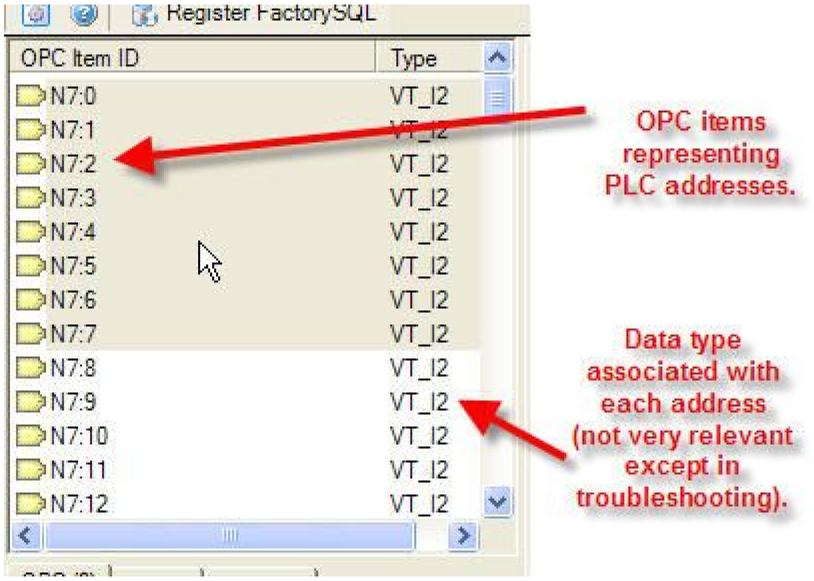
Table Name: compressors

- ( ) insert new record
- (•) update/select
  - ( ) first record
  - ( ) last record
  - (•) custom

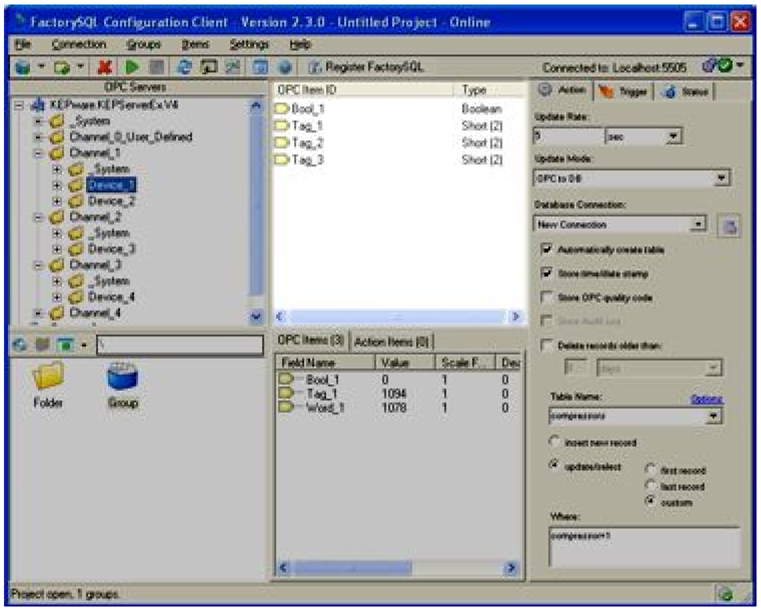
Where: compressor=1

Browsing Pane

The **Browsing Pane** is the area on the FactorySQL screen that displays OPC items while browsing**.** The easiest way to populate **transaction groups** with OPC items is to drag the items down from the **Browsing Pane** to the the **OPC Group Tab**.


Register FactorySQL
OPC items representing PLC addresses.
Data type associated with each address (not very relevant except in troubleshooting).

OPC Item ID	Type
N7:0	VT_I2
N7:1	VT_I2
N7:2	VT_I2
N7:3	VT_I2
N7:4	VT_I2
N7:5	VT_I2
N7:6	VT_I2
N7:7	VT_I2
N7:8	VT_I2
N7:9	VT_I2
N7:10	VT_I2
N7:11	VT_I2
N7:12	VT_I2



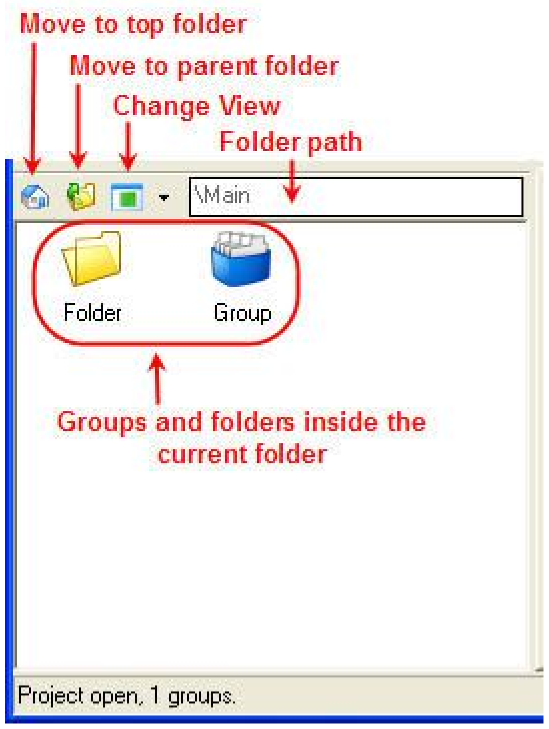
OPC Item ID	Type
Bool_1	Boolean
Tag_1	Short (2)
Tag_2	Short (2)
Tag_3	Short (2)

Field Name	Value	Scale F...	Dec
Bool_1	0	1	0
Tag_1	1094	1	0
Word_1	1078	1	0

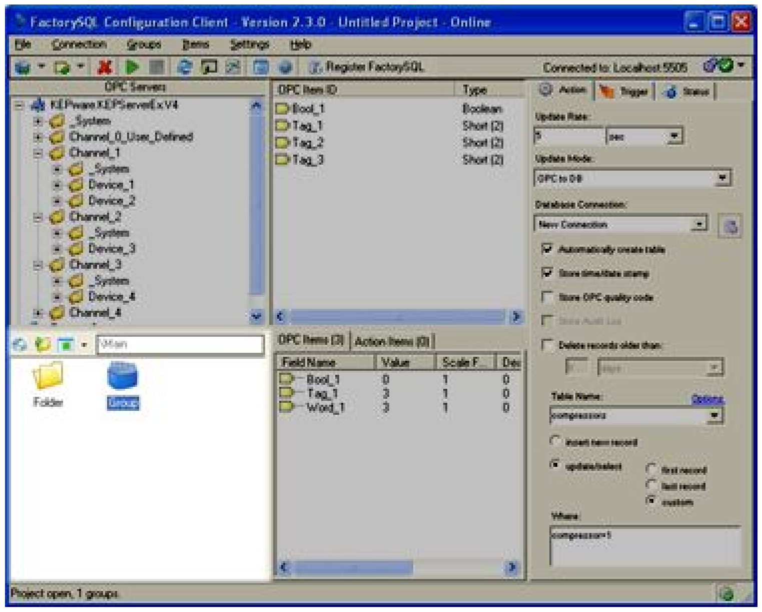
Update Rate: 5 sec
Update Mode: OPC to DB
Database Connection: New Connection
[✓] Automatically create table
[✓] Store time/date stamp
[ ] Store OPC quality code
[ ] Store audit log
[ ] Delete records older than:
Table Name: compressors
Options
( ) insert new record
(✓) update/select
( ) first record
( ) last record
(✓) custom
Where: compressor=1

Project View Pane

The **Folder Pane** is the area on the FactorySQL Frontend where the developer organizes his work. It houses **groups** and **folders** and controls which groups are running.



Project open, 1 groups.



Connected to: Localhost 5505

OPC Item ID	Type
Bool_1	Boolean
Tag_1	Short (2)
Tag_2	Short (2)
Tag_3	Short (2)

OPC Items (3) | Action Items (0)

Field Name	Value	Scale F...	Dec
Bool_1	0	1	0
Tag_1	3	1	0
Word_1	3	1	0

Update Rate: 5 sec

Update Mode: OPC to DB

Database Connection: New Connection

[✓] Automatically create table

[✓] Store time/date stamp

[ ] Store OPC quality code

[ ] Store Audit Log

[ ] Delete records older than:

Table Name: compressors Options

( ) insert new record

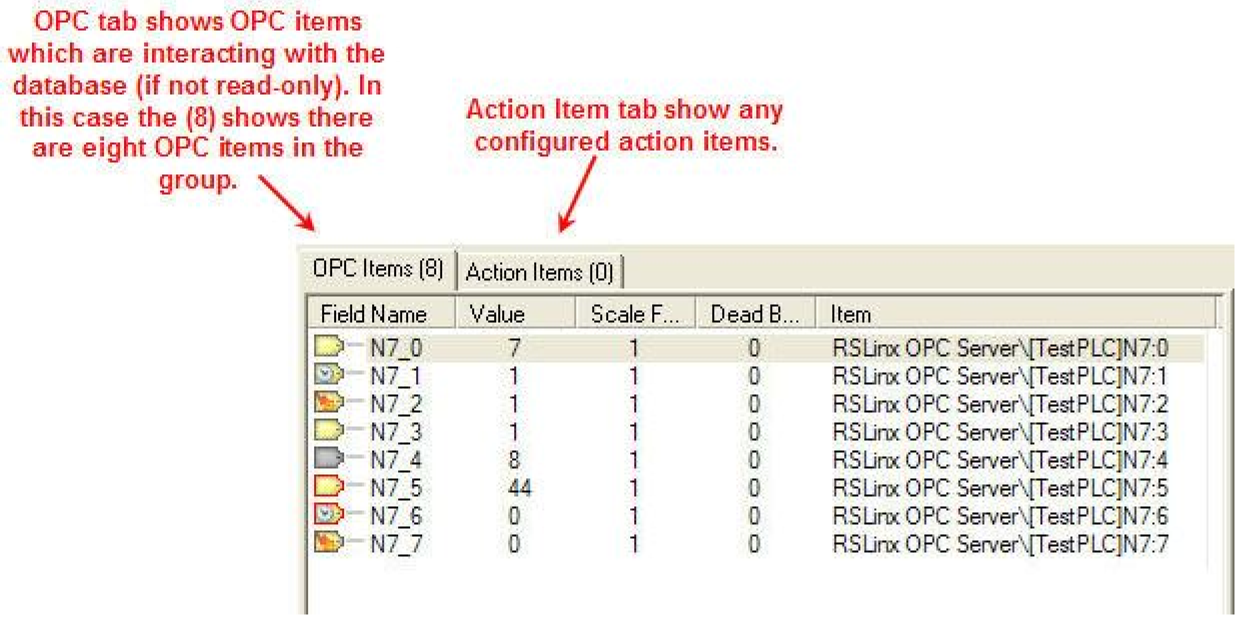
[✓ update/select] ( ) first record ( ) last record [✓ custom]

Where: compressor=1

Project open, 1 groups.

Group Item Pane

The group item pane has two tabs. Each group of items executes according to its update interval. Each group can have OPC items and/or Action Items. OPC item values can be embedded into Action Items. Furthermore the results of Action Items can be cascaded into subsequent items. This makes for very powerful processing capability. Each tab shows the number of items it has in the parenthesis.


OPC tab shows OPC items which are interacting with the database (if not read-only). In this case the (8) shows there are eight OPC items in the group.
Action Item tab show any configured action items.
OPC Items (8)
Action Items (0)

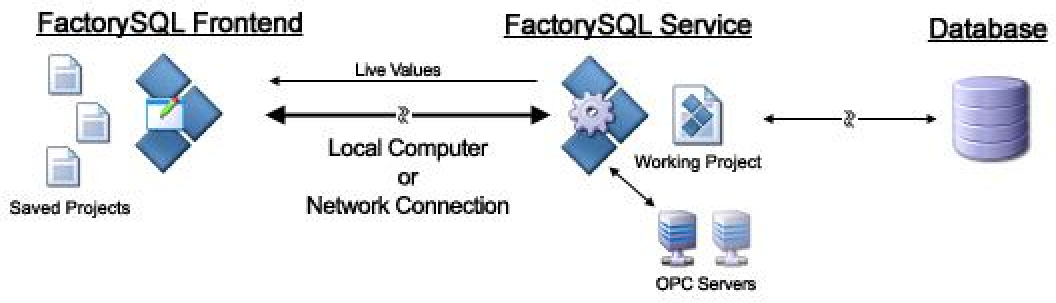
Field Name	Value	Scale F...	Dead B...	Item
N7_0	7	1	0	RSLinx OPC Server\[TestPLC]N7:0
N7_1	1	1	0	RSLinx OPC Server\[TestPLC]N7:1
N7_2	1	1	0	RSLinx OPC Server\[TestPLC]N7:2
N7_3	1	1	0	RSLinx OPC Server\[TestPLC]N7:3
N7_4	8	1	0	RSLinx OPC Server\[TestPLC]N7:4
N7_5	44	1	0	RSLinx OPC Server\[TestPLC]N7:5
N7_6	0	1	0	RSLinx OPC Server\[TestPLC]N7:6
N7_7	0	1	0	RSLinx OPC Server\[TestPLC]N7:7

Architecture Overview

The term **FactorySQL** actually refers to several pieces of software. Though they are used in conjunction, in is important to be aware of the difference, and of which piece performs which functions.

There are three main programs: the **FactorySQL Frontend**, the **FactorySQL Service**, and the **FactorySQL Service Application**. However, the different between the **Service** and **Service Application** is very small, and is covered below. For now, assume they are the same, and are simply called the **Service**.

The following image displays a logical layout:



The FactorySQL Frontend

The **Frontend** is the graphical application where you will perform all of the manual tasks in FactorySQL. The Frontend operates on **Project** files, either "offline" files that you open manually, or the "working project" that you open by Connecting to a Service. For more information on project files, click Here.

The FactorySQL Frontend does very little execution work itself. Tasks such as running the groups, interacting with the database, and even browsing the OPC servers are all the responsibility of the **Service**.

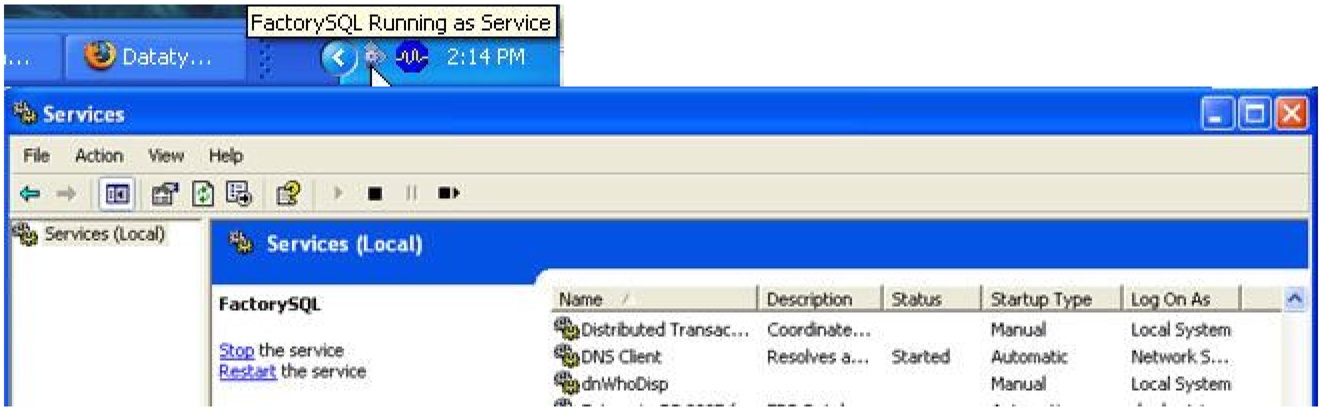
The FactorySQL Service

The **Service** is the execution side of FactorySQL. It has one project, the **Working Project**, which is configured or sent over from the Frontend. When the Frontend is connected to the Service, almost all operations directly affect the service: new groups and items immediately get saved to the working project, starting and stopping groups actually affect execution, and so on.

The service is also the component that interacts directly with OPC servers. The Frontend has no access to the server, it simply receives values and browsing data from the service. Therefore, if something is not working as expected, it is better to disconnect and try restarting the service instead of just restarting the frontend.

Service vs. Service Application

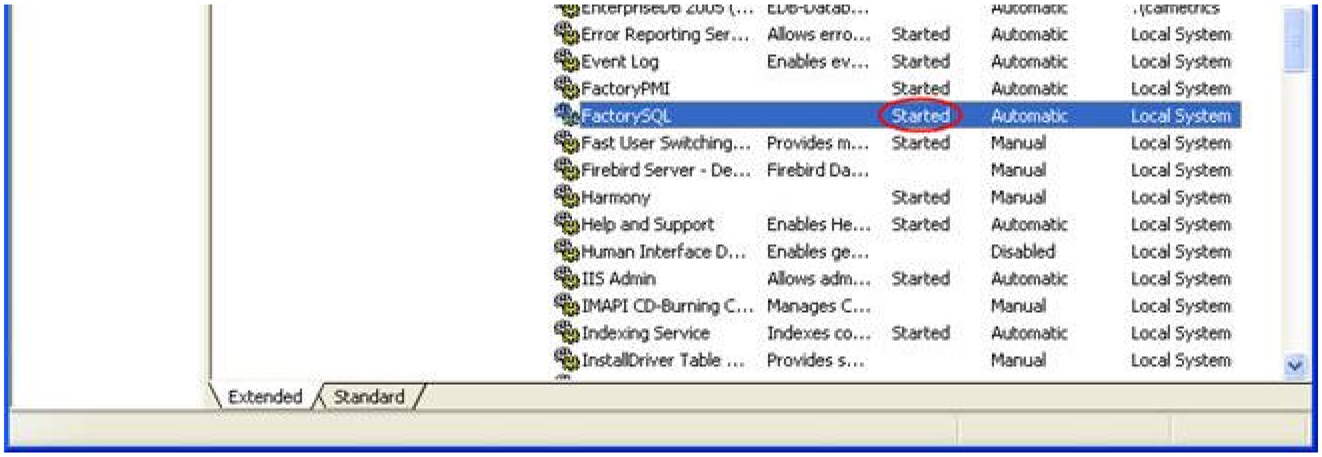
The difference between the **Service** and the **Service Application** comes down to whether FactorySQL is set to run as an **Application** or a **Windows Service**. If set to run as a Windows Service, the FactorySQL service will run in the background, and will be shown as "Running" in the Windows Service Manager. If the Service Application is open (is present in the system tray), it will also show that FactorySQL is running as a service:



FactorySQL

Stop the service
Restart the service

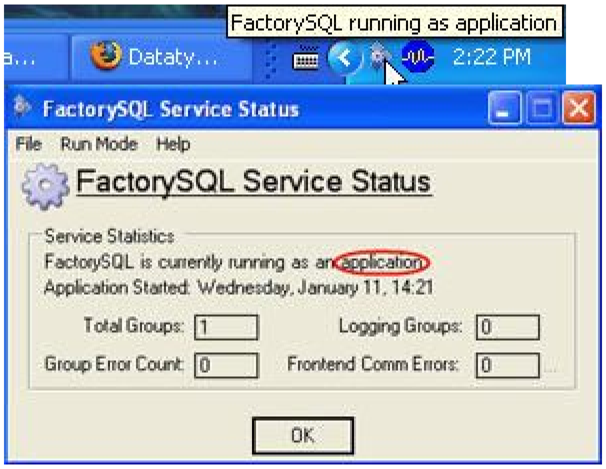
Name	Description	Status	Startup Type	Log On As
Distributed Transac...	Coordinate...		Manual	Local System
DNS Client	Resolves a...	Started	Automatic	Network S...
dnWhoDisp			Manual	Local System



EnterpriseDB 2005 (...	EDB-Datab...		Automatic	.\caimetrics
Error Reporting Ser...	Allows erro...	Started	Automatic	Local System
Event Log	Enables ev...	Started	Automatic	Local System
FactoryPMI		Started	Automatic	Local System
FactorySQL		Started	Automatic	Local System
Fast User Switching...	Provides m...	Started	Manual	Local System
Firebird Server - De...	Firebird Da...		Manual	Local System
Harmony		Started	Manual	Local System
Help and Support	Enables He...	Started	Automatic	Local System
Human Interface D...	Enables ge...		Disabled	Local System
IIS Admin	Allows adm...	Started	Automatic	Local System
IMAPI CD-Burning C...	Manages C...		Manual	Local System
Indexing Service	Indexes co...	Started	Automatic	Local System
InstallDriver Table ...	Provides s...		Manual	Local System

FactorySQL running as a Service. Service application (in tray) is only informational.

When running as an Application, the Service Application displayed in the System Tray has a much more important role. It *IS* the FactorySQL Service, if it is closed, FactorySQL will no longer function.



FactorySQL running as application
Dataty...
2:22 PM
FactorySQL Service Status
File
Run Mode
Help
FactorySQL Service Status
Service Statistics
OK

FactorySQL is currently running as an application

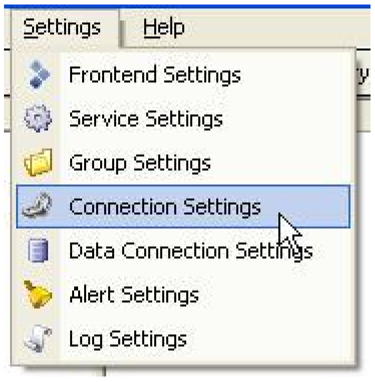
Application Started: Wednesday, January 11, 14:21

Total Groups:	1	Logging Groups:	0
Group Error Count:	0	Frontend Comm Errors:	0

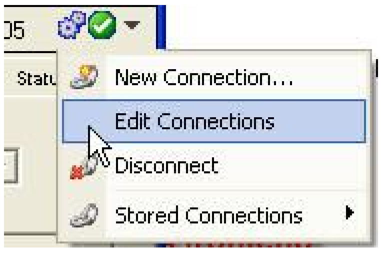
Running as an application. This status screen is accessed by double-clicking on the tray icon.

Configuring Service Connections

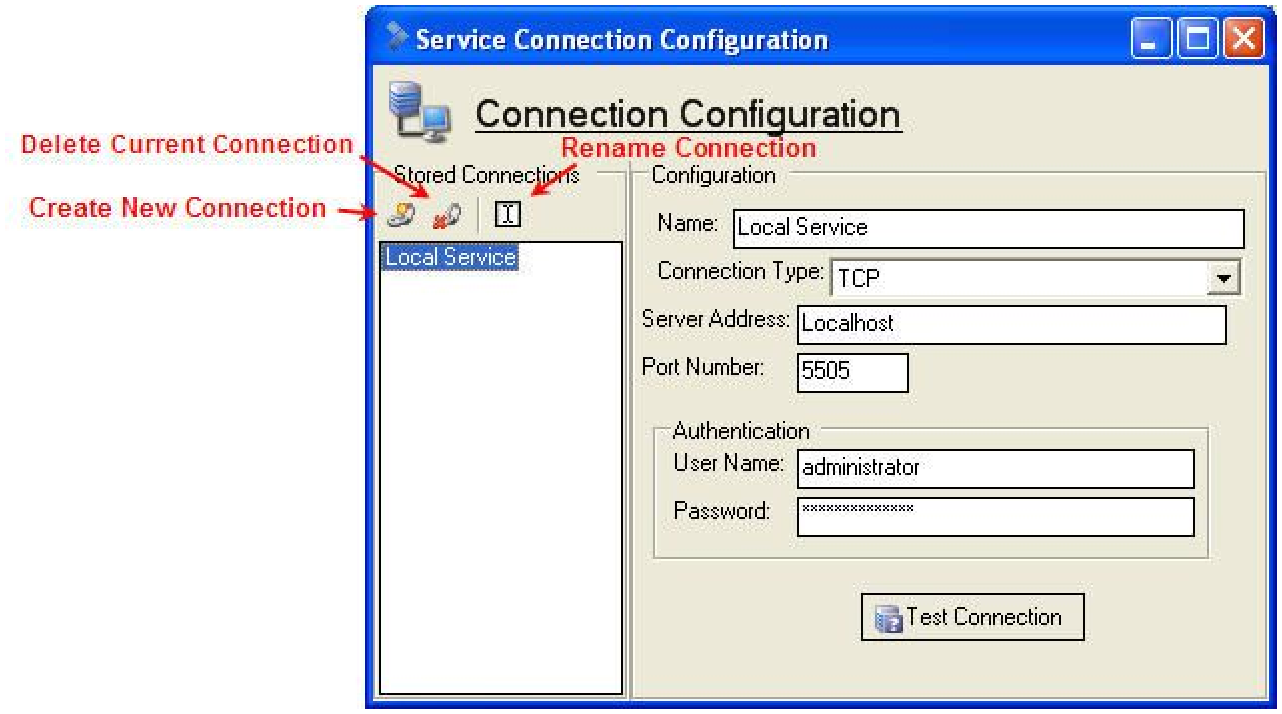
FactorySQL Service Connections allow the developer to create connection profiles to administer multiple remote FactorySQL installations from one point.
To edit the connection settings, choose **Connection Settings** from the settings menu, or click **Edit Connections** from the service connection menu:



or



The configuration screen looks as follows:



Local Service

Name: Local Service
Connection Type: TCP
Server Address: Localhost
Port Number: 5505
User Name: administrator
Password: **************

Configuration Options

Name: Specifies the name to reference the connection.

Connection Type: Protocol used for communication. As of this document, TCP is the only option.

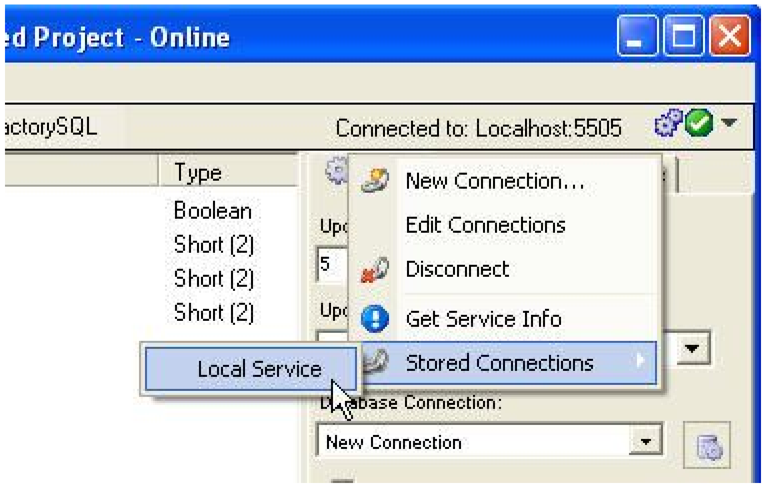
Server Address: Address of the FactorySQL service. This expects an IP address or "Localhost", which resolves to 127.0.0.1 (the adapter itself).

Port Number: The port that the FactorySQL Frontend will try to connect over.

Username/Password: The FactorySQL Frontend will attempt to connect with the given username and password.

Connecting the Frontend to a Service

FactorySQL is set up to allow local or remote configuration over TCP/IP. The developer can create and test FactorySQL connections in the **Service Connection Settings**.



Connecting

Connecting refers to the linking between the FactorySQL client and FactorySQL service. Connecting will allow the developer to browse the available OPC servers and configure the project on the FactorySQL service.

To connect, go through the connection icon on the upper right portion of the FactorySQL frontend.
The project on the remote service will be retrieved, and can now be modified. All changes are saved automatically.

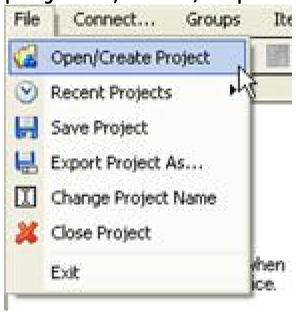
Projects

A FactorySQL **Project** is simply a collection of configured Groups. Projects are configured through the Frontend, and are saved with a ".FSP" file extension.

The most important concept to understand concerning **Projects** is that it is the **FactorySQL Service** that executes a project, and it can only run one project at a time - the **Working Project**. You may open and edit project files in the Frontend, but in order for them to actually execute, they must be sent to the service and made the working project. See below for more information on the Working Project, and on sending projects to the Service.

Creating / Opening Projects

You may create and open offline project files from the **File Menu**. This menu also lets you open recently accessed projects, save/export projects, and change the Project Name.

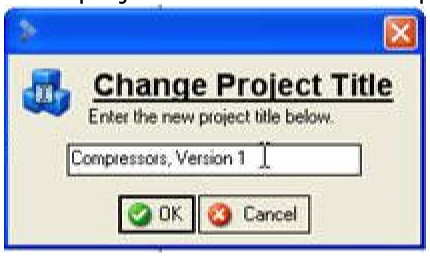


Saving/Exporting Projects

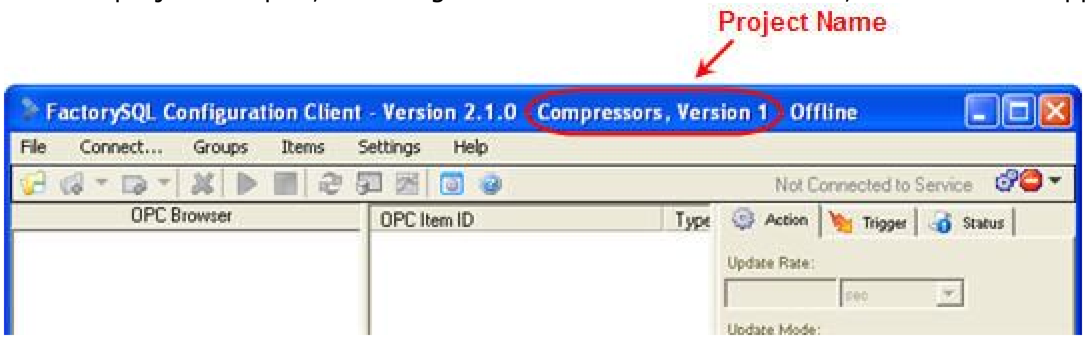
Projects are saved automatically as changes are made, so there is no need to explicitly save them. If you wish to save a project as a different file, you may use the **Export Project As...** item under the **File/Export-Backup** menu. Note that this is called "Export" instead of "Save As" because after the save, you will still be working on the original file, not the new one.

Changing Project Name

Each project can have a descriptive name, which is settable from the **Change Project Name** option under **File**:



When a project is open, including when connected to a service, the name will appear in the FactorySQL title bar:



About the "Working Project"

A FactorySQL service *always* has a working project. This is the project that it executes, and is the project that is opened by the Frontend when a user connects to a Service.
In normal use, it is usually unnecessary for a user to actually open a project and edit it offline. Editing projects in

such a manner does not alter what the FactorySQL service does at all, and the project must be sent to the service first in order to run (see Send Project to Service for information on how to do this). Instead, a normal workflow will consist of the user opening the FactorySQL frontend, Connecting to a Service, and modifying the project that automatically opens on connection.

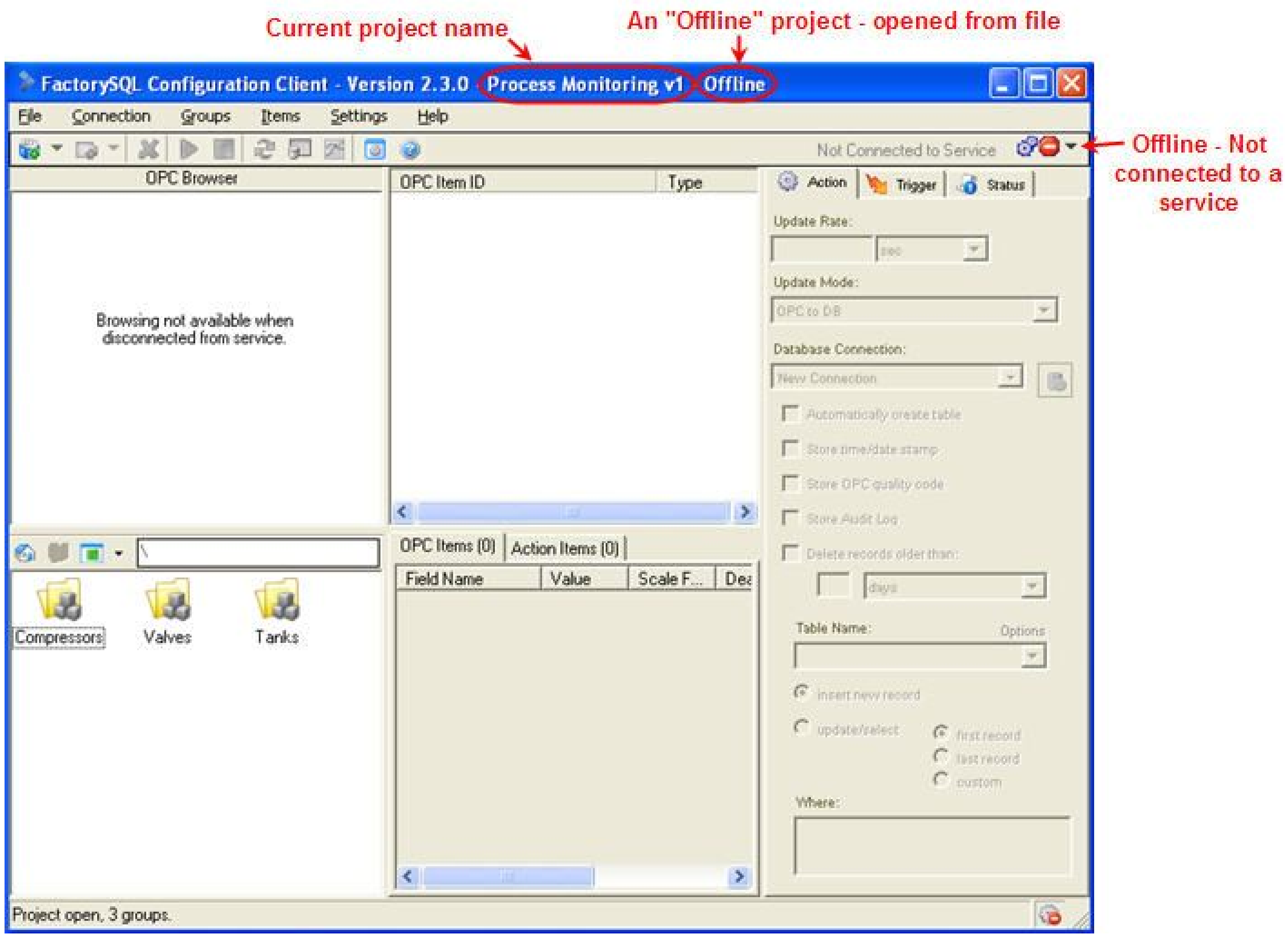
In very rare circumstances, it may be necessary to edit the working project offline. The project is located in the internal database, "system_database.fdb", under the install directory (usually Program Files\Inductive Automation\FactorySQL). Remember, changes made manually to this file will not take effect until the service is restarted.

Sending a Project to the Service

In order to actually run a project, it must reside on the FactorySQL Service. A service can only have one project, so sending one over will overwrite the current one.

The process to send a project over is simple: just open the project, and connect to the service. You will be prompted to send the file, and if you choose "YES", and confirm your decision on the following screen, the project will be sent, and will become the working project on the service.

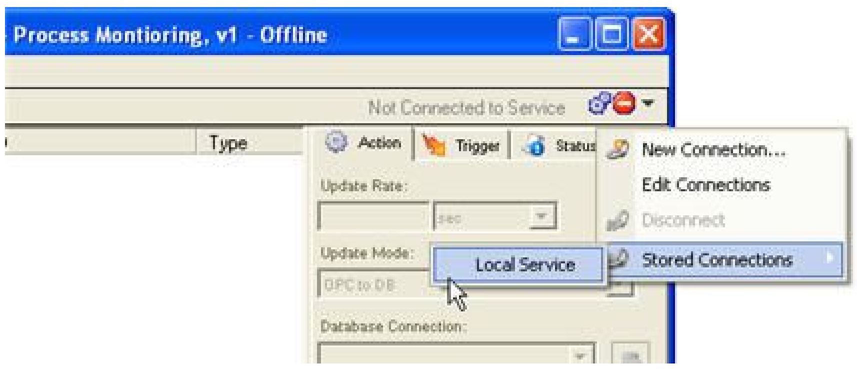
Consider the following example, where we've opened a project from "File - Open", and wish to send it to the service:



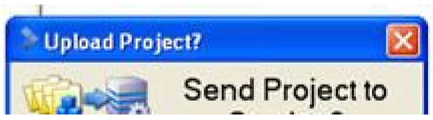
Browsing not available when disconnected from service.

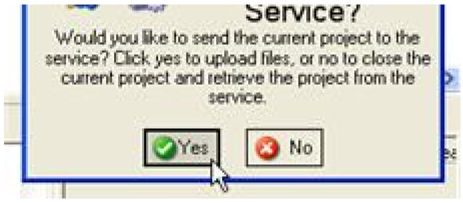
Project open, 3 groups.

1. Project open, offline



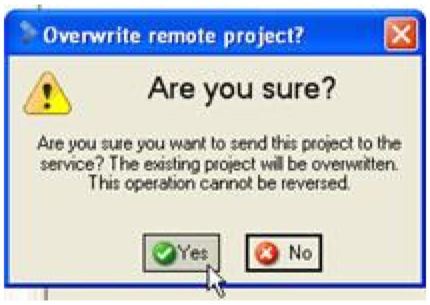
2. Selecting service to connect to.




Service?

Would you like to send the current project to the service? Click yes to upload files, or no to close the current project and retrieve the project from the service.

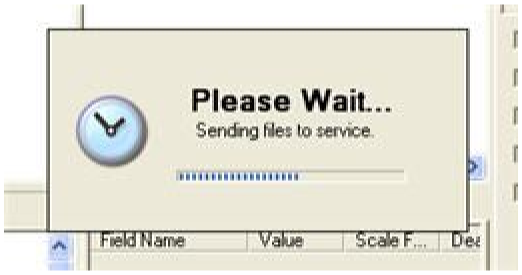
3. Choosing to send to service.


Overwrite remote project?

Are you sure?

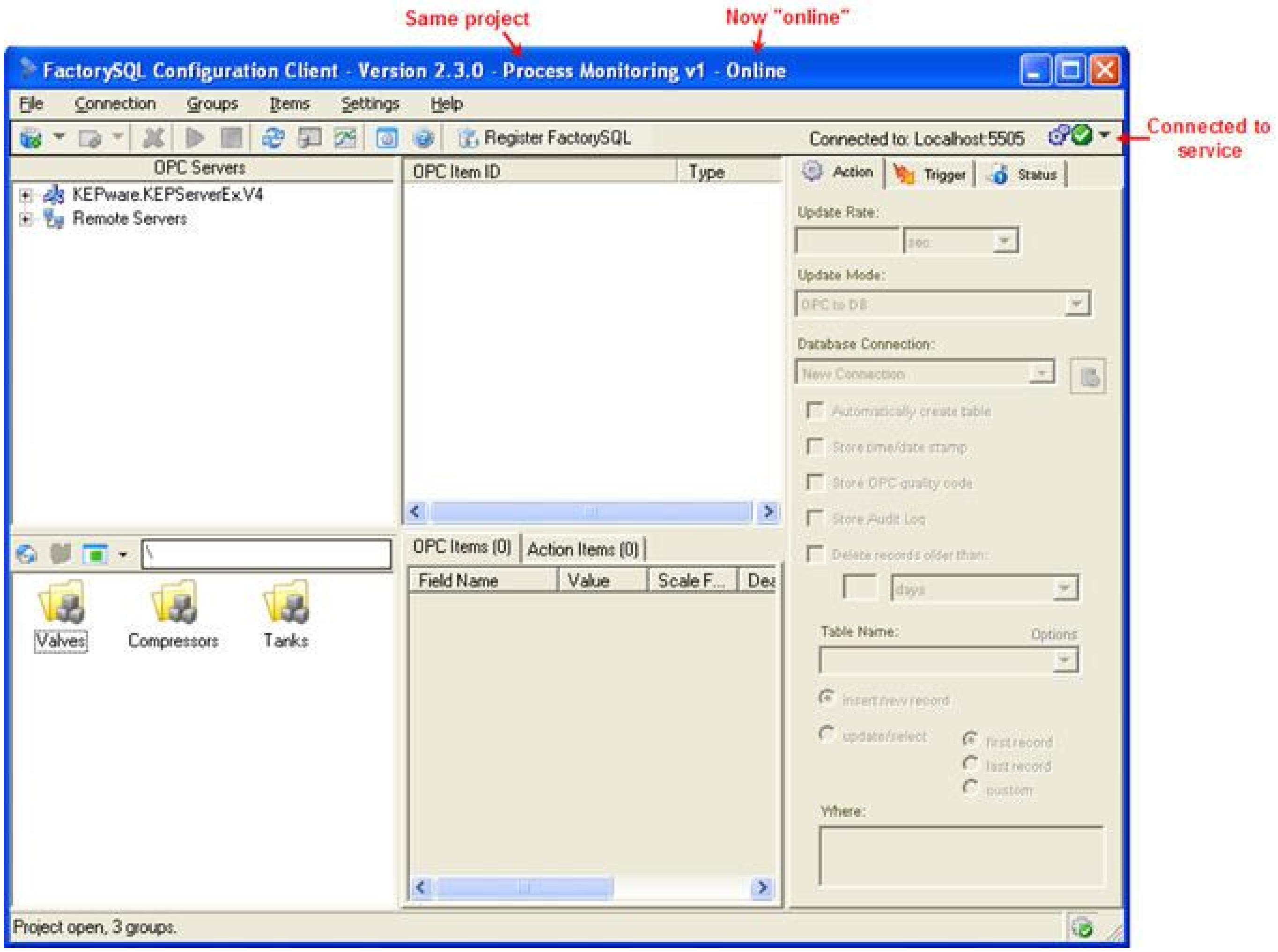
Are you sure you want to send this project to the service? The existing project will be overwritten. This operation cannot be reversed.

4. Confirming send.


Please Wait...

Sending files to service.

5. Sending to service...

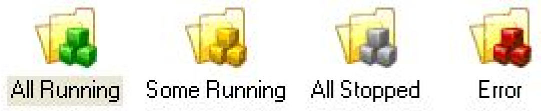


6. Now "Online", as the current working project.

Folders

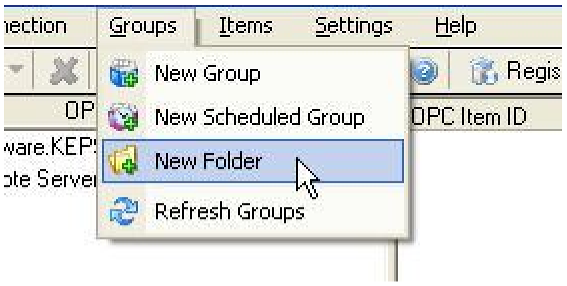
Folders logically organize FactorySQL groups. They have no inherit properties, but provide a convenient mechanism to start and stop groups, as well as a quick indication of running status.

Folder status is indicated by the color of its icon as follows:

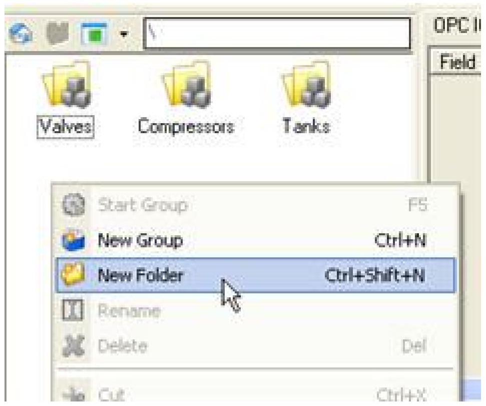


Note: the "error" icon takes precedence.

You may create group folders from the **Groups** menu on the main toolbar, or by right clicking on the group folder pane:



Or



To add groups to the folder, you may select the folder and create a new group, drag groups into it (hold CONTROL to create a copy), or use cut/copy/paste.

Groups

Groups form the basis of FactorySQL projects. They contain **OPC Items** and **Action Items**, and execute based on the settings in the **Action Tab** and **Trigger Tab**. A group's status information can be found on the **Status Tab**.

Group Explaination

The easiest way to think of a **group** is as a set of OPC addresses that all get synchronized with the database in the same way. For example, a "logging group" might consist of 10 PLC registers that all get written to the database every 2 minutes.

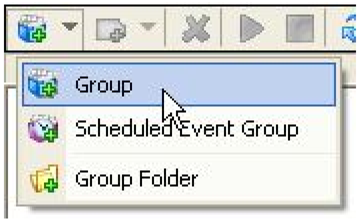
Items in a **group** write to the same row, in the same table, of the same database. They also share a common update interval. The idea is that a **group** should be thought of as a single logical unit.

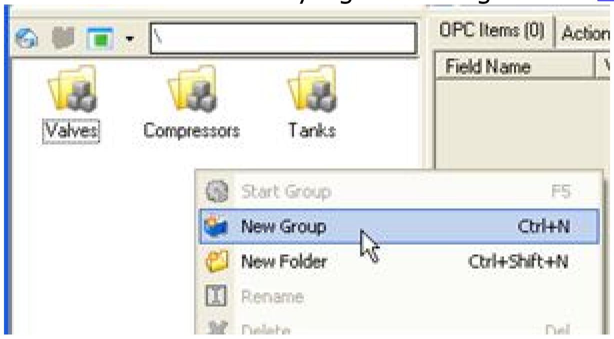
From there you can have individual item modes, Action Items, and a **dynamic where clause**. These features allow the designer the flexibility to make **groups** represent whole field devices/logical autonomous units.

For example, it normally wouldn't make sense to have a group for "compressor 1 display" and a different group for "compressor 1 control" even though some items might be OPC->DB while others are DB->OPC. An apt developer would create a single group, "compressor 1", and switch the update mode of some of the OPC items.

Adding a group

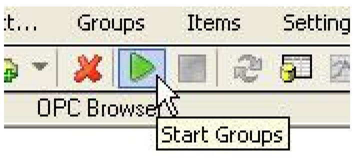
A new group can be created from the toolbar or by right-clicking on the Group Pane.

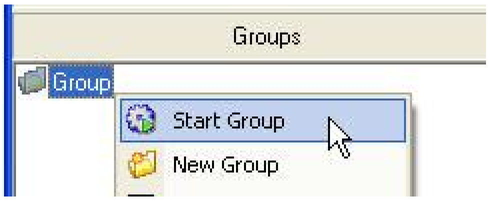




Running a group

Starting a group in FactorySQL causes it to "run". You may start the group from the toolbar, the group right-click menu, or by hitting F5:

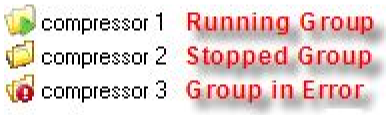




"Running" in FactorySQL refers to a group that is actively inspecting the PLC and database on its update rate, and synchronizing the two, as configured on the group's action tab.

Similarly, **stopping** a group does not correspond with physically stopping a field device, it simply indicates that FactorySQL is no longer synchronizing the database with those PLC registers.

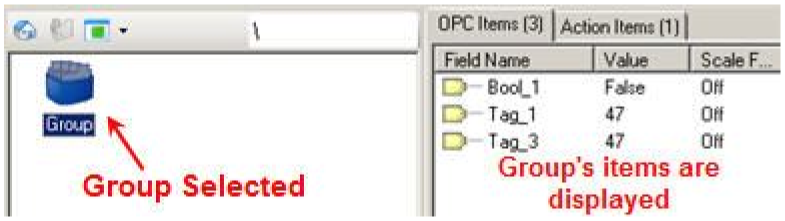
Groups have the following visual indicators:



More information on error conditions on **status tab**.

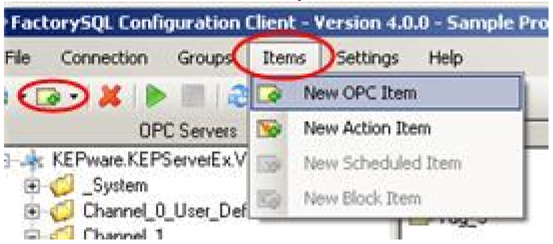
Group Items

All FactorySQL groups are made up of **Items**. When a group is selected in the Group Pane, it's corresponding items will be displayed to the right, in the Group Item Pane.



Field Name	Value	Scale F...
Bool_1	False	Off
Tag_1	47	Off
Tag_3	47	Off

The **Items** menu entry and toolbar button provide an easy way to add items to the group:



Item Types

There are several types of items in FactorySQL. A specific item type may not be available in all types of groups, or may vary slightly depending on the group type.

- OPC Items - Represents the value at an OPC address, and thus usually the value of a specific PLC register. See OPC Items for more information.
- Action Items - Can either be expressions or the result of an SQL query. Provides an opportunity to manipulate data or use it in a create way, before optionally writting to the database, or back to an OPC address. See Action Items for more information.
- Block Data Items - A collection of OPC items in on or more **segments**, defining a verticle column of data. See Block Data Items for more information.

- Scheduled Items - Actually stored as rows in the scheduled group table, these are queries that are run at a specific time, and optionally repeated on a schedule. See Scheduled Items for more info.

Database Connections

As the name implies, the primary purpose of FactorySQL is to interact with databases. In order to do this, a connection must be configured. There are 2 main types of connections: **Native** and **DSN**. Additionally, **Aggregate Connections** allow you to create a fail-over connection wrapper.

Native Connections

Native Connections use a native Microsoft .NET driver to connect to the database. The versions bundled with FactorySQL or available from Inductive Automation tend to be better tested, faster, and optimized for use with FactorySQL, making them your best first bet.

For an example of setting up a native connection, see the Quick Start.

Driver Type: This specifies which native driver to use. It will be specific to the database you are using.

Translator (advanced): FactorySQL uses translation files to mediate differences in SQL syntax between brands of servers. In almost every case this value should be left as 'Automatic', but in certain cases it may be necessary to choose a specialized translator.

Host: The address where the db server is located.

Port: The port that the db server runs on.

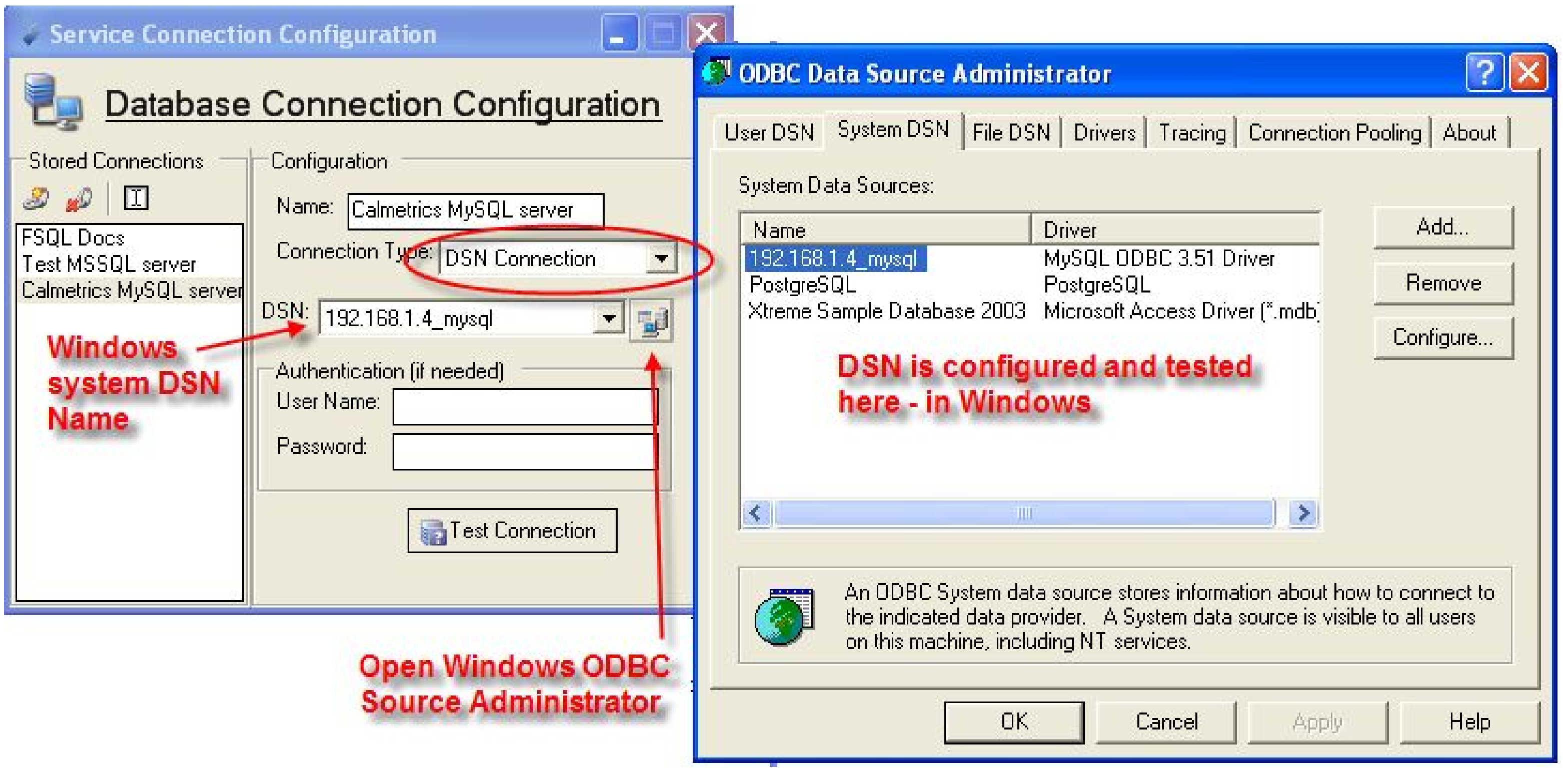
Database: The database to use on the server.

Extra Connection Parameters: Other parameters that will be passed along during connection. These are defined by the database driver, so you will have to consult the manual of the driver you are using for possible options.

Authentication: The username/password to use when connecting to the database.

ODBC Connections

ODBC connections are installed and configured in Windows. Simply install third party ODBC drivers to connect to almost any database.


Service Connection Configuration
Database Connection Configuration
Stored Connections
Configuration
Authentication (if needed)
Test Connection
Windows system DSN Name
Open Windows ODBC Source Administrator
ODBC Data Source Administrator
User DSN
System DSN
File DSN
Drivers
Tracing
Connection Pooling
About
Add...
Remove
Configure...
DSN is configured and tested here - in Windows
OK
Cancel
Apply
Help

Stored Connections: FSQL Docs; Test MSSQL server; Calmetrics MySQL server

Name: Calmetrics MySQL server
Connection Type: DSN Connection
DSN: 192.168.1.4_mysql
User Name:
Password:

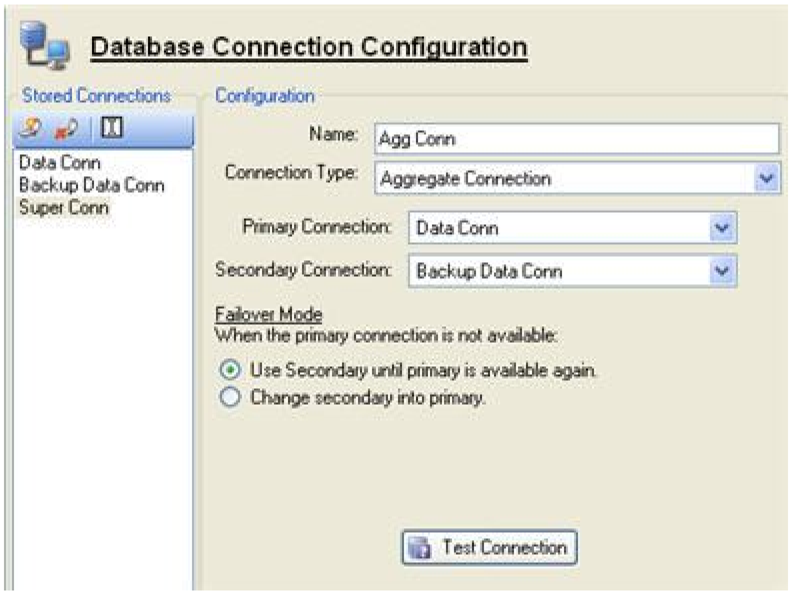
System Data Sources:

Name	Driver
192.168.1.4_mysql	MySQL ODBC 3.51 Driver
PostgreSQL	PostgreSQL
Xtreme Sample Database 2003	Microsoft Access Driver (*.mdb)

An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users on this machine, including NT services.

Aggregate Connections

The **Aggregate Connection Type** allows you to specify 2 data connections, a primary and secondary. When the primary is not available, it will fail over to the secondary connection. The **Failover Mode** determines what happens when the primary is available again. Multiple aggregate connections can be chained together to create a longer list of failover databases.


Database Connection Configuration

Stored Connections
- Data Conn
- Backup Data Conn
- Super Conn

Configuration

Name: Agg Conn

Connection Type: Aggregate Connection

Primary Connection: Data Conn

Secondary Connection: Backup Data Conn

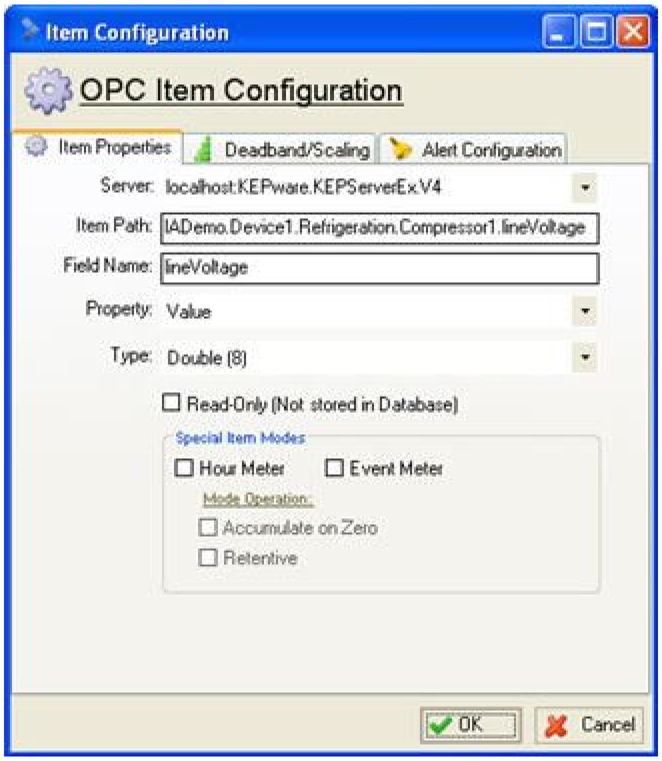
Failover Mode
When the primary connection is not available:
- [✓ Use Secondary until primary is available again.]
- Change secondary into primary.

Failover Mode: Determines what happens when the primary connection is available again. The first (and default) option is to switch back to the primary connection. Optionally, you can choose to continue using the current connection, in essence turning the former primary connection into a new secondary.

OPC Items

OPC Items are the fundamental building block of a FactorySQL project. The represent an address in the OPC server, such as the value of a PLC register.
The following outlines some basic settings of an OPC item, and some that pertain to Standard Groups. The actual options available for an OPC item will depend on the type of group being used.

The configuration of a standard group's opc item looks as follows:



Server: localhost:KEPware.KEPServerEx.V4
Item Path: IADemo.Device1.Refrigeration.Compressor1.lineVoltage
Field Name: lineVoltage
Property: Value
Type: Double (8)
☐ Read-Only (Not stored in Database)
Special Item Modes
☐ Hour Meter ☐ Event Meter
Mode Operation:
☐ Accumulate on Zero
☐ Retentive

OPC Item Settings

Server: The OPC server to use.
Item Path: The OPC item path. The syntax of this path will depend on the specific OPC server being used.
Field Name/Item Name: If the item is being written to the database (that is, is not Read Only and belongs to a group that supports it), this value will indicate the name of the column to write to. Otherwise, it will simply be the item's name, for item referencing purposes.
Property: The property to retrieve from the tag. Default is Value, but it is also possible to retrieve Quality, Timestamp and Item path information. **Type:** The datatype of the item. This is passed to the OPC as the requested item type.
Mode: If supported by the group, this allows you to override the group's execution mode. For example, on a status group that writes OPC->DB, you could change one item to "Bi-directional" in order to support control for that point.
Read-Only: The item will be available to reference in the group (in action items, alert setpoints, where clause, etc), but will not be stored **to the database**. NOTE: Even if an OPC item is set to Read-Only, it can still be written to through action items and trigger/handshakes.
Special Item Modes: See **Hour Meter** and **Event Meter**.

Deadband/Scaling

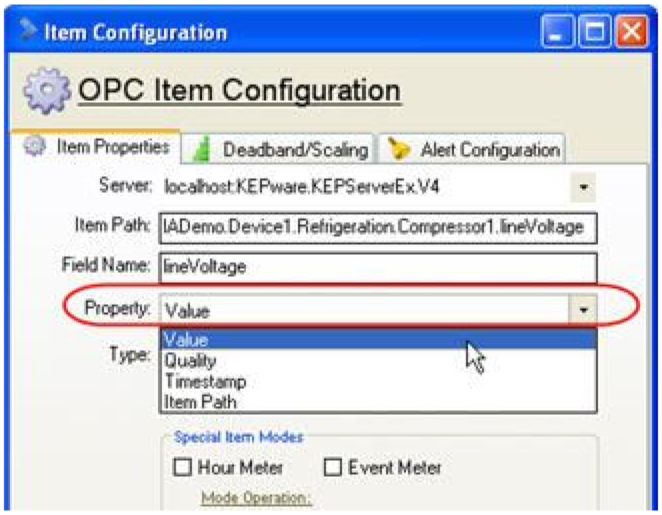
See Deadband & Scaling

Alerting

Alerting based on the value of an OPC item is possible. The check will be evaluated every time the group is run. Learn about alerting **here**.

OPC Item Property

The **Property** setting allows you to retrieve useful information about a tag.

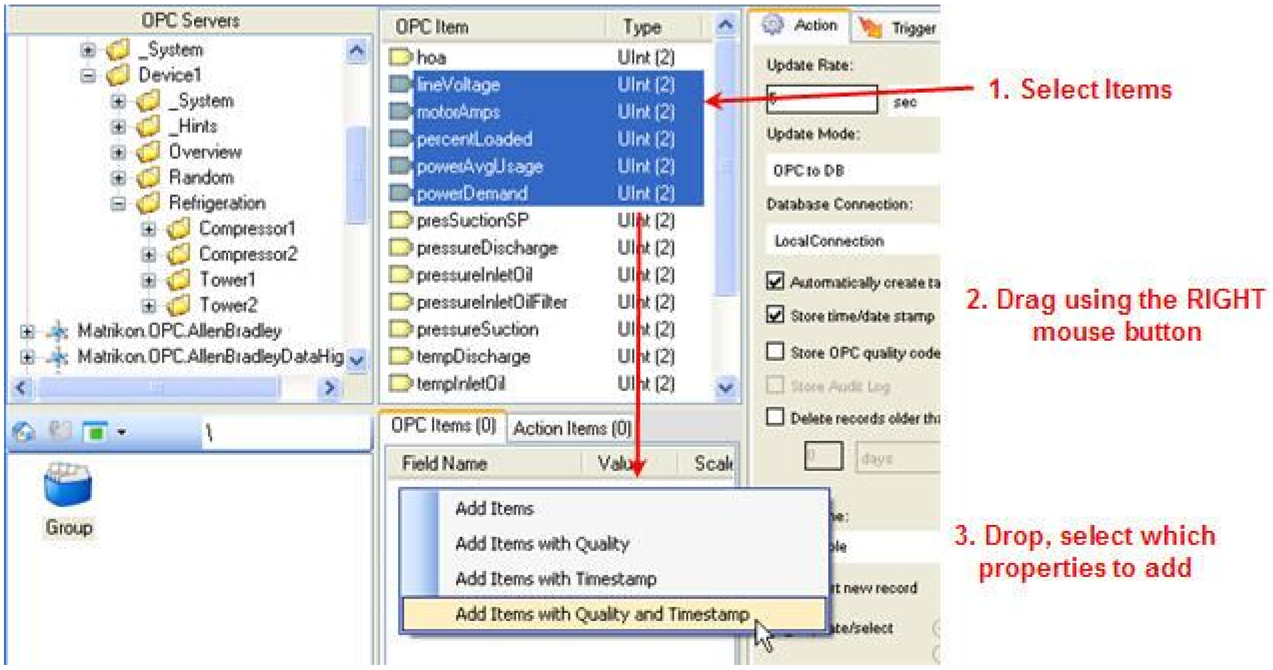


It is important to remember a few points about the properties:

- **Datatype should be set:** It is important that the datatype is correct for the **type of the property** (not the source tag). The datatype is automatically set for the value property when the item is added to the group, but may need to be changed if the property is changed. It should be as follows: Quality - Int, Timestamp - Date, Item Path - String.
- **Timestamp is different than group timestamp:** The group timestamp indicates when the value was written, whereas the item's timestamp property comes from the OPC server and is an indication of when the value last changed.
- **It is possible to have the same item multiple times in a group:** Therefore, you can mix multiple item properties in a group. Simply copy and paste items, and modify them to fit your needs.

Adding Item Properties While Browsing

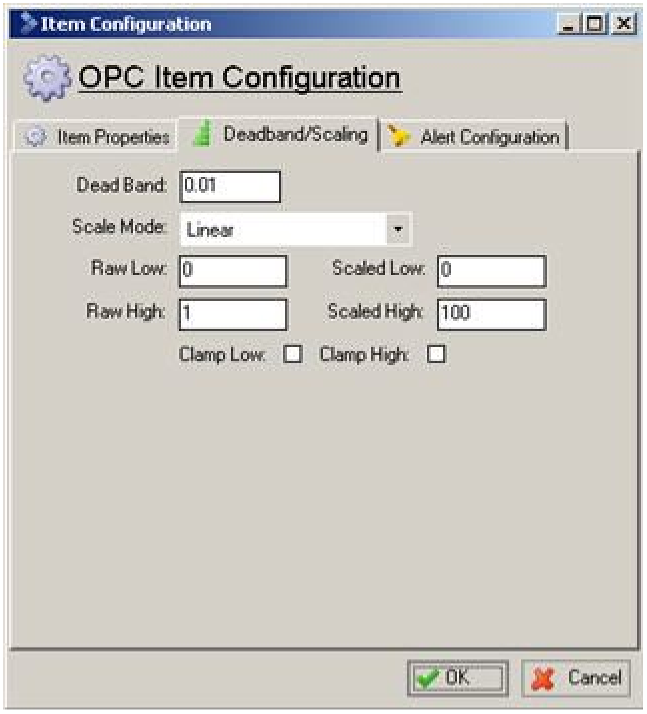
It is possible to quickly add item properties while browsing by dragging & dropping items with the **right mouse button**.



When you drop the items, a menu asks which properties to add. Duplicate copies of the item with the appropriate property and type settings will be created.

Deadband & Scaling

OPC items allow you to specify deadband and scaling settings.



Dead Band: 0.01
Scale Mode: Linear
Raw Low: 0
Scaled Low: 0
Raw High: 1
Scaled High: 100
Clamp Low: ☐
Clamp High: ☐

Deadband: Prevents data changes from occuring until the value has changed by the specified amount.

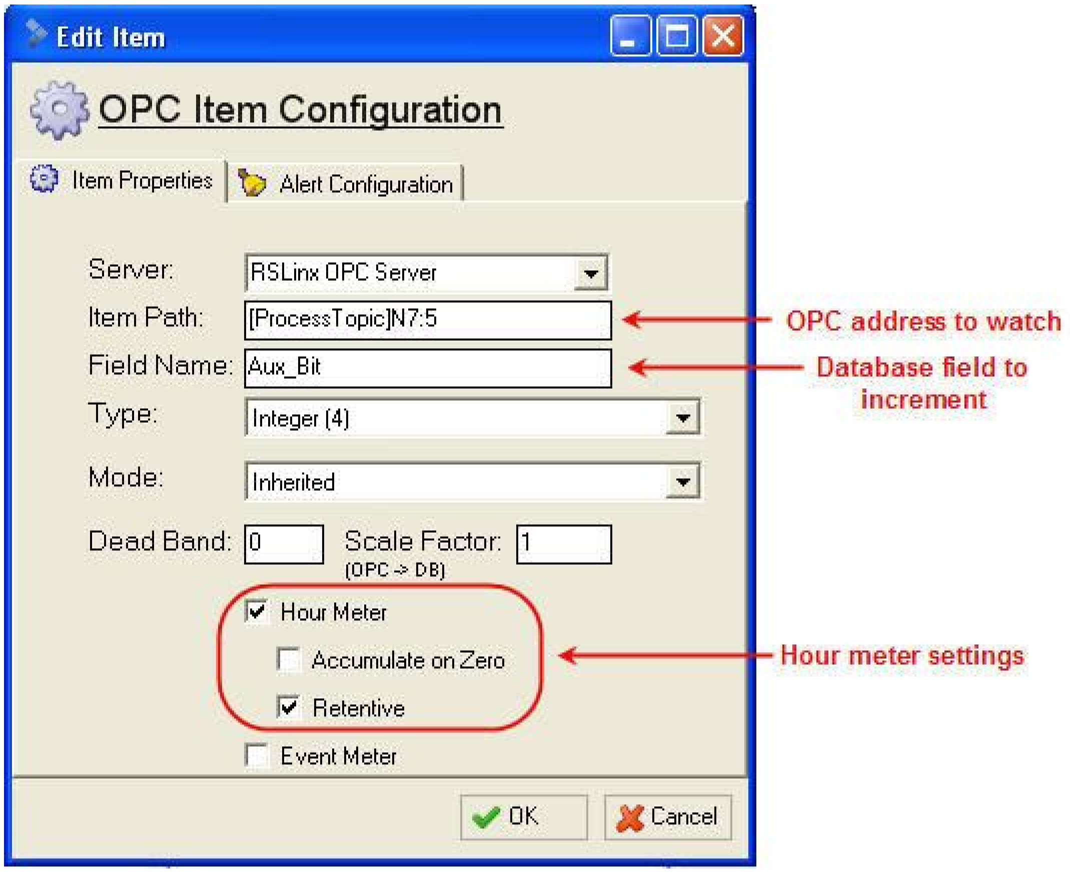
Scalemode: Off, Linear, Square Root
Raw Low, High, Scaled Low, High: The values to scale between, from the perspective of OPC to DB. For instance, in the screenshot above, the raw goes from 0 to 1, and scaled from 0 to 100, in linear mode. Essentially, FactorySQL will multiple each value coming from the OPC server by 100, and will divide any value coming from the database by the same amount.

Clamp: Prevents the scaled value from moving outside of the specified bounds. Continuing the example from above, if we clamped both Low and High, the value would always be between 0-100.

Hour Meter

An **Hour Meter** is a special kind of **OPC Item** that increments the database field based on a PLC register state. It is used to keep time on devices.

Suppose we wanted to track the running time of a motor based on its "aux" bit. If the group was set to run every 30 seconds, then every time that bit was 1 FactorySQL would increment the database field by 30 seconds.



Server: RSLinx OPC Server
Item Path: [ProcessTopic]N7:5
Field Name: Aux_Bit
Type: Integer (4)
Mode: Inherited
Dead Band: 0
Scale Factor: 1 (OPC -> DB)

[✓] Hour Meter
[ ] Accumulate on Zero
[✓] Retentive
[ ] Event Meter

Settings:

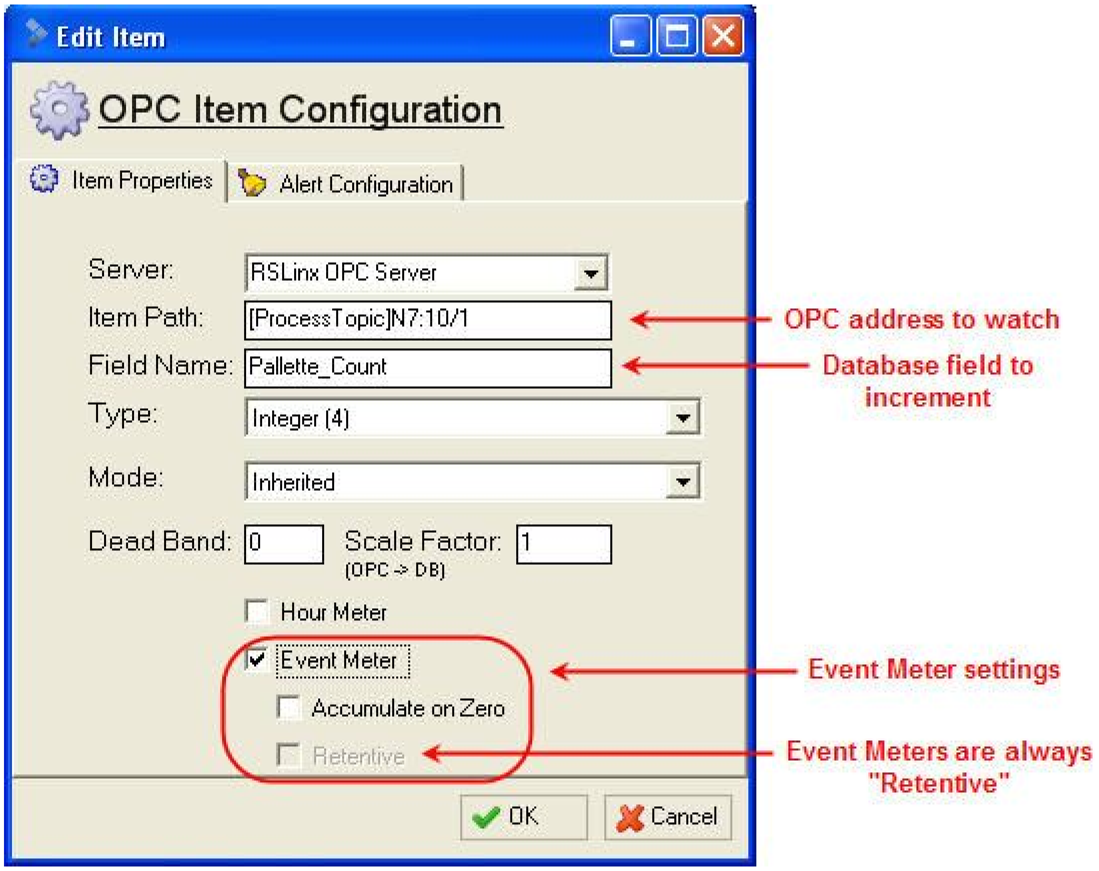
Hour Meter: This checkbox makes an **OPC Item** behave as an Hour Meter.

Accumulate on Zero: Increments database when the OPC Item value is 0. Hour Meters increment on any non-zero by default.

Retentive: When this is set FactorySQL will always just increment the field. If **retentive** is not set FactorySQL will clear the value whenever the group runs or the Hour Meter condition goes false.

Event Meter

An **Event Meter** is a special kind of **OPC Item** that increments the database field by 1 whenever a PLC register state goes from False to True. It is used to keep a count, such as how many pallets have gone by a photoeye.



Server: RSLinx OPC Server
Item Path: [ProcessTopic]N7:10/1
Field Name: Pallette_Count
Type: Integer (4)
Mode: Inherited
Dead Band: 0
Scale Factor: (OPC -> DB) 1
[ ] Hour Meter
[✓] Event Meter
[ ] Accumulate on Zero
[ ] Retentive

Settings:

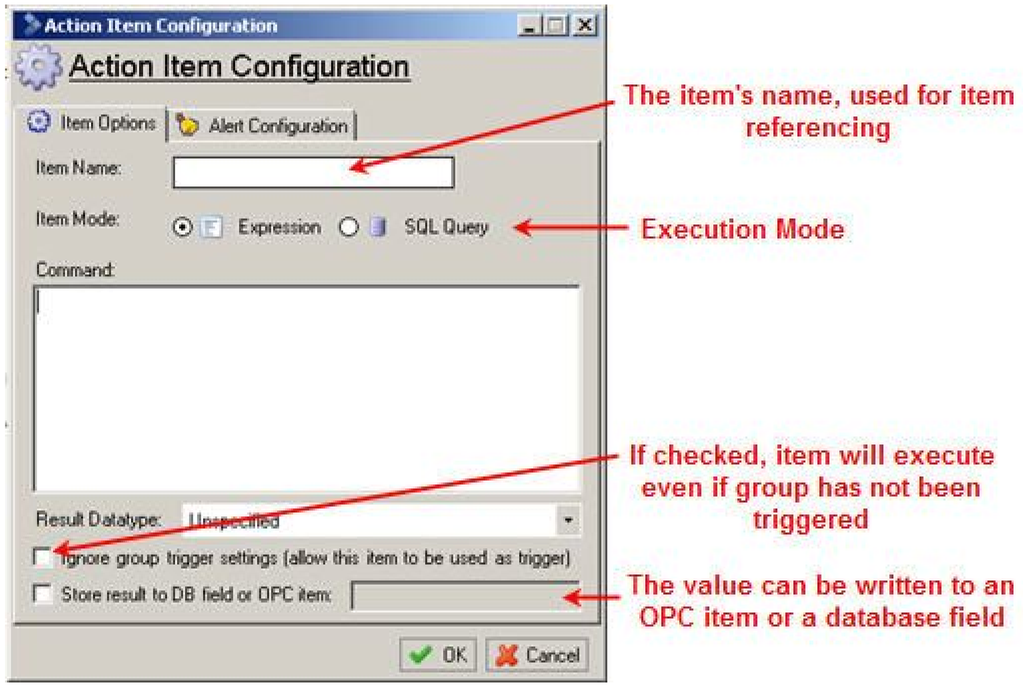
Event Meter: This checkbox makes an **OPC Item** behave as an Event Meter.

Accumulate on Zero: Increments database when the OPC Item value is 0. Event Meters increment on any non-zero by default.

Action Items

Action Items provide a way to execute some action, either an expression or SQL query, and use the result in your group. This provides you with a wide range of flexibility and a great deal of power in accomplishing complex tasks in your groups, especially since the expression language can be extended with the Scripting API.

When you add an Action Item, or double click on one to edit it, you are presented with the following screen:


Action Item Configuration
Action Item Configuration
Item Options
Alert Configuration
Item Name:
Item Mode:
Expression
SQL Query
Command:
Result Datatype:
Unspecified
Ignore group trigger settings (allow this item to be used as trigger)
Store result to DB field or OPC item:
OK
Cancel
The item's name, used for item referencing
Execution Mode
If checked, item will execute even if group has not been triggered
The value can be written to an OPC item or a database field

Item Substitution

Options:

Item Name: Logically names Action item. This is the display name of the item in the group and when using **Item Substitution**. Keep in mind that this *does not* correspond with a column name as the name of an OPC item does. To accomplish that result, type the desired name in the **Store result to DB field...** field.

Item Mode: This options changes how the **Command** is interpreted. When "Expression" is selected (the default), the command will be evaluated as an expression language statement. As an "SQL Query", the command will be run against the groups data connection. If the query returns more than one value, the first will be used for the value of the item.

Command: This is the "meat" of an Action Item. This query gets run on the selected **Data Connection**. The syntax of this field, as mentioned above, depends on the **Item Mode**. Expression statements support most basic arithmetic operators, as well as bitwise and logical operators. For their exact syntax, see Expression Language Syntax. As an SQL statement, you may use any normal command.
Irregardless of the execution mode used, it's important to realize that the key to effectively using Action Items is to use **Item Substitution**. This allows you to reference the values of other items, to perform calculations, and create complex logic statments that you can then trigger on. Really, with a little creativity, the sky's the limit!

Result Datatype: Specifies the data type to convert the value to before returning. Normally "Unspecified" will work fine.

Ignore group trigger settings: When this option is selected, the action item will be executed on every update interval (defined in the group options), even if the group does not run because of trigger conditions. This way, the item may be used as a trigger for the group. If it is not selected, it will only be executed when the group is in an active trigger state. Of course, this option is insignificant if the group is not triggered.

Store result to DB field or OPC item: This option will write back the result of the Action Item to the database or PLC. When writing to the database, FactorySQL automatically writes to the same connection, table, and row as the group is writing to. This is useful to make an Action Item behave like an OPC item. To write to an OPC item, simply use CNTL+SPACE to bring up the **Item Substitution Menu**.

Alerting

It is possible to alert on the value of an Action Item, just like an OPC item. The check will be evaluated every time the group is run. Learn about alerting **here**.

Groups

Groups form the basis of FactorySQL projects. They contain **OPC Items** and **Action Items**, and execute based on the settings in the **Action Tab** and **Trigger Tab**. A group's status information can be found on the **Status Tab**.

Group Explaination

The easiest way to think of a **group** is as a set of OPC addresses that all get synchronized with the database in the same way. For example, a "logging group" might consist of 10 PLC registers that all get written to the database every 2 minutes.

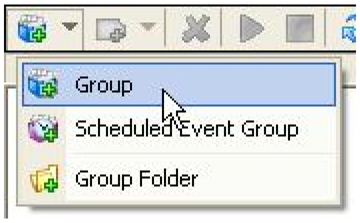
Items in a **group** write to the same row, in the same table, of the same database. They also share a common update interval. The idea is that a **group** should be thought of as a single logical unit.

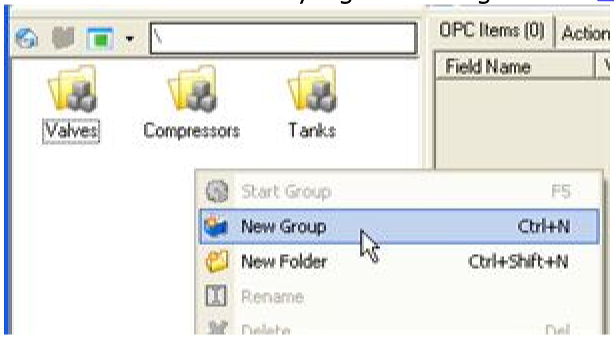
From there you can have individual item modes, Action Items, and a **dynamic where clause**. These features allow the designer the flexibility to make **groups** represent whole field devices/logical autonomous units.

For example, it normally wouldn't make sense to have a group for "compressor 1 display" and a different group for "compressor 1 control" even though some items might be OPC->DB while others are DB->OPC. An apt developer would create a single group, "compressor 1", and switch the update mode of some of the OPC items.

Adding a group

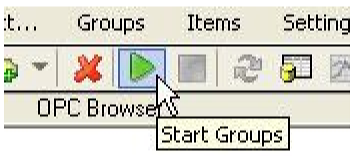
A new group can be created from the toolbar or by right-clicking on the Group Pane.

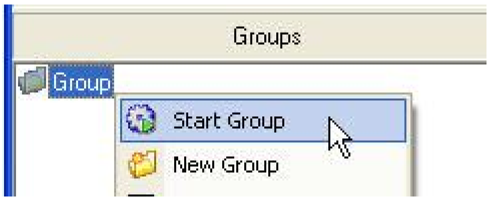




Running a group

Starting a group in FactorySQL causes it to "run". You may start the group from the toolbar, the group right-click menu, or by hitting F5:

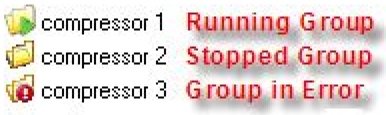




"Running" in FactorySQL refers to a group that is actively inspecting the PLC and database on its update rate, and synchronizing the two, as configured on the group's action tab.

Similarly, **stopping** a group does not correspond with physically stopping a field device, it simply indicates that FactorySQL is no longer synchronizing the database with those PLC registers.

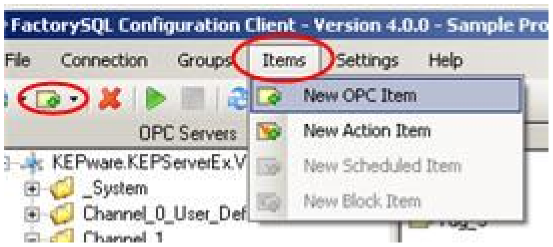
Groups have the following visual indicators:


compressor 1 Running Group
compressor 2 Stopped Group
compressor 3 Group in Error

More information on error conditions on **status tab**.

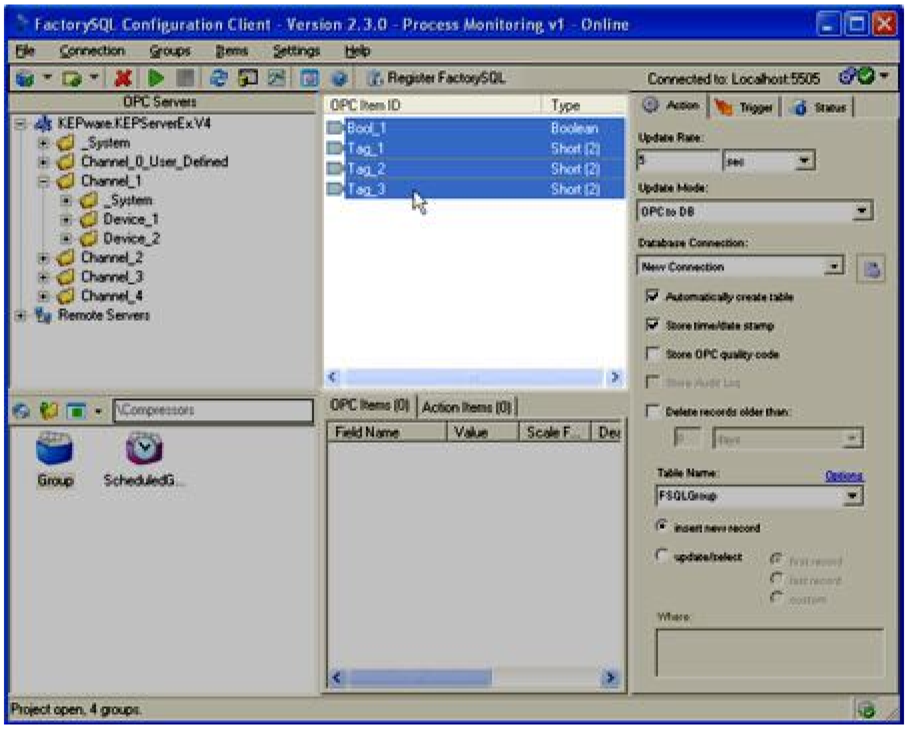
Adding Items

Items can be added easily to transaction groups through the Menu, the Toolbar, or by right-clicking the Group Item Pane.



Drag & Drop OPC Items

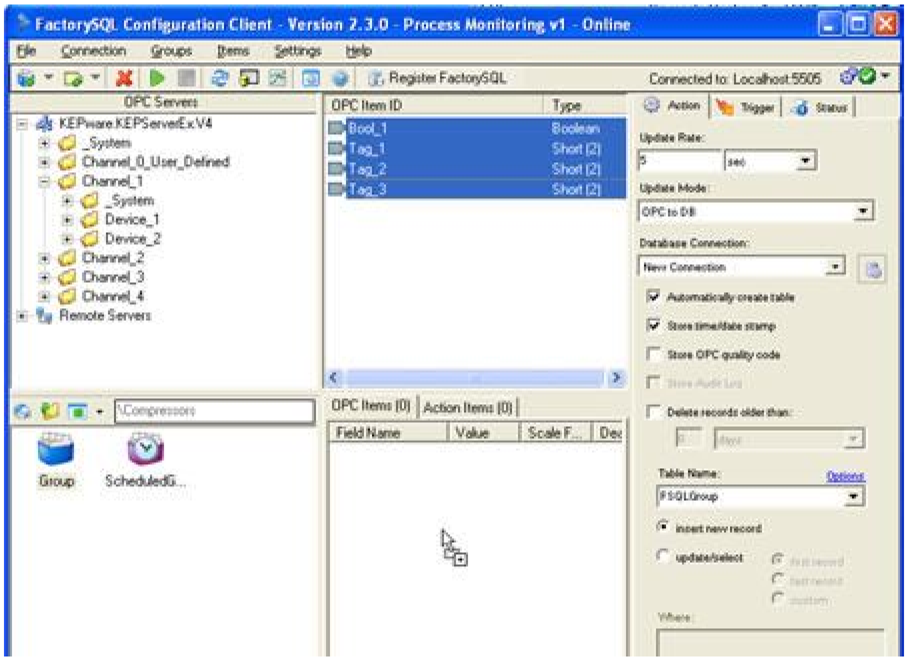
It is possible to add OPC items very quickly using FactorySQL's server browsing and drag & drop functionality:



OPC Item ID	Type
Bool_1	Boolean
Tag_1	Short (2)
Tag_2	Short (2)
Tag_3	Short (2)

Update Rate: 5 sec
Update Mode: OPC to DB
Database Connection: New Connection
[✓] Automatically create table
[✓] Store time/date stamp
[ ] Store OPC quality code
[ ] Store Audit Log
[ ] Delete records older than: 0 days
Table Name: FSQLGroup (Options)
(●) insert new record
( ) update/select (● first record, ○ last record, ○ custom)
Where:

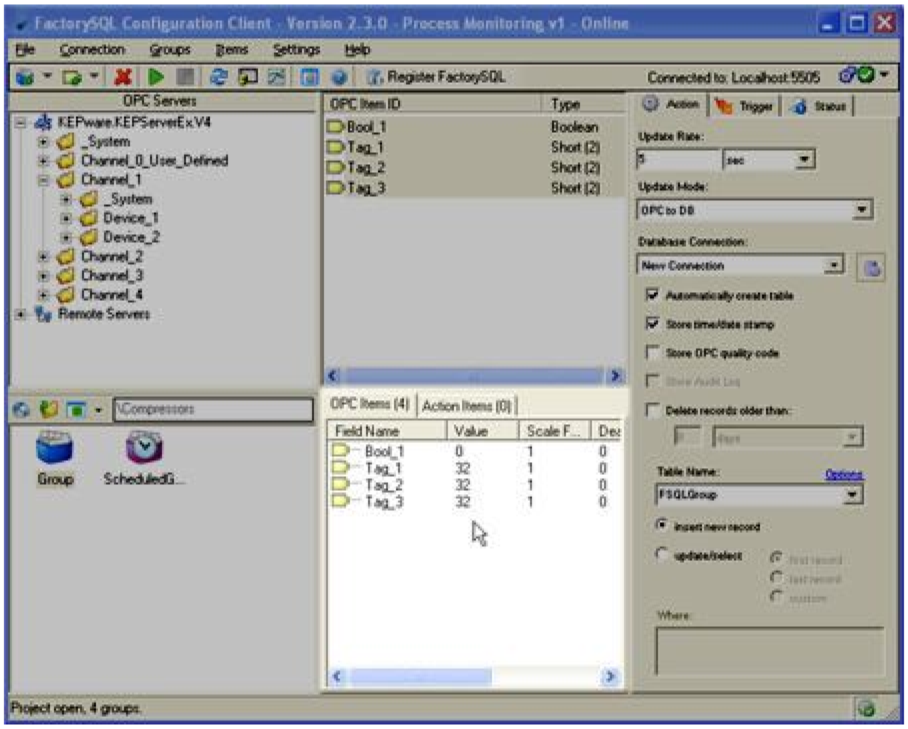
1. Browse to items you wish to add.



OPC Item ID	Type
Bool_1	Boolean
Tag_1	Short (2)
Tag_2	Short (2)
Tag_3	Short (2)

Update Rate: 5 sec
Update Mode: OPC to DB
Database Connection: New Connection
[✓] Automatically create table
[✓] Store time/date stamp
[ ] Store OPC quality code
[ ] Store Audit Log
[ ] Delete records older than: 0 days
Table Name: FSQLGroup (Options)
(●) insert new record
( ) update/select (● first record, ○ last record, ○ custom)
Where:

2. Drag items into Group Item Pane.



OPC Item ID	Type
Bool_1	Boolean
Tag_1	Short (2)
Tag_2	Short (2)
Tag_3	Short (2)

Field Name	Value	Scale F...	Des
Bool_1	0	1	0
Tag_1	32	1	0
Tag_2	32	1	0
Tag_3	32	1	0

Update Rate: 5 sec

Update Mode: OPC to DB

Database Connection: New Connection

[✓] Automatically create table

[✓] Store time/date stamp

[ ] Store OPC quality code

[ ] Store Audit Log

[ ] Delete records older than:

Table Name: FSQLGroup

Options

(●) insert new record

( ) update/select

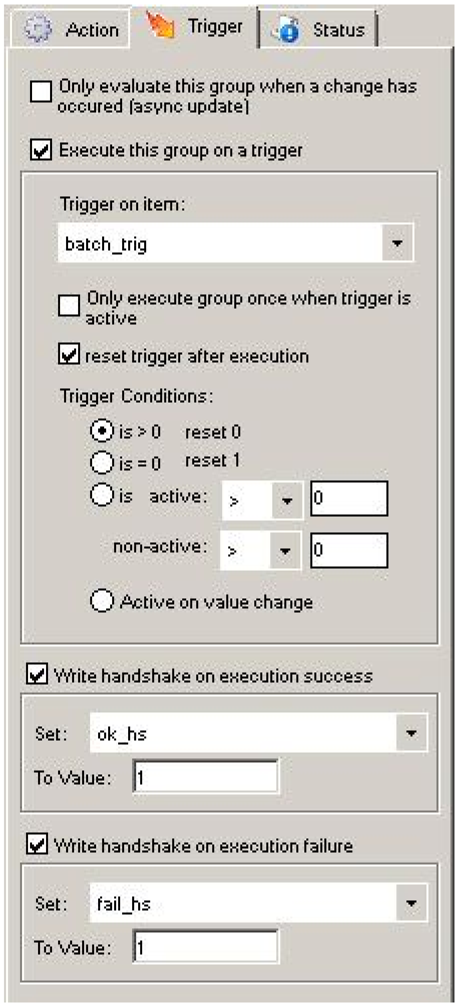
( ) first record ( ) last record ( ) custom

Where:

3. Release mouse - items are added! Field name will default to a database-safe version of the item's name.

Group Options - Trigger Tab

Triggering allows you to specify more precisely when a group will be executed. The **Trigger Tab** allows you to configure the condition that will allow the group to run, along with a few other related settings.



☐ Only evaluate this group when a change has occured (async update)

☑ Execute this group on a trigger

Trigger on item: batch_trig

☐ Only execute group once when trigger is active

☑ reset trigger after execution

Trigger Conditions:

● is > 0 reset 0

○ is = 0 reset 1

○ is active: > 0

non-active: > 0

○ Active on value change

☑ Write handshake on execution success

Set: ok_hs

To Value: 1

☑ Write handshake on execution failure

Set: fail_hs

To Value: 1

Only evaluate this group when a change has occurred: When checked, the group will only be evaluated when a change has occurred to an OPC value since the last update. This is useful when you only want to log changes, and not just on a set interval. This setting takes precedence over the rest of the trigger settings.

Execute this group on a trigger: This option specifies to use the trigger. If it is not selected, the group will run every update interval as specified in the **Action Tab**. If it is selected, the trigger condition must be true for the group to do anything (except run an Action Item that is set to run every update interval).

Trigger Options:

Trigger on item: select an item to be used as the trigger. This can be an OPC item or an Action Item set to run every interval.

Only execute group once when trigger is active: Only allows the transaction group to run once when the trigger is true and then not again until the trigger goes to a false state, then returns true. Also known as "One-Shot" triggering.

Reset trigger after execution: writes a predefined false value to the trigger address after execution. Only applies to the first 2 trigger conditions - "is > 0" and "is = 0".

Prevent trigger caused by group start: certain trigger conditions can cause the group to execute each time it is started, for instance "active on value change". This setting prevents this execution from occurring. If the group is counting items or being triggered after the production of an item, detection of a condition, etc. it is usually not desirable to log on startup. However, if the group is logging general history, it may be useful to log on startup.

Trigger Conditions:

These define exactly when the group is considered to in and out of trigger.

Is > 1, Reset 0: If the trigger item is >1, the group is triggered and will execute. If "Reset Trigger" is active, a 0 will be written after execution.

Is = 0, Reset 1: Group will execute if value is 0, and a 1 will be written to the trigger item if "Reset Trigger" is selected.

Is active, non-active: These conditions specify when the group is considered to be active and not active. There is no actual trigger reset in this mode. Note that once a group is "active", it will continue to log until the "non-active" condition is true. That is, it does not stop simply when the "active" condition is false.

Active on value change: The group will be activated for 1 cycle when the trigger value changes. This is the only trigger condition that naturally only executes the group 1 time.

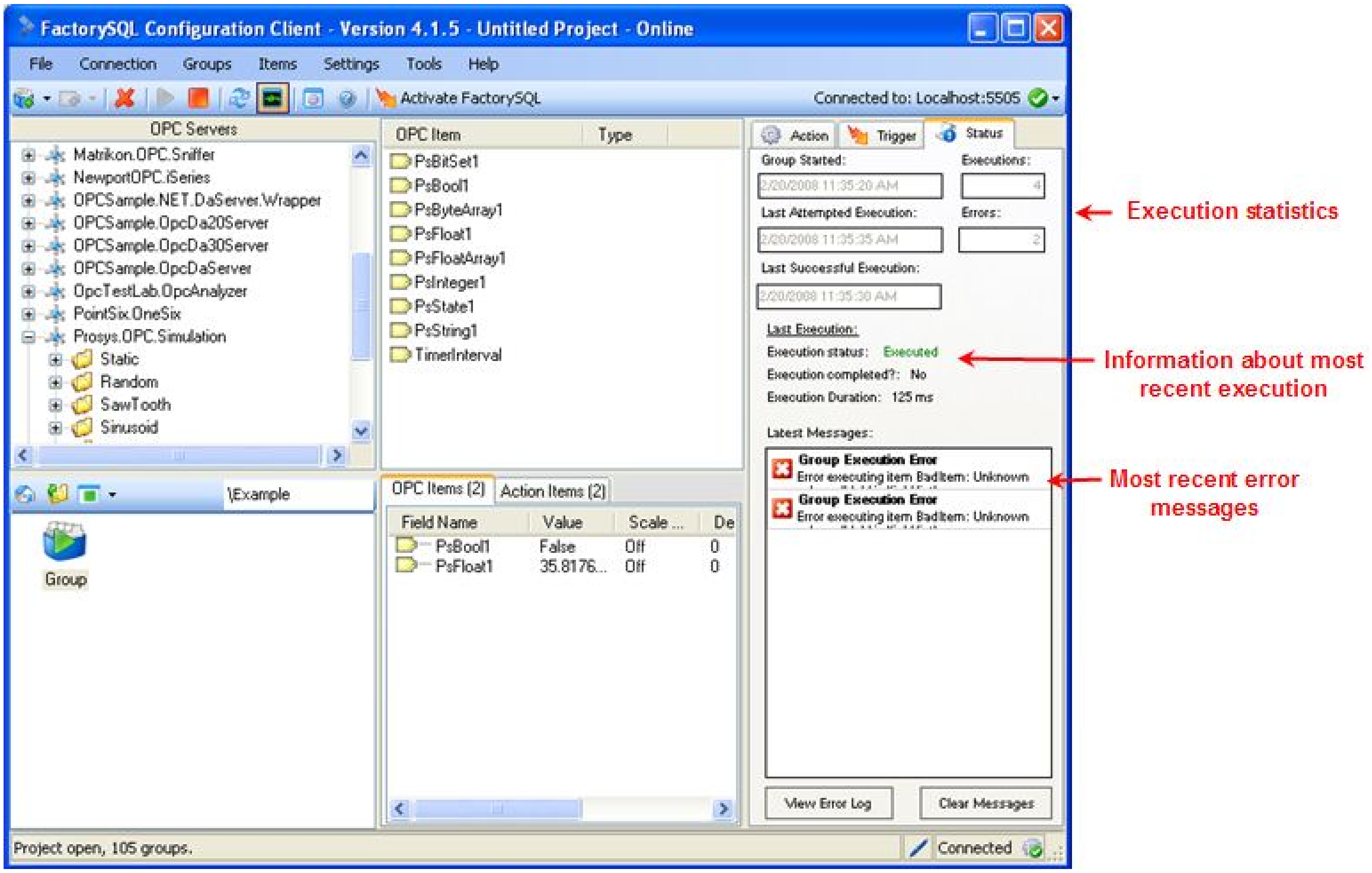
Handshakes

A handshake is a way to communicate the success or failure of a group to the PLC. There are 2 handshakes available to a group: a success handshake, and a failure handshake. They allow the user to write a specific value to a specific OPC item at the end of the group's execution cycle to communicate whether execution completed without problem, or whether some error occurred.

Note: Resetting the trigger is often used to indicate success. This is completely fine, and the success handshake can be used side by side with this mechanism to write extra information, if desired.

Group Status

The **Status Tab** provides information pertaining to how long the group has been running, how many times it has executed, and errors that may have occured. It is the first place to look when a group is not working as expected.


FactorySQL Configuration Client - Version 4.1.5 - Untitled Project - Online
File Connection Groups Items Settings Tools Help
Activate FactorySQL
Connected to: Localhost:5505
OPC Servers
Action Trigger Status
View Error Log
Clear Messages
Execution statistics
Information about most recent execution
Most recent error messages

OPC Servers: Matrikon.OPC.Sniffer, NewportOPC.iSeries, OPCSample.NET.DaServer.Wrapper, OPCSample.OpcDa20Server, OPCSample.OpcDa30Server, OPCSample.OpcDaServer, OpcTestLab.OpcAnalyzer, PointSix.OneSix, Prosys.OPC.Simulation (Static, Random, SawTooth, Sinusoid)

OPC Item	Type
PsBitSet1	
PsBool1	
PsByteArray1	
PsFloat1	
PsFloatArray1	
PsInteger1	
PsState1	
PsString1	
TimerInterval	

\Example — Group

OPC Items (2) | Action Items (2)

Field Name	Value	Scale ...	De
PsBool1	False	Off	0
PsFloat1	35.8176...	Off	0

Group Started: 2/20/2008 11:35:20 AM
Executions: 4
Last Attempted Execution: 2/20/2008 11:35:35 AM
Errors: 2
Last Successful Execution: 2/20/2008 11:35:30 AM

Last Execution:
Execution status: Executed
Execution completed?: No
Execution Duration: 125 ms

Latest Messages:
Group Execution Error
Error executing item BadItem: Unknown
Group Execution Error
Error executing item BadItem: Unknown

Project open, 105 groups. Connected

- **Group Started:** When the group was first "started" for execution- either by clicking "start group", or when the service was started.
- **Executions:** How many times the group has attempted to execute. NOTE: If a trigger is used, this number will only increment when the trigger is active.
- **Last Attempted Execution:** The last time the group was checked. This is updated irregardless of the trigger, therefore it should expected to update at a rate similar to the group update rate (note: the status refresh may be slower than the group rate, so some display lag may occur)
- **Errors:** The number of times the group's execution has errored out. Generally when an error occures in a group, execution is immediately halted.
- **Last Successful Execution:** The last time the group successfully executed completely. That is, the trigger was active (if applicable) and no errors occured.

Last Execution

- **Execution Status:** Whether the group executed, or didn't due to trigger/async settings.
- **Execution Completed?:** YES if the group executed without error, NO otherwise.
- **Execution Duration:** How long the group took to execute.

Latest messages: The last messages reported by the group, usually errors.

Item Substitution

Item Substitution is a powerful feature of FactorySQL that allows the user to "plug in" an OPC value into another field, such as the Where Clause, and Action Item statement, or an analog alarm point.

In general, to insert a value into a field, press **Control+Space** to bring up the menu. You may also right click a valid field and choose "Reference Item".

In the WHERE Clause

A common application is to use a PLC register to select a database record to use for the group. This is done by using a **Custom WHERE Clause** as described in the Action tab settings and substituting in an OPC value. This is very useful for recipe systems or other situations where you want to use the database as a repository of values that the PLC can choose to load.

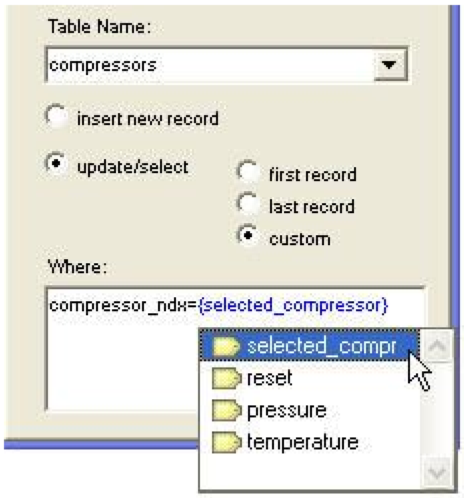


Table Name: compressors

( ) insert new record
(•) update/select — ( ) first record ( ) last record (•) custom

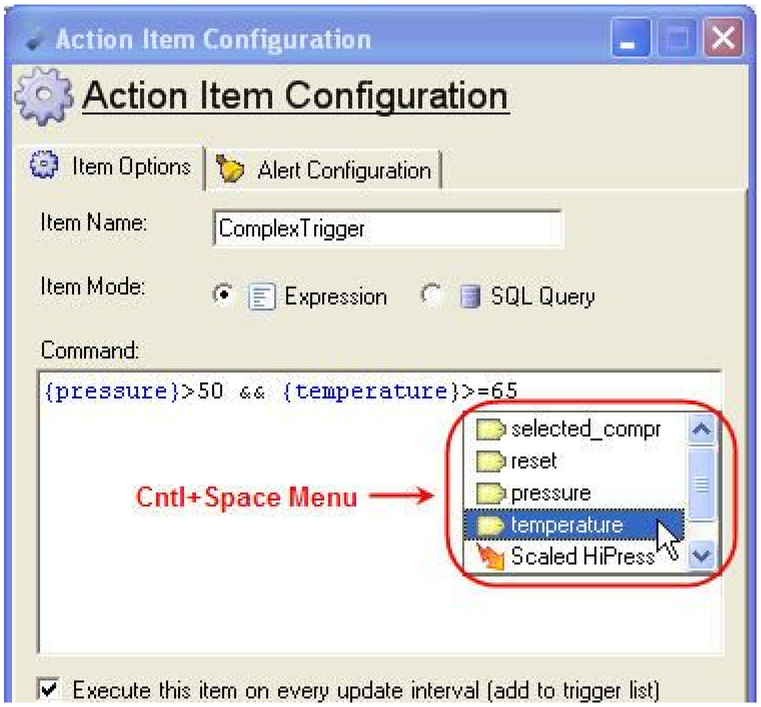
Where:

```
compressor_ndx={selected_compressor}
```

Within Action Items

OPC path values can also be plugged directly into Action Items.

This is typically used to scale/perform calculations on OPC values, or to perform logical operations for triggering or alerting. The result can be stored in the database or written back to the PLC.



Item Name: ComplexTrigger

Item Mode: (•) Expression ( ) SQL Query

Command:

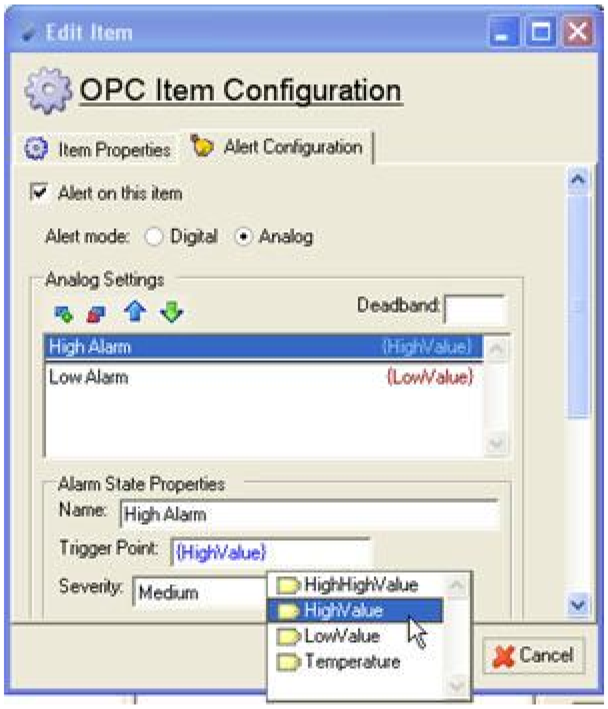
```
{pressure}>50 && {temperature}>=65
```

[✓] Execute this item on every update interval (add to trigger list)

Store result to DB field or OPC item: [ ]

Within Alert Items

You can use Item Substitution to set the trigger point for analog alert items. This allows you to have dynamic alert levels that can be changed from the PLC. When using Item Substitution, you can use the Up and Down arrows to position the item exactly as you'd like. For more information on alerting, see Alert Configuration.



[✓] Alert on this item

Alert mode: Digital [✓ Analog]

Analog Settings

Deadband:

High Alarm	{HighValue}
Low Alarm	{LowValue}

Alarm State Properties

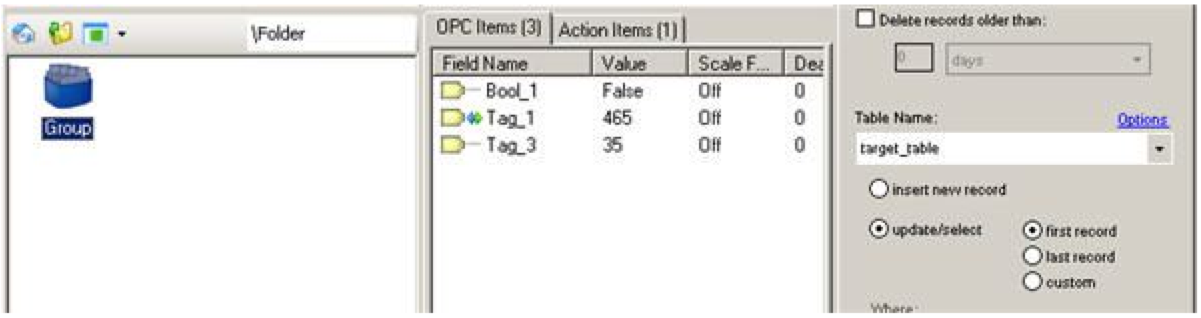
Name: High Alarm

Trigger Point: {HighValue}

Severity: Medium

Standard Groups

Standard Groups are the core group type in FactorySQL. They are very versitile, and provide a wide range of features.



\Folder

Group

OPC Items (3) | Action Items (1)

Field Name	Value	Scale F...	De
Bool_1	False	Off	0
Tag_1	465	Off	0
Tag_3	35	Off	0

☐ Delete records older than: 0 days

Table Name: target_table

Options

○ insert new record
[✓ update/select] — [✓ first record] ○ last record ○ custom

Where:

Features

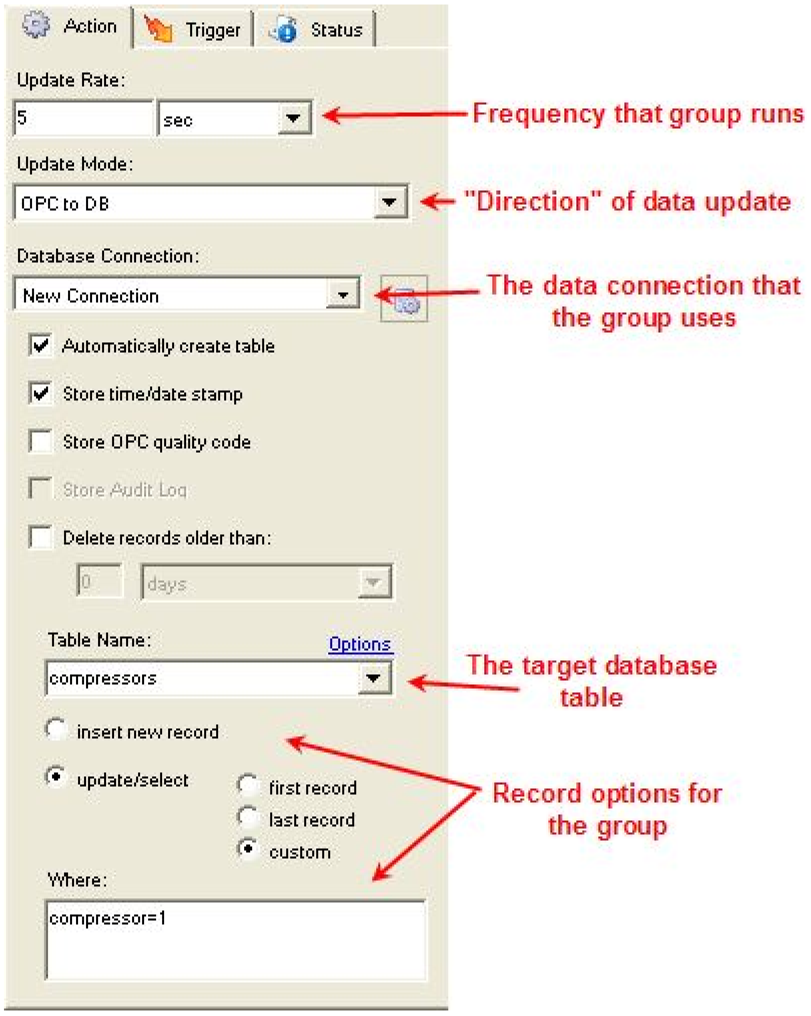
- **Easy, flexible mapping between OPC and DB**: Drag and drop items, quickly change their column names, target table & row, and start synchronizing.
- **Historical Logging**: Simply set table mode to "Insert new record" and your instantly logging historical data.
- **Full bi-directional support**: Keep OPC values and database values perfectly synched.
- **Column definable bi-directional mode**: Mix and match bi-directional columns to create complete status and control systems in one group.

Items

- **OPC Items**: Pull in OPC data. Can be mapped to a database field, or used internally for trigger, expressions, where clause lookup, etc.
- **Action Items**: Can be expressions or SQL queries, can reference other item values, and can write their value to the database or OPC server.

Group Options - Action Tab

The Action tab is where the core settings for a group's execution are configured. You can set the update interval and mode, the database connection used, and the table in the database used. You can also select other features and data that will be tied to the group.


Action
Trigger
Status
Frequency that group runs
"Direction" of data update
The data connection that the group uses
The target database table
Record options for the group
Options

Update Rate: 5 sec

Update Mode: OPC to DB

Database Connection: New Connection

[✓] Automatically create table

[✓] Store time/date stamp

[ ] Store OPC quality code

[ ] Store Audit Log

[ ] Delete records older than: 0 days

Table Name: compressors

( ) insert new record

(●) update/select — ( ) first record ( ) last record (●) custom

Where: compressor=1

Update Rate: Defines a number and interval unit for the timing of a transactional group. The group will run once when you start the group then again on that interval. For example, selecting 2 minutes means that FactorySQL will inspect the group for changes every two minutes.

Update Mode:

Defines the "direction" that data is updated between OPC addresses and the database

OPC->DB: Reads the PLC and writes values to the database. This mode is most often used for data logging with the "insert new record" option, or to show realtime status with "update/select" a given record selected.

DB->OPC: Reads the database and writes changed values to the PLC.

Bi-directional (OPC wins): Reads the database and PLC. If either side has changed during the update interval and the other hasn't, both will take the new value. If both sides have changed, the database value will be overwritten by the PLC value.

Bi-directional (DB wins): Reads the database and PLC. If either side has changed during the update interval and the other hasn't, both will take the new value. If both sides have changed, the PLC value will be overwritten by the database value.

Additional Options:

Automatically create table: When you start a group FactorySQL will create a table in the database if it doesn't exist. It will also automatically create columns in the database for each OPC item that doesn't exist. If the table already exists, FactorySQL will ask the user if they want to add the columns.

Store time/date stamp: FactorySQL will maintain a "datetime" data type called *t_stamp* that updates to the current time whenever FactorySQL writes data in either direction. This is essential for data logging and often useful in any mode.

Store OPC quality code: Every value that comes from the OPC server has a "quality code" associated with it, that defines if it is a vaild value, or if an error occured along the way. If you select this box, FactorySQL will maintain the code in the database under the column *quality_code.* This can be useful for troubleshooting problems between the OPC server and device, or for monitoring a connection to a PLC. The codes from all items will be logically ANDed together, so if one item is bad, the value in the database will reflect a bad quality. This is important to keep in mind when mixing items from different OPC servers in the same group, when some items may be good and others bad at the same time.

Store Audit Log: Inserts a record into an "audit table" in the database whenever a value gets written from the database to the PLC. This is useful to track operator settings changes. More on **Auditing**

Delete records older than: Purges data in the database table that is older than the specified time duration. This is convenient for a rapid data logging group, so that the table does not grow to an unmanagable or unnecessary size.

Record Options:

Determine which record in the database will be updated for the transaction group

Table Name: The name of the table in the database to use.

insert new record (OPC->DB only): FactorySQL will insert a new record into the database every time the group runs. Typically, this option is used to log data at a regular interval. When combined with trigger options, the group can be set to either log at certain times, or be used to create "snapshots" on some condition.

update/select record: In every "update mode" (except when logging data) FactorySQL groups tie (map) a single database record to a set of OPC paths. This setting defines how the group finds its record.

update/select first: FactorySQL uses the first record as defined by *tablename*_ndx. It is the best option if there is only one record in the table, usually for data that isn't templated.

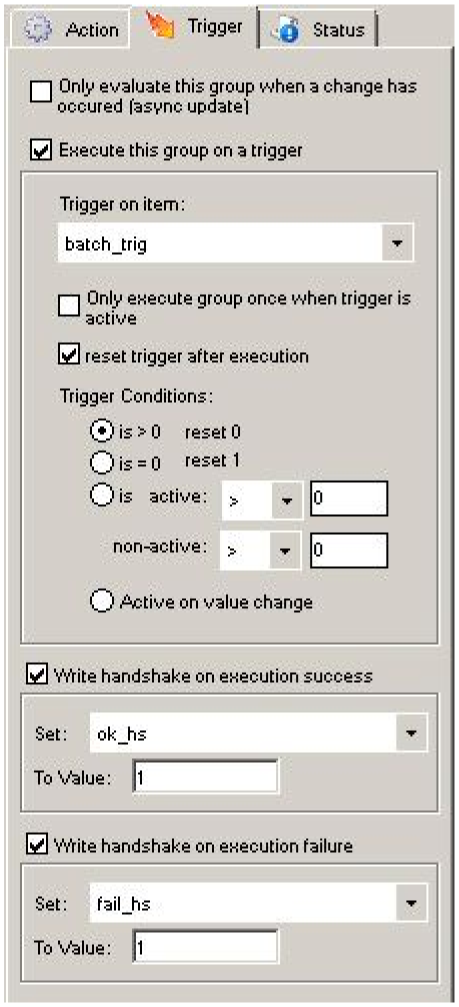
update/select last: FactorySQL uses the last record as defined by *tablename*_ndx. This is especially useful for a rolling incremental "current group". For example, a bottling line may have a current production count that is always the last record. The group will always keep the last record up to date, while another group adds a new record every shift. The result, a table that shows a historical snapshot of production at the end of each shift as well as realtime indication on the last record.

update/select custom: FactorySQL uses an SQL WHERE clause to determine which record to use. The typical usage is to choose where an index column equals some value. Click here for an example.

A powerful feature of this is to use the "item substitution" feature for indirect addressing. This allows for things like having a PLC register point to a recipe in a batching system, then using the database to maintain many different recipes.

Group Options - Trigger Tab

Triggering allows you to specify more precisely when a group will be executed. The **Trigger Tab** allows you to configure the condition that will allow the group to run, along with a few other related settings.



Action | Trigger | Status

☐ Only evaluate this group when a change has occured (async update)

☑ Execute this group on a trigger

Trigger on item: batch_trig

☐ Only execute group once when trigger is active

☑ reset trigger after execution

Trigger Conditions:

● is > 0 reset 0
○ is = 0 reset 1
○ is active: > 0
non-active: > 0
○ Active on value change

☑ Write handshake on execution success

Set: ok_hs
To Value: 1

☑ Write handshake on execution failure

Set: fail_hs
To Value: 1

Only evaluate this group when a change has occurred: When checked, the group will only be evaluated when a change has occurred to an OPC value since the last update. This is useful when you only want to log changes, and not just on a set interval. This setting takes precedence over the rest of the trigger settings.

Execute this group on a trigger: This option specifies to use the trigger. If it is not selected, the group will run every update interval as specified in the **Action Tab**. If it is selected, the trigger condition must be true for the group to do anything (except run an Action Item that is set to run every update interval).

Trigger Options:

Trigger on item: select an item to be used as the trigger. This can be an OPC item or an Action Item set to run every interval.

Only execute group once when trigger is active: Only allows the transaction group to run once when the trigger is true and then not again until the trigger goes to a false state, then returns true. Also known as "One-Shot" triggering.

Reset trigger after execution: writes a predefined false value to the trigger address after execution. Only applies to the first 2 trigger conditions - "is > 0" and "is = 0".

Prevent trigger caused by group start: certain trigger conditions can cause the group to execute each time it is started, for instance "active on value change". This setting prevents this execution from occurring. If the group is counting items or being triggered after the production of an item, detection of a condition, etc. it is usually not desirable to log on startup. However, if the group is logging general history, it may be useful to log on startup.

Trigger Conditions:

These define exactly when the group is considered to in and out of trigger.

Is > 1, Reset 0: If the trigger item is >1, the group is triggered and will execute. If "Reset Trigger" is active, a 0 will be written after execution.

Is = 0, Reset 1: Group will execute if value is 0, and a 1 will be written to the trigger item if "Reset Trigger" is selected.

Is active, non-active: These conditions specify when the group is considered to be active and not active. There is no actual trigger reset in this mode. Note that once a group is "active", it will continue to log until the "non-active" condition is true. That is, it does not stop simply when the "active" condition is false.

Active on value change: The group will be activated for 1 cycle when the trigger value changes. This is the only trigger condition that naturally only executes the group 1 time.

Handshakes

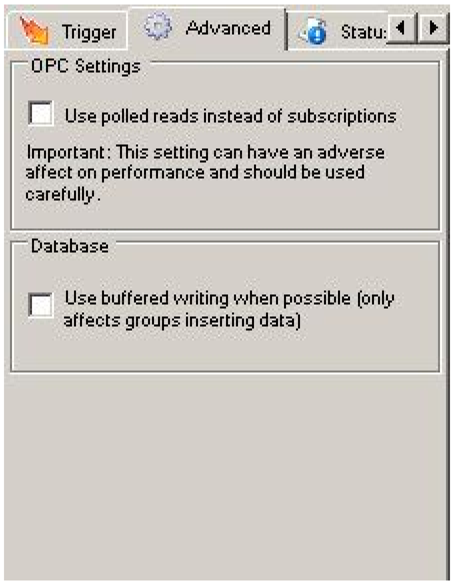
A handshake is a way to communicate the success or failure of a group to the PLC. There are 2 handshakes available to a group: a success handshake, and a failure handshake. They allow the user to write a specific value to a specific OPC item at the end of the group's execution cycle to communicate whether execution completed without problem, or whether some error occurred.

Note: Resetting the trigger is often used to indicate success. This is completely fine, and the success handshake can be used side by side with this mechanism to write extra information, if desired.

Group Options - Advanced Tab

IMPORTANT: The Advanced Options tab is not displayed by default. It must first be enabled in Frontend Settings.

The advanced tab houses a few additional settings that affect group execution. They are not necessarily advanced in what they do, but are designated as such in order to reduce confusion among new users.



OPC Settings

☐ Use polled reads instead of subscriptions

Important: This setting can have an adverse affect on performance and should be used carefully.

Database

☐ Use buffered writing when possible (only affects groups inserting data)

OPC Settings

Use polled reads instead of subscriptions: Normally FactorySQL acquires data from the OPC server on a subscription basis. That is, FactorySQL provides the server with a list of items to monitor (the items in a group, for instance) and the server provides notifications whenever the value or quality of an item changes.
By selecting this option, FactorySQL will instead call a Read function on the server each execution in order to retrieve the values. This behavior can result in much more work for both FactorySQL and the server, and is generally discouraged by the OPC Foundation. However, there are certain situations in which this option is very useful. In particular, it is useful for batch operations, which are usually triggered. When the trigger goes high, FactorySQL will read the values, and you can know for certain that the values retrieved are the most recent. Using subscriptions there is the slight possibility that the trigger may arrive before the other data, and thus the older data would be logged.

Even with this option selected, certain items **will still be subscribed**:

- The trigger
- Any item that is referenced by a run-alway action item (assuming the group is triggered).

This way, FactorySQL is able to monitor the trigger or any items necessary for the trigger, and only call read when the group actually needs to execute.

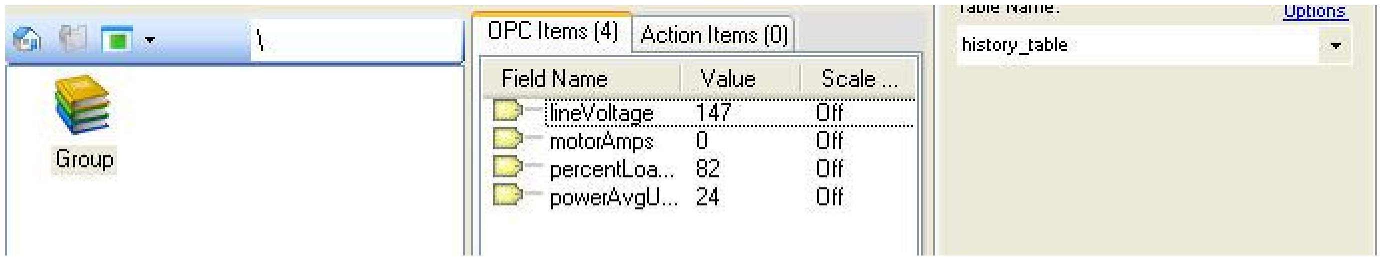
Database Settings (Standard group only)

Use buffered writing when possible: If selected, FactorySQL will write historical data to a buffer instead of directly to the database. By doing this, it is less likely that data will be lost when the database connection goes down. Even with data caching enabled, there is a certain period of time required to determine connection failure. Without buffering, the group cannot execute during this time. With buffering enabled, however, the data will be queued until the switch over to the data cache is complete.

On the downside, the buffer executors currently only use 1 thread per connection, so concurancy is more limited than with normal unbuffered execution.

Historical Groups

Historical Groups make it quick and easy to log historical OPC data. Configuration is as simple as drag and drop.



Field Name	Value	Scale ...
lineVoltage	147	Off
motorAmps	0	Off
percentLoa...	82	Off
powerAvgU...	24	Off

history_table

Features

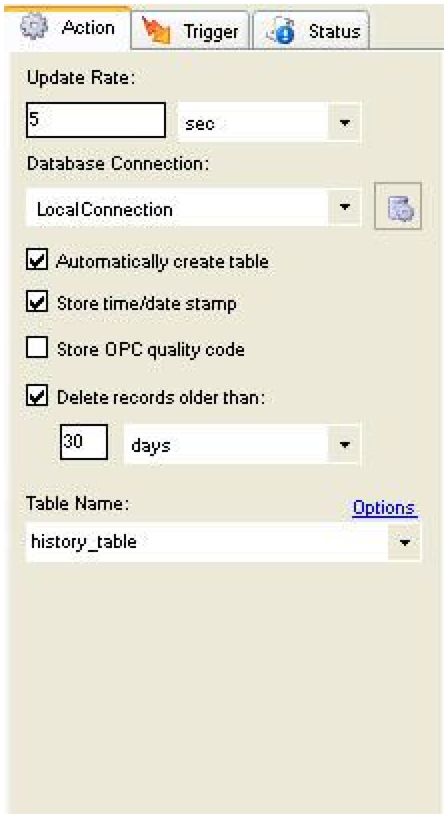
- **Easy configuration**: Drag and drop items, select a table and an update rate, and start logging!
- **Buffered data writing**: Data is written through a buffer, which works hand-in-hand with the data cache to prevent data loss should the database become unavailable.
- **Powerful item modes**: Hourmeter and event meter item modes make it quick and easy to perform downtime and event tracking.
- **Full trigger support**: Utilize triggers to intelligently log data and track events.
- **Action item support**: Create expression-based tags, perform mathmatical functions, run sql queries and more.

Items

- **OPC Items**: Pull in OPC data. Can be mapped to a database field, or used internally for trigger, expressions, etc.
- **Action Items**: Can be expressions or SQL queries, can reference other item values, and can write their value to the database or OPC server.

Historical Group Options - Action Tab

The Action tab is where the core settings for the group's execution are configured. Primarily you will set how often the data is logged, at to which table in a specific database.



Update Rate: 5 sec

Database Connection: LocalConnection

☑ Automatically create table

☑ Store time/date stamp

☐ Store OPC quality code

☑ Delete records older than: 30 days

Table Name: history_table

Update Rate: Defines a number and interval unit for the timing of a historical group. When running, the group will be checked at this interval. If the trigger condition is valid, or if a trigger is not defined, data will be logged.

Database Connection: Which database connection to log data to.

Table Name: The name of the table that the data will log to.

Additional Options:

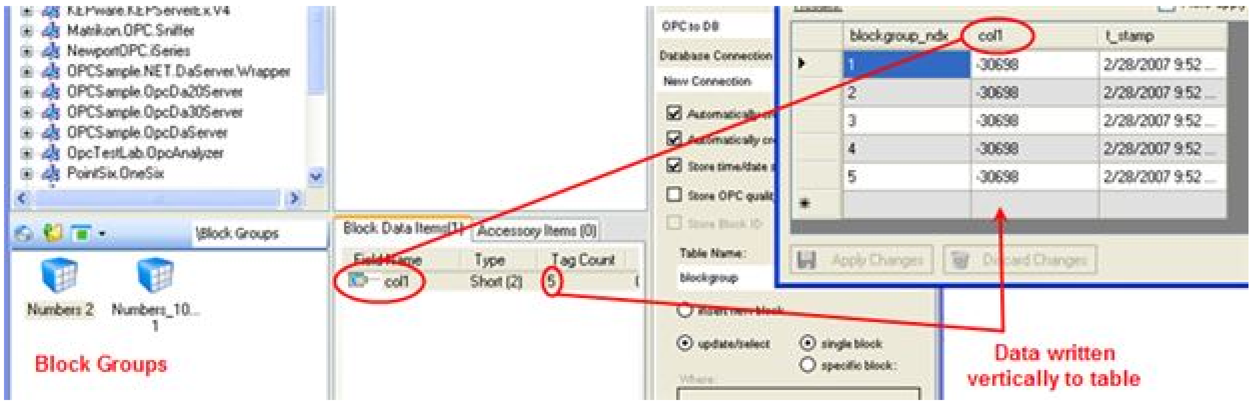
Automatically create table: When you start a group FactorySQL will create a table in the database if it doesn't exist. It will also automatically create columns in the database for each item that doesn't exist. If the table already exists, FactorySQL will ask the user if they want to add the columns.

Store time/date stamp: FactorySQL will maintain a "datetime" that indicates the last time FactorySQL logged data. This is usually very important for data logging situations.

Store OPC quality code: Every value that comes from the OPC server has a "quality code" associated with it, that defines if it is a vaild value, or if an error occured along the way. If you select this box, FactorySQL will maintain the code in the database. This can be useful for troubleshooting problems between the OPC server and device, or for monitoring a connection to a PLC. The codes from all items will be logically ANDed together, so if one item is bad, the value in the database will reflect a bad quality. This is important to keep in mind when mixing items from different OPC servers in the same group, when some items may be good and others bad at the same time. **Note:** To store individual qualities, copy and paste the items, and then select the "Quality" property under their configuration windows.

Block Data Groups

Block Data Groups are very similar to standard groups, except they are designed to write to multiple rows of a table at one time, creating a very efficient "data block".



blockgroup_ndx	col1	t_stamp
1	-30698	2/28/2007 9:52 ...
2	-30698	2/28/2007 9:52 ...
3	-30698	2/28/2007 9:52 ...
4	-30698	2/28/2007 9:52 ...
5	-30698	2/28/2007 9:52 ...

Features

Block Data Groups support most standard group features, and some extras:

- **Full bi-directional support**: Very useful for creating efficient recipe systems.
- **Column definable bi-directional mode**: Mix and match bi-directional columns to create complete status and control systems in one group.
- **Multiple segments per column**: Allows you to stack data from different memory block or OPC servers in the same column.
- **Efficient**: By blocking the data, throughput is greatly increased over using multiple standard groups.

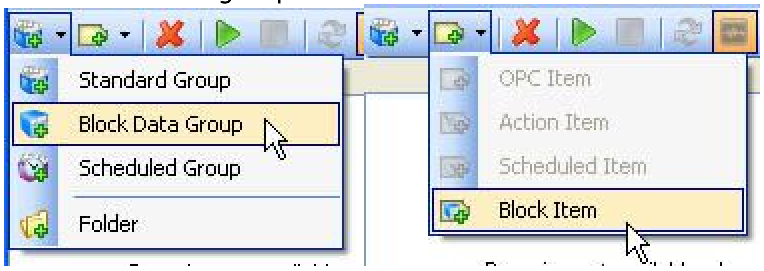
Items

Block data groups contain 2 distinctive groups of items:

- **Block Data Items**: These items represent the columns (and data) that the group will read & write. Each block item defines its column name, data type, and mode, and contain one or more **Segments**, which define the actual OPC addresses of the data.
- **Accessory Items**: These are standard **OPC Items** and **Action Items**, that can be used by the block data group as a trigger or in the where clause. They are not stored to the database.

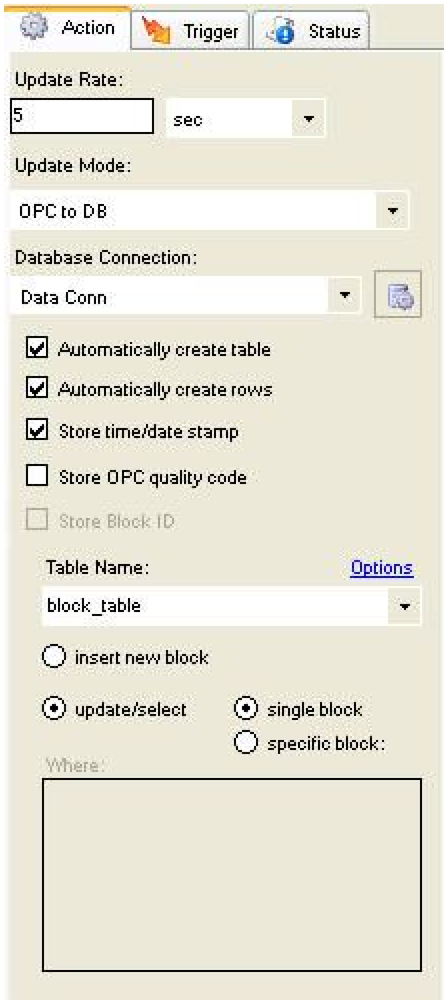
Creating Block Data Groups

New block data groups can be created from the toolbar or by right clicking on the project pane:



Block Data Group Options

Most of the options on the block data group action tab perform the same function as they do for standard groups. There are a few settings, however, that are unique to block groups.



Update Rate: 5 sec

Update Mode: OPC to DB

Database Connection: Data Conn

[✓] Automatically create table

[✓] Automatically create rows

[✓] Store time/date stamp

[ ] Store OPC quality code

[ ] Store Block ID

Table Name: block_table Options

( ) insert new block

(●) update/select (●) single block ( ) specific block:

Where:

Block Data Group Options

Automatically create rows: Automatically add the rows necessary to run the group. If false, FactorySQL will ask to create rows when starting the group.

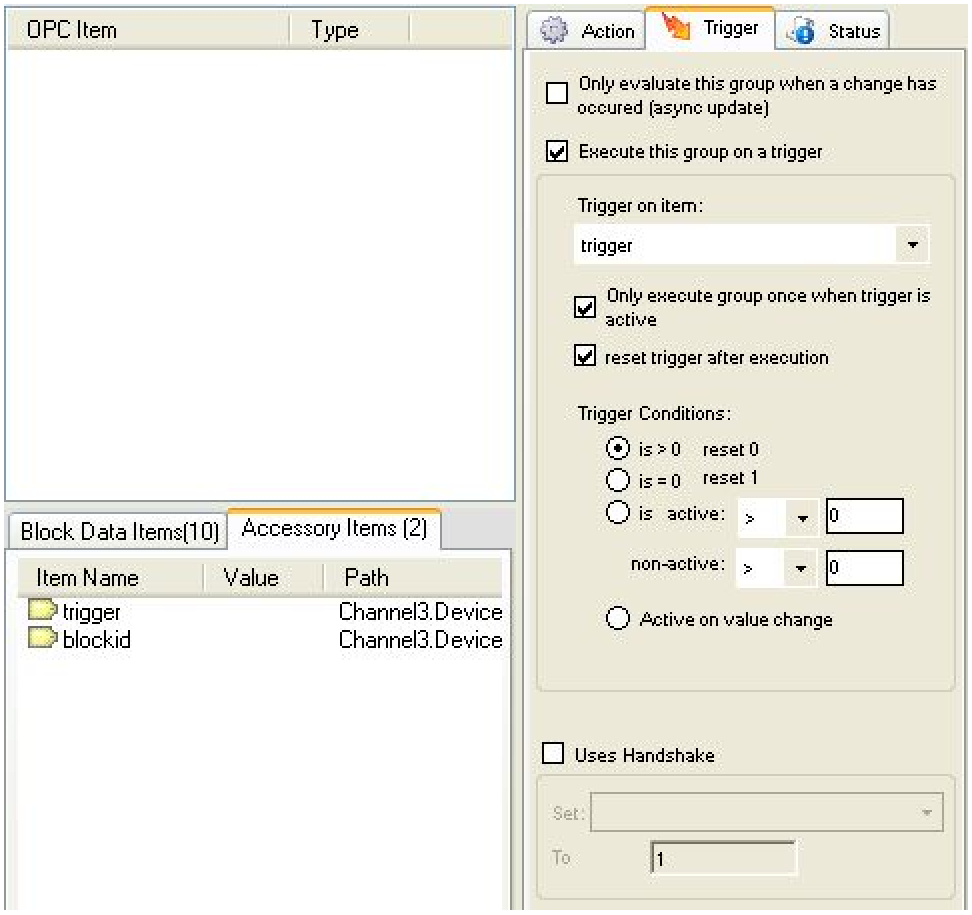
Store Block ID: Only available when inserting block, this option stores a sequential identifier that is unique for each data block.

Insert new block (OPC->DB only): FactorySQL will insert the block into the database every time the group runs. Typically, this option is used to log data at a regular interval. When combined with trigger options, the group can be set to either log at certain times, or be used to create "snapshots" on some condition.

Update/select: Selects a specific block to read and write from. Either the first block (**single block**), or a block selected by a specific SQL where clause. You may also use the "item substitution" feature for indirect addressing. This allows for things like having a PLC register point to a recipe in a batching system, then using the database to maintain many different recipes.

Triggering

Block group trigger works exactly the same as Standard Group Triggering. The only nuance is that the trigger item must be an **Accessory Tag**. If you would like to use a tag that is part of your datablock as the trigger, you'll have to add a duplicate into the accessory item list. There is very little overhead in having duplicate tags in block groups, as FactorySQL treats these tags as 1 point.



Item Name	Value	Path
trigger		Channel3.Device
blockid		Channel3.Device

☐ Only evaluate this group when a change has occured (async update)

☑ Execute this group on a trigger

Trigger on item: trigger

☑ Only execute group once when trigger is active

☑ reset trigger after execution

Trigger Conditions:

[✓ is > 0 reset 0]
is = 0 reset 1
is active: > 0
non-active: > 0
Active on value change

☐ Uses Handshake

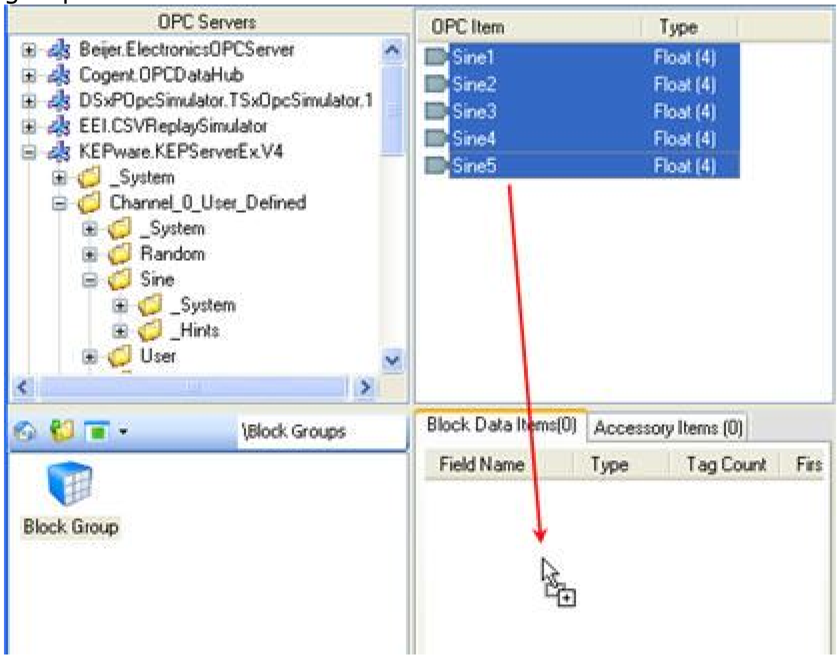
Set:
To 1

Block Data Items & Segments

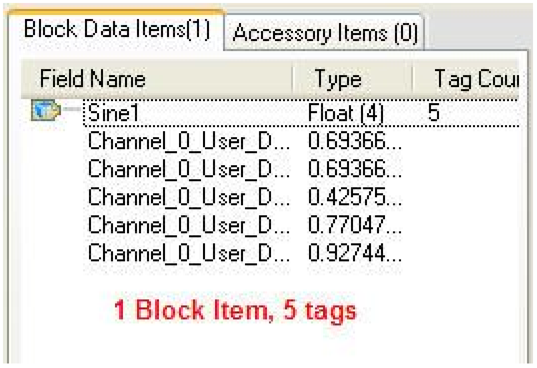
A **Block Data Item** defines a database column to read & write to, a data type, an operation mode, and a set of addresses to use. Addresses are defined as **Segments**. Each block data item contains one or more **Segments**. A segment defines a set of OPC addresses. A block data item can contain any number of segments, for the sake execution they will be compiled down to a single list of addresses. There are currently 2 types of segments: the **Address List** segment, and the **Range** segment.

Creating Block Data Items

Block data items can be created from the item toolbar menu, or by dragging tags from the OPC browser into the group.



Which creates:

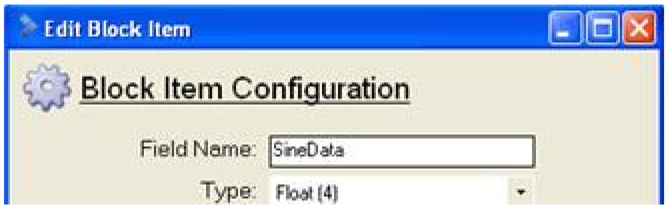


Field Name	Type	Tag Cou
Sine1	Float (4)	5
Channel_0_User_D...	0.69366...	
Channel_0_User_D...	0.69366...	
Channel_0_User_D...	0.42575...	
Channel_0_User_D...	0.77047...	
Channel_0_User_D...	0.92744...	

Note: To view addresses contained by an item, and their values, you have to **Expand** the item, by using the right click menu or CTRL-E.

Address List Segment

This type of segment is simply an ordered list of OPC addresses. There is no need for the addresses to be contiguous.

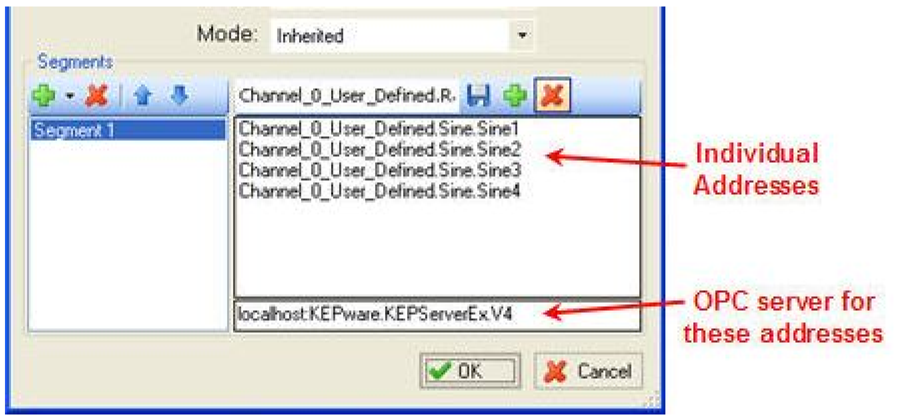


Edit Block Item

Block Item Configuration

Field Name: SineData

Type: Float (4)


Mode: Inherited

Segments

Segment 1

Channel_0_User_Defined.R

Channel_0_User_Defined.Sine.Sine1
Channel_0_User_Defined.Sine.Sine2
Channel_0_User_Defined.Sine.Sine3
Channel_0_User_Defined.Sine.Sine4

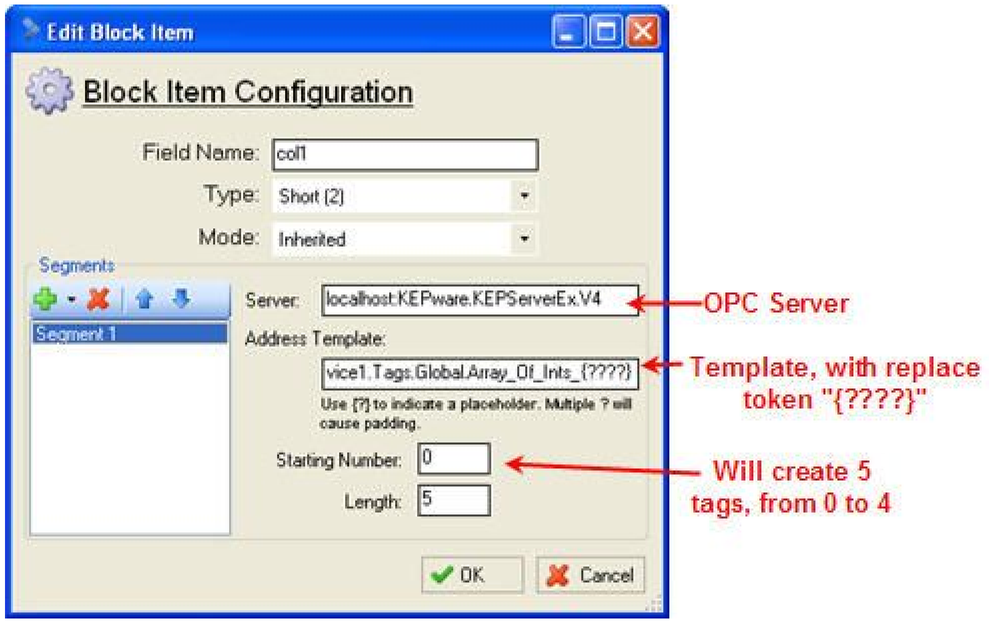
localhost:KEPware.KEPServerEx.V4

Range Segment

The range segment allows you to define an address template, a starting address, and a length. In this way, you can specify a set of contiguous addresses quickly.

Parameters:
Server: The OPC server this segment uses.
Address Template: The template that will be used to create the item paths. Use "{?}" to denote the unique part of the address. Multiple "?" will cause padding.
For example:
TagPath.DataPoint{?} will become TagPath.DataPoint0
TagPath.DataPoint{???} will become TagPath.DataPoint000

Starting number: The address to begin with.
Length: The number of addresses to create.


Edit Block Item

Block Item Configuration

Field Name: col1
Type: Short (2)
Mode: Inherited

Segments

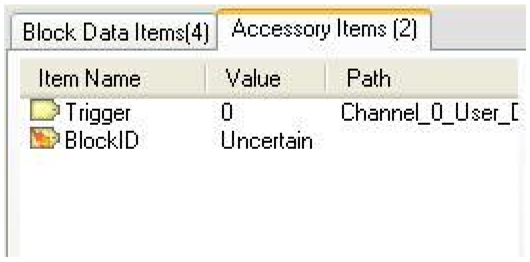
Segment 1

Server: localhost:KEPware.KEPServerEx.V4
Address Template: vice1.Tags.Global.Array_Of_Ints_{????}
Use {?} to indicate a placeholder. Multiple ? will cause padding.
Starting Number: 0
Length: 5

OPC Server
Template, with replace token "{????}"
Will create 5 tags, from 0 to 4
OK
Cancel

Block Data Accessory Items

Accessory Items are simply **OPC Items** and **Action Items** that can be used by block data groups for tasks such as **triggering** and **dynamic where clauses**. The only difference is that accessory items cannot currently be written to the database.

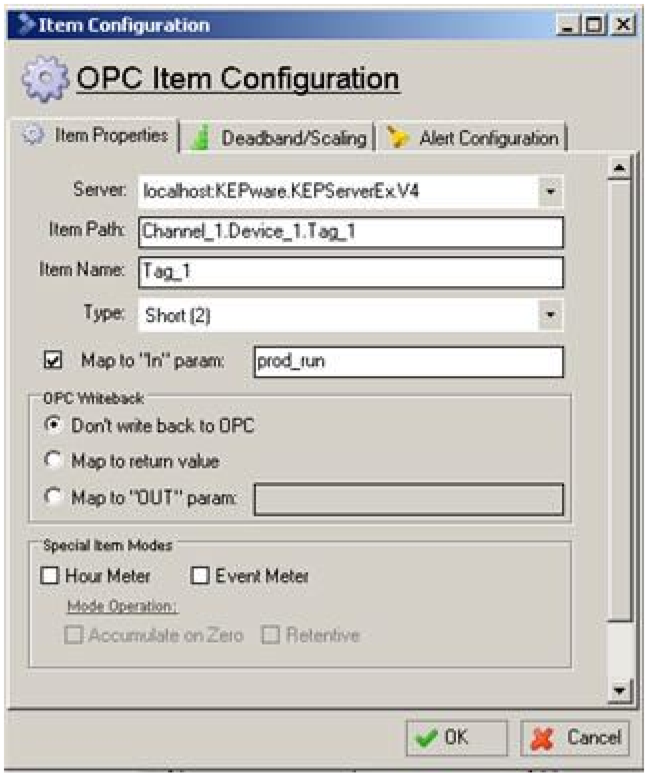


Item Name	Value	Path
Trigger	0	Channel_0_User_[
BlockID	Uncertain	

Mapping Parameters

Both OPC items and Action items can be mapped to stored procedure parameters. It is done through the item's configuration dialog.

OPC Items

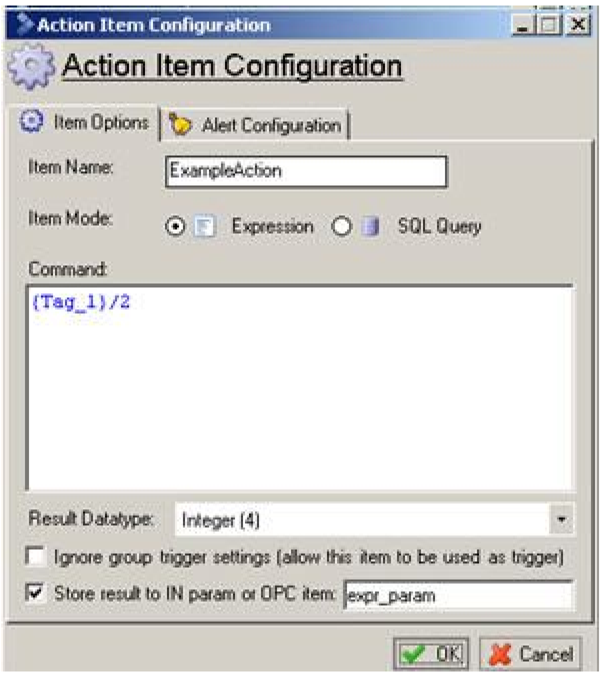


Server: localhost:KEPware.KEPServerEx.V4
Item Path: Channel_1.Device_1.Tag_1
Item Name: Tag_1
Type: Short (2)
[✓] Map to "In" param: prod_run
[✓ Don't write back to OPC] Map to return value / Map to "OUT" param:
[ ] Hour Meter [ ] Event Meter
[ ] Accumulate on Zero [ ] Retentive

OPC Items can be mapped to and from the stored procedure.
To map the value to an **IN** parameter, simply select the check box for "In Param" and set the parameter name.
NOTE: FactorySQL will automatically add any special parameter name character neccessary.
Coming back from the procedure, the item may be bound to an **OUT** parameter, or to the **Return value**.

You may also bind to an **INOUT** parameter by using the same parameter name on both the IN and OUT fields.

Action Items



Item Name: ExampleAction
Item Mode: [✓ Expression] SQL Query
Command:

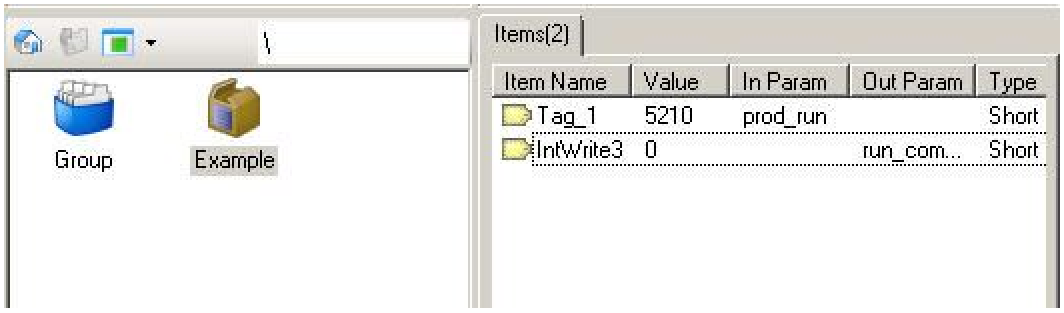
```
{Tag_1}/2
```

Result Datatype: Integer (4)
[ ] Ignore group trigger settings (allow this item to be used as trigger)
[✓] Store result to IN param or OPC item: expr_param

Action items can be mapped to **IN** parameters by selecting the "Store result..." checkbox and entering the parameter name.

Stored Procedure Groups

Stored Procedure Groups make it quick and easy to map OPC values to and from stored procedure parameters. With the stored procedure group, you can effectively leverage the power of stored procedures.



Item Name	Value	In Param	Out Param	Type
Tag_1	5210	prod_run		Short
IntWrite3	0		run_com...	Short

Features

Stored procedure groups support standard group features like triggering and handshaking, along with the following:

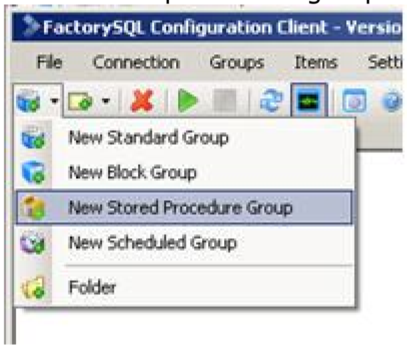
- **Full bi-directional support**: Not only can you map values to inputs, you can also map outputs and return values to OPC items.
- **Action items as inputs**: Map the value of action items to parameter inputs.
- **Include timestamp and quality parameters**

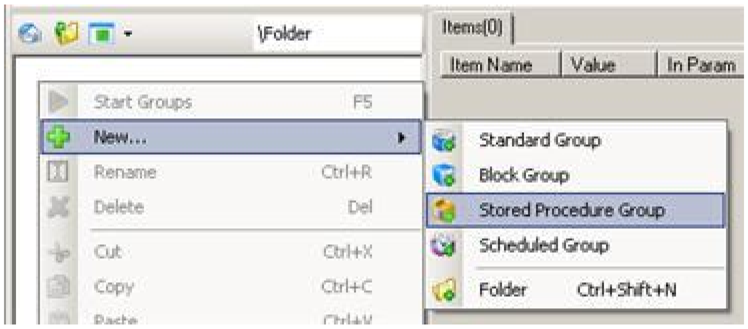
Items

Stored procedure groups support slightly modified **OPC Items** and **Action Items**.

Creating Stored Procedure Groups

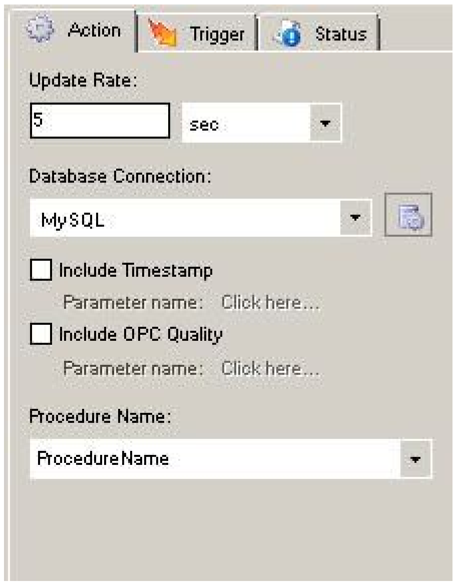
New stored procedure groups can be created from the toolbar or by right clicking on the project pane:





Stored Procedure Group Options

The stored procedure group's action tab is relatively simple. There is the standard update rate and database connection, and then several options specific to SP groups.



Update Rate: 5 sec

Database Connection: MySQL

☐ Include Timestamp
Parameter name: Click here...

☐ Include OPC Quality
Parameter name: Click here...

Procedure Name: ProcedureName

Stored Procedure Group Options

Include Timestamp: If selected, the current timestamp will be included as an input parameter to the procedure. The parameter name is selected by clicking the link.

Include Quality: The data quality of the items will be included in the parameter specified.

Procedure Name: The name of the procedure to target.

Trigger Tab

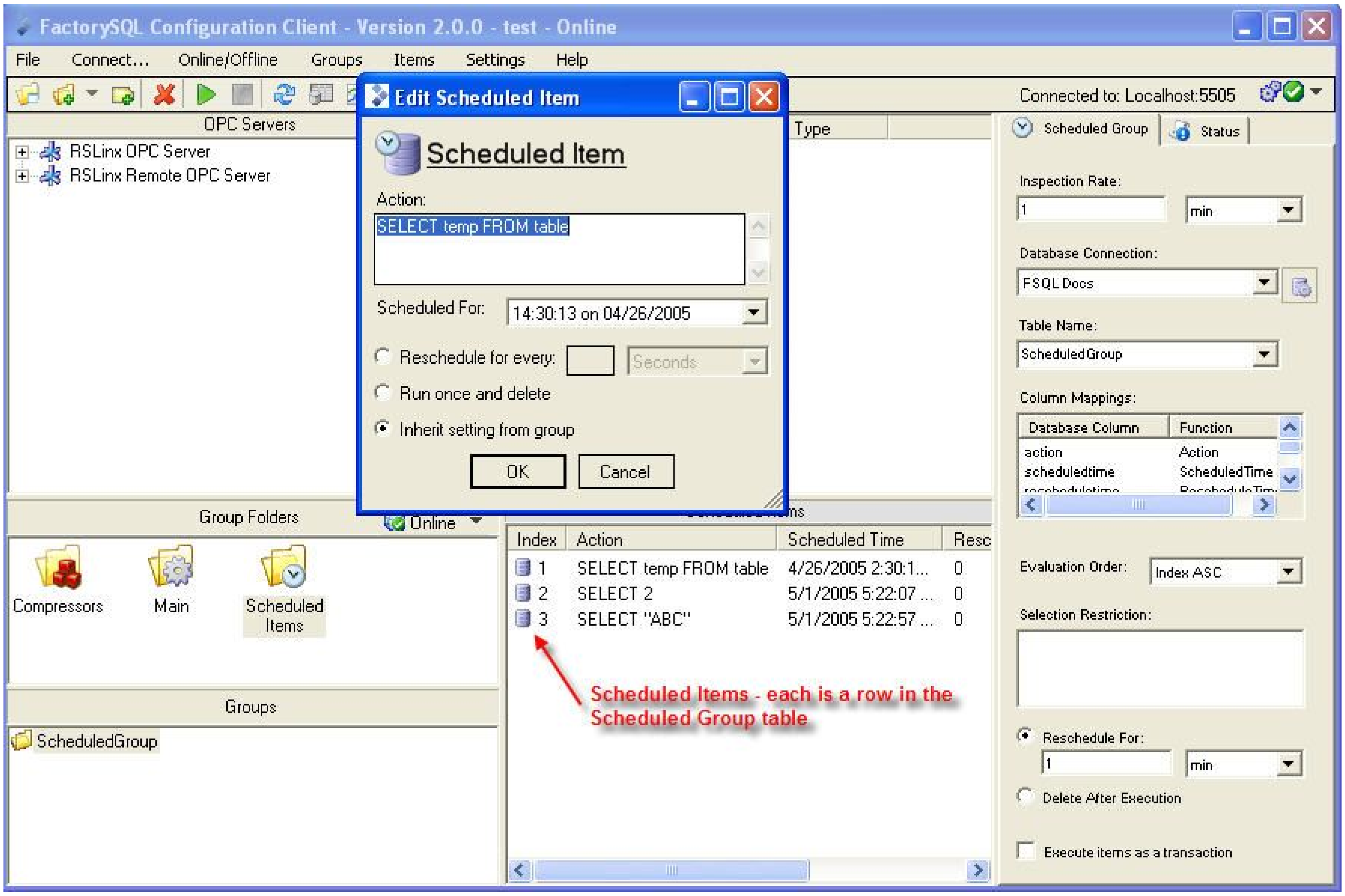
The Stored Procedure group shares the same trigger tab as the other groups.

Scheduled Groups

NOTE: As of FactorySQL 4.0 Scheduled Groups are DEPRECATED. It is advised that they should only be used if absolutely necessary. Usually, the functionality of scheduled groups can be accomplished using action items.

Scheduled Groups read queries from the database then run them. The queries are defined inside of **scheduled items**.

For more information on configuring scheduled groups, see **Group Options**.



Edit Scheduled Item

Scheduled Item

Action: SELECT temp FROM table

Scheduled For: 14:30:13 on 04/26/2005

( ) Reschedule for every: [ ] Seconds

( ) Run once and delete

(●) Inherit setting from group

OK | Cancel

Index	Action	Scheduled Time	Resc
1	SELECT temp FROM table	4/26/2005 2:30:1...	0
2	SELECT 2	5/1/2005 5:22:07 ...	0
3	SELECT "ABC"	5/1/2005 5:22:57 ...	0

Inspection Rate: 1 min

Database Connection: FSQL Docs

Table Name: ScheduledGroup

Column Mappings:

Database Column	Function
action	Action
scheduledtime	ScheduledTime

Evaluation Order: Index ASC

Selection Restriction:

(●) Reschedule For: 1 min

( ) Delete After Execution

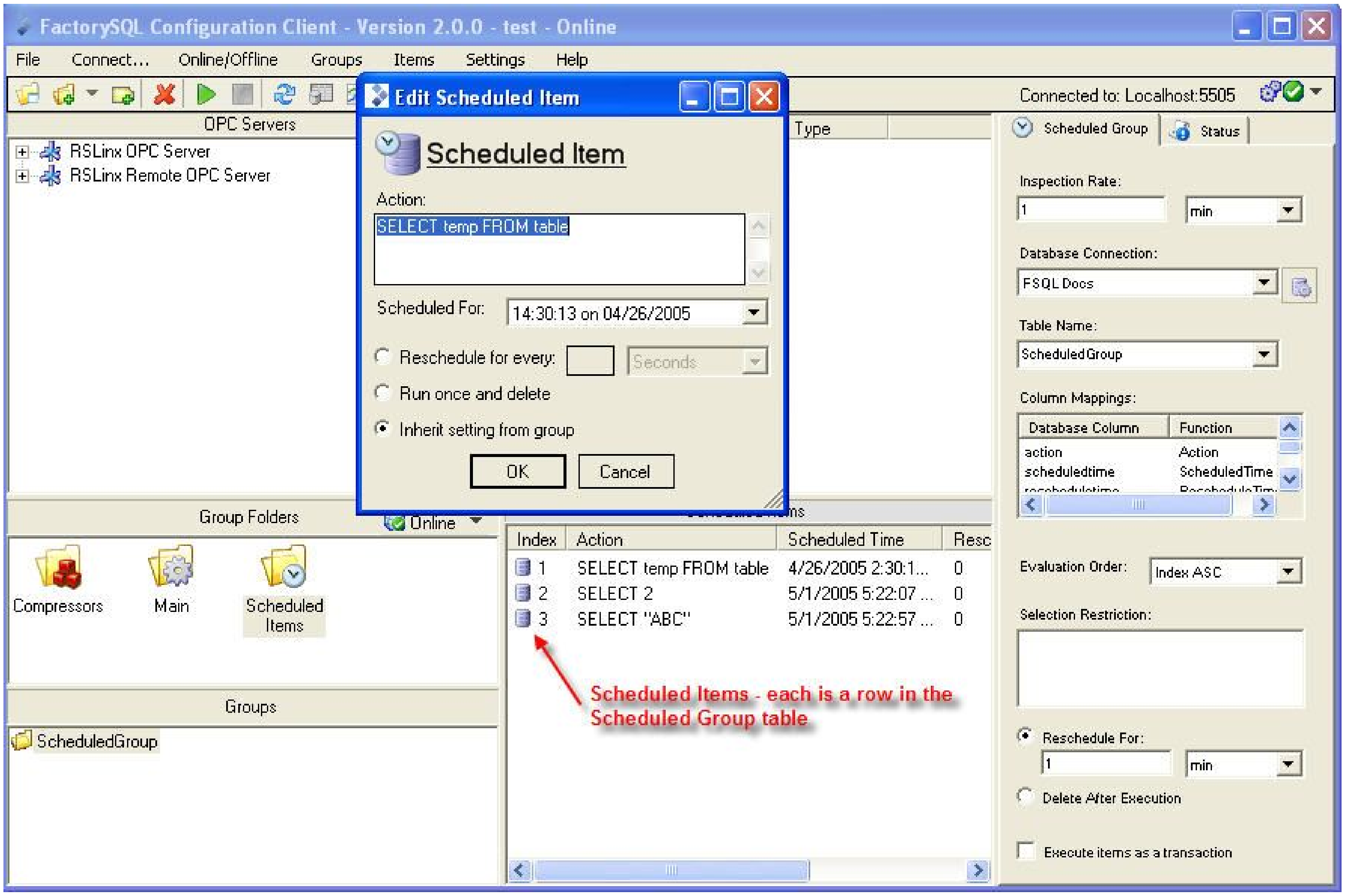
[ ] Execute items as a transaction

Scheduled Groups

NOTE: As of FactorySQL 4.0 Scheduled Groups are DEPRECATED. It is advised that they should only be used if absolutely necessary. Usually, the functionality of scheduled groups can be accomplished using action items.

Scheduled Groups read queries from the database then run them. The queries are defined inside of scheduled items.

For more information on configuring scheduled groups, see Group Options.



Action: SELECT temp FROM table

Scheduled For: 14:30:13 on 04/26/2005

( ) Reschedule for every: Seconds

( ) Run once and delete

(•) Inherit setting from group

Index	Action	Scheduled Time	Resc
1	SELECT temp FROM table	4/26/2005 2:30:1...	0
2	SELECT 2	5/1/2005 5:22:07 ...	0
3	SELECT "ABC"	5/1/2005 5:22:57 ...	0

Scheduled Items - each is a row in the Scheduled Group table

Inspection Rate: 1 min

Database Connection: FSQL Docs

Table Name: ScheduledGroup

Column Mappings:

Database Column	Function
action	Action
scheduledtime	ScheduledTime
rescheduletime	RescheduleTim

Evaluation Order: Index ASC

Selection Restriction:

(•) Reschedule For: 1 min

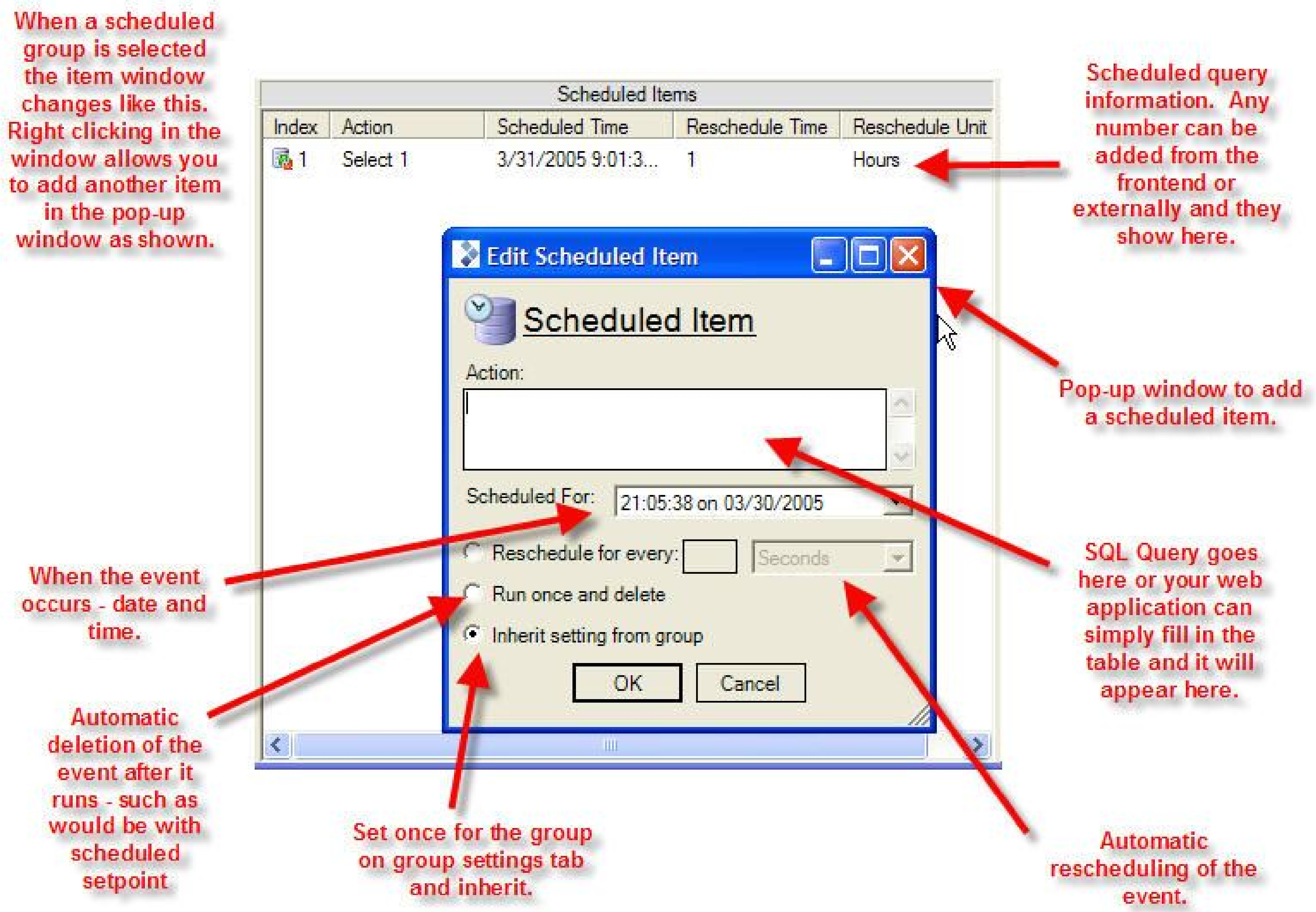
( ) Delete After Execution

[ ] Execute items as a transaction

Scheduled Items

Scheduled Items are similar to SQL "stored procedures". They run a fixed query at a scheduled date and time and have the option of being automatically periodically rescheduled.

When a schedule group is selected the group item window changes as below. From this window you can add actions to execute at scheduled time.

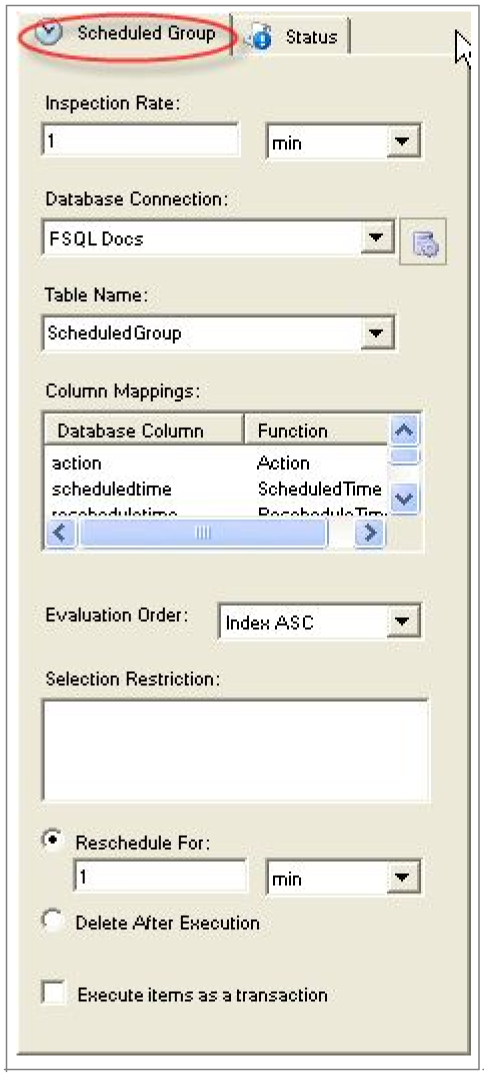


When a scheduled group is selected the item window changes like this. Right clicking in the window allows you to add another item in the pop-up window as shown.
Scheduled Items
Index
Action
Scheduled Time
Reschedule Time
Reschedule Unit
1
Select 1
3/31/2005 9:01:3...
1
Hours
Scheduled query information. Any number can be added from the frontend or externally and they show here.
Edit Scheduled Item
Scheduled Item
Action:
Pop-up window to add a scheduled item.
Scheduled For:
21:05:38 on 03/30/2005
Reschedule for every:
Seconds
Run once and delete
Inherit setting from group
OK
Cancel
When the event occurs - date and time.
SQL Query goes here or your web application can simply fill in the table and it will appear here.
Automatic deletion of the event after it runs - such as would be with scheduled setpoint
Set once for the group on group settings tab and inherit.
Automatic rescheduling of the event.

It is important to note that the scheduled items are simply rows in the table specified in the Group Settings Tab. Thus, you can use any external source to add new events on the fly, and have FactorySQL execute them at the correct time.

Scheduled Group Options

Scheduled groups read and execute SQL queries This provides a way for users to maintain scheduled SQL queries external to FactorySQL. For example, you could run a scheduled query to retrieve data from another database with the days production schedule. Or you could run a query which initiates a database backup.



Inspection Rate: 1 min

Database Connection: FSQL Docs

Table Name: ScheduledGroup

Column Mappings:

Database Column	Function
action	Action
scheduledtime	ScheduledTime

Evaluation Order: Index ASC

Selection Restriction:

(●) Reschedule For: 1 min

( ) Delete After Execution

[ ] Execute items as a transaction

Inspection Rate: How often the group will check the database to see if new items are ready to run (have passed their scheduled time).

Database Connection, Table Name: Specifies where the group will look for scheduled items.

Column Mapping: Allows you to map the necessary fields to different names, in case you are trying to use an exisiting table, or otherwise need different column names. Default values should be fine in most cases.

Evaluation Order: Specifies how items will be executed if more than 1 have expired. In the picture, items will be executed according to their database index, ascending.

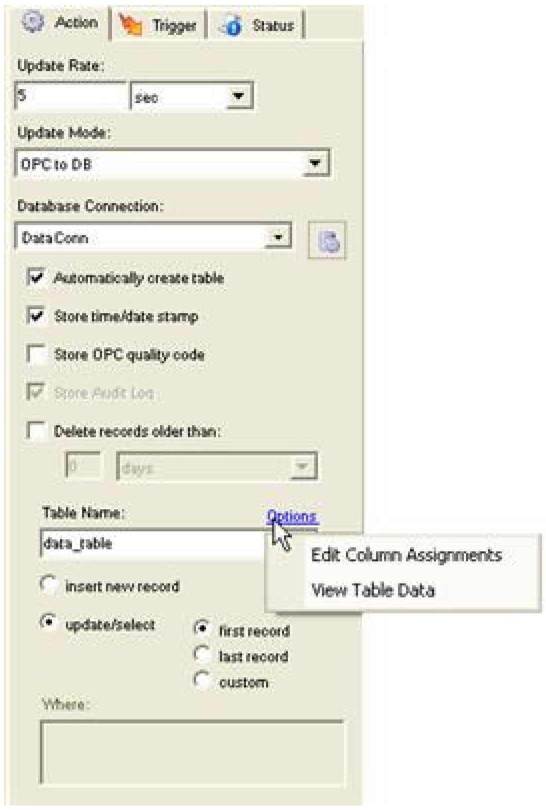
Selection Restriction: An SQL "Where Clause" for selecting items.

Reschedule For/Delete: After running, items will either be rescheduled or delete. This setting is mostly a default, as individual items can specify their own settings.

Execute Items as a Transaction: If the database supports it, all active items will be executed as a *transaction*, meaning that if one item fails to execute, all items will be undone.

Table Options

The **Table Options** link provides access to 2 useful functions: viewing table data, and editing column assignments.



Update Rate: 5 sec
Update Mode: OPC to DB
Database Connection: DataConn
[✓] Automatically create table
[✓] Store time/date stamp
[ ] Store OPC quality code
[✓] Store Audit Log
[ ] Delete records older than: 0 days
Table Name: data_table
( ) insert new record
(●) update/select — (●) first record ( ) last record ( ) custom
Where:

Edit Column Assignments: Allows you to change the default column for several FactorySQL functions. There are currently 4 columns that FactorySQL looks for by default: The table index, the timestamp, the quality code, and the audit log column. The necessity of these columns depend on the group settings (for instance, you cannot set the audit log user column if auditing is not selected in the group). See Table Column Assignments for more information.

View Table Data: This will bring up the FactorySQL query browser, allowing you to view the data in the currently selected table. From there, you may run other queries or adjust the data. See Table Data for more information.

Table Column Assignment

The **Table Column Assignment** screen allows you to change the columns that FactorySQL uses for certain functionality.

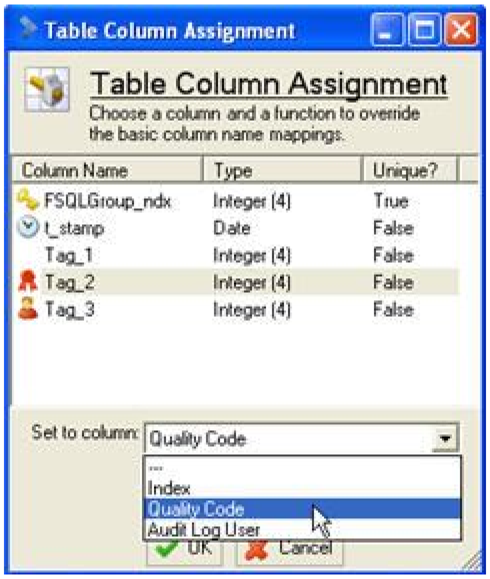


Table Column Assignment

Choose a column and a function to override the basic column name mappings.

Column Name	Type	Unique?
FSQLGroup_ndx	Integer (4)	True
t_stamp	Date	False
Tag_1	Integer (4)	False
Tag_2	Integer (4)	False
Tag_3	Integer (4)	False

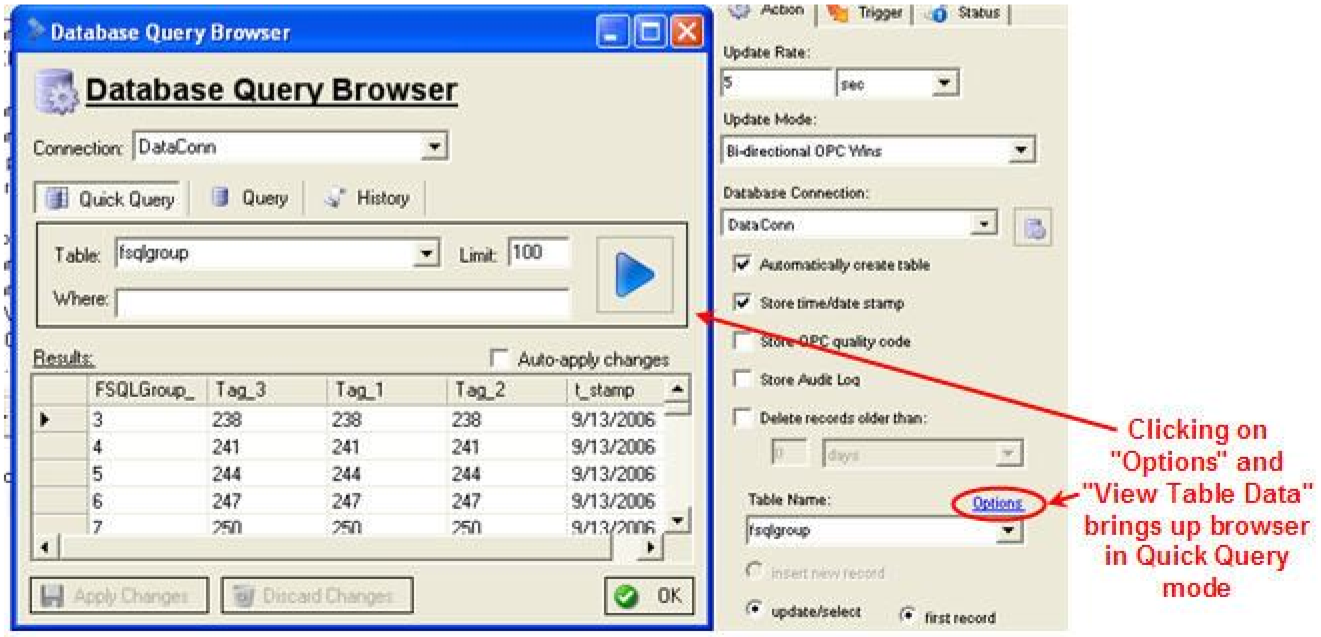
Set to column: Quality Code

This tool is particularly useful when trying to log to an existing table that cannot be altered. In this case, you can re-assign necessary columns, such as the index, to the existing columns of the table.
The only column that is required for logging is the Index column. The other 3 columns (timestamp, quality code, audit log user) depend on the settings of the group.

Query Browser/View Table Data

The **FactorySQL Query Brower** allows you to execute database queries, and view/edit table data.

You may get to it by clicking on Table Options, and then "View Table Data":



Database Query Browser

Connection: DataConn

Table: fsqlgroup Limit: 100

Where:

Results: ☐ Auto-apply changes

	FSQLGroup_	Tag_3	Tag_1	Tag_2	t_stamp
▶	3	238	238	238	9/13/2006
	4	241	241	241	9/13/2006
	5	244	244	244	9/13/2006
	6	247	247	247	9/13/2006
	7	250	250	250	9/13/2006

Update Rate: 5 sec

Update Mode: Bi-directional OPC Wins

Database Connection: DataConn

☑ Automatically create table
☑ Store time/date stamp
☐ Store OPC quality code
☐ Store Audit Log
☐ Delete records older than: 0 days

Table Name: fsqlgroup

○ insert new record
● update/select ● first record

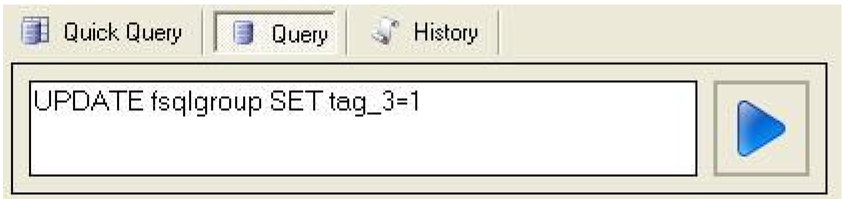
As the image shows, the browser will be opened in **Quick Query** mode, with the base parameters filled in according to the table settings.

Quick Query

The Quick Query tab allows you to exeute a simple query quickly. You can select the table, the number of rows returned, and build a simple where clause.
Note that data editing is only enabled when using the quick query tab.

Editing Data
If the table selected in the quick query tab is a FactorySQL compatible table (that is, has a valid index row specified) it is editable in the results table. To edit the data, double click on a cell and change the value as desired. After changing cells, or hitting enter, the **Apply Changes** and **Discard Changes** buttons will become active. At this point, the changes have not yet been applied to the database. Click the Apply Changes button to commit the values.
Alternatively, you may select the **Auto-apply** check box in order to automatically write changes to the database as they occur.

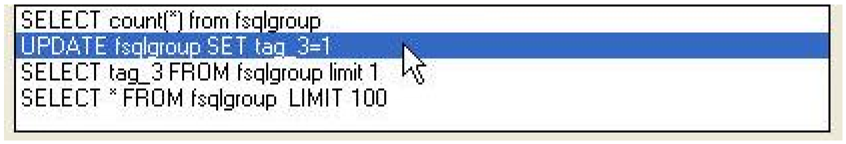
Query



```
UPDATE fsqlgroup SET tag_3=1
```

The Query tab allows you to enter a free form query against the selected connection. This query does not need to be a SELECT statement, it can be any statement that is valid for the target database.

History



```
SELECT count(*) from fsqlgroup
UPDATE fsqlgroup SET tag_3=1
SELECT tag_3 FROM fsqlgroup limit 1
SELECT * FROM fsqlgroup  LIMIT 100
```

This tab shows a list of the most recently run queries, including those generated by the Quick Query tab. You may double-click on an entry to load it into the query tab so it may be run again.

Alerting Overview

Alerting is the process of recognizing unusual events, or conditions to which you'd like to respond in a special way. FactorySQL supports basic alerting, and provides a number of useful mechanisms for accessing alert data.

The alerting system provides a history table, a status table, and a basic email notification system. There are 2 different types of history/status tables:

Standard - The default scheme. 2.3

Alert History - Provides all of the information for an alert in one row, such as active/clear time, acknowledgement info, etc.

Alert Status - Provides information on the current state of each configured alarm, and the last time that the state changed.

Legacy - The scheme for FactorySQL 2.2 and earlier.

Alert History - Each alert event (active, clear) is it's own row. No inherent support for acknowledgement.

Alert Status - Provides the state of each configured alert, as well as its acknowledgement state.

You may choose the alerting scheme from the Alert Options page:

There are 2 topics which are applicable to the alert system in general:

- **Configuring Alert Items** - The first step you must take to use alerting is to define what an alert is. Both OPC and Action Items can be used for alerts. By editing their alert configurations, you can choose the conditions when they are "In Alert" or "Clear".
- **Alert Notification** - FactorySQL support simple alert notification through e-mail, including to mobile devices such as cell phones. This section describes the notification message, and how to set it up.

Alerting Overview

Alerting is the process of recognizing unusual events, or conditions to which you'd like to respond in a special way. FactorySQL supports basic alerting, and provides a number of useful mechanisms for accessing alert data.

The alerting system provides a history table, a status table, and a basic email notification system. There are 2 different types of history/status tables:

Standard - The default scheme. 2.3

Alert History - Provides all of the information for an alert in one row, such as active/clear time, acknowledgement info, etc.

Alert Status - Provides information on the current state of each configured alarm, and the last time that the state changed.

Legacy - The scheme for FactorySQL 2.2 and earlier.

Alert History - Each alert event (active, clear) is it's own row. No inherent support for acknowledgement.

Alert Status - Provides the state of each configured alert, as well as its acknowledgement state.

You may choose the alerting scheme from the Alert Options page:

There are 2 topics which are applicable to the alert system in general:

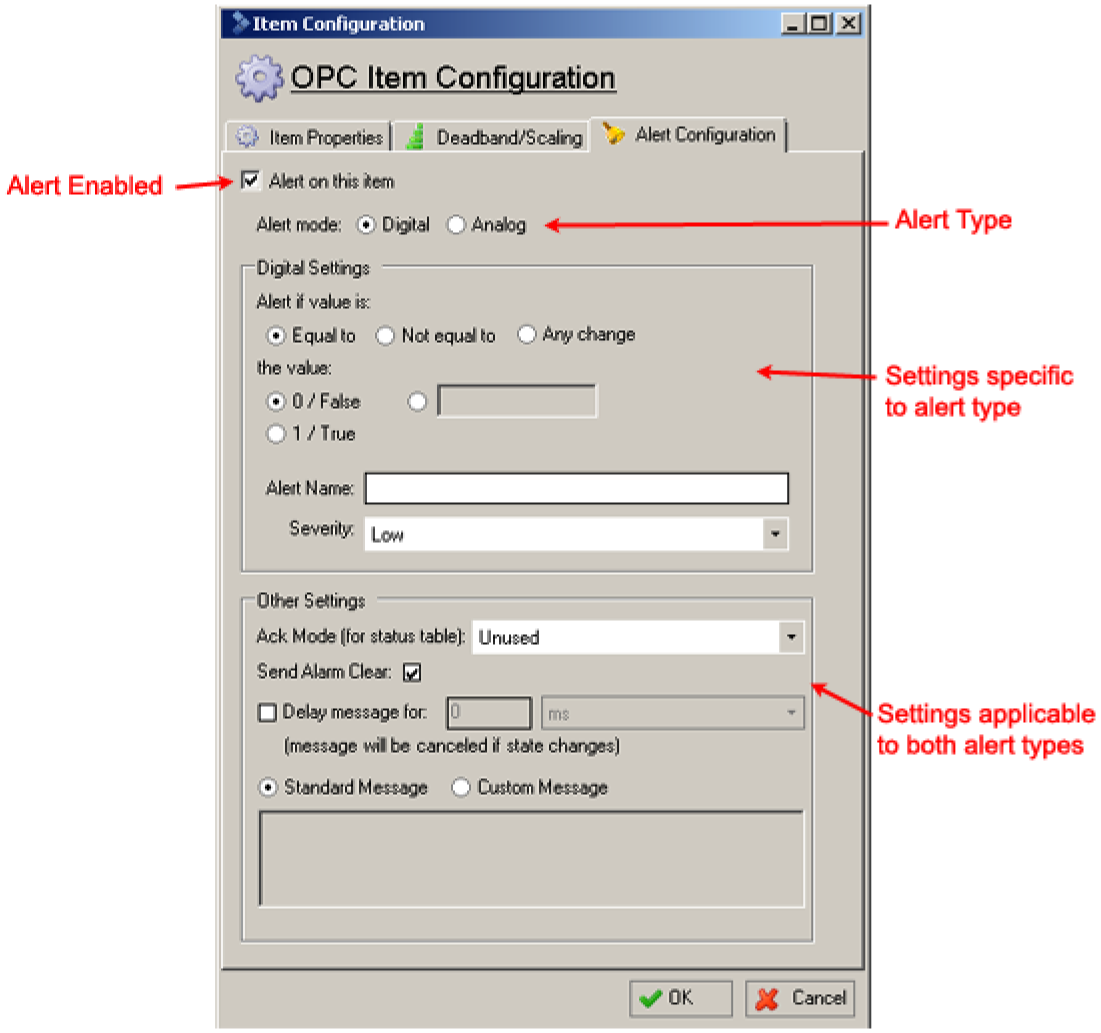
- **Configuring Alert Items** - The first step you must take to use alerting is to define what an alert is. Both OPC and Action Items can be used for alerts. By editing their alert configurations, you can choose the conditions when they are "In Alert" or "Clear".
- **Alert Notification** - FactorySQL support simple alert notification through e-mail, including to mobile devices such as cell phones. This section describes the notification message, and how to set it up.

Configuring an Alert Item

An **Alert Item** is not a different type of item, but instead any normal item (OPC or Action) that has been configured to cause an alert. The process for configuring them is the same for each one.

Alert configuration screen

To open the alert configuration screen, either double-click on the item you want in the Group Item Window, or right-click on the item and choose "Edit". Click on the "Alert Configuration" tab. If you were to click the box labeled "Alert on this item", the screen would look as follows:



[✓] Alert on this item

Alert mode: [✓ Digital] Analog

Digital Settings

Alert if value is: [✓ Equal to] Not equal to Any change

the value: [✓ 0 / False] 1 / True

Alert Name:

Severity: Low

Other Settings

Ack Mode (for status table): Unused

Send Alarm Clear: [✓]

[ ] Delay message for: 0 ms

(message will be canceled if state changes)

[✓ Standard Message] Custom Message

Digital Mode

Digital mode means that there are only 2 states: in alarm or not. Therefore, the alert is configured for a specific condition.

Step 1: Select condition logic. The item can be in alarm if the value is **Equal to** a specified value, **Not Equal to** that value, or on **Any change**.

Step 2: Select the condition value. (Not applicable for any-change). There are 3 choices:

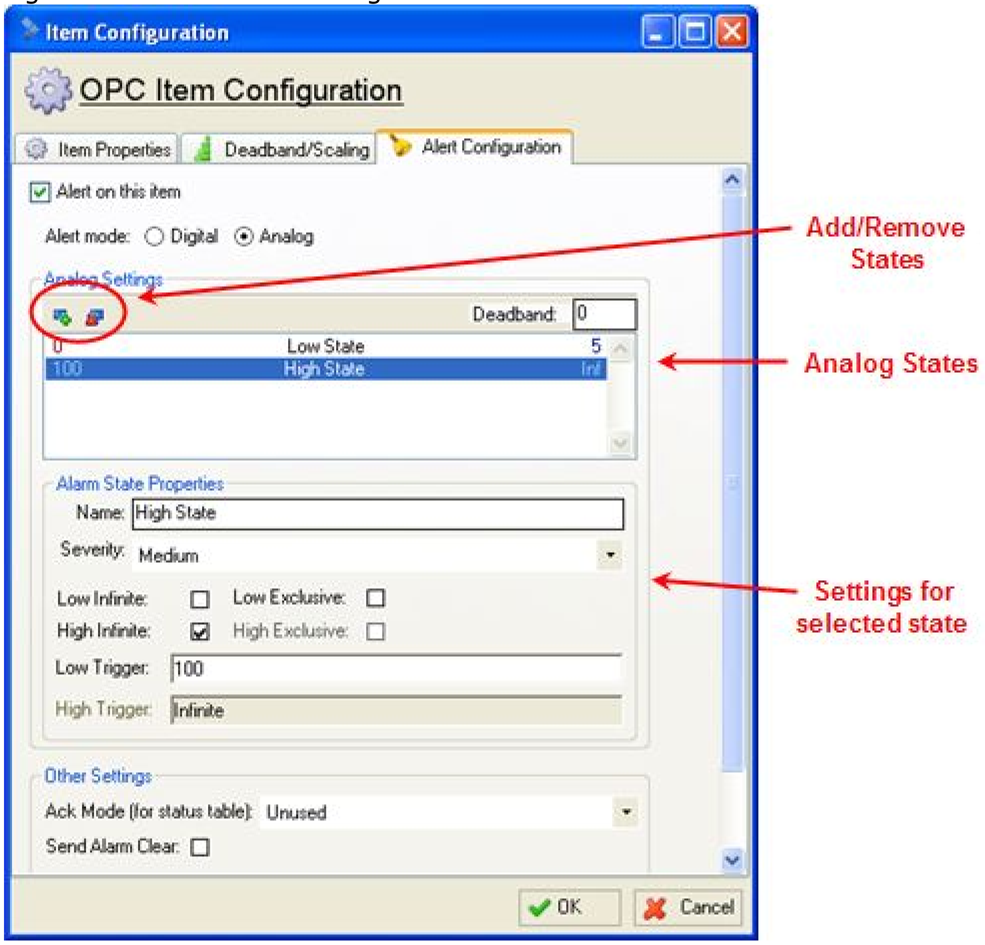
- 0/False: The value is equal to 0 if numeric, or "false" if boolean.
- 1/True: The value is equal to 1 if numeric, or "true" if boolean.
- Free form: Any numerical value, or **item substitution**.

Name: A name for this alarm, which can be used for display purposes from the status table, or sent with a notification over email.

Severity: A way for you indicate how severe this alarm is. You are able to adjust the alert settings so that only very important messages are emailed to you.

Analog Alerts

An analog alert allows many different states, or levels of alarm. Each state can have its own severity, allowing a good deal of flexibility. For instance, you can keep a record of all minor state changes, but only receive an email for significant alerts. The config screen looks as follows:



[✓] Alert on this item

Alert mode: Digital [✓ Analog]

Analog Settings

Deadband: 0

0	Low State	5
100	High State	Inf

Alarm State Properties

Name: High State

Severity: Medium

Low Infinite: ☐ Low Exclusive: ☐

High Infinite: ☑ High Exclusive: ☐

Low Trigger: 100

High Trigger: Infinite

Other Settings

Ack Mode (for status table): Unused

Send Alarm Clear: ☐

General concept: The alarm state window shows a list of states, each of which has a **low** and **high** setpoint. These setpoints can be inclusive or exclusive, and can be infinity. When the value of the item is between the low and high setpoint, the state is active, and the item is "in alarm". It is therefore possible to have several states active at one time.

Deadband: This value "pads" each of the analog states, so that you don't end up with overly active alarm states when the precise value of the object is instable. If your alarm state trigger point is X, the item will alarm as soon as the value is more than X. However, it will only clear again when the value is less than X-Deadband.

Name: Just like digital items, the name provides useful information in the alarm status and log tables, and is included in the notification email.

Low trigger/High trigger: The range in which the state will be active.

Severity: Used to distinguish between different levels of alerts.

Other Settings

These settings apply to both digital and analog alerts.

Ack Mode: The meaning of this setting differs a bit depending on which system behavior you're using. Using the standard system, it will determine how FactorySQL acknowledges alerts in the log table, but under the legacy system it will affect the "unacknowledged" field of the status table.

Unused - The alarm is always acknowledged. This means that for the standard system it is immediately marked, and for legacy system no action is taken.

Auto - The alert is automatically acknowledged when the alert clears.

Manual - The "unacknowledged" field is set, and never automatically cleared.

Send alarm clear: This will create another message when the alarm goes inactive. It will be logged to the history table, and will be emailed if the Alert Settings allow it.

Delay Message: This will delay the sending of an email alert for the specified amount of time. The message will be canceled if the alert is cleared during the delay. Therefore, it is a useful setting to avoid sending many emails for alerts that commonly bounce in and out.

Message Type: Determines what message will be sent, if emailing is enabled. A **Standard** message includes basic information, and looks like:
ALERT 01:12:07 - Main.AlarmPerfTest.Out1 High Value SP 56 deg F
whereas a **Custom** message is freeform.

Custom Messages

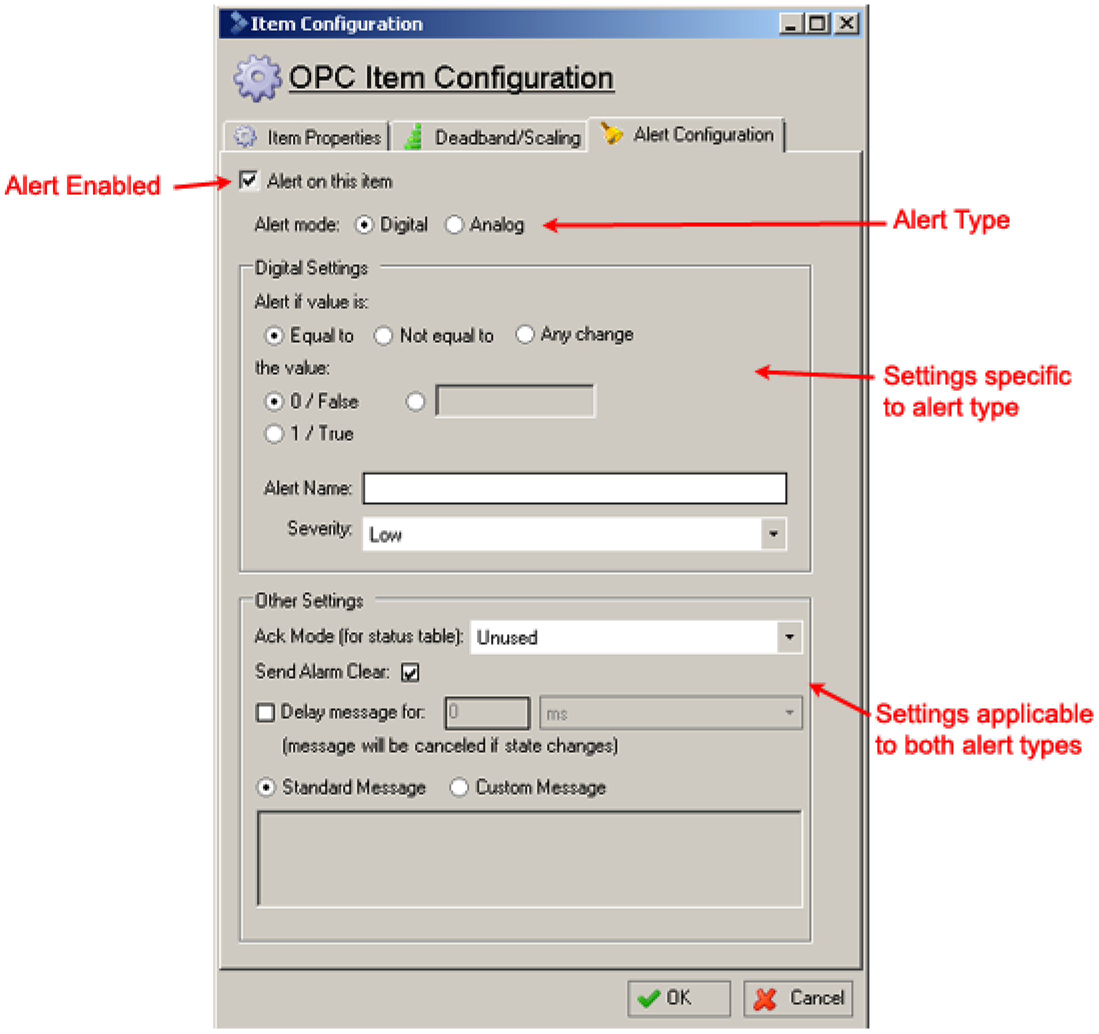
Custom messages allow you to create user friendly messages for emailed alerts. When creating a custom message, it is very useful to use CTRL-Space (or right-click) to reference various important properties of the alert.
Available Properties:
Date
Time
Value
Previous Value
Item Name
State Name
Item Units
Item Path - The path of the item **inside of the group**, such as the OPC address for an OPC item.
Severity - Will be the text interpretation, such as "Low", "Medium-Low", "Medium", etc.
Folder Path - The full path of the folder that the group is in.
Group Name - The name of the group that this alert is in.
Alert Type - This will either be "Alert" or "Clear", to indicate the state of the alert.
Additionally, you may refer to other items in the group in order to include their value.

Configuring an Alert Item

An **Alert Item** is not a different type of item, but instead any normal item (OPC or Action) that has been configured to cause an alert. The process for configuring them is the same for each one.

Alert configuration screen

To open the alert configuration screen, either double-click on the item you want in the Group Item Window, or right-click on the item and choose "Edit". Click on the "Alert Configuration" tab. If you were to click the box labeled "Alert on this item", the screen would look as follows:



[✓] Alert on this item

Alert mode: [✓ Digital] Analog

Digital Settings

Alert if value is: [✓ Equal to] Not equal to Any change

the value: [✓ 0 / False] 1 / True

Alert Name:

Severity: Low

Other Settings

Ack Mode (for status table): Unused

Send Alarm Clear: [✓]

Delay message for: 0 ms

(message will be canceled if state changes)

[✓ Standard Message] Custom Message

Digital Mode

Digital mode means that there are only 2 states: in alarm or not. Therefore, the alert is configured for a specific condition.

Step 1: Select condition logic. The item can be in alarm if the value is **Equal to** a specified value, **Not Equal to** that value, or on **Any change**.

Step 2: Select the condition value. (Not applicable for any-change). There are 3 choices:

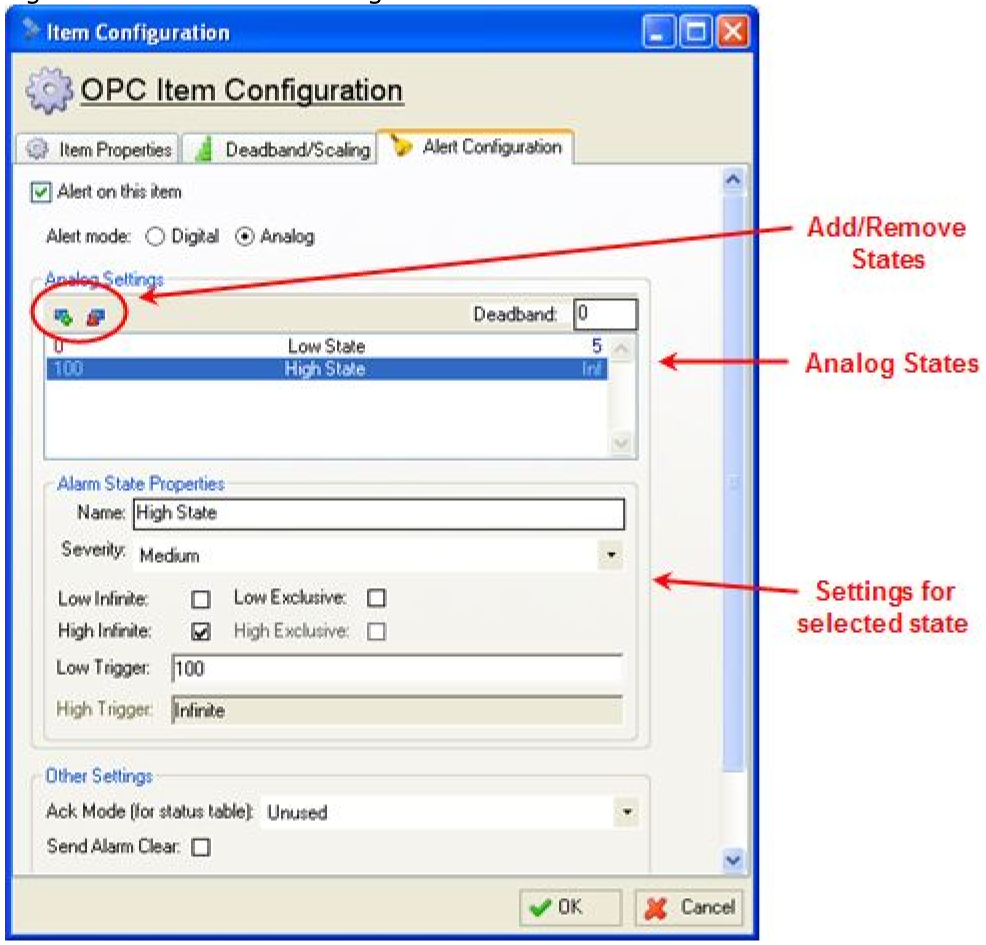
- 0/False: The value is equal to 0 if numeric, or "false" if boolean.
- 1/True: The value is equal to 1 if numeric, or "true" if boolean.
- Free form: Any numerical value, or **item substitution**.

Name: A name for this alarm, which can be used for display purposes from the status table, or sent with a notification over email.

Severity: A way for you indicate how severe this alarm is. You are able to adjust the alert settings so that only very important messages are emailed to you.

Analog Alerts

An analog alert allows many different states, or levels of alarm. Each state can have its own severity, allowing a good deal of flexibility. For instance, you can keep a record of all minor state changes, but only receive an email for significant alerts. The config screen looks as follows:



[✓] Alert on this item

Alert mode: Digital [✓ Analog]

Analog Settings

Deadband: 0

0	Low State	5
100	High State	Inf

Alarm State Properties

Name: High State

Severity: Medium

Low Infinite: ☐ Low Exclusive: ☐

High Infinite: ☑ High Exclusive: ☐

Low Trigger: 100

High Trigger: Infinite

Other Settings

Ack Mode (for status table): Unused

Send Alarm Clear: ☐

General concept: The alarm state window shows a list of states, each of which has a **low** and **high** setpoint. These setpoints can be inclusive or exclusive, and can be infinity. When the value of the item is between the low and high setpoint, the state is active, and the item is "in alarm". It is therefore possible to have several states active at one time.

Deadband: This value "pads" each of the analog states, so that you don't end up with overly active alarm states when the precise value of the object is instable. If your alarm state trigger point is X, the item will alarm as soon as the value is more than X. However, it will only clear again when the value is less than X-Deadband.

Name: Just like digital items, the name provides useful information in the alarm status and log tables, and is included in the notification email.

Low trigger/High trigger: The range in which the state will be active.

Severity: Used to distinguish between different levels of alerts.

Other Settings

These settings apply to both digital and analog alerts.

Ack Mode: The meaning of this setting differs a bit depending on which system behavior you're using. Using the standard system, it will determine how FactorySQL acknowledges alerts in the log table, but under the legacy system it will affect the "unacknowledged" field of the status table.

Unused - The alarm is always acknowledged. This means that for the standard system it is immediately marked, and for legacy system no action is taken.

Auto - The alert is automatically acknowledged when the alert clears.

Manual - The "unacknowledged" field is set, and never automatically cleared.

Send alarm clear: This will create another message when the alarm goes inactive. It will be logged to the history table, and will be emailed if the Alert Settings allow it.

Delay Message: This will delay the sending of an email alert for the specified amount of time. The message will be canceled if the alert is cleared during the delay. Therefore, it is a useful setting to avoid sending many emails for alerts that commonly bounce in and out.

Message Type: Determines what message will be sent, if emailing is enabled. A **Standard** message includes basic information, and looks like:
ALERT 01:12:07 - Main.AlarmPerfTest.Out1 High Value SP 56 deg F
whereas a **Custom** message is freeform.

Custom Messages

Custom messages allow you to create user friendly messages for emailed alerts. When creating a custom message, it is very useful to use CTRL-Space (or right-click) to reference various important properties of the alert.
Available Properties:
Date
Time
Value
Previous Value
Item Name
State Name
Item Units
Item Path - The path of the item **inside of the group**, such as the OPC address for an OPC item.
Severity - Will be the text interpretation, such as "Low", "Medium-Low", "Medium", etc.
Folder Path - The full path of the folder that the group is in.
Group Name - The name of the group that this alert is in.
Alert Type - This will either be "Alert" or "Clear", to indicate the state of the alert.
Additionally, you may refer to other items in the group in order to include their value.

Alert Notification

Alert Notification sends an email when alerts occur under specified conditions. The email may either be a standard message, consisting of the time, OPC path, value, and units, or a custom message that is user-defined.

The recipients may either be a statically defined list, a list pulled from the database, or in various rule based groups using **Distribution Groups**.

Most wireless providers support email to text messages in alphanumeric pagers and cell phones.

How to configure

You can set up email alerts in the **Alert Settings** screen.
Pay special attention to the "Severity" and "Send Alarm Clear" fields, as these settings are the most common reason users don't recieve the alerts they expect.

Sample standard email alert

From: FactorySQL Alerts
Date: May 16, 2005 6:12 AM
Subject: Out1 Alert
To:
ALERT 01:12:07 - Main.AlarmPerfTest.Out1 High Value SP 56 deg F

Distribution Lists

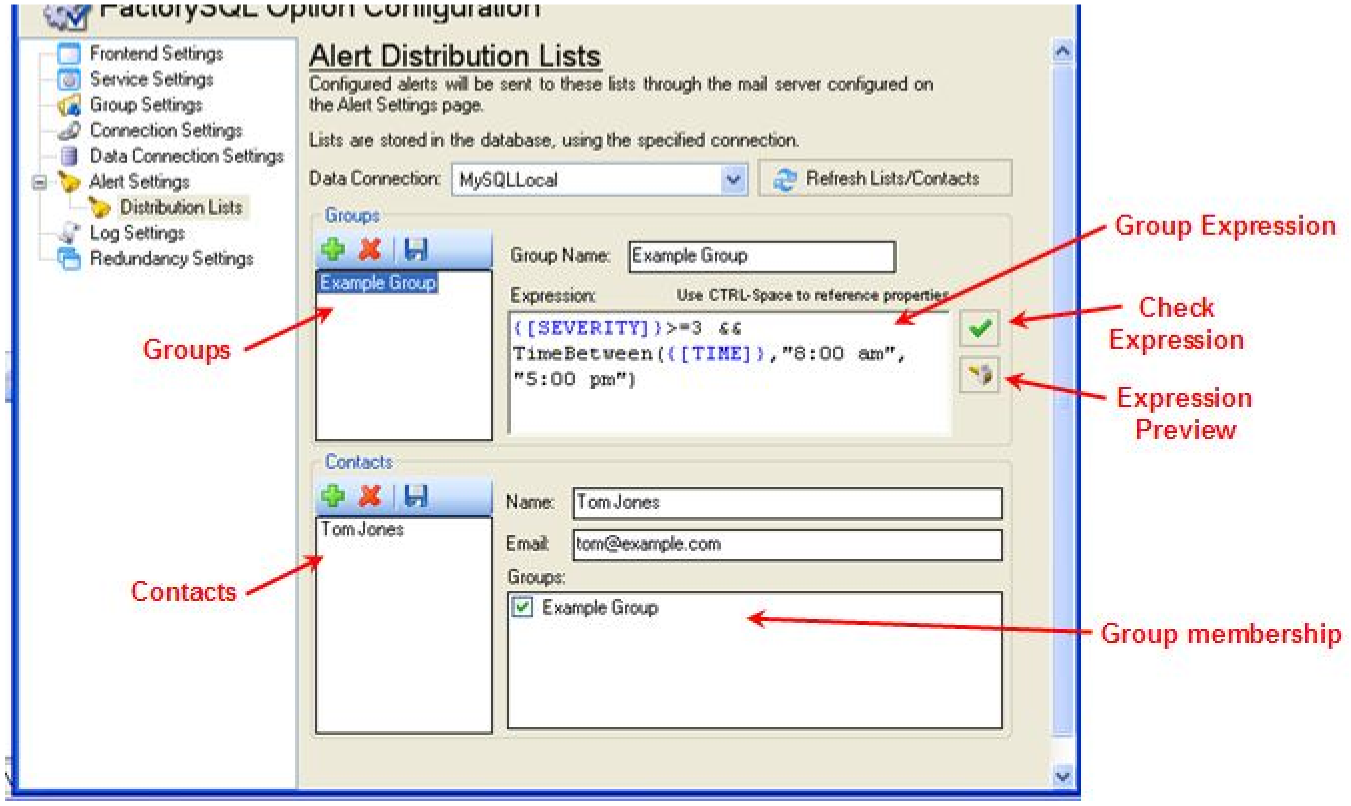
Distribution lists allow you to define **Groups** and **Contacts**, and rules that define when they receive various alerts.

Groups: Above all, a group has an expression that defines which alerts it will receive. Whenever an alert occurs that passes the group's expression, an email is sent to all contacts who belong to that group.
The **Expression** is the same as an action item expression, and can use all of the functions and operators available to them. See Expression Language Syntax for more information.
Unlike action items, the distribution group expression can refer to the properties of an alert, such as its Value, Time, Severity, etc.

Properties available to expressions:
(Note: these are the same properties available to **Custom Alert Messages**, but in some cases the values are different (such as severity, which is text for messages, but numerical here).
Date
Time
Value
Previous Value
Item Name
State Name
Item Units
Item Path - The path of the item inside of the group, such as the OPC address for an OPC item.
Severity - The severity of the alert, in numerical format (from 1 [LOW] to 5 [HIGH])
Folder Path - The full path of the folder that the group is in.
Group Name - The name of the group that this alert is in.
Alert Type - This will be numeric, 1 for ALERT, 2 for CLEAR

Contacts: A contact consists of a name and email address, and a list of groups that they belong to.
Note: only one email will be sent to each contact per alert, therefore it is OK to have contacts that are members of groups whose expressions may overlap.

Distribution lists are configured under Alert Settings, in the Program Settings dialog.
The screen looks as follows:



FactorySQL Option Configuration

Alert Distribution Lists

Configured alerts will be sent to these lists through the mail server configured on the Alert Settings page.

Lists are stored in the database, using the specified connection.

Data Connection: MySQLLocal

Group Name: Example Group

Expression:

```
{[SEVERITY]}>=3 &&
TimeBetween({[TIME]},"8:00 am",
"5:00 pm")
```

Name: Tom Jones

Email: tom@example.com

Groups: ✓ Example Group

Alert Notification

Alert Notification sends an email when alerts occur under specified conditions. The email may either be a standard message, consisting of the time, OPC path, value, and units, or a custom message that is user-defined.

The recipients may either be a statically defined list, a list pulled from the database, or in various rule based groups using **Distribution Groups**.

Most wireless providers support email to text messages in alphanumeric pagers and cell phones.

How to configure

You can set up email alerts in the **Alert Settings** screen.
Pay special attention to the "Severity" and "Send Alarm Clear" fields, as these settings are the most common reason users don't recieve the alerts they expect.

Sample standard email alert

From: FactorySQL Alerts
Date: May 16, 2005 6:12 AM
Subject: Out1 Alert
To:
ALERT 01:12:07 - Main.AlarmPerfTest.Out1 High Value SP 56 deg F

Distribution Lists

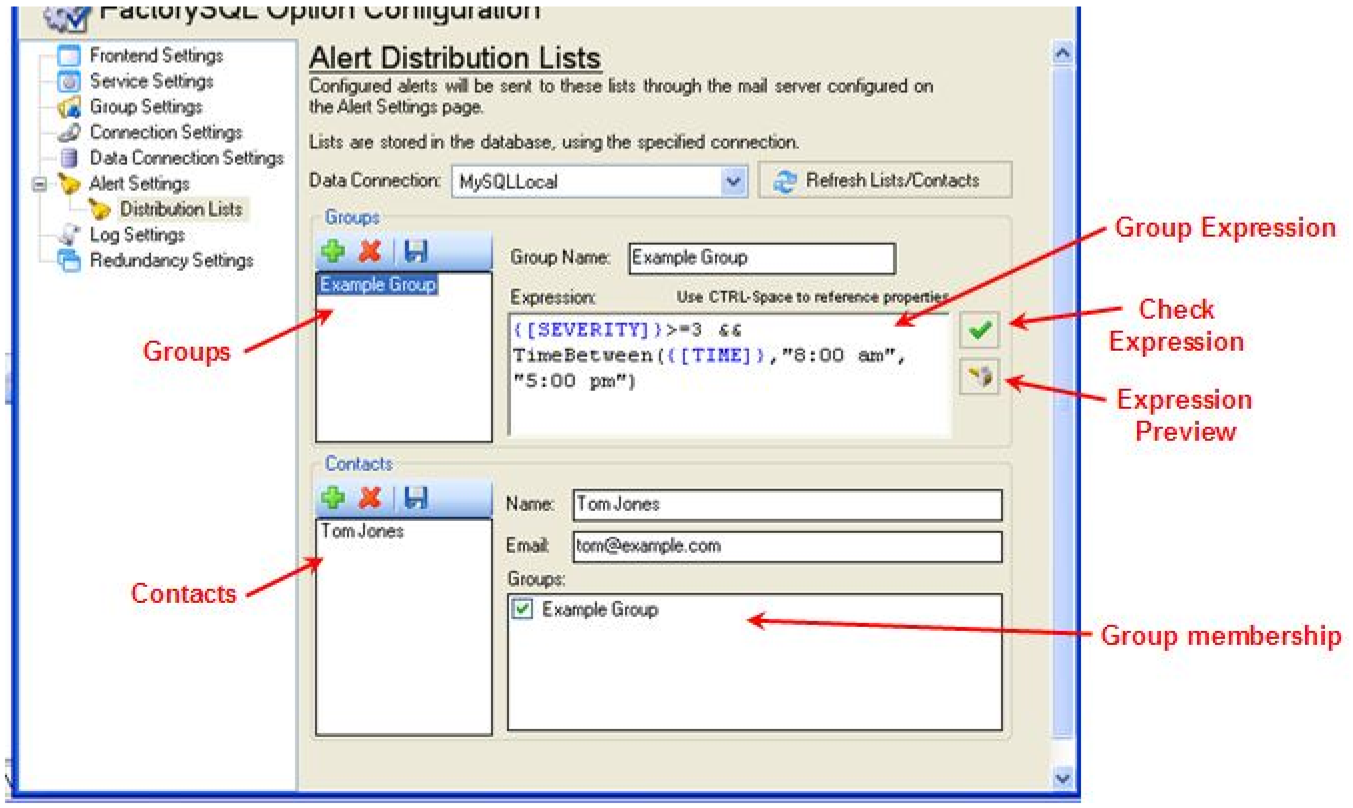
Distribution lists allow you to define **Groups** and **Contacts**, and rules that define when they receive various alerts.

Groups: Above all, a group has an expression that defines which alerts it will receive. Whenever an alert occurs that passes the group's expression, an email is sent to all contacts who belong to that group.
The **Expression** is the same as an action item expression, and can use all of the functions and operators available to them. See Expression Language Syntax for more information.
Unlike action items, the distribution group expression can refer to the properties of an alert, such as its Value, Time, Severity, etc.

Properties available to expressions:
(Note: these are the same properties available to **Custom Alert Messages**, but in some cases the values are different (such as severity, which is text for messages, but numerical here).
Date
Time
Value
Previous Value
Item Name
State Name
Item Units
Item Path - The path of the item inside of the group, such as the OPC address for an OPC item.
Severity - The severity of the alert, in numerical format (from 1 [LOW] to 5 [HIGH])
Folder Path - The full path of the folder that the group is in.
Group Name - The name of the group that this alert is in.
Alert Type - This will be numeric, 1 for ALERT, 2 for CLEAR

Contacts: A contact consists of a name and email address, and a list of groups that they belong to.
Note: only one email will be sent to each contact per alert, therefore it is OK to have contacts that are members of groups whose expressions may overlap.

Distribution lists are configured under Alert Settings, in the Program Settings dialog.
The screen looks as follows:


FactorySQL Option Configuration

Alert Distribution Lists

Configured alerts will be sent to these lists through the mail server configured on the Alert Settings page.

Lists are stored in the database, using the specified connection.

Data Connection: MySQLLocal

Refresh Lists/Contacts

Groups

Example Group

Group Name: Example Group

Expression: Use CTRL-Space to reference properties

```
{[SEVERITY]}>=3 &&
TimeBetween({[TIME]},"8:00 am",
"5:00 pm")
```

Contacts

Tom Jones

Name: Tom Jones

Email: tom@example.com

Groups:

[✓] Example Group

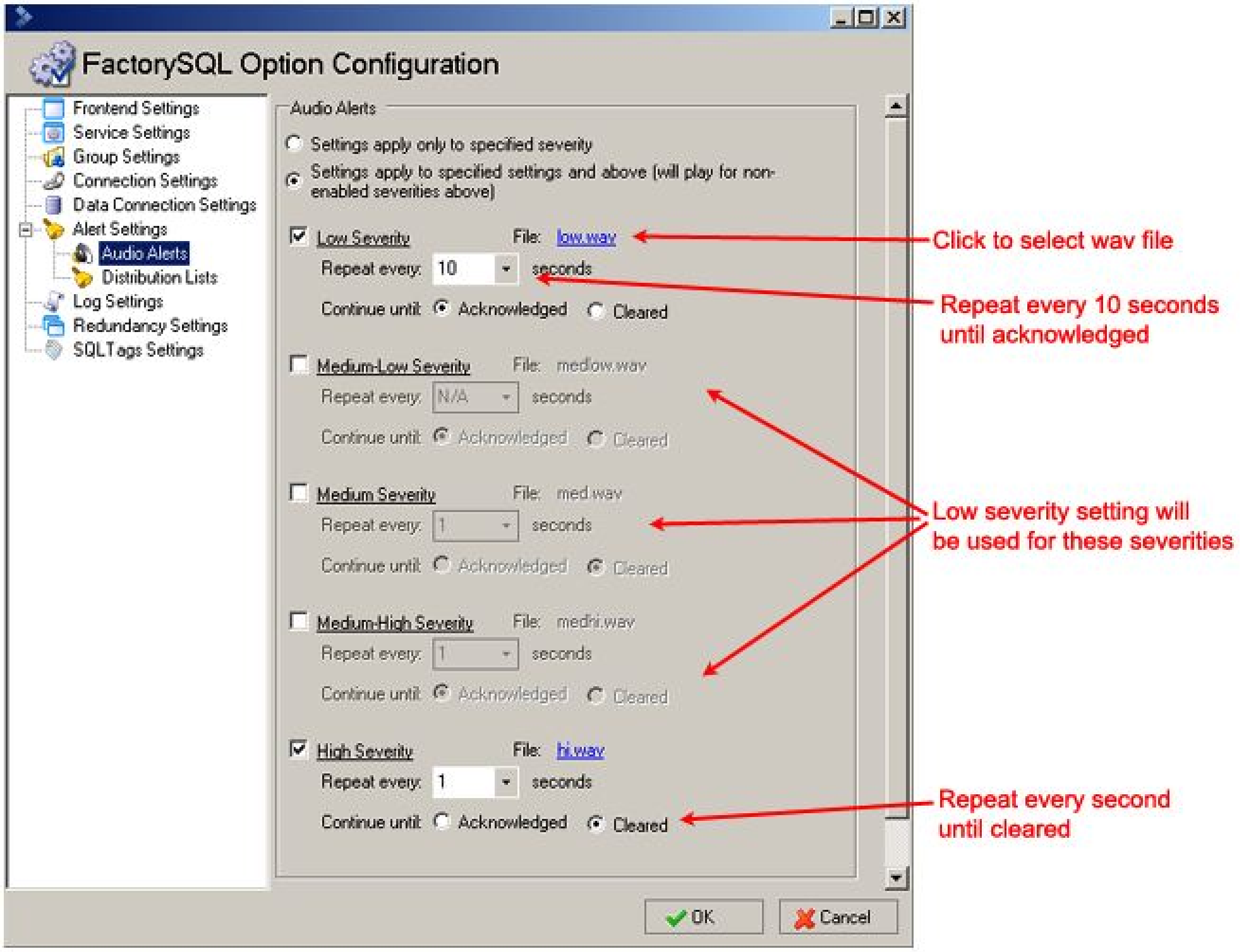
Audio Alerts

Audio Alerts allow you configure sounds to be played when there is an active alert. A different sound my be specified for each severity level, and can be configured to play once or to repeat at a specified interval until all alarms of the given severity are acknowledged or cleared.

How to configure

Audio alerts are configured by selecting the "Audio Alerts" sub-item in the **Alert Settings** screen.

The configuration screen looks as follows:



FactorySQL Option Configuration

Frontend Settings
Service Settings
Group Settings
Connection Settings
Data Connection Settings
Alert Settings
Audio Alerts
Distribution Lists
Log Settings
Redundancy Settings
SQLTags Settings

Audio Alerts

( ) Settings apply only to specified severity
(●) Settings apply to specified settings and above (will play for non-enabled severities above)

[✓] Low Severity — File: low.wav — Repeat every: 10 seconds — Continue until: (●) Acknowledged ( ) Cleared

[ ] Medium-Low Severity — File: medlow.wav — Repeat every: N/A seconds — Continue until: (●) Acknowledged ( ) Cleared

[ ] Medium Severity — File: med.wav — Repeat every: 1 seconds — Continue until: ( ) Acknowledged (●) Cleared

[ ] Medium-High Severity — File: medhi.wav — Repeat every: 1 seconds — Continue until: (●) Acknowledged ( ) Cleared

[✓] High Severity — File: hi.wav — Repeat every: 1 seconds — Continue until: ( ) Acknowledged (●) Cleared

OK
Cancel
Click to select wav file
Repeat every 10 seconds until acknowledged
Low severity setting will be used for these severities
Repeat every second until cleared

General Settings

The only general setting specifies how the severity settings are applied. There are two choices: apply settings only to the specified alert severity, or apply them to all greater severities that do not have their own setting. For example, by choosing to apply the setting to all severities (the second option), you only need to specify a sound for the lowest severity in order to hear it for any alert that occurs.

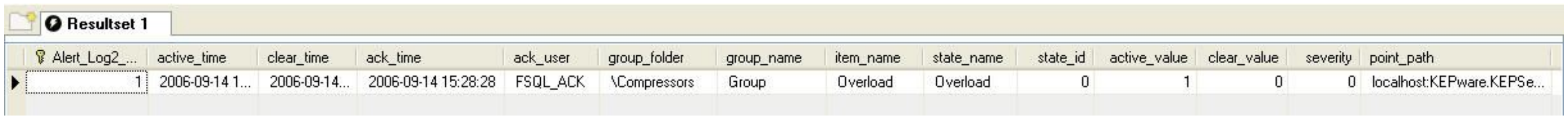
Severity Settings

There are five sets of identical settings, one for each severity.
Enabled: Enables audio alerts for the specified severity.
File: The audio file to play.
Repeat every X seconds: The time period at which to loop the audio. Audio will continue to loop until all alerts of the specified severity (or higher, depending on the general setting) are acknowledged or cleared (depending on the following setting). If set to 0, the sound will only be played once.

Continue until...: Specifies when to stop looping the audio, if applicable.

Standard Alert History Table

The **Standard Alert History** table consists of records that represent the life cycle of an item's alerts. That is, each row has information about an particular alert's active, clear, and acknowledged events.



Resultset 1

Alert_Log2_...	active_time	clear_time	ack_time	ack_user	group_folder	group_name	item_name	state_name	state_id	active_value	clear_value	severity	point_path
1	2006-09-14 1...	2006-09-14...	2006-09-14 15:28:28	FSQL_ACK	\Compressors	Group	Overload	Overload	0	1	0	0	localhost:KEPware.KEPSe...

Specify the alert history table **here** under **log table**.

Columns

Active Time: The date and time that the alert became active.

Clear Time: The date and time that the alarm was cleared (became inactive).

Ack Time: The date and time that the alert was acknowledged. The value of this field depends on the Ack Mode setting.

Ack User: The user that acknowledged the alarm. If ACK Mode is set to unused or automatic, this value will be "FSQL_ACK".

Group Folder: Indicates the Group folder where the alarm item resides.

Group Name: Indicates the Group where the alarm item resides.

Item Name: Indicates the item that changed alarm state.

State Name: The user defined text string to describe the alarm state. For **digital alerts**, this describes the alarm. For **analog alerts**, it indicates which alarm state the item went into.
Previous Value: The last value that FactorySQL read of the item before it changed alarm state.

State ID: The id of the alert state. For digital, this will be 0 or 1. For analog, this can be positive or negative, corresponding to how far away the alert state is from the base state.

Active Value: The value of the item that caused the item to become active.

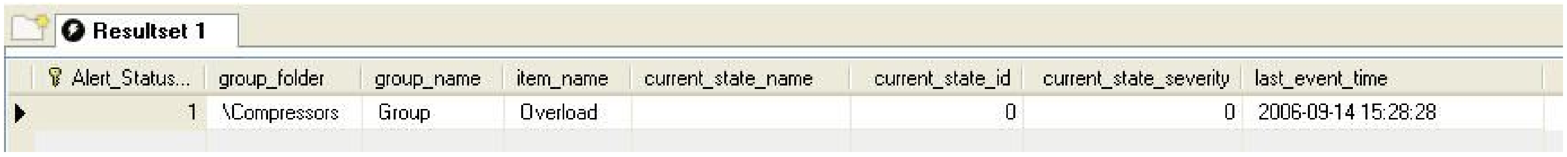
Clear Value: The value of the item that caused the alert to clear.

Severity: A numeric representation of the severity states from 1 (low) to 5 (high).

Point Path: The full OPC path to the item.

Standard Alert Status Table

The **Standard Alert Status** table displays alert status of items that have been configured for alerting. It includes which items are in alarm, severity, etc. It is useful for displaying the realtime status of alert items in a web or HMI system.



Alert_Status...	group_folder	group_name	item_name	current_state_name	current_state_id	current_state_severity	last_event_time
1	\Compressors	Group	Overload		0	0	2006-09-14 15:28:28

Specify the alert stus table **here** under **Alert Status Table**.

Columns

Group Folder: Indicates the Group folder where the alert item resides.

Group Name: Indicates the Group where the alert item resides.

Item Name: Indicates the item that changed alert state.

Current State Name: The user defined text string to describe the alert state. For **digital alerts**, this describes the alert. For **analog alert**, it indicates which alert state that the item went into.

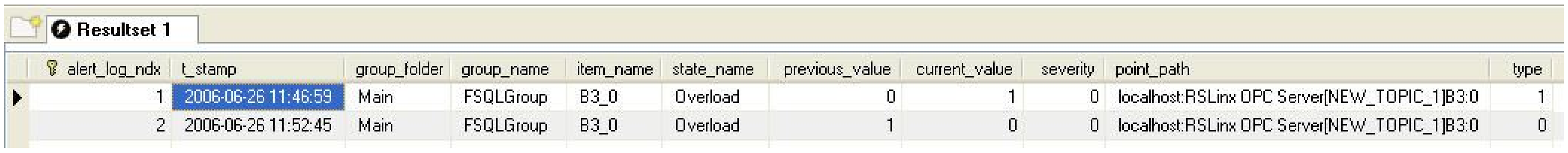
Current State Id: Provides the ID of the current alert state. Alert states are automatically assigned ID numbers, which can be used to identify them and distinguish between them. For digital, it will either be a 0 (indicating no alert), or 1 (indicating in-alert). For analog, 0 indicates that the value is within the base range, and the item is not in alert. Alert states are numbered sequentially, going up for rising-edge alarms, and down (into negative numbers) for falling edge alert states.

Current State Severity: A numeric representation of the severity states from 1 (low) to 5 (high).

Last Event Time: Provides the date and time that the alert last changed state.

Legacy Alert History Table

The **Alert History** table consists of records that represent an item going into an alarm state, changing alarm states, or going out of an alarm state, depending on configuration. This is useful for observing trouble spots of a system over time.



Resultset 1

alert_log_ndx	t_stamp	group_folder	group_name	item_name	state_name	previous_value	current_value	severity	point_path	type
1	2006-06-26 11:46:59	Main	FSQLGroup	B3_0	Overload	0	1	0	localhost:RSLinx OPC Server[NEW_TOPIC_1]B3:0	1
2	2006-06-26 11:52:45	Main	FSQLGroup	B3_0	Overload	1	0	0	localhost:RSLinx OPC Server[NEW_TOPIC_1]B3:0	0

Specify the alert history table **here** under **log table**.

Columns

t_stamp: Date and time of the alarm.

Group Folder: Indicates the Group folder where the alarm item resides.

Group Name: Indicates the Group where the alarm item resides.

Item Name: Indicates the item that changed alarm state.

State Name: The user defined text string to describe the alarm state. For **digital alerts**, this describes the alarm. For **analog alerts**, it indicates which alarm state the item went into.

Previous Value: The last value that FactorySQL read of the item before it changed alarm state.

Current Value: The value of the item immediately after it changed alarm state.

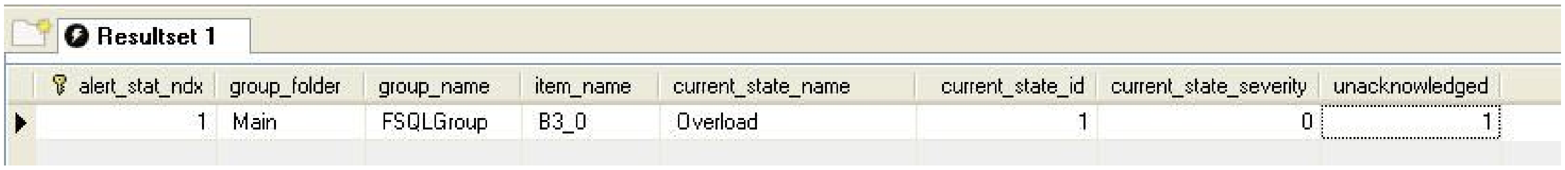
Severity: A numeric representation of the severity states from 1 (low) to 5 (high).

Point Path: The full OPC path to the item.

Type: Indicates going into alert (1) or clearing (0).

Legacy Alert Status Table

The **Alert Status** table displays alert status of items that have been configured for alerting. It includes which items are in alarm, severity, etc. It is useful for displaying the realtime status of alert items in a web or HMI system.



Resultset 1

alert_stat_ndx	group_folder	group_name	item_name	current_state_name	current_state_id	current_state_severity	unacknowledged
1	Main	FSQLGroup	B3_0	Overload	1	0	1

Specify the alert stus table **here** under **Alert Status Table**.

Columns

Group Folder: Indicates the Group folder where the alert item resides.

Group Name: Indicates the Group where the alert item resides.

Item Name: Indicates the item that changed alert state.

current_state_name: The user defined text string to describe the alert state. For **digital alerts**, this describes the alert. For **analog alert**, it indicates which alert state that the item went into.

current_state_id: Provides the ID of the current alert state. Alert states are automatically assigned ID numbers, which can be used to identify them and distinguish between them. For digital, it will either be a 0 (indicating no alert), or 1 (indicating in-alert). For analog, 0 indicates that the value is within the base range, and the item is not in alert. Alert states are numbered sequentially, going up for rising-edge alarms, and down (into negative numbers) for falling edge alert states.

current_state_severity: A numeric representation of the severity states from 1 (low) to 5 (high).

Unacknowledged: Indicates whether the current alert has been acknowleged. Provided for use by the display system, which is also responsible for setting it. 1 indicates that the alarm has NOT been acknowledged, whereas 0 indicates it has. Behavior is also contingent on the ACK Mode setting on the Alert Configuration Page.

Redundancy Overview

FactorySQL Redundancy provides a way to boost uptime in critical systems, by running multiple nodes that can execute a shared project if needed. Setting up redundancy is fairly straightforward, but there is a good amount of flexibility in designing and implementing different redundant schemes. A solid understanding of the principals and components of FactorySQL redundancy can help in making these design choice.

Definitions & Terminology

In discussing redundancy, it's useful to have the basic terminology well defined. The following definitions are listed in logical order instead of alphabetical.

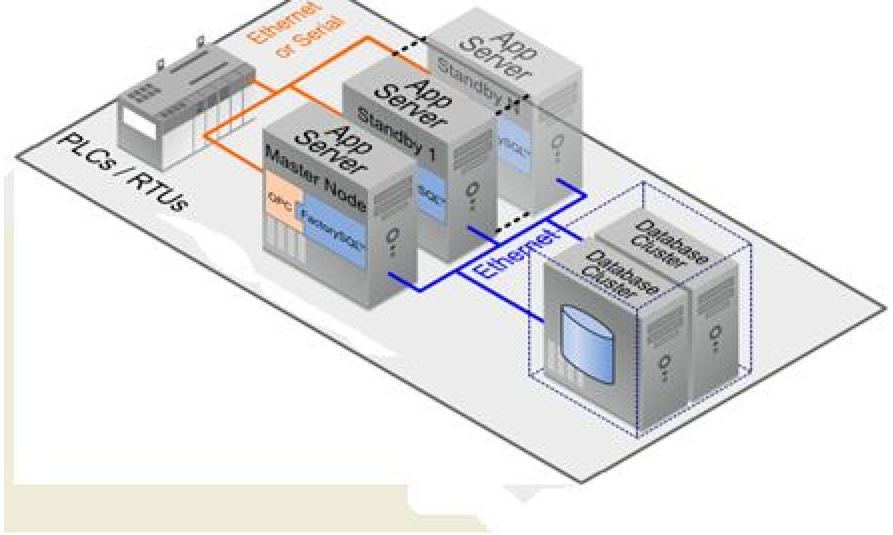
Node: An instance of FactorySQL, running in redundant mode.
Responsibility: What a node should be- master, backup, provisional master, etc.

Shared Project: The project that is read & executed by all redundant nodes. Changes made on any node will be recognized by all of the other nodes.
Redundancy/Shared Database: The database that is used by the nodes to communicated project information, node state, etc.

Master: The instance of FactorySQL that is executing the project.
Provisional Master: An instance of FactorySQL running in a reduced version of master. Caused by an inability to determine with certainty what the responsibilty should be.
Slave/Backup: An instance of FactorySQL that is not currently executing the project, but could start should the master fail.
Warm Backup: A backup node that has all of the OPC items connected and groups started- but is not currently executing them.
First Backup: A backup node that is the next up to run, should the master fail. Usually runs warm.
Node State : The operating state of the node- Running, errored, join_wait, etc.
Project State: The operating state of the project on a specific node: Ready, Warm, Active, etc.

Basic Principals

A basic redundant setup might look as follows:



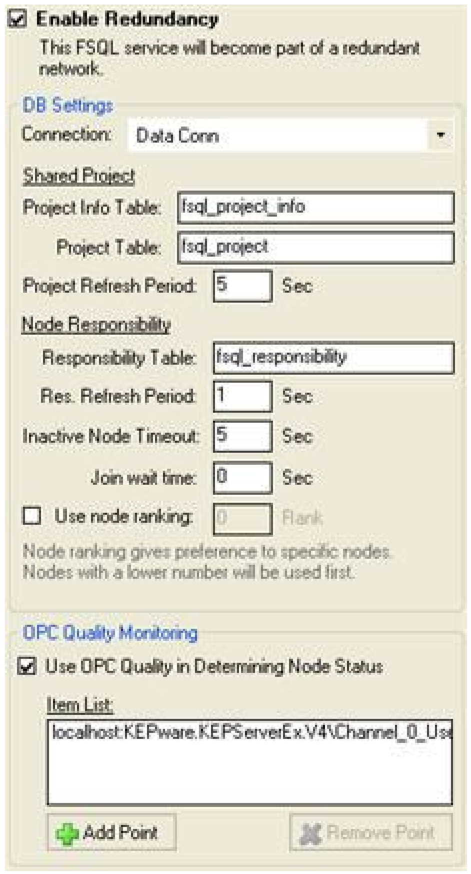
At the core there is a **shared database** that all of the nodes can see and communicate through. This can be a clustered/replicated/mirrored database for increased reliability. It does not have to be the same database that the project writes to, but that is often the case.
Each node will register itself in the **responsibility table**. This table contains key information about the nodes, and is queried in order to determine who should be master. If a particular node does not update its row within the proper amount of time, it is determined to be dead, and is removed. The next node in line will become master.

When a node is master, it will run the shared project as if it were a normal, local project. If the node is not master, but instead a backup, the project will be loaded but not executed. A **warm backup** will connect all OPC points, but

won't execute groups.
If the shared database is not available, there will be no way for the nodes to communicate. In this case, each node will run in **Provisional Master** mode, in which case it will run and cache historical data. When the shared database is available again, the node who becomes master will write the cache to the database, and all other nodes will discard their data.

Redundancy Settings/Configuration

Redundancy system configuration is fairly straightforward, as the default values can be used in most cases. Since all communication occurs through the database, it is important to choose the correct connection. It is also very important that the settings are the same on each node.



☑ Enable Redundancy
This FSQL service will become part of a redundant network.

DB Settings
Connection: Data Conn

Shared Project
Project Info Table: fsql_project_info
Project Table: fsql_project
Project Refresh Period: 5 Sec

Node Responsibility
Responsibility Table: fsql_responsibility
Res. Refresh Period: 1 Sec
Inactive Node Timeout: 5 Sec
Join wait time: 0 Sec
☐ Use node ranking: 0 Rank
Node ranking gives preference to specific nodes. Nodes with a lower number will be used first.

OPC Quality Monitoring
☑ Use OPC Quality in Determining Node Status
Item List:
localhost:KEPware.KEPServerEx.V4\Channel_0_Us

Shared Project

The **shared project** settings affect where FactorySQL finds the redundant project information, and how often it checks for updates.

Project Info Table: The table where general project information is stored.
Project Table:: Where groups are stored.
Project Refresh Period: How often to check for project updates.

Node Responsibility

The **node responsibility** settings specify how FactorySQL determines the rank of nodes- whose master, backup, etc. Pay careful attention when setting them, and make sure they are the same on each node.

Responsibility Table: The table that holds information about the nodes.
Responsibility Refresh Period: How often to update node status and check for responsibility changes.
Inactive Node Timeout: How long to wait before deciding a node is dead. In other words, nodes who have not updated their status within this amount of time will be marked as dead.
Join wait time: The node will maintain a "JOIN_WAIT" status in the responsibility table after startup for this amount of time. This is useful in preventing unnecessary responsibility switches in some cases.
Use Node Ranking/Node Rank: This allows you to give and use a ranking to the node. Lower ranked (numerically) nodes will be prefered as master. This option circumvents the normal node ranking, so it should only be used in cases where you want to make sure a certain machine is master if it's running. Nodes with the same ranking are ordered according to the normal principals.

OPC Quality Monitoring

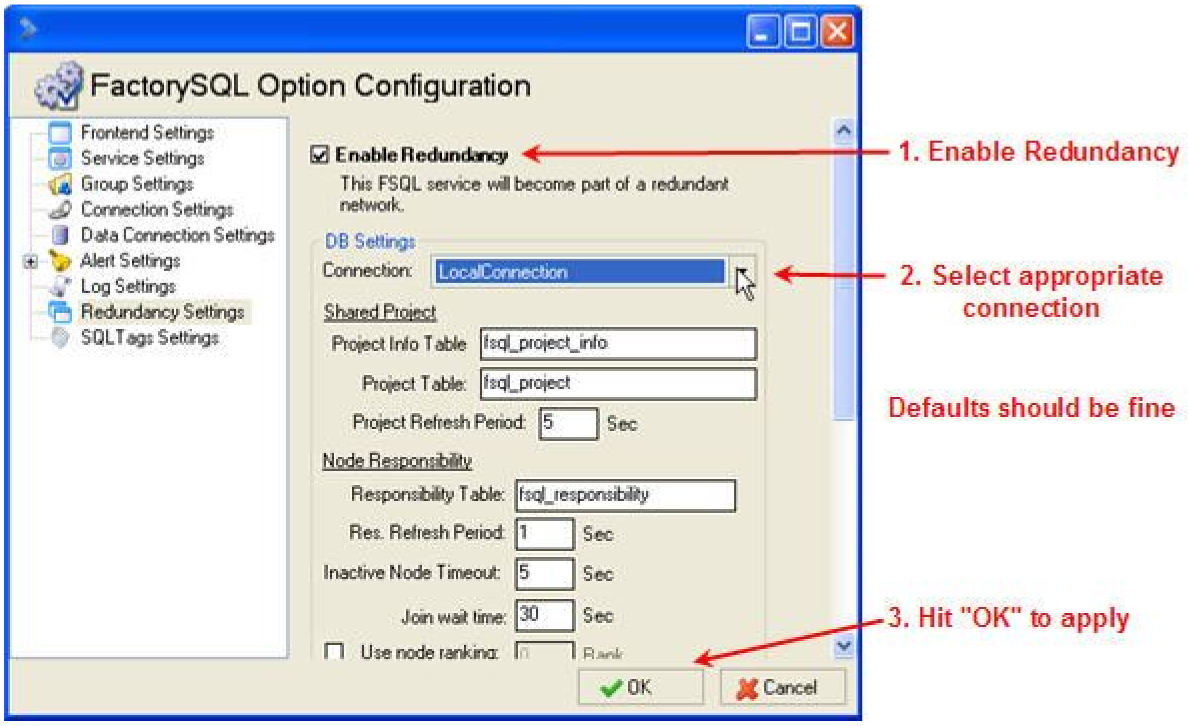
OPC Quality Monitoring allows you to specify tags that will be monitored for quality. This quality will be inserted into the node status table, and used to determine node responsibility. In this way, if one node has valid data when another one does not, it will become master.
To add a point to monitor, simply click the Add button, and enter in the server and path. It is suggested that you add a tag from each PLC/Topic/Device you are reading from.

Redundancy - Getting Started

The redundancy system is designed to be very easy to get started. Essentially, all you need to do is turn it on and populate the shared project.
This quick overview assumes you would like to use your current local project as the redundant project.

Step 1: Enable

Enable redundancy under Settings - Redundancy Settings.


FactorySQL Option Configuration

☑ Enable Redundancy
This FSQL service will become part of a redundant network.

DB Settings
Connection: LocalConnection

Shared Project
Project Info Table: fsql_project_info
Project Table: fsql_project
Project Refresh Period: 5 Sec

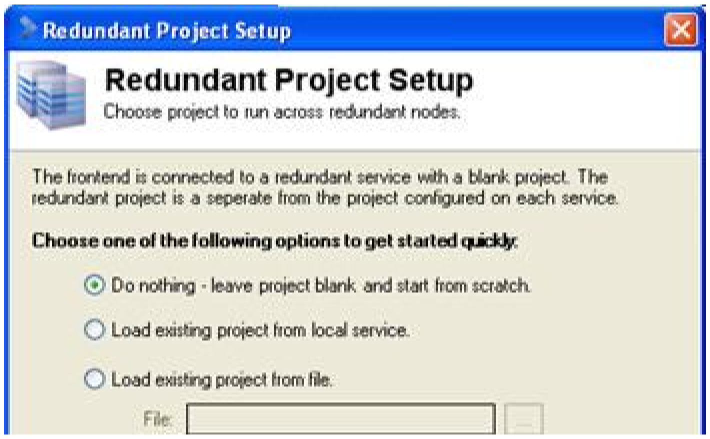
Node Responsibility
Responsibility Table: fsql_responsibility
Res. Refresh Period: 1 Sec
Inactive Node Timeout: 5 Sec
Join wait time: 30 Sec
☐ Use node ranking 0 Rank

Select the data connection that you will be using. Normally the default settings are fine, but if you need to change them you can refer to the Redundancy Settings section for more information.
Hitting "OK" will apply the settings and close the window.

Step 2: Configure Shared Project

The **shared project** is the project that is used by all of the nodes. It is different than the project held by any particular node. That is, when running redundantly, the FactorySQL service uses the shared project instead of its internal project, but the internal project is not lost. By disabling redundancy you can restore the internal project. It is currently not possible to run both the internal and shared project simulataneously.

After enabling redundancy for the first time, the shared project will be blank, and the **Redundant Project Setup Dialog** will appear. This dialog allows you to quickly populate the shared project with an existing project.


Redundant Project Setup

Redundant Project Setup
Choose project to run across redundant nodes.

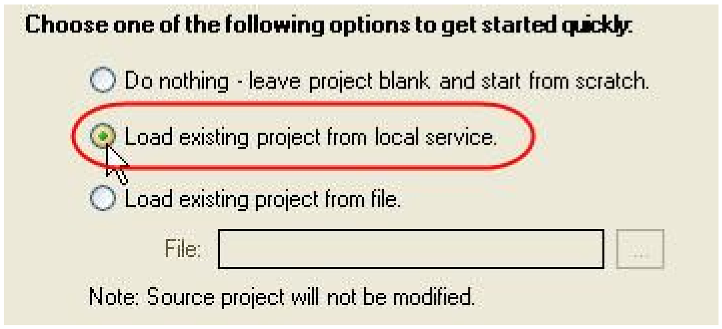
The frontend is connected to a redundant service with a blank project. The redundant project is a seperate from the project configured on each service.

Choose one of the following options to get started quickly:

[✓ Do nothing - leave project blank and start from scratch.]
Load existing project from local service.
Load existing project from file.
File:

Note: Source project will not be modified.

There are 3 options: do nothing, load from service, load from file. The first option, "Do nothing", will not load a project, and you will be able to start with a blank slate. The other 2 will load an existing project.
Normally you'll want to load the local service's project:


Choose one of the following options to get started quickly:

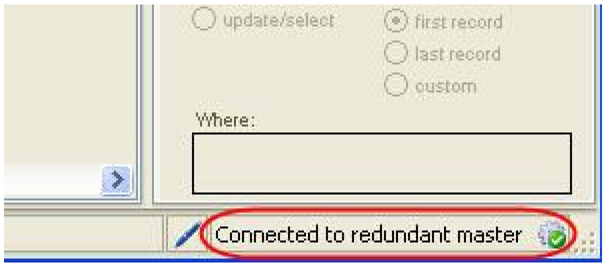
Do nothing - leave project blank and start from scratch. [✓ Load existing project from local service.] Load existing project from file.

File:

Note: Source project will not be modified.

Step 3: Done

After the project is loaded, you should see a screen that looks very similar to when you started. In the lower right corner, however, you'll notice that you are now running in a redundant state. Due to your setup and the other nodes on the network, you may or may not be currently connected to the master node.



Connected to redundant master

SQLTags^{TM} Overview

SQLTags provides a powerful link between FactorySQL and FactoryPMI, while maintaining all of the benefits of a database-centric architecture.

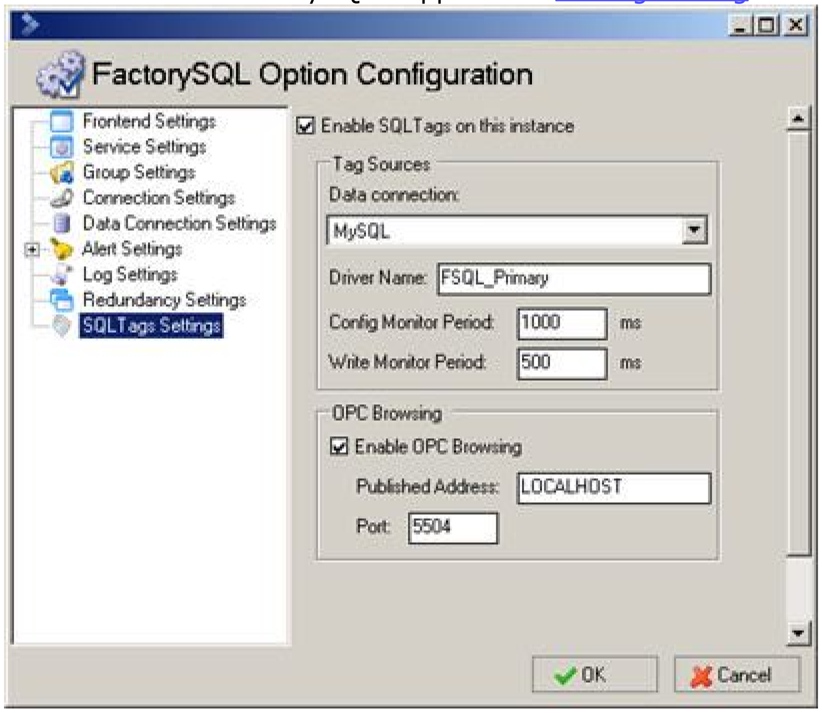
History & Overview

In the past, configuring systems using FactorySQL and FactoryPMI focused heavily on the central SQL database. Tags were mapped in FactorySQL from the OPC server to a particular location in the database, and then it was up to the designer to interpret the database values in a meaningful way in FactoryPMI. This direct approach was very powerful, but had several drawbacks. Primarily, there was a disconnect between addresses in the OPC server and their display/control counterparts in the PMI project, creating significant room for error and difficulty in maintenance. Additionally, there was a significant learning curve for users with little previous SQL experience.

SQLTags aims to address these issues, as well as provide a range of other benefits not previously possible in FactorySQL/PMI. Essentially, SQLTags creates a tag based architecture directly in FactoryPMI, allowing users to develop systems rapidly while conceptually "skipping over" the database and FactorySQL configuration. Users can browse OPC servers, configure expressions and alerts, and otherwise benefit from all of the features they would use in FactorySQL, without leaving the FactoryPMI design environment. Additionally, there is better support for control systems with an improved write response system built in.

Configuration

SQLTags is designed to be easy to use. Therefore, there is very little setup in FactorySQL. The SQLTags settings can be found in FactorySQL's application settings dialog.



☑ Enable SQLTags on this instance

Tag Sources

Data connection: MySQL

Driver Name: FSQL_Primary

Config Monitor Period: 1000 ms

Write Monitor Period: 500 ms

OPC Browsing

☑ Enable OPC Browsing

Published Address: LOCALHOST

Port: 5504

Enable SQLTags: Turns on/off SQLTags for the current FactorySQL service.
Data connection: The database connection used for SQLTags. The necessary tables will be created if they do not already exist.

Driver Name: FactorySQL will only execute SQLTags whose driver name matches its own driver name. All other tags will run as "external tags"- their values will be monitored, but it they should be executed by a different instance.
Config Monitor Period: How often, in milliseconds, the database should be checked for tag configuration changes.
Write Monitor Period: How often the database should be checked for OPC write requests.

Enable OPC browsing: FactorySQL can provide OPC browsing services to the FactoryPMI gateway. Enabling this feature requires an open port that is visible from the FactoryPMI gateway.
Published Address: The address that FactoryPMI will use to contact the FactorySQL service for OPC browsing. If

both services are on the same machine, LOCALHOST should work fine.
Port: The port that will be used by FactoryPMI to browse.

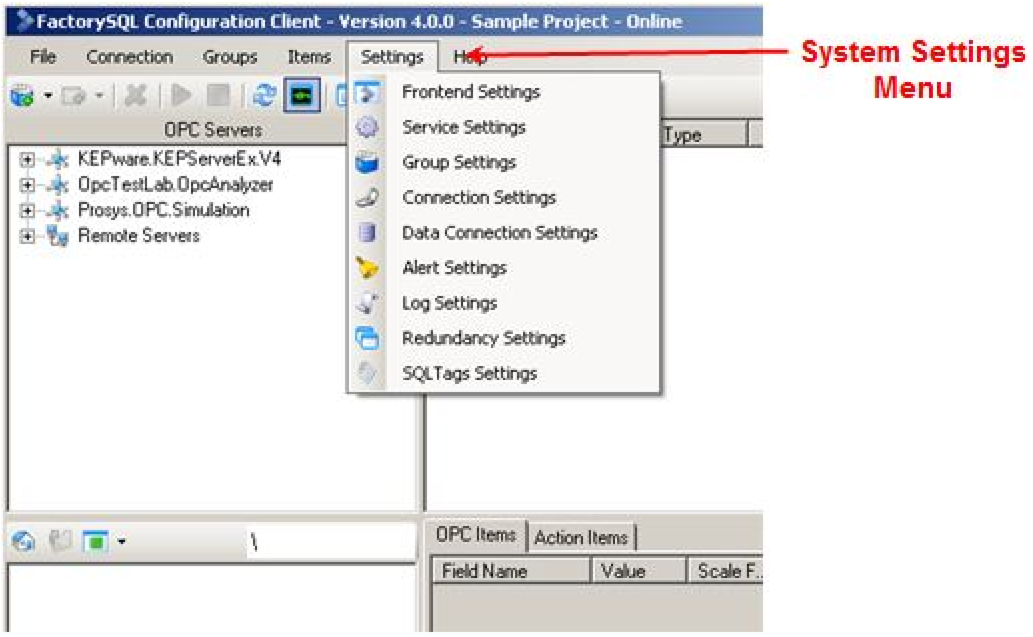
SQLTags and Redundancy

SQLTags automatically supports redundancy. When redundancy is enabled, FactorySQL will only execute SQLTags while it is the master. The primary consideration when setting up redundant SQLTags systems is that **every instance's "driver name" should be the same**. Otherwise, some nodes may view tags as external tags instead of properly executing them.

Application Settings

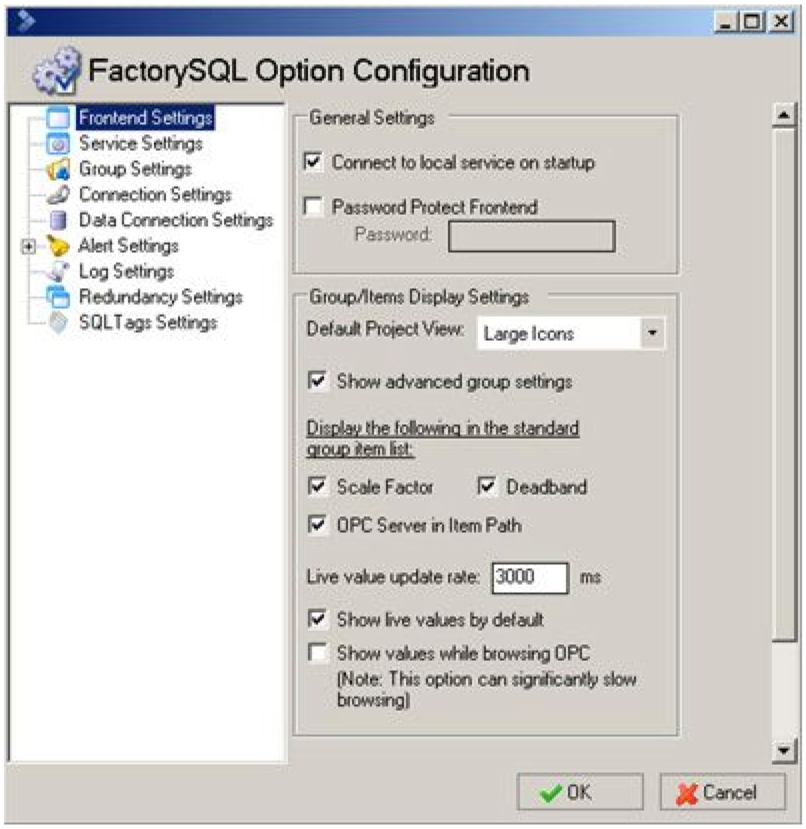
- Frontend Settings
- Service Settings
- Group Settings
- Connection Settings
- Data Connection Settings
- Alert Settings
- Log Settings
- Redundancy Settings
- SQLTags Settings

"Settings" refer to application wide settings for FactorySQL. These include program defaults, data connections, and security. These are configured via the top menu and should not be confused with group settings on the right tabs.



Frontend Settings

FactorySQL frontend settings affect only the frontend running on the local machine, regardless of FactorySQL service connections.



General Settings

[✓] Connect to local service on startup

[ ] Password Protect Frontend

Password:

Group/Items Display Settings

Default Project View: Large Icons

[✓] Show advanced group settings

Display the following in the standard group item list:

[✓] Scale Factor [✓] Deadband

[✓] OPC Server in Item Path

Live value update rate: 3000 ms

[✓] Show live values by default

[ ] Show values while browsing OPC (Note: This option can significantly slow browsing)

General Settings

Connect to local service on startup: The FactorySQL Frontend will connect to the same machine upon startup.

Password Protect Frontend: This FactorySQL frontend will require a password to open. This has no effect on the FactorySQL Service running.

Group/Items Display Settings

Default Project View: How the project is displayed in the frontend by default.

Show advanced group settings: Determines whether advanced settings tabs are displayed for certain group types.

Display the following...: Sets whether the specified columns are shown by default in the OPC Item Pane.

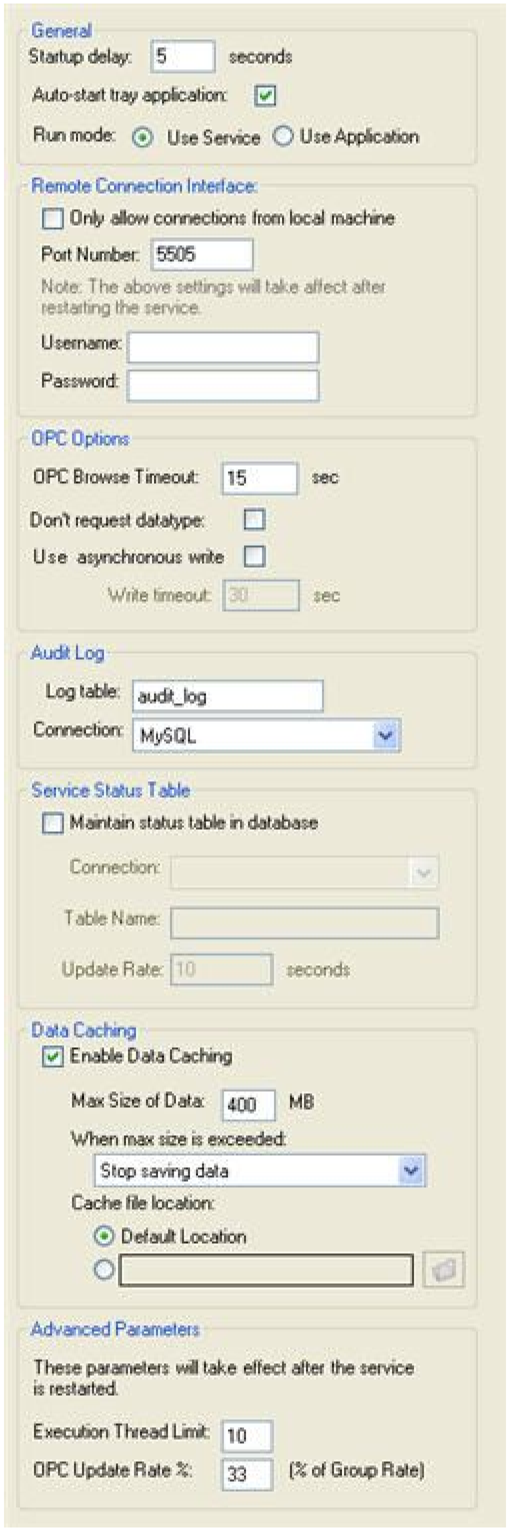
Live value update rate: Frequency in milliseconds that the FactorySQL frontend will update live value display.

Show live values by default: Whether live values will normally be turned on after connecting to a service.

Show values while browsing OPC: If selected, FactorySQL will attempt to show a snapshot of values in the OPC Browse Pane when browsing. Whether or not this option works depends on the OPC server being browsed. It can also slow browsing, as the server must go retrieve the value of each tag.

Service Settings

FactorySQL Service settings affect the service settings on any machine or connection. This *can* be managed remotely.


General
Startup delay: 5 seconds
Auto-start tray application: ✓
Run mode: [✓ Use Service] Use Application

Remote Connection Interface:
☐ Only allow connections from local machine
Port Number: 5505
Note: The above settings will take affect after restarting the service.
Username:
Password:

OPC Options
OPC Browse Timeout: 15 sec
Don't request datatype: ☐
Use asynchronous write ☐
Write timeout: 30 sec

Audit Log
Log table: audit_log
Connection: MySQL

Service Status Table
☐ Maintain status table in database
Connection:
Table Name:
Update Rate: 10 seconds

Data Caching
✓ Enable Data Caching
Max Size of Data: 400 MB
When max size is exceeded:
Stop saving data
Cache file location:
[✓ Default Location]

Advanced Parameters
These parameters will take effect after the service is restarted.
Execution Thread Limit: 10
OPC Update Rate %: 33 (% of Group Rate)

Service Options:

Startup delay: FactorySQL will wait for this many seconds before starting and connecting to any OPC servers. The delay is sometimes necessary to ensure that the OPC server has been fully loaded by Windows before FactorySQL requests values.

Auto-start tray application: Whether Windows will start the tray application on startup/login. The tray application makes sure the service is running, and can run the project if the service is unable to start. It is also the process that runs the project if run mode is set to "Application".

Run Mode: The **FactorySQL Service** can run as either a **Windows Service** or an **application**. **Note:** As of version 2.2, this feature has been deprecated. It is still possible to run as an application, but it is not support out of the box. For more information, see the Inductive Automation Support Forums.

Remote Connection Interface (Security):

Only allow connections from local machine: FactorySQL will no longer listen for connections that allow remote administration.

Port Number: The port that FactorySQL listens on for frontend connections. Often IT departments will want to set this to adhere to corporate firewall standards.

Username/Password: Requires FactorySQL Frontend to specify given username and password to connect to this FactorySQL Service. Note: This applies to both local and remote connections.

OPC Options

Browse Timeout: How long to wait while browsing before canceling the operation. OPC servers can vary widely in how long browse operations take. For example, some servers only query the device for address structure at the time of the browse, potentially causing the operations to take a minute or more to return. Setting this value higher will accomidate these types of servers.

Don't request datatype: Normally FactorySQL will pass datatype setting information along to the OPC server. The server, in turn, will try to return the data in the requested format. Selecting this option will prevent FactorySQL from requesting the type, causing the OPC server to send the data in its native format.

Use asynchronous write: The OPC specification support Synchronous and Asynchronous writing methods. By default FactorySQL will use synchronous, causing write operations to block until the server returns a result. Choosing this option will cause FactorySQL to use asynchronous writes.

Write Timeout: If asynchronous writes are used, they can be canceled after the specified timeout if not completed.

Audit and Status Settings

Audit log table: Table name used for the audit log. Any groups configured to use auditing will write to this table.
Connection: The data connection used to access the table.

Maintain status table in database: FactorySQL will create and maintain a **status table** with the given **Table Name** on the given **Connection**.

Data Caching

Data Caching allows FactorySQL to buffer historical data when the connection to the database fails. It is enabled by default.

Enable Data Caching: Turn caching on/off.

Max Size of Data: The maximum amount of disk space caching will use.

When max size is exceeded...: What to do when the max is reached. There are currently 2 options: Overwrite old data, or Stop caching new data.

Advanced Parameters

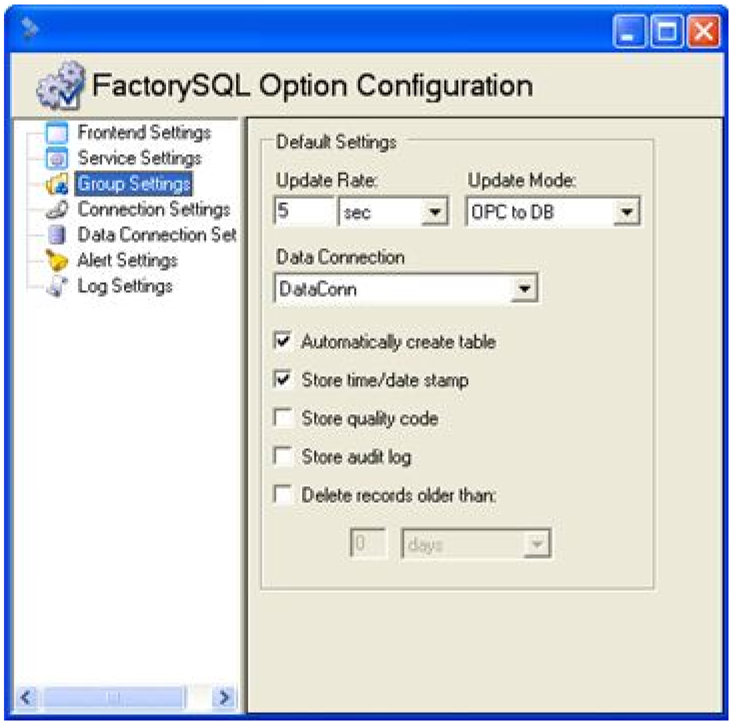
Execution Thread Limit: The max number of threads FactorySQL will use to process work. The minimum is 1, and the maximum is 25. This value should be changed with care.

OPC Update Rate %: The percentage of the group update rate that the OPC values will update. Recommended

<50%, but may need to be set higher depending on the number of tags being used and the OPC server.

Group Settings

Group settings define defaults for new transactional groups. All settings are described under the Action Tab.



Default Settings

Update Rate: 5 sec

Update Mode: OPC to DB

Data Connection: DataConn

☑ Automatically create table

☑ Store time/date stamp

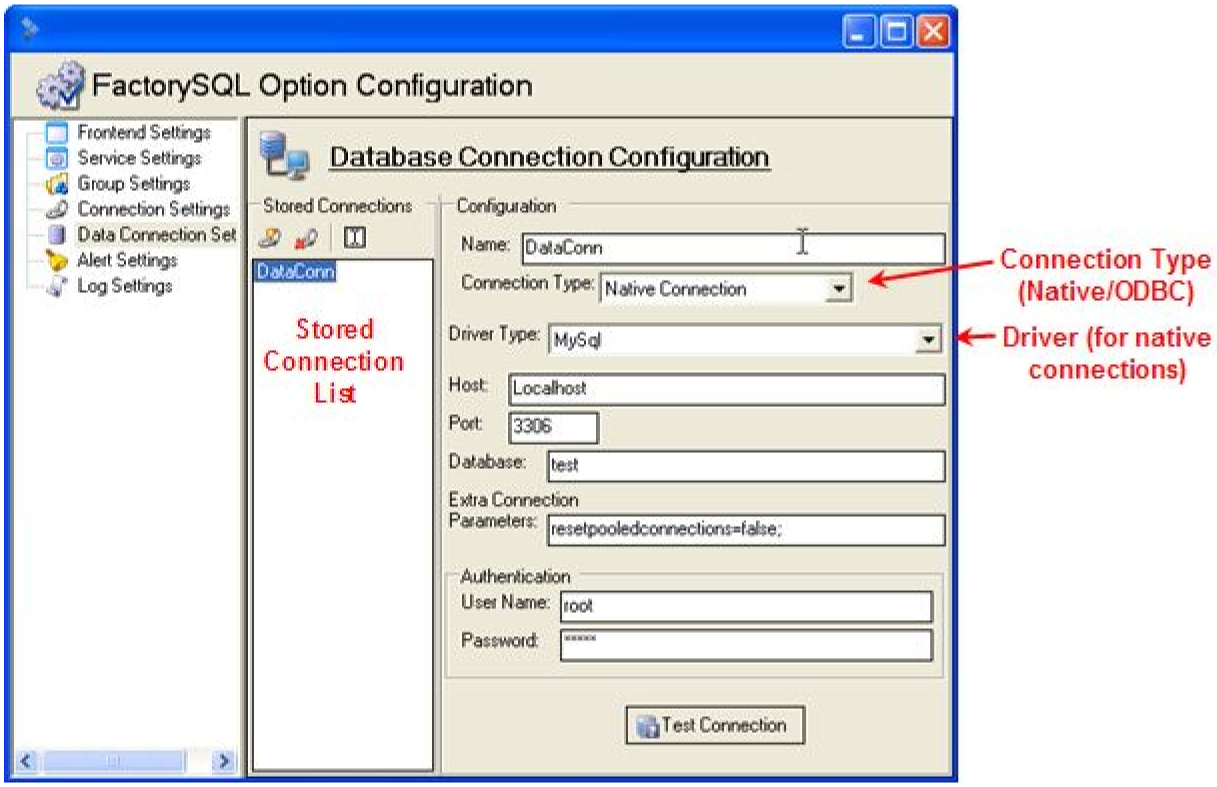
☐ Store quality code

☐ Store audit log

☐ Delete records older than: 0 days

Database Connection Configuration

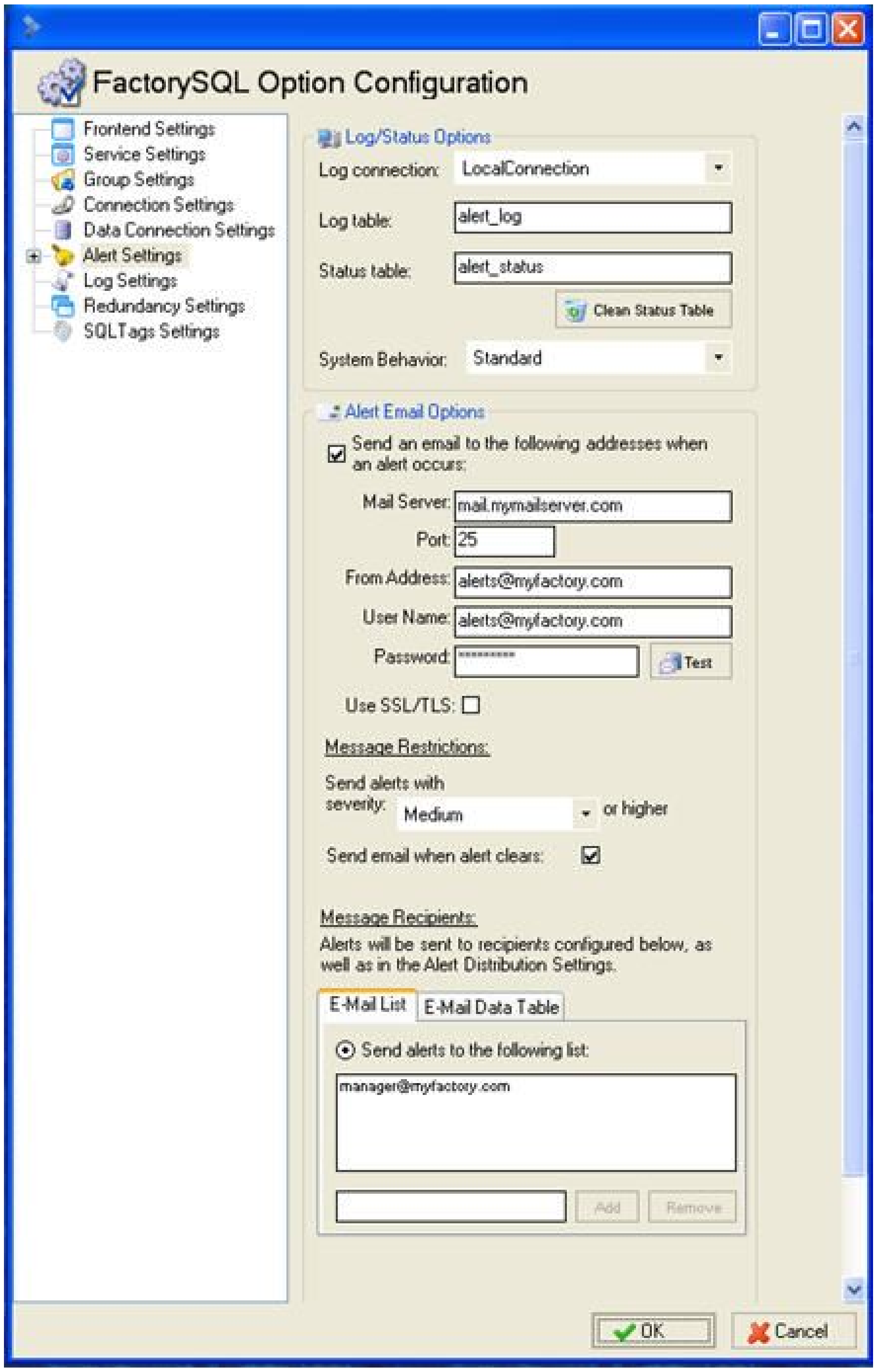
Database Connection Configuration allows the developer to create stored database connection profiles that contain all information required to make a connection to a database. The developer can create any number of database connections. These can be different database types on any machine that FactorySQL can access. For more information, see Data Connections.



Name: DataConn
Connection Type: Native Connection
Driver Type: MySql
Host: Localhost
Port: 3306
Database: test
Extra Connection Parameters: resetpooledconnections=false;
User Name: root
Password: *****

Alert Settings

The **Alert Settings Page** allows you to configure the various settings for the Alert System.


FactorySQL Option Configuration

- Frontend Settings
- Service Settings
- Group Settings
- Connection Settings
- Data Connection Settings
- Alert Settings
- Log Settings
- Redundancy Settings
- SQLTags Settings

Log/Status Options

Log connection: LocalConnection

Log table: alert_log

Status table: alert_status

Clean Status Table

System Behavior: Standard

Alert Email Options

[✓] Send an email to the following addresses when an alert occurs:

Mail Server: mail.mymailserver.com

Port: 25

From Address: alerts@myfactory.com

User Name: alerts@myfactory.com

Password: **********

Test

Use SSL/TLS: [ ]

Message Restrictions:

Send alerts with severity: Medium or higher

Send email when alert clears: [✓]

Message Recipients:

Alerts will be sent to recipients configured below, as well as in the Alert Distribution Settings.

E-Mail List | E-Mail Data Table

(●) Send alerts to the following list:

manager@myfactory.com

Add Remove

OK Cancel

Log/Status Options

Log connection: Specifies database connection to use for the alert log and alert status tables.

Log table: Table name of historical **alert log**

Status table: Table name of realtime **alert status**.

System Behavior: Specifies how the alert log and status tables should behave. See Alerting for more information.

Alert Email Options

Send email to the following addresses when an alert occurs: Enables alerting e-mail notifications. When a qualified alert occurs (according to to the severity restriction below), an email will be sent through the specified mail server, to the specified accounts. **E-mail Recipients List:** You can specify two sources for the list of addresses to send to: a static list, or a database column.

- **Static List**: Simply enter the addresses of the recipients in the box, and click "Add".
- **Data Table**: With this option, you can specify a connection name, table, and column to select each time an alert needs to be sent. This allows you to easily change the recipient list from outside of FactorySQL. Additionally, with a database that supports views, you could create a view that returns a dynamic list, based on system conditions such as date and time. With some creativity, there's a great deal of power in this feature.

Mail Server: SMTP server to use to send email.

Port: Normally 25, but some mail servers require a different port number. **User Name/Password:** to send email. May be optional, best way to tell is with the "Test" button

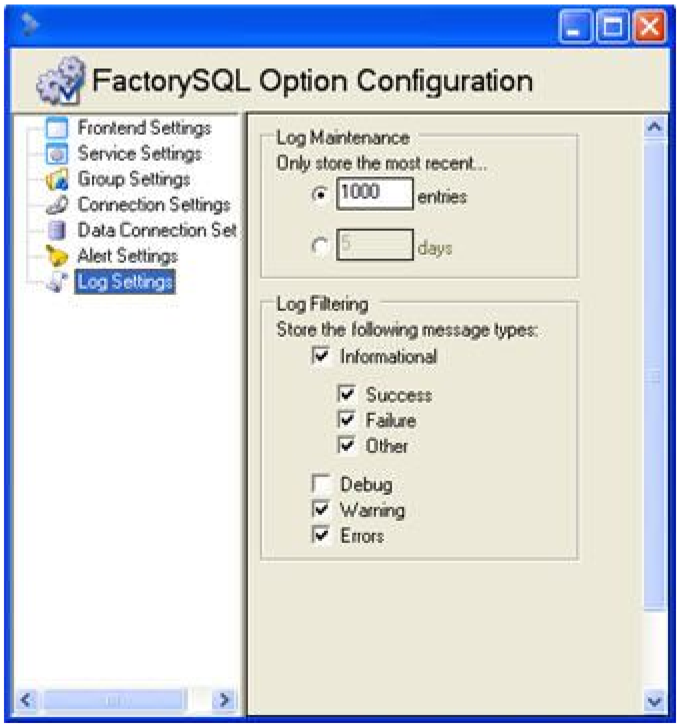
Send alerts with severity: Will only send an email for alerts of the selected severity or greater.

Send email when alert clears: Will send another email when alerts of the selected severity or greater clear.

Advanced - Use SSL/TLS: Will use encryption when talking to the server. Required by some services, such as GMail.

Log Settings

The **Log Settings Page** allows you to configure the various settings for the FactorySQL Error/Event logging system. The logging system log errors and messages to the internal database, here you can specify what to log, and how long to keep it.



Log Maintenance
Only store the most recent...
(●) 1000 entries
( ) 5 days

Log Filtering
Store the following message types:
[✓] Informational
[✓] Success
[✓] Failure
[✓] Other
[ ] Debug
[✓] Warning
[✓] Errors

Log Maintenance

Store X entries: Will only store the specified number of messages.

Store X days: Will store the specified number of days worth of events, no matter how many there are.

Log Filtering

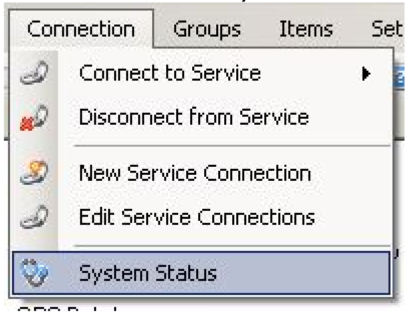
Specifies what type of messages to store. The default set includes everything except for Debug messages, as there can be a great number of debug messages generated.

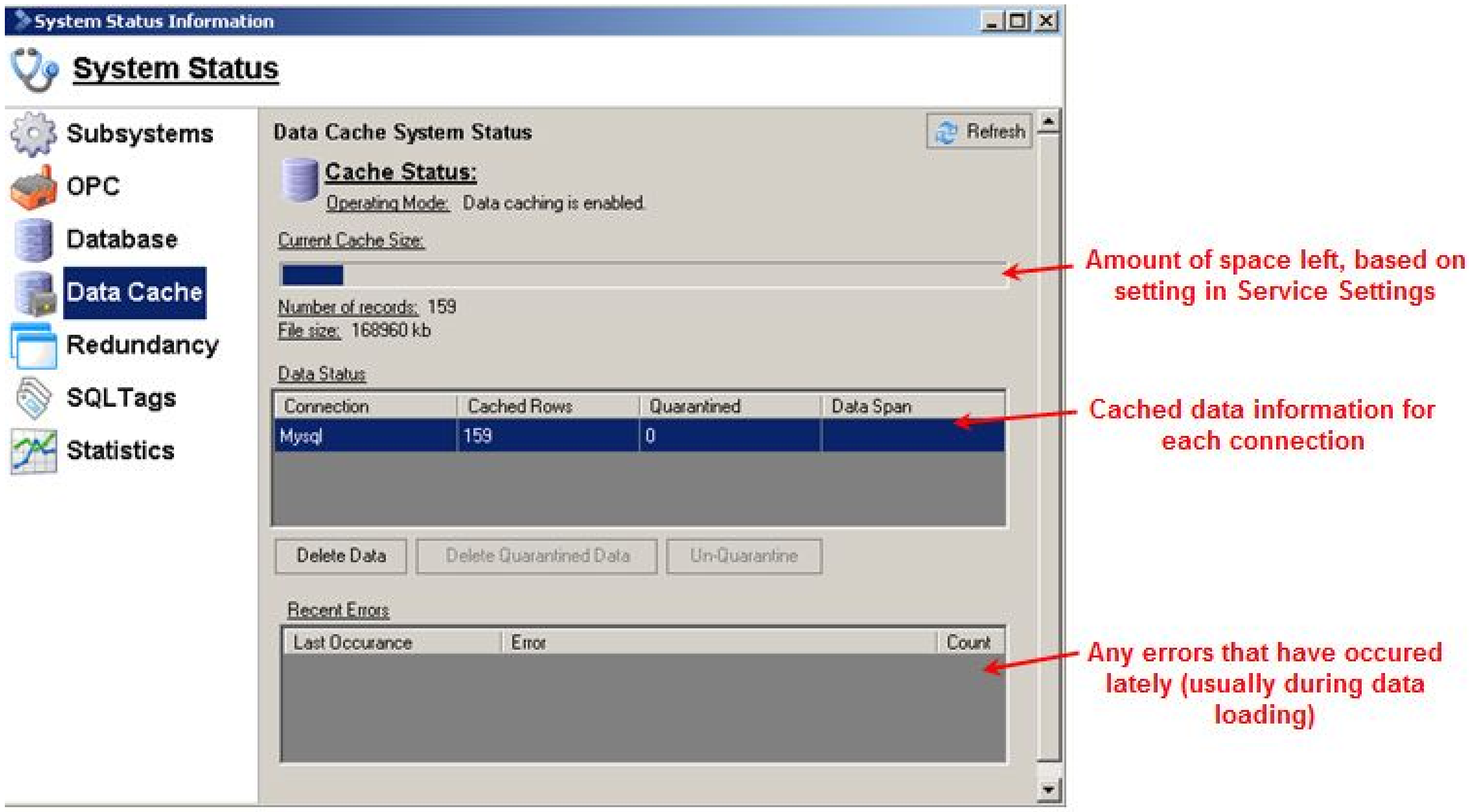
Data Cache

The **data cache** provides local storage for historical data while the target database connection is unavailable. When enabled through Service Settings (it is enabled by default), historical queries generated by groups and alerting will be redirected to the cache should the database connection go down. When it is available again the data will be loaded into the target database. Any timestamp fields in the queries (such as the data for the t_stamp column) will be adjusted and accurate for the target database, so the data will appear as if nothing occured (assuming the data is sorted by timestamp).

Data Cache Status

The data cache system status can be found under the System Status dialog:




System Status Information
System Status
Subsystems
OPC
Database
Data Cache
Redundancy
SQLTags
Statistics
Data Cache System Status
Refresh
Cache Status:
Delete Data
Delete Quarantined Data
Un-Quarantine
Amount of space left, based on setting in Service Settings
Cached data information for each connection
Any errors that have occured lately (usually during data loading)

Operating Mode: Data caching is enabled.

Current Cache Size:

Number of records: 159
File size: 168960 kb

Data Status

Connection	Cached Rows	Quarantined	Data Span
Mysql	159	0	

Recent Errors

Last Occurance	Error	Count

The data cache status page provides important information about data in the cache. You can see how much data is stored, and how much data has been quarantined. From this screen you can choose to delete data and un-quarantine it, as well as view any errors that may be causing data to become quarantined.

Quarantined Data

Data becomes quarantined when the data cache system is unable to load it after several attempts. These queries usually would cause an executing group to error out, but were stored to the cache. Common causes include references to tables or columns that don't exist, or data issues- such as trying to write a NULL to a non-null column.

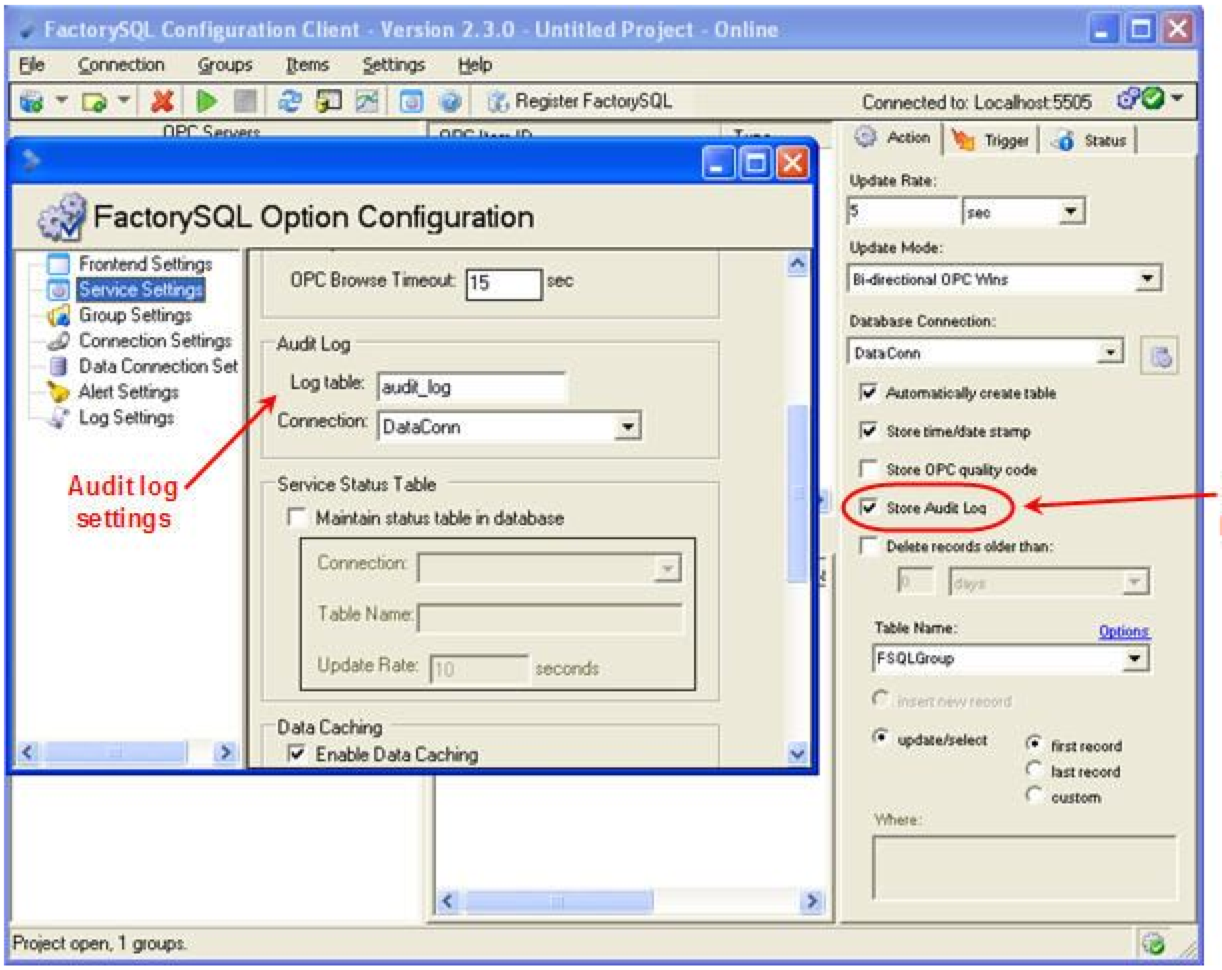
Once quarantined, FactorySQL won't attempt to load the data again. By going to the status screen, you can view the errors that occured, and either delete the quarantined data or try to fix the errors and un-quarantine it. When you un-quarantine the data the system will attempt to load it in the next load cycle (which may take up to a minute to occur).

Auditing

Auditing inserts a record into the audit log table that indicates whenever a PLC register is written to. It is up to the user to update the "log_usr" column in the database row corresponding to the value(s) that will be written to the PLC. This is necessary because FactorySQL has no notion of who is logged in, and the system expects the possibility of multiple concurrent users .

When auditing is enabled on a group that writes to the PLC, the following occurs:
1. FactorySQL keeps track of the old value of each item.
2. FactorySQL keeps track of the new value of each item.
3. FactorySQL reads "log_usr" from the database in the same row as the new value, expecting that the HMI/web page correctly updated that upon writing that value down.
4. FactorySQL writes a new record in the audit log for **each** value that changed. Each entry has all of the information below. The table uniquely identifies where the change came from in the database, when it occurred, the user who made the change, and the new and old values.

Settings are configured **here**.


FactorySQL Configuration Client - Version 2.3.0 - Untitled Project - Online

Connected to: Localhost:5505

FactorySQL Option Configuration

OPC Browse Timeout: 15 sec

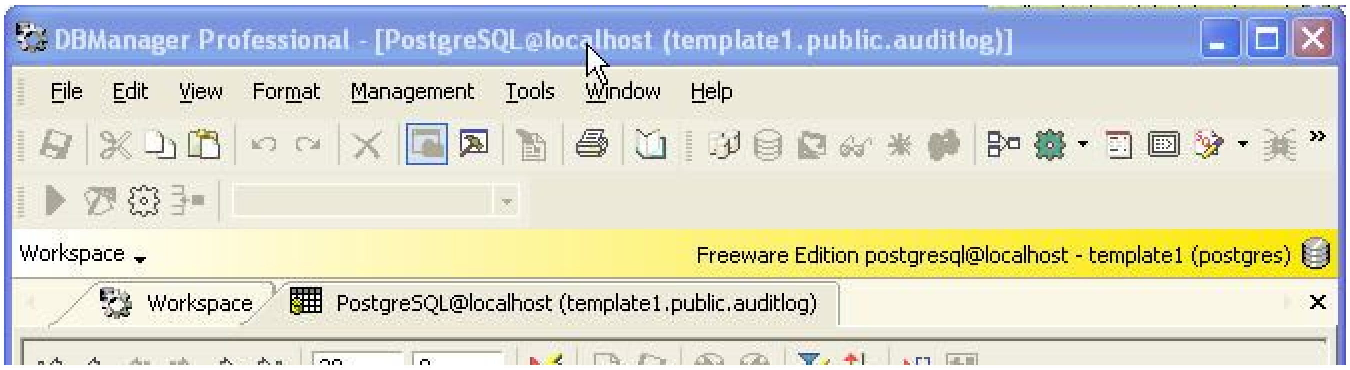
Audit Log
Log table: audit_log
Connection: DataConn

Service Status Table
Maintain status table in database
Connection:
Table Name:
Update Rate: 10 seconds

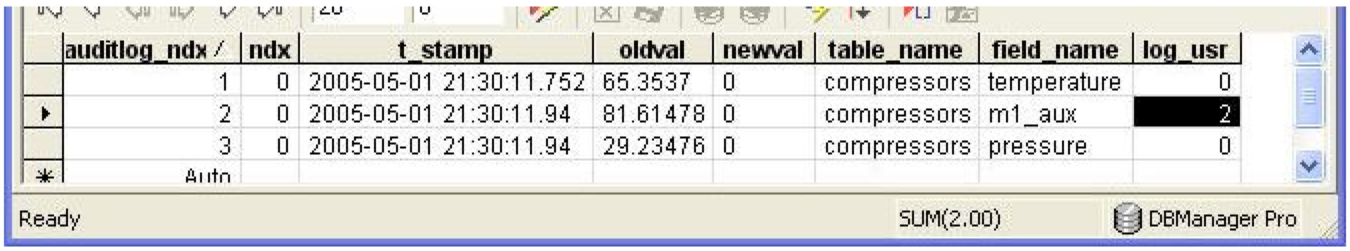
Data Caching
Enable Data Caching ✓

Update Rate: 5 sec
Update Mode: Bi-directional OPC Wins
Database Connection: DataConn
✓ Automatically create table
✓ Store time/date stamp
Store OPC quality code
✓ Store Audit Log
Delete records older than: 0 days
Table Name: FSQLGroup
insert new record
● update/select
● first record
last record
custom
Where:

Project open, 1 groups.


DBManager Professional - [PostgreSQL@localhost (template1.public.auditlog)]

Freeware Edition postgresql@localhost - template1 (postgres)



auditlog_ndx	ndx	t_stamp	oldval	newval	table_name	field_name	log_usr
1	0	2005-05-01 21:30:11.752	65.3537	0	compressors	temperature	0
2	0	2005-05-01 21:30:11.94	81.61478	0	compressors	m1_aux	2
3	0	2005-05-01 21:30:11.94	29.23476	0	compressors	pressure	0
Auto							

Ready SUM(2.00) DBManager Pro

Columns:

t_stamp:Date/time value was written.

oldval: Old value in the PLC.

newval: Value written to the PLC.

table_name: Table in the database that value was read from.

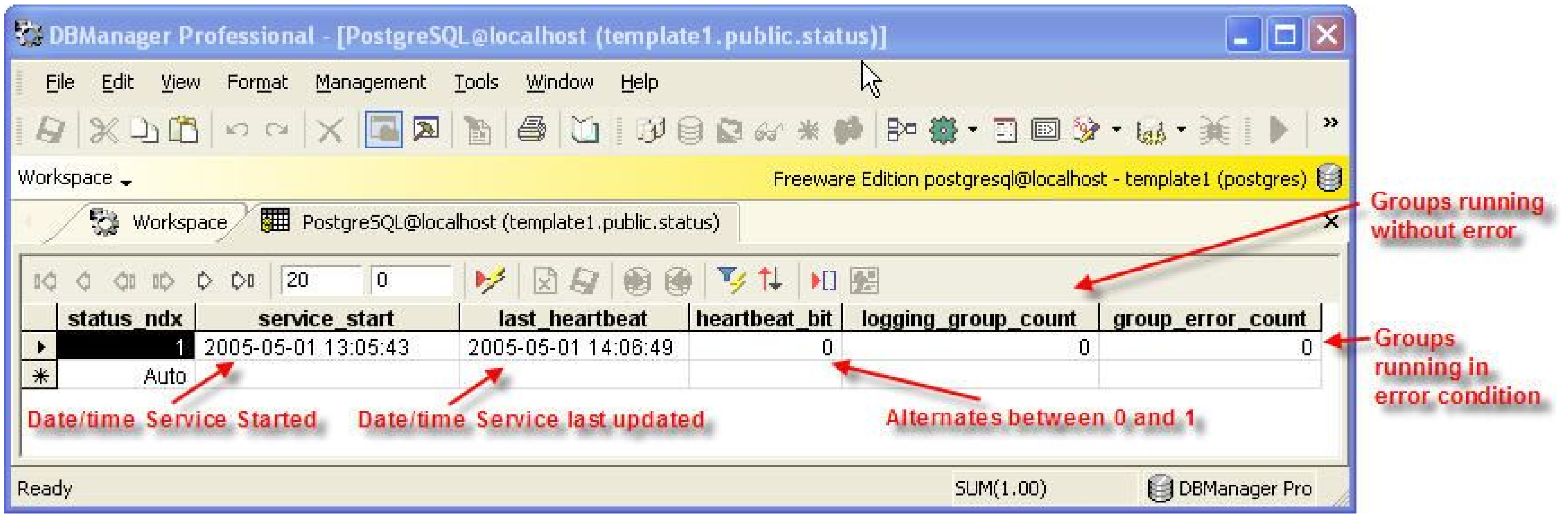
fieldname: Field in the database that value was read from.

ndx: The index (unique identifier column) value in the database that value was read from.

log_usr: Last user to write a value down according to database record. FactorySQL does not track this.

Status Table

The FactorySQL status table indicates the current running status of FactorySQL in a database table. Settings are configured in the **Service Settings** area.


DBManager Professional - [PostgreSQL@localhost (template1.public.status)]
File Edit View Format Management Tools Window Help
Workspace
Freeware Edition postgresql@localhost - template1 (postgres)
Workspace
PostgreSQL@localhost (template1.public.status)
20
0
Groups running without error
Groups running in error condition
Date/time Service Started
Date/time Service last updated
Alternates between 0 and 1
Ready
SUM(1.00)
DBManager Pro

	status_ndx	service_start	last_heartbeat	heartbeat_bit	logging_group_count	group_error_count
▸	1	2005-05-01 13:05:43	2005-05-01 14:06:49	0	0	0
*	Auto					

Status Fields

service_start: Date/time the FactorySQL service started.

last_heartbeat: Date/time FactorySQL last wrote to the database.

heartbeat_bit: Alternates value between 0 and 1.

logging_group_count: Groups running successfully.

group_error_count: Groups running that have at least one error.

Expression Language Syntax

The FactorySQL Action Item expression language is a simple language designed to perform basic calculations, and to support drop-in function plugins created with the ScriptingAPI.

It is important to keep in mind that every expression will be evaluated down to 1 value. Although it is possible to save and use "variables" with the Save/GetVariable functions, there is no assignment operator.

The following table outlines the operators available:

Operator	Operation
Arithmetic	
+	Addition
-	Subtraction
*****	Multiplication
/	Division
^	Power
%	Modulus (remainder)
Bitwise Operators	
&	Bitwise AND
|	Bitwise OR
XOR	Bitwise XOR
<<	Bitshift Left
>>	Bitshift Right
Logical Operators	
&&	Logical AND
||	Logical OR
! or **NOT**	NOT (boolean inverse)
= or ==	Equals
!=	Not Equal
>	Greater-Than
>=	Greater-Than or Equal to
<	Less-Than
<=	Less-Than or Equal to

Notes on Arithmetic Operators

All of these operators return a value whose datatype is the same as the most precise datatype used in the operation. For instance, a "Long (8 bytes) + Integer (4 bytes)" returns a Long. Along this line though, it is important to note that if writing back to an OPC item, that value will be cast to whatever the data type of the Item is set to. So if you perform the operation "3.4 + 7" and write it to an Integer, even though the value of the action item would be 10.4 (a double), it would be changed to an Integer for the purpose of writing to the item, and thus only 10 would be stored.

Notes on Bitwise Operators

These operation will return integers or Longs, depending on the operands.
The bitshift operators work as follows: Value "<<|>>" Number_Of_Shifts . For instance, 1 << 1 = 2, 1 << 2 = 4, 8 >> 3 = 1, etc.

Notes on Data Types

The scripting language inherantly supports, and automatically casts between the following datatypes: Integers, Longs, Floats, Doubles, Booleans and Strings.

As mentioned above, calculation between mixed types will default to the most accurate type. In the case of booleans, if used in a calculation, they will be converted to 0 for false, and 1 for true.

The following keywords are supported for booleans (typed without quotes):
"true","True","TRUE","false","False","FALSE".

NOTE: There are more datatypes supported inside of the expression, for the purpose of calling Functions, but the ultimate value must be one of these supported types. What this means is that it is possible to use an exterior function that returns a non supported type, such as a Date, as a parameter to another function accepting that type, as long as the final function returns something that is supported. See the Scripting API documentation for more information.

Notes on Strings

Strings are defined with either single or double quotes. Thus 'hello' and "hello" are the same.
There are 2 operators that are valid for strings: concatenation (+), and removal (-). There are defined as follows:
Concatenation (+): Takes both sides, and returns one string. If one of the sides is not a string, it will be converted to one.
Example: "Hello" + " World" = "Hello World", 5 + " times" = "5 times".
Removal (-): This will remove all instances of the right side from the left side.
Example: "Hello" - "l" = "Heo", " padded " - " " = "padded"

Item Substitution

Item Substitution works the same in the expression language as everywhere else in FactorySQL. That is, you can reference the value of any item in your group by typing "{*item_name*}", or by pressing **CTRL+SPACE** to bring up the item list. Items may be reference anywhere, in calculations, function call, etc. However, you cannot use a reference inside of a reference.

Function Calls

If you have a suitable function plug-in installed, you may call the function like *functionName*(*Argument 1*, *Argument 2*)
The function name is not case sensitive, and of course the number of arguments must match those required by the function, as defined by your plug-in's documentation.

Database Functions

In many of the following database functions, an optional *ConnectionName* may be passed in (optional parameters are indicated with square brackets []). If not provided, the function will use the connection specified for the group under which it is executed.

IsConnectionAvailable(*[ConnectionName]*)

Returns: a boolean indicating whether the connection is available.
ConnectionName is optional, if it is not provided the connection for the group is tested.

ExecuteScalarQuery(*Query, [ConnectionName]*)

Executes a query, and returns the first value encountered.

Returns: The first value encountered in the result set. If the query produces multiple rows or columns, all except the first will be discarded.

ExecuteUpdateQuery(*Query, [ConnectionName]*)

Executes a query that is not expected to return results, such as **update** or **delete** queries.

Returns: the number of rows affected by the query.

ExecuteQuery(*Query, DataSetName, [ConnectionName]*)

Executes a query that generates a set of data (columns and rows), and stores the dataset to the name provided. The GetDataSetValue... functions may then be used to retrieve the data.

This mechanism works in a similar manner to the **StoreVariable/GetVariable** functions. **However**, the dataset variables are normally local to the group that created them. This is different than the variable functions, which place the data globally. Therefore, multiple groups can use the same *DataSetName* without overwriting each others data. To make a dataset global, prefix the name with an underscore. For example, "_globalData" would be shared amongst all groups.

Returns: The number of rows in the result set.

GetDatasetValueByIndex(*DataSetName, RowIndex, ColIndex, DefaultValue*)

Retrieves the value at the given row index and column index, in a dataset created by calling ExecuteQuery. The *DefaultValue* will be returned if the dataset does not exist, or if the requested row/column does not exist.

Row and column indices start at 0.

GetDatasetValueByName(*DataSetName, RowIndex, ColName, DefaultValue*)

Retrieves the value at the given row index, for the given column name, in a dataset created by calling ExecuteQuery. The *DefaultValue* will be returned if the dataset does not exist, or if the requested row/column does not exist.

Column names can be the actual name of the column, though it is often easier (and sometimes necessary) to "alias" the column in the SQL statement. For example, the query:
`SELECT min(val), max(val) FROM table`
uses aggregate expressions, whose resulting column names may or may not be easy to guess. It would be better to instead give them names in the query such as:
`SELECT min(val) as "minval", max(val) as "maxval" FROM table`
Now you could easily get their values with calls such as: `GetDatasetValueByName("dsetName", 0, "minval",0)`

Data cache

GetCachedRowCount()

Returns: the number of rows stored in the data cache. Does not include any quarantined (errored) rows.

GetCachedRowCountForConnection(ConnectionName)

Returns: the number of rows stored in the data cache for the given connection name. Does not include any quarantined (errored) rows.

Date & Time Functions

CurrentDateTime()
Returns the current date and time of the local machine. Also available: Now()

CurrentUTCDateTime()
Returns the current date and time of the local machine, converted to universal coordinated time.

DateArithmetic(*Date, ChangeAmount, CalendarField*)

Parameters

Date

The date to operate on.

ChangeAmount

The integer amount, positive or negative, to add to the date.

CalendarField

The field of the `Date` to add `ChangeAmount` to. Possible options are:

- `"second"`
- `"sec"`
- `"minute"`
- `"min"`
- `"hour"`
- `"hr"`
- `"day"`
- `"week"`
- `"month"`
- `"year"`

DateArithmetic will add the amount specified to the field of the date specified. Note that the date fields can be pluralized.

Examples:

DateArithmetic(CurrentDateTime(), 5, "hour")

...returns a new date, five hours after the current time.

DateArithmetic({ItemReference}, -8, "days")

...returns a new date, eight days before the value of the referenced item.

DateDiff(*Date1, Date2, CalendarField*)

Parameters

Date1

The first date to compare.

Date2

The second date to compare.

CalendarField

The field of the `Date` to calculate the difference for. Possible options are:

- `"second"`
- `"sec"`
- `"minute"`
- `"min"`
- `"hour"`
- `"hr"`
- `"day"`
- `"week"`
- `"month"`
- `"year"`

DateDiff will calculate the amount of time between the given two dates in the unit given. The return value will be a floating point value, meaning that parts of units are considered. The exception to this rule is for the months and years fields, which will always return an integral difference. If Date2 is after Date1, the return value will be positive, otherwise it will be negative.

Examples:

```
DateArithmetic(ToDate("2008-2-24 8:00:00"), ToDate("2008-2-24 8:15:30"), "minute")
```

...returns a 15.5

```
DateArithmetic(ToDate("2008-2-24 8:00:00"), ToDate("2008-3-12 9:28:00"), "month")
```

...returns a 1.0

```
DateArithmetic(ToDate("2008-2-24 8:00:00"), ToDate("2008-3-12 9:28:00"), "day")
```

...returns a 17.02

DateExtract(*Date, CalendarField*)

Parameters

Date

The date from which to extract the value of a calendar field.

CalendarField

The field of the `Date` to return. Possible options are:

- `"second"`
- `"sec"`
- `"minute"`
- `"min"`
- `"hour"`
- `"hr"`
- `"day"`
- `"week"`
- `"month"`
- `"year"`

DateExtract will return the integer value of a specific field inside of a date. For instance, the year.

DateFormat(*Date, FormatString*)

Formats the date according to the given string.
Common format string parameters:

Format specifier	Description
d	Day (1-31)
h	Hour (12-hour format)
H	Hour (24-hour format)
m	Minute
s	seconds
M	Month (1-12)
MMMM	Month, full name
tt	AM/PM display
yy	2 digit year
yyyy	4 digit year

Note: many specifiers can be repeated to increase information. For example, "s" would display seconds without an initial 0, but "ss" would always display it as 2 characters.

ExtractDay(*Time*)

Returns the day from the given date.

ExtractMonth(*Time*)

Returns the month from the given date.

ExtractYear(*Time*)

Returns the year from the given date.

ExtractHour(*Time*)

Returns the hour from the given date.

ExtractMinute(*Time*)

Returns the minute from the given date.

ExtractSecond(*Time*)

Returns the second from the given date.

TimeBetween(*Time, StartTime, EndTime*)

Checks to see if the given time is between the start and end times.
The given times are expected as strings, and may include dates. Note: dates will be parsed according to the default system culture.
If only the time is provided, and the start time is greater than the end time, start time will be assumed to refer to the day before.

Examples:

```
TimeBetween("2:00:00 pm","9:00:00 am","5:00:00 pm")
```

...returns TRUE

```
TimeBetween("14:00:00","09:00:00","17:00:00")
```

...returns TRUE

```
TimeBetween("6/10/2006 2:00:00 pm","06/3/06 9:00:00 am","2006-06-12 5:00:00 pm")
```

...returns TRUE
(Note: This example also shows the variety of date formats accepted)

```
TimeBetween("11:30:00 pm","10:00:00 pm","6:00:00 am")
```

...returns TRUE
(Note: Start time is later than end time, so it refers to 10pm the day before)

Logic Functions

If(*Condition, TrueReturn, FalseReturn*)

This function evaluates the expression condition, and returns the value of trueReturn or falseReturn depending on the boolean value of condition.

Parameters

Condition

A boolean expression to evaluate.

TrueReturn

The value (may be another expression) to return if condition is true.

FalseReturn

The value to return if condition is false.

Example:

```
if(1, "Yes", "No")
```

...would return "Yes"

```
if(0, "Yes", "No")
```

...would return "No"

```
if({Root Container.CheckBox.selected}, "Selected", "Not Selected")
```

...would return the a description of the state of the checkbox

Switch(*value, case1, case2,...,caseN, return1, return2,...,returnN, returnDefault*)

This function acts like the switch statement in C-like programming languages. It takes the return value of the value expression, and compares it to each of the case1 through caseN expressions. If value is equal to caseX, then switch returns valueX. If value is not equal to any of the case1..N, then returnDefault is returned. For Example:

```
switch(
15, // value
1,  // case 1
24, // case 2
15, // case 3
44, // return 1
45, // return 2
46, // return 3
-1) // default
```

...would return 46 because the value (15) matched case 3, so the third return (46) was returned, while:

```
switch(
35, // value
50,  // case 1
51, // case 2
60, // return 1
60, // return 2
100) // default
```

...would return 100 because the value (35) didn't match any of the cases. Also note that the return values need not be the same type as the case values.

```
switch(
2, // value
0,  // case 1
1, // case 2
2, // case 3
"Off", // return 1
"Running", // return 2
"Fault", // return 3
"Unknown State") // default
```

...would return "Fault".

BinEnum(*boolean1, boolean2, ...*)

This function, whose name stands for "binary enumeration", takes a list of booleans, and returns the index of the first parameter that evaluates to `true`. For example:

```
binEnum(false,true,false)
```

...returns 2 (the index of the true parameter)

```
binEnum(0,false,15,0,23)
```

...returns 3 (the index of the 15 - the first true parameter)

This function is basically a shortcut for a large switch - if you have a bunch of boolean conditions, and you need to turn them into an integer.

BinEnc(*boolean1, boolean2, ...*)

This function, whose name stands for "binary encoder", takes a list of booleans, and treats them like the bits in a binary number. It returns an integer representing the decimal value of the number. The digits go from least significant to most significant.

```
binEnc(0,0,1,0)
```

...returns 4 (the value of `0100`)

```
binEnc(true,0,1,1,0)
```

...returns 13 (the value of `01101`)

GetBit(*Value, Pos*)

This function returns the value of a bit at the given position in the value. For example:

```
GetBit(5,0)
```

...returns 1

```
GetBit(5,1)
```

...returns 0

```
GetBit(5,2)
```

...returns 1

Coalesce(*value1, value2, ...*)

Returns the first value that is not NULL.

Math Functions

The math functions are all very similar, except for round and logx. They all take 1 expression as a parameter, which must be a number.

The functions are:

- cos (Cosine of an angle in radians)
- sin (Sine of an angle in radians)
- abs (Absolute value of a number)
- acos (Arc Cosine)
- asin (Arc Sine)
- tan (Tangent of an angle in radians)
- atan (Arc Tangent)
- ceil (Ceiling - returns the smallest integer value that is not less than the argument)
- floor (Floor - returns the largest integer value that is not greater than the argument)
- exp (Exponent - returns *e* raised to the power of the argument)
- log (Performs a natural log (base *e*) of the argument)
- sqrt (Square Root - returns the square root of the argument)
- todegrees (Converts the argument from radians to degrees)
- toradians (Converts the argument from degrees to radians)

Round(*value, decimals*)
Rounds to the specified number of decimals.

LogX(*value, base*)
Returns the log of the specified value, using the specified base.

There are also 2 functions that return constants:

PI()
Returns PI.

E()
Returns *e*.

String Functions

Concat(*value1, value2,..., valueN*)

Concatenates all of the parameters into one string. The plus `(+)` operator does the same.

Len(*value*)

Returns the number of characters in the String `value`.

Repeat(*string*, *count*)

Returns the given string, repeated some number of times.

```
repeat("hello", 2)
```
...returns "hellohello"
```
repeat("hello", 0)
```
...returns ""

Substring(*String, StartIndex, [EndIndex]*)

Substring will return the portion of the string from the *StartIndex* to the *EndIndex*, or end of the string if *EndIndex* is not specified. All indexes start at 0, so in the string "Test", "s" is at index 2.

Examples:
```
Substring("unhappy", 2)
```
...returns "happy"
```
Substring("Harbison", 3)
```
...returns "bison"
```
Substring("emptiness", 9)
```
...returns ""
```
Substring("hamburger", 4, 8)
```
...returns "urge"
```
Substring("smiles", 1, 5)
```
...returns "mile"

IndexOf(*String, Subvalue, [Start]*)

Returns the index of the first instance of *SubValue* inside of *Value*, optionally starting at *Start*. Like substring, all indexes start at 0, so in the string "Test", "s" is at index 2.
This function will return -1 if the subvalue is not found.
Examples:
```
IndexOf("Result: Good", ":")
```
...returns 6
```
IndexOf("Test Test", "Test")
```
...returns 0
```
IndexOf("Test Test", "Test", 1)
```
...returns 5
```
IndexOf("failed", "success")
```
...returns -1

LastIndexOf(*String, Subvalue*)

Returns the index of the last occurance of *SubValue* inside of *Value*.

Replace(*String, OldValue, NewValue*)

Replaces all instances of *OldValue* in *String* with *NewValue*

Examples:
```
Replace("Status: Good", "Good","Bad")
```
...returns "Status: Bad"
```
Replace("Please wait...", ".","!")
```
...returns "Please wait!!!"

ToUpper(*value*)
Returns `value` in upper case letters.

ToLower(*value*)
Returns `value` in lower case letters.

Type Functions

The Expression Language is type-safe. This means that expressions such as "hello" / 15 are invalid, because division does not work on Strings. Sometimes, however, it may be useful to try and coerce a value of one type to another. For example, suppose you have a text box that a user will type a number into. The text box's bound property is a String, not a number, so you couldn't use this value in a mathematical expression, or bind an integer property to it. With the toInt casting function, you can try to interpret the string as an integer.

All cast functions take 1 parameter, the value to convert.

ToBoolean(*value*)

ToDate(*value*)

ToDouble(*value*)

ToFloat(*value*)

ToInt(*value*)

ToLong(*value*)

ToString(*value*)

Variable Functions

A word about variables: These basic functions provide a rudimentary way for groups to share information between themselves, as normally groups are completely isolated. A "variable" is essentially nothing more than a named slot in a value table. When StoreVariable is called, a slot will be created if not already present. It will only be removed on a call to DeleteVariable or ClearVariables.
Generally speaking, it is rare that variables are required, and there is usually a better way to accomplish a goal. Variables tend to make groups difficult to maintain, as it can be confusing as to where the data is coming from. Therefore, care should be taken when working with these functions.

StoreVariable(*Name, Value*)
Stores the value under the given name. Variable will be created if not already defined.

GetVariable(*Name, DefaultValue*)
Retrieves the specified variable, or will return the default value if the name is not defined.

IsVariableDefined(*Name*)
Returns a boolean TRUE or FALSE as to whether the given name is defined.

DeleteVariable(*Name*)
Un-defines a variable that has been defined with StoreVariable. If the variable is not defined, does nothing.

ClearVariables()
Removes all defined variables from memory.

Scripting API

Introduction

The FactorySQL Scripting API provides a method for developers to link their own custom libraries into FactorySQL. Developers can create class libraries based on the Microsoft .Net framework and expose functions that may be called from Action Items. Additionally, components may be developed that run behind-the-scenes, and do not have a public interface.

Fundamentals

Using the API is very simple, you simply create a .Net project with a reference to ScriptingAPI dll (ScriptingAPI.dll, found in the FactorySQL install directory), and then mark which classes and functions you'd like to expose with *Attributes*.
Once you've built your dll, simply drop it into the *scripting\plugins\* directory in the FactorySQL install directory, restart FactorySQL, and your function will now be accessible (and visible from the right-click menu in the action item editor).

The following table outlines the basic attributes:

Attribute	Scope	Function	Parameters (optional)
ScriptClass	Class	Denotes that the class is visible to FactorySQL. Also allows you to specify when the class is instantiated.	**InstantiationStyle** - See below for a description of the different instantiation styles.
ScriptFunction	Function	Marks the function visible to FactorySQL. Allows you to set various parameters.	**FunctionName** - An alias for the function. Necessary if you're trying to use a reserved keyword for the function name. **VariableArguments** - Boolean specifying whether the arguments are variable (such as when using the *optional* keyword in VB.Net). **Category** - The sub-menu that the function will show up under in the frontend. **Description** - A description of the function.
ScriptClassStartup	Function	Will cause the function to be called when the class is first used.	None
ScriptClassTeardown	Function	Will cause the function to be called when the class is done being used.	None

The *InstantiationStyle* property of the ScriptClass attribute allows you to specify when instances are created. This can be very important for classes that need to persist information between calls.
The following table outlines the options for instantiation:

Instantiation Style Enumeration	
Static	The class is not instantiated. All functions marked as ScriptFunctions must also be static.
OncePerServer	The class is instantiated one time, and is shared by all items that call its functions. **Caution:** Because FactorySQL is multi-threaded, any exposed function in a class that is instantiated in this way must be thread-safe.
OncePerGroup	Each group will have its own object that will be shared by all items in the group.
OncePerItem	Each item that uses the class will have its own object.
OncePerCall	The class will be instantiated each time one of its functions is called.

Basic Example

This C# example shows an extremely basic plugin that exposes a single function. The function receives an integer, and returns its inverse:

```
using System;
using FactorySQL.Scripting;
namespace PluginExample
```

```
{
   [ScriptClass()]
   public class PluginExample
   {
      public PluginExample()
      {
      }

      [ScriptFunction()]
      public float Invert(float Value)
      {
         return 1/Value;
      }
   }
}
```

Enhanced Example

The following is a piece of code from the Base Function Library, included with FactorySQL. It shows the usage of most of the attribute properties, including optional parameters.

```
using System;
using FactorySQL.Scripting;

namespace FactorySQL.Scripting
{
   /// The FunctionLibrary class is the static class that provides all of our functions.
   /// There is no reason to create an instance of it.
   [ScriptClass(InstantiationStyle=InstantiationStyles.Static)]
   public class FunctionLibrary
   {
      public FunctionLibrary()
      {
      }

      //String Functions
      [ScriptFunction(FunctionName="Concat",Description="Concatenates all parameters together.",
         Category="String Functions",VariableArguments=true)]
      public static string Concat(params object[] values)
      {
         System.Text.StringBuilder sb = new System.Text.StringBuilder();

         for(int i=0; i<values.Length; i++)
            if(values[i]!=null)
               sb.Append(values[i].ToString());

         return sb.ToString();
      }

      [ScriptFunction(FunctionName="Len",Description="Returns the length of the string.",
         Category="String Functions")]
      public static int Len(string Value)
      {
         return Value.Length;
      }
   }
}
```

How To Get Help

There are many ways to get help from Inductive Automation! We recommend trying the following order:

Support Forum

The IA support forum is an excellent place to ask questions, find answers, and to learn something new! The forum can be found at:
http://forum.inductiveautomation.com
The forum is regularly monitored by Inductive Automation staff, and all posts should receive a response within 24 hours. Forum support is always free.

Email Us

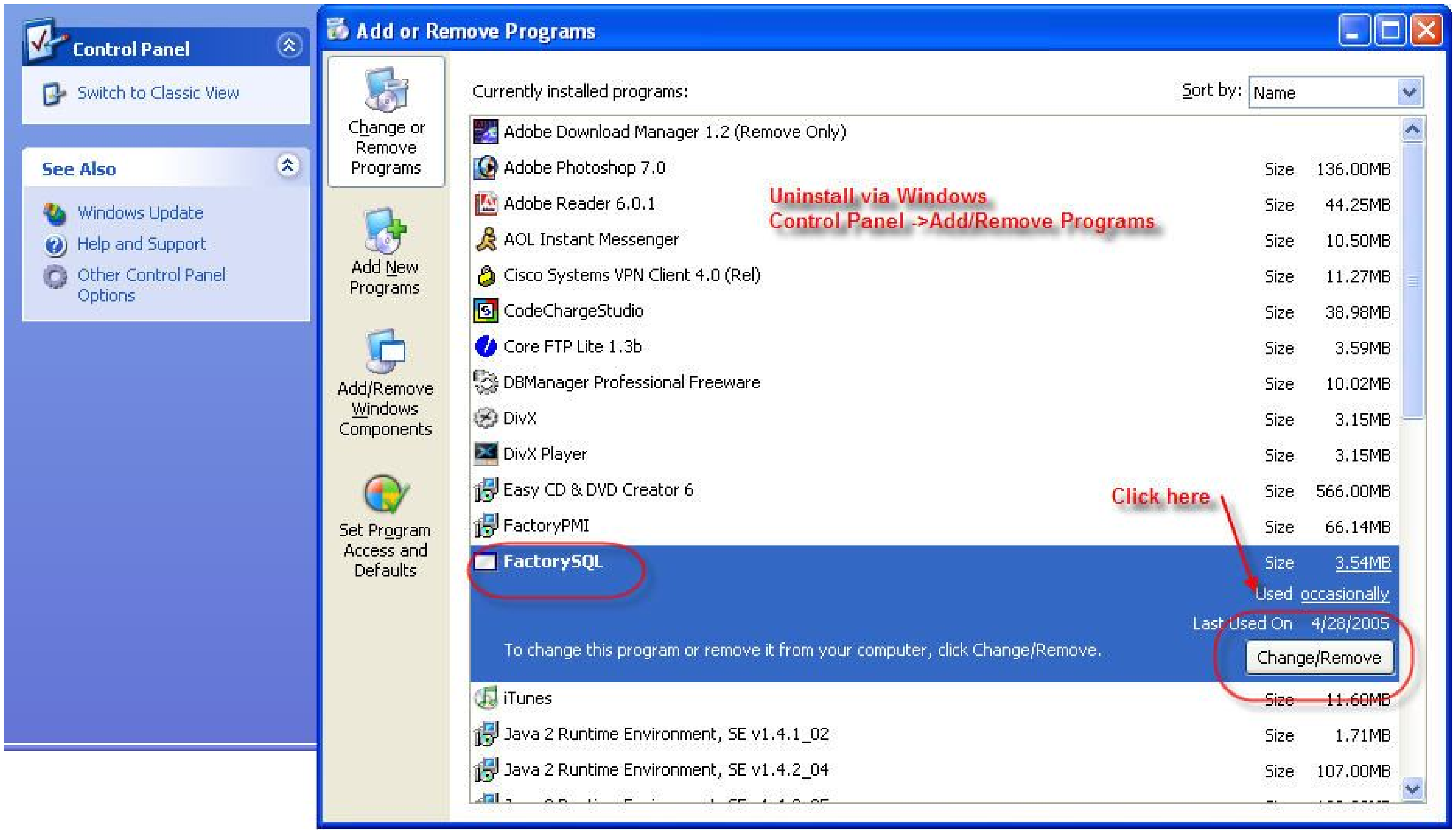
Feel free to email us at support@inductiveautomation.com. Email support is always free, and we are committed to a fast response time. All inquiries should receive a response within 36 hours.

Call Us

You may call us directly at 1-800-266-7798. A support contract may be required for phone support.

Uninstalling FactorySQL

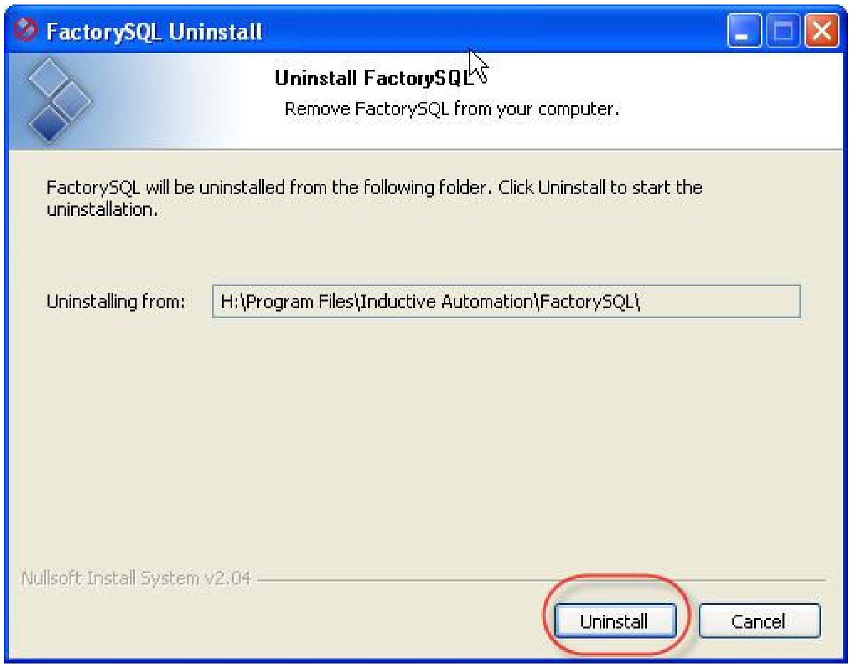
To uninstall FactorySQL simply follow the standard procedure to uninstall a program in Windows. Go to the Control Panel. Open "Add or Remove Programs". Select FactorySQL. Click "Change/Remove". Follow uninstallation instructions.


Control Panel
Switch to Classic View
See Also
Windows Update
Help and Support
Other Control Panel Options
Add or Remove Programs
Change or Remove Programs
Add New Programs
Add/Remove Windows Components
Set Program Access and Defaults
Sort by: Name
Uninstall via Windows Control Panel ->Add/Remove Programs
Click here
Change/Remove

Currently installed programs:

Adobe Download Manager 1.2 (Remove Only)		
Adobe Photoshop 7.0	Size	136.00MB
Adobe Reader 6.0.1	Size	44.25MB
AOL Instant Messenger	Size	10.50MB
Cisco Systems VPN Client 4.0 (Rel)	Size	11.27MB
CodeChargeStudio	Size	38.98MB
Core FTP Lite 1.3b	Size	3.59MB
DBManager Professional Freeware	Size	10.02MB
DivX	Size	3.15MB
DivX Player	Size	3.15MB
Easy CD & DVD Creator 6	Size	566.00MB
FactoryPMI	Size	66.14MB
FactorySQL	Size	3.54MB
	Used	occasionally
	Last Used On	4/28/2005
iTunes	Size	11.60MB
Java 2 Runtime Environment, SE v1.4.1_02	Size	1.71MB
Java 2 Runtime Environment, SE v1.4.2_04	Size	107.00MB

To change this program or remove it from your computer, click Change/Remove.



Uninstall FactorySQL

Remove FactorySQL from your computer.

FactorySQL will be uninstalled from the following folder. Click Uninstall to start the uninstallation.

Uninstalling from: H:\Program Files\Inductive Automation\FactorySQL\

Register FactorySQL

The FactorySQL configuration client is provided free of charge. It can connect to any FactorySQL service application, locally or remotely. The FactorySQL service application will run for two hours at a time without registration. After two hours all logging groups & SQLTags will stop running. To restart, you must reconnect to the FactorySQL service application and choose to "Restart Demo Period".

After registering your FactorySQL service application the two hour runtime limitation will be removed. This liberal registration strategy allows you to develop complete SQL logging applications without any fee, and lets you be sure that FactorySQL is right for your system.

You can purchase FactorySQL online at: http://www.inductiveautomation.com

Or by calling: (800)266-7798

Purchase Benefits:

- Use the software without any limitation
- 3 months of free phone support.
- Guaranteed 36-hour turn around on e-mail support.
- Free upgrades for the life of the major version.

Item Substitution is a powerful feature of FactorySQL that allows the user to "plug in" an OPC value into another field, such as the Where Clause, and Action Item statement, or an analog alarm point.

In general, to insert a value into a field, press **Control+Space** to bring up the menu. You may also right click a valid field and choose "Reference Item".

In the WHERE Clause

A common application is to use a PLC register to select a database record to use for the group. This is done by using a **Custom WHERE Clause** as described in the Action tab settings and substituting in an OPC value. This is very useful for recipe systems or other situations where you want to use the database as a repository of values that the PLC can choose to load.

Within Action Items

OPC path values can also be plugged directly into Action Items.

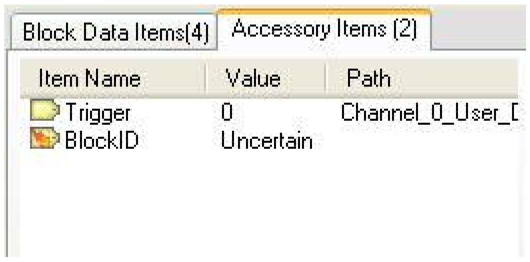
This is typically used to scale/perform calculations on OPC values, or to perform logical operations for triggering or alerting. The result can be stored in the database or written back to the PLC.

Within Alert Items

You can use Item Substitution to set the trigger point for analog alert items. This allows you to have dynamic alert levels that can be changed from the PLC. When using Item Substitution, you can use the Up and Down arrows to position the item exactly as you'd like. For more information on alerting, see Alert Configuration.

Block Data Accessory Items

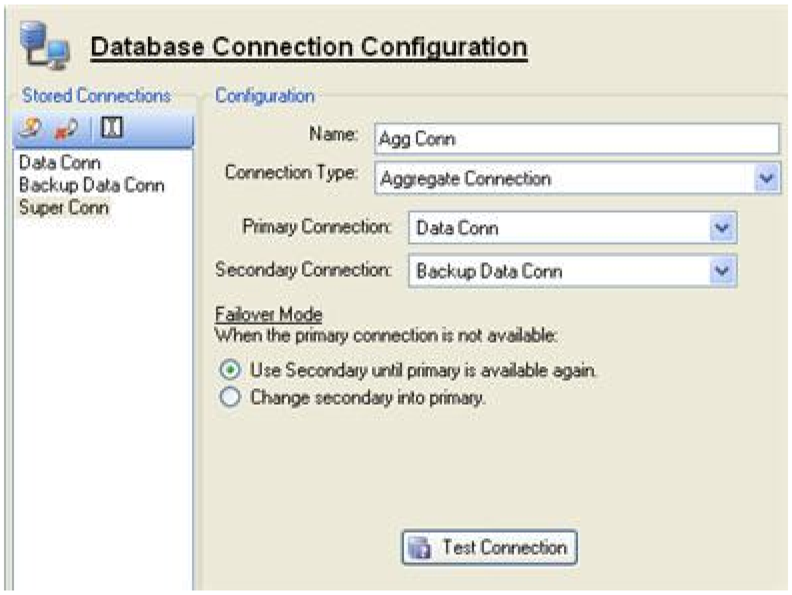
Accessory Items are simply **OPC Items** and **Action Items** that can be used by block data groups for tasks such as **triggering** and **dynamic where clauses**. The only difference is that accessory items cannot currently be written to the database.



Item Name	Value	Path
Trigger	0	Channel_0_User_[
BlockID	Uncertain	

Aggregate Connections

The **Aggregate Connection Type** allows you to specify 2 data connections, a primary and secondary. When the primary is not available, it will fail over to the secondary connection. The **Failover Mode** determines what happens when the primary is available again. Multiple aggregate connections can be chained together to create a longer list of failover databases.



Data Conn
Backup Data Conn
Super Conn

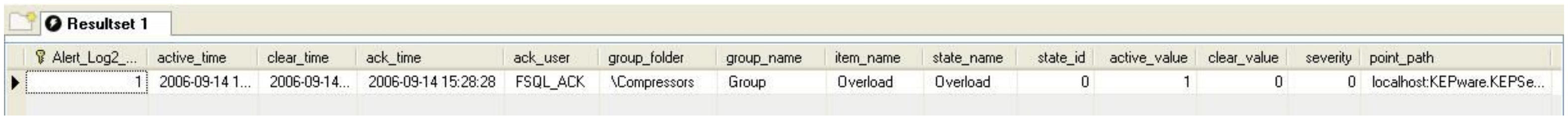
Name: Agg Conn
Connection Type: Aggregate Connection
Primary Connection: Data Conn
Secondary Connection: Backup Data Conn

Failover Mode
When the primary connection is not available:
[✓ Use Secondary until primary is available again.]
Change secondary into primary.

Failover Mode: Determines what happens when the primary connection is available again. The first (and default) option is to switch back to the primary connection. Optionally, you can choose to continue using the current connection, in essence turning the former primary connection into a new secondary.

Standard Alert History Table

The **Standard Alert History** table consists of records that represent the life cycle of an item's alerts. That is, each row has information about an particular alert's active, clear, and acknowledged events.



Alert_Log2_...	active_time	clear_time	ack_time	ack_user	group_folder	group_name	item_name	state_name	state_id	active_value	clear_value	severity	point_path
1	2006-09-14 1...	2006-09-14...	2006-09-14 15:28:28	FSQL_ACK	\Compressors	Group	Overload	Overload	0	1	0	0	localhost:KEPware.KEPSe...

Specify the alert history table **here** under **log table**.

Columns

Active Time: The date and time that the alert became active.

Clear Time: The date and time that the alarm was cleared (became inactive).

Ack Time: The date and time that the alert was acknowledged. The value of this field depends on the Ack Mode setting.

Ack User: The user that acknowledged the alarm. If ACK Mode is set to unused or automatic, this value will be "FSQL_ACK".

Group Folder: Indicates the Group folder where the alarm item resides.

Group Name: Indicates the Group where the alarm item resides.

Item Name: Indicates the item that changed alarm state.

State Name: The user defined text string to describe the alarm state. For **digital alerts**, this describes the alarm. For **analog alerts**, it indicates which alarm state the item went into.
Previous Value: The last value that FactorySQL read of the item before it changed alarm state.

State ID: The id of the alert state. For digital, this will be 0 or 1. For analog, this can be positive or negative, corresponding to how far away the alert state is from the base state.

Active Value: The value of the item that caused the item to become active.

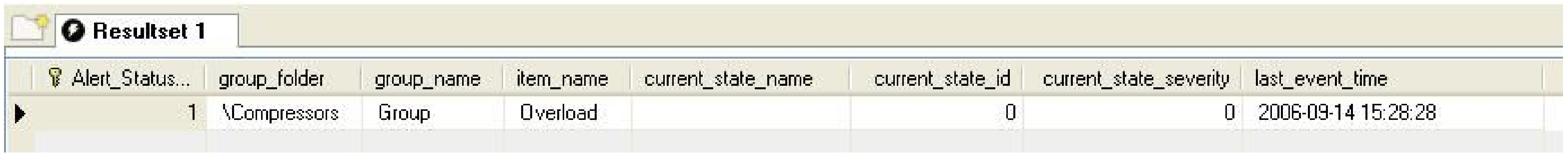
Clear Value: The value of the item that caused the alert to clear.

Severity: A numeric representation of the severity states from 1 (low) to 5 (high).

Point Path: The full OPC path to the item.

Standard Alert Status Table

The **Standard Alert Status** table displays alert status of items that have been configured for alerting. It includes which items are in alarm, severity, etc. It is useful for displaying the realtime status of alert items in a web or HMI system.


Resultset 1

Alert_Status...	group_folder	group_name	item_name	current_state_name	current_state_id	current_state_severity	last_event_time
1	\Compressors	Group	Overload		0	0	2006-09-14 15:28:28

Specify the alert stus table **here** under **Alert Status Table**.

Columns

Group Folder: Indicates the Group folder where the alert item resides.

Group Name: Indicates the Group where the alert item resides.

Item Name: Indicates the item that changed alert state.

Current State Name: The user defined text string to describe the alert state. For **digital alerts**, this describes the alert. For **analog alert**, it indicates which alert state that the item went into.

Current State Id: Provides the ID of the current alert state. Alert states are automatically assigned ID numbers, which can be used to identify them and distinguish between them. For digital, it will either be a 0 (indicating no alert), or 1 (indicating in-alert). For analog, 0 indicates that the value is within the base range, and the item is not in alert. Alert states are numbered sequentially, going up for rising-edge alarms, and down (into negative numbers) for falling edge alert states.

Current State Severity: A numeric representation of the severity states from 1 (low) to 5 (high).

Last Event Time: Provides the date and time that the alert last changed state.

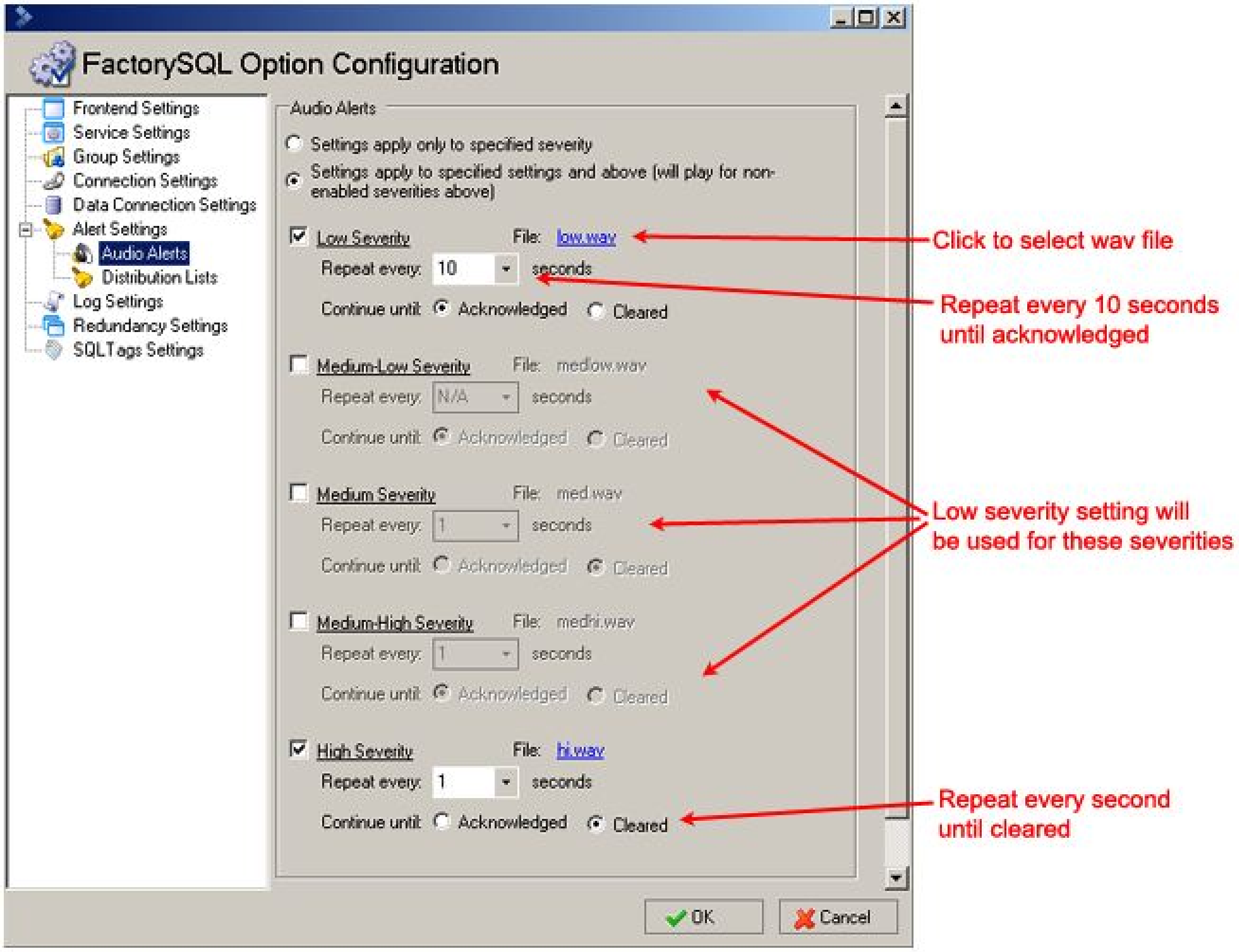
Audio Alerts

Audio Alerts allow you configure sounds to be played when there is an active alert. A different sound my be specified for each severity level, and can be configured to play once or to repeat at a specified interval until all alarms of the given severity are acknowledged or cleared.

How to configure

Audio alerts are configured by selecting the "Audio Alerts" sub-item in the **Alert Settings** screen.

The configuration screen looks as follows:



Audio Alerts

Settings apply only to specified severity [✓ Settings apply to specified settings and above (will play for non-enabled severities above)]

[✓] Low Severity File: low.wav
Repeat every: 10 seconds
Continue until: [✓ Acknowledged] Cleared

[ ] Medium-Low Severity File: medlow.wav
Repeat every: N/A seconds
Continue until: [✓ Acknowledged] Cleared

[ ] Medium Severity File: med.wav
Repeat every: 1 seconds
Continue until: Acknowledged [✓ Cleared]

[ ] Medium-High Severity File: medhi.wav
Repeat every: 1 seconds
Continue until: [✓ Acknowledged] Cleared

[✓] High Severity File: hi.wav
Repeat every: 1 seconds
Continue until: Acknowledged [✓ Cleared]

General Settings

The only general setting specifies how the severity settings are applied. There are two choices: apply settings only to the specified alert severity, or apply them to all greater severities that do not have their own setting. For example, by choosing to apply the setting to all severities (the second option), you only need to specify a sound for the lowest severity in order to hear it for any alert that occurs.

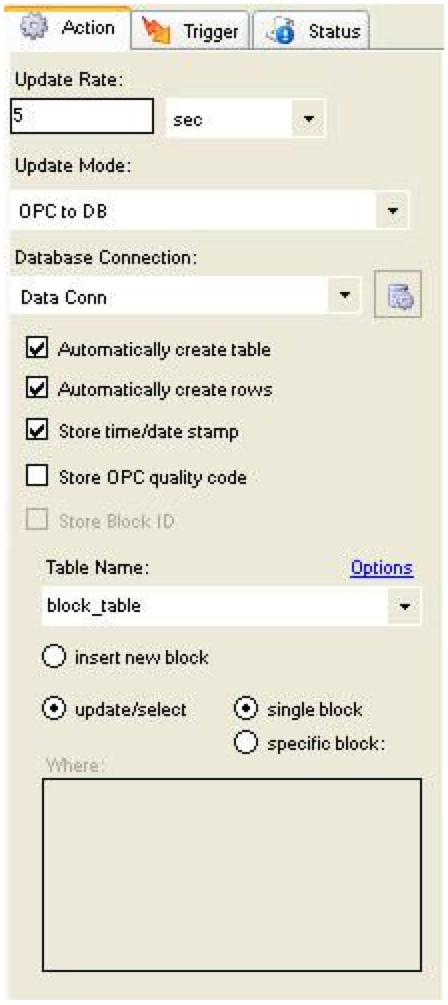
Severity Settings

There are five sets of identical settings, one for each severity.
Enabled: Enables audio alerts for the specified severity.
File: The audio file to play.
Repeat every X seconds: The time period at which to loop the audio. Audio will continue to loop until all alerts of the specified severity (or higher, depending on the general setting) are acknowledged or cleared (depending on the following setting). If set to 0, the sound will only be played once.

Continue until...: Specifies when to stop looping the audio, if applicable.

Block Data Group Options

Most of the options on the block data group action tab perform the same function as they do for standard groups. There are a few settings, however, that are unique to block groups.



Update Rate:
5 sec

Update Mode:
OPC to DB

Database Connection:
Data Conn

[✓] Automatically create table
[✓] Automatically create rows
[✓] Store time/date stamp
[ ] Store OPC quality code
[ ] Store Block ID

Table Name: Options
block_table

( ) insert new block
(●) update/select
(●) single block
( ) specific block:

Where:

Block Data Group Options

Automatically create rows: Automatically add the rows necessary to run the group. If false, FactorySQL will ask to create rows when starting the group.

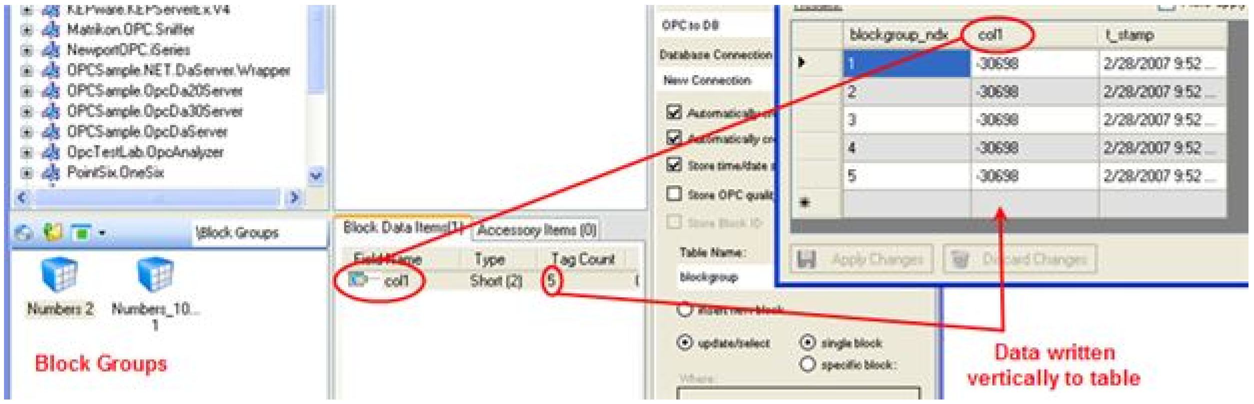
Store Block ID: Only available when inserting block, this option stores a sequential identifier that is unique for each data block.

Insert new block (OPC->DB only): FactorySQL will insert the block into the database every time the group runs. Typically, this option is used to log data at a regular interval. When combined with trigger options, the group can be set to either log at certain times, or be used to create "snapshots" on some condition.

Update/select: Selects a specific block to read and write from. Either the first block (**single block**), or a block selected by a specific SQL where clause. You may also use the "item substitution" feature for indirect addressing. This allows for things like having a PLC register point to a recipe in a batching system, then using the database to maintain many different recipes.

Block Data Groups

Block Data Groups are very similar to standard groups, except they are designed to write to multiple rows of a table at one time, creating a very efficient "data block".



Features

Block Data Groups support most standard group features, and some extras:

- **Full bi-directional support**: Very useful for creating efficient recipe systems.
- **Column definable bi-directional mode**: Mix and match bi-directional columns to create complete status and control systems in one group.
- **Multiple segments per column**: Allows you to stack data from different memory block or OPC servers in the same column.
- **Efficient**: By blocking the data, throughput is greatly increased over using multiple standard groups.

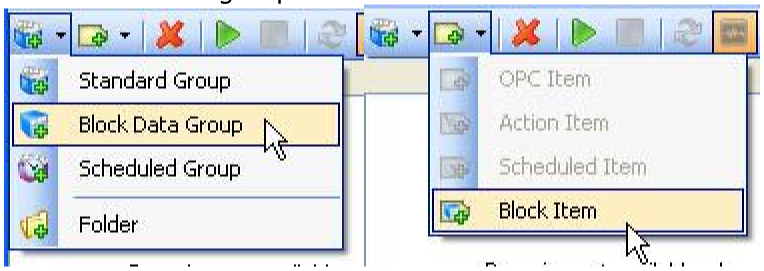
Items

Block data groups contain 2 distinctive groups of items:

- **Block Data Items**: These items represent the columns (and data) that the group will read & write. Each block item defines its column name, data type, and mode, and contain one or more **Segments**, which define the actual OPC addresses of the data.
- **Accessory Items**: These are standard **OPC Items** and **Action Items**, that can be used by the block data group as a trigger or in the where clause. They are not stored to the database.

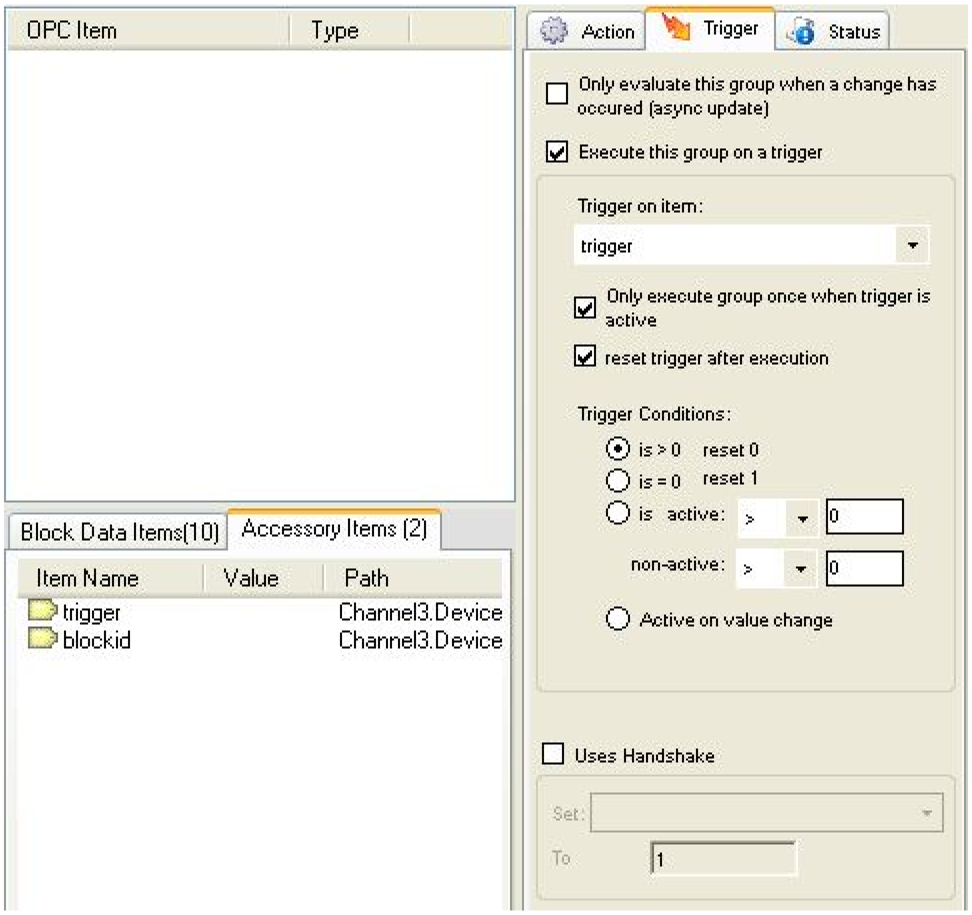
Creating Block Data Groups

New block data groups can be created from the toolbar or by right clicking on the project pane:



Triggering

Block group trigger works exactly the same as Standard Group Triggering. The only nuance is that the trigger item must be an **Accessory Tag**. If you would like to use a tag that is part of your datablock as the trigger, you'll have to add a duplicate into the accessory item list. There is very little overhead in having duplicate tags in block groups, as FactorySQL treats these tags as 1 point.



Item Name	Value	Path
trigger		Channel3.Device
blockid		Channel3.Device

[ ] Only evaluate this group when a change has occured (async update)

[✓] Execute this group on a trigger

Trigger on item: trigger

[✓] Only execute group once when trigger is active

[✓] reset trigger after execution

Trigger Conditions:

[✓ is > 0 reset 0]
is = 0 reset 1
is active: > 0
non-active: > 0
Active on value change

[ ] Uses Handshake

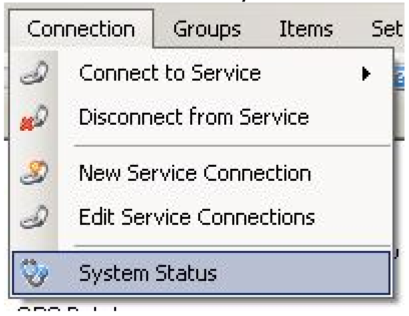
Set:
To 1

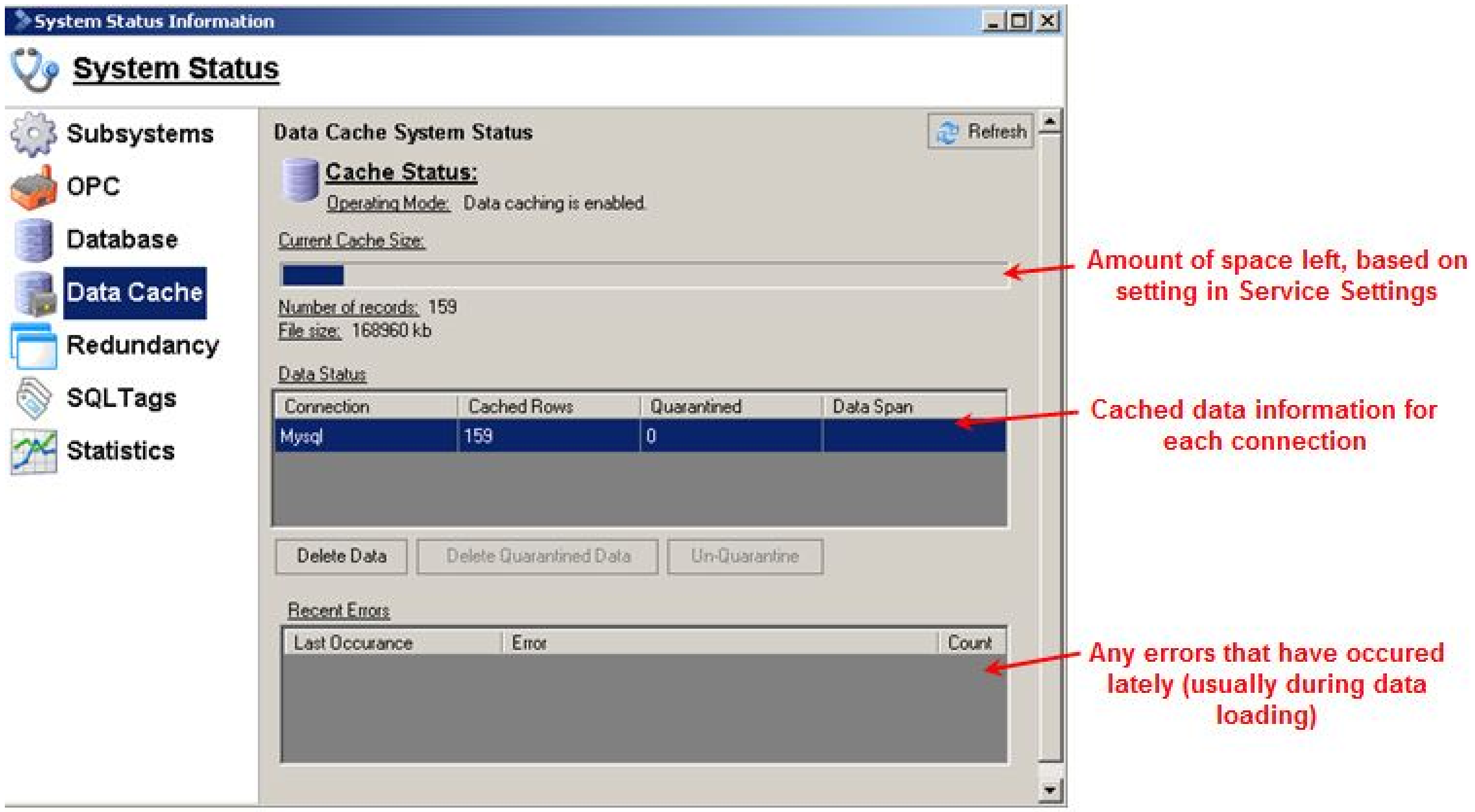
Data Cache

The **data cache** provides local storage for historical data while the target database connection is unavailable. When enabled through Service Settings (it is enabled by default), historical queries generated by groups and alerting will be redirected to the cache should the database connection go down. When it is available again the data will be loaded into the target database. Any timestamp fields in the queries (such as the data for the t_stamp column) will be adjusted and accurate for the target database, so the data will appear as if nothing occured (assuming the data is sorted by timestamp).

Data Cache Status

The data cache system status can be found under the System Status dialog:





System Status Information
System Status
Subsystems
OPC
Database
Data Cache
Redundancy
SQLTags
Statistics
Data Cache System Status
Refresh
Cache Status:
Current Cache Size:
Delete Data
Delete Quarantined Data
Un-Quarantine
Amount of space left, based on setting in Service Settings
Cached data information for each connection
Any errors that have occured lately (usually during data loading)

Operating Mode: Data caching is enabled.

Number of records: 159
File size: 168960 kb

Data Status

Connection	Cached Rows	Quarantined	Data Span
Mysql	159	0	

Recent Errors

Last Occurance	Error	Count

The data cache status page provides important information about data in the cache. You can see how much data is stored, and how much data has been quarantined. From this screen you can choose to delete data and un-quarantine it, as well as view any errors that may be causing data to become quarantined.

Quarantined Data

Data becomes quarantined when the data cache system is unable to load it after several attempts. These queries usually would cause an executing group to error out, but were stored to the cache. Common causes include references to tables or columns that don't exist, or data issues- such as trying to write a NULL to a non-null column.

Once quarantined, FactorySQL won't attempt to load the data again. By going to the status screen, you can view the errors that occured, and either delete the quarantined data or try to fix the errors and un-quarantine it. When you un-quarantine the data the system will attempt to load it in the next load cycle (which may take up to a minute to occur).

Database Connections

As the name implies, the primary purpose of FactorySQL is to interact with databases. In order to do this, a connection must be configured. There are 2 main types of connections: **Native** and **DSN**. Additionally, **Aggregate Connections** allow you to create a fail-over connection wrapper.

Native Connections

Native Connections use a native Microsoft .NET driver to connect to the database. The versions bundled with FactorySQL or available from Inductive Automation tend to be better tested, faster, and optimized for use with FactorySQL, making them your best first bet.

For an example of setting up a native connection, see the Quick Start.

Driver Type: This specifies which native driver to use. It will be specific to the database you are using.

Translator (advanced): FactorySQL uses translation files to mediate differences in SQL syntax between brands of servers. In almost every case this value should be left as 'Automatic', but in certain cases it may be necessary to choose a specialized translator.

Host: The address where the db server is located.

Port: The port that the db server runs on.

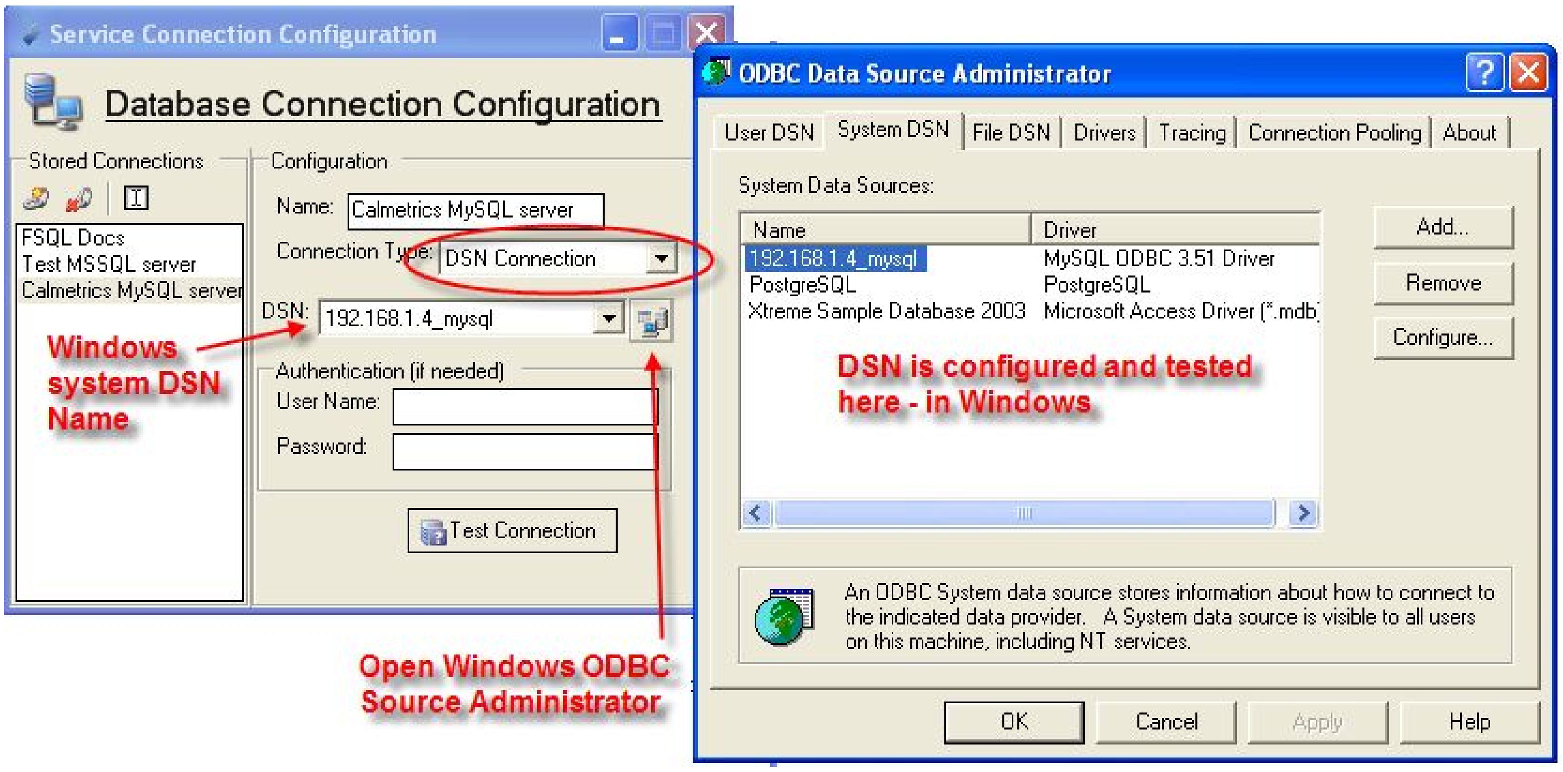
Database: The database to use on the server.

Extra Connection Parameters: Other parameters that will be passed along during connection. These are defined by the database driver, so you will have to consult the manual of the driver you are using for possible options.

Authentication: The username/password to use when connecting to the database.

ODBC Connections

ODBC connections are installed and configured in Windows. Simply install third party ODBC drivers to connect to almost any database.


Service Connection Configuration
Database Connection Configuration
Stored Connections
Configuration
Authentication (if needed)
User Name:
Password:
Test Connection
Windows system DSN Name
Open Windows ODBC Source Administrator
ODBC Data Source Administrator
User DSN
System DSN
File DSN
Drivers
Tracing
Connection Pooling
About
Add...
Remove
Configure...
DSN is configured and tested here - in Windows
OK
Cancel
Apply
Help

FSQL Docs
Test MSSQL server
Calmetrics MySQL server

Name: Calmetrics MySQL server
Connection Type: DSN Connection
DSN: 192.168.1.4_mysql

System Data Sources:

Name	Driver
192.168.1.4_mysql	MySQL ODBC 3.51 Driver
PostgreSQL	PostgreSQL
Xtreme Sample Database 2003	Microsoft Access Driver (*.mdb)

An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users on this machine, including NT services.

Historical Groups

Historical Groups make it quick and easy to log historical OPC data. Configuration is as simple as drag and drop.

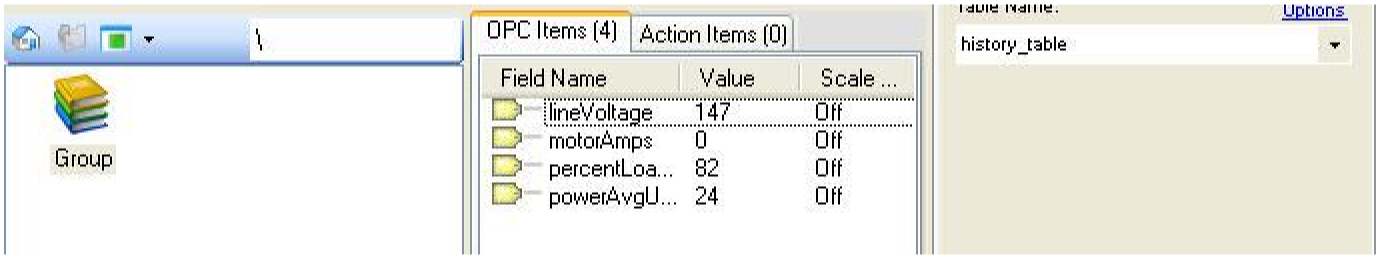


Table Name:

history_table

Field Name	Value	Scale ...
lineVoltage	147	Off
motorAmps	0	Off
percentLoa...	82	Off
powerAvgU...	24	Off

Features

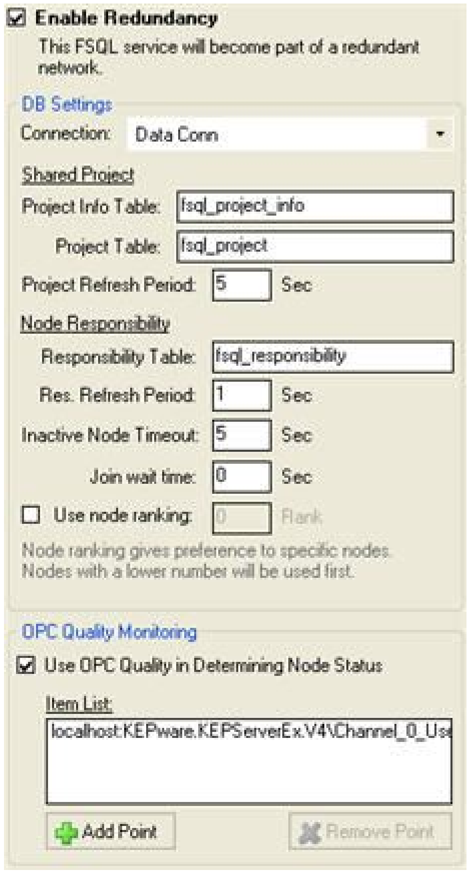
- **Easy configuration**: Drag and drop items, select a table and an update rate, and start logging!
- **Buffered data writing**: Data is written through a buffer, which works hand-in-hand with the data cache to prevent data loss should the database become unavailable.
- **Powerful item modes**: Hourmeter and event meter item modes make it quick and easy to perform downtime and event tracking.
- **Full trigger support**: Utilize triggers to intelligently log data and track events.
- **Action item support**: Create expression-based tags, perform mathmatical functions, run sql queries and more.

Items

- **OPC Items**: Pull in OPC data. Can be mapped to a database field, or used internally for trigger, expressions, etc.
- **Action Items**: Can be expressions or SQL queries, can reference other item values, and can write their value to the database or OPC server.

Redundancy Settings/Configuration

Redundancy system configuration is fairly straightforward, as the default values can be used in most cases. Since all communication occurs through the database, it is important to choose the correct connection. It is also very important that the settings are the same on each node.


☑ Enable Redundancy
This FSQL service will become part of a redundant network.

DB Settings
Connection: Data Conn

Shared Project
Project Info Table: fsql_project_info
Project Table: fsql_project
Project Refresh Period: 5 Sec

Node Responsibility
Responsibility Table: fsql_responsibility
Res. Refresh Period: 1 Sec
Inactive Node Timeout: 5 Sec
Join wait time: 0 Sec
☐ Use node ranking: 0 Rank
Node ranking gives preference to specific nodes. Nodes with a lower number will be used first.

OPC Quality Monitoring
☑ Use OPC Quality in Determining Node Status
Item List:
localhost:KEPware.KEPServerEx.V4\Channel_0_Us

Shared Project

The **shared project** settings affect where FactorySQL finds the redundant project information, and how often it checks for updates.

Project Info Table: The table where general project information is stored.
Project Table:: Where groups are stored.
Project Refresh Period: How often to check for project updates.

Node Responsibility

The **node responsibility** settings specify how FactorySQL determines the rank of nodes- whose master, backup, etc. Pay careful attention when setting them, and make sure they are the same on each node.

Responsibility Table: The table that holds information about the nodes.
Responsibility Refresh Period: How often to update node status and check for responsibility changes.
Inactive Node Timeout: How long to wait before deciding a node is dead. In other words, nodes who have not updated their status within this amount of time will be marked as dead.
Join wait time: The node will maintain a "JOIN_WAIT" status in the responsibility table after startup for this amount of time. This is useful in preventing unnecessary responsibility switches in some cases.
Use Node Ranking/Node Rank: This allows you to give and use a ranking to the node. Lower ranked (numerically) nodes will be prefered as master. This option circumvents the normal node ranking, so it should only be used in cases where you want to make sure a certain machine is master if it's running. Nodes with the same ranking are ordered according to the normal principals.

OPC Quality Monitoring

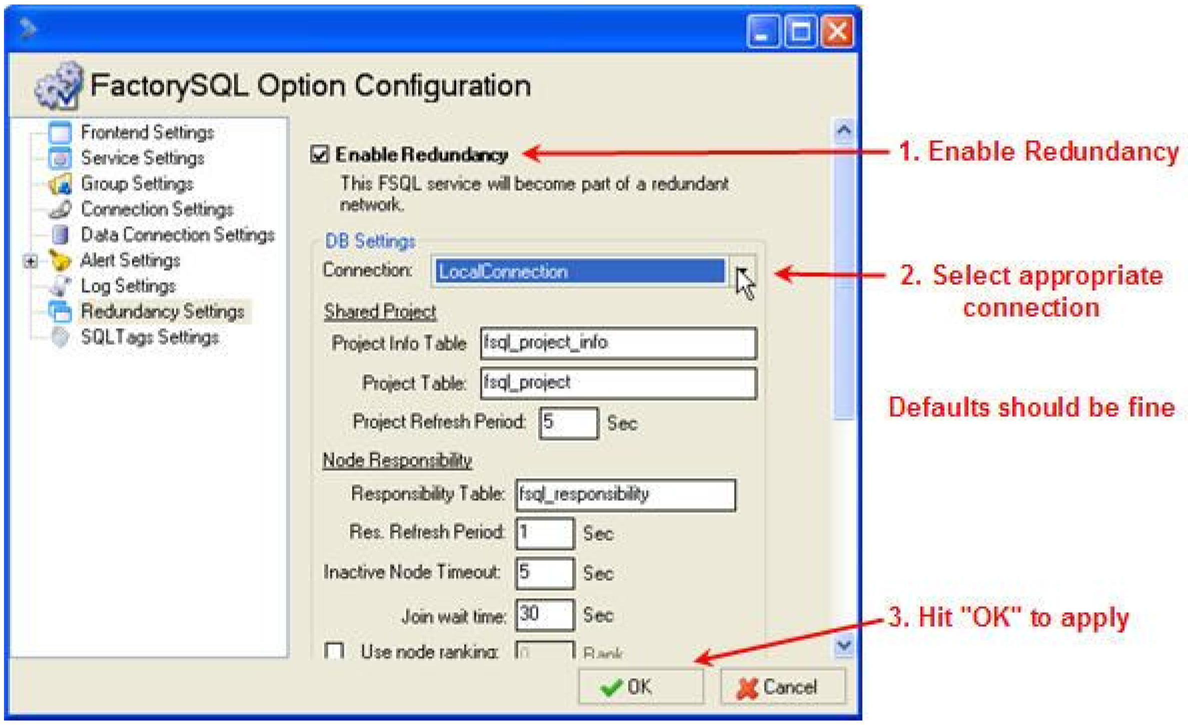
OPC Quality Monitoring allows you to specify tags that will be monitored for quality. This quality will be inserted into the node status table, and used to determine node responsibility. In this way, if one node has valid data when another one does not, it will become master.
To add a point to monitor, simply click the Add button, and enter in the server and path. It is suggested that you add a tag from each PLC/Topic/Device you are reading from.

Redundancy - Getting Started

The redundancy system is designed to be very easy to get started. Essentially, all you need to do is turn it on and populate the shared project.
This quick overview assumes you would like to use your current local project as the redundant project.

Step 1: Enable

Enable redundancy under Settings - Redundancy Settings.



☑ Enable Redundancy
This FSQL service will become part of a redundant network.

DB Settings
Connection: LocalConnection

Shared Project
Project Info Table: fsql_project_info
Project Table: fsql_project
Project Refresh Period: 5 Sec

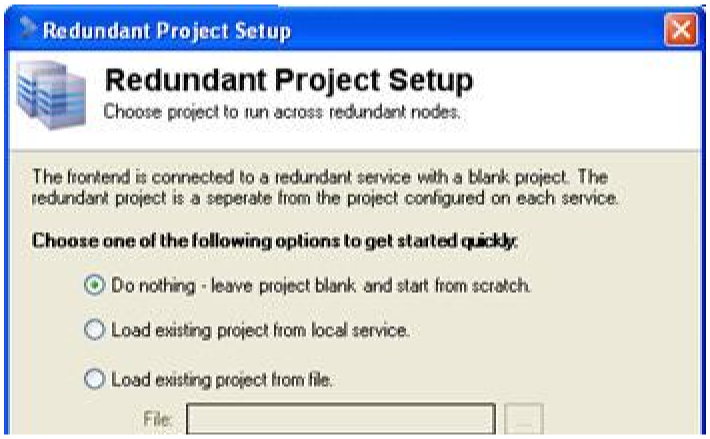
Node Responsibility
Responsibility Table: fsql_responsibility
Res. Refresh Period: 1 Sec
Inactive Node Timeout: 5 Sec
Join wait time: 30 Sec
☐ Use node ranking 0 Rank

Select the data connection that you will be using. Normally the default settings are fine, but if you need to change them you can refer to the Redundancy Settings section for more information.
Hitting "OK" will apply the settings and close the window.

Step 2: Configure Shared Project

The **shared project** is the project that is used by all of the nodes. It is different than the project held by any particular node. That is, when running redundantly, the FactorySQL service uses the shared project instead of its internal project, but the internal project is not lost. By disabling redundancy you can restore the internal project. It is currently not possible to run both the internal and shared project simulataneously.

After enabling redundancy for the first time, the shared project will be blank, and the **Redundant Project Setup Dialog** will appear. This dialog allows you to quickly populate the shared project with an existing project.



Redundant Project Setup
Choose project to run across redundant nodes.

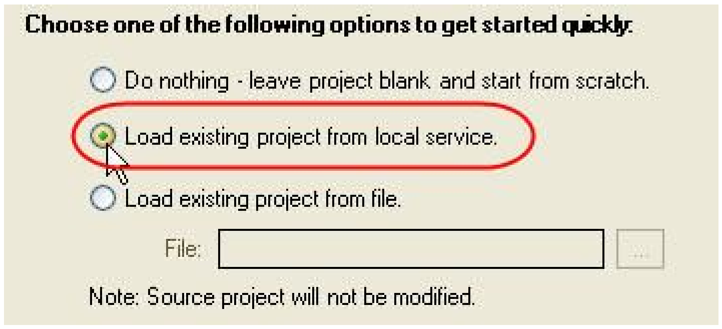
The frontend is connected to a redundant service with a blank project. The redundant project is a seperate from the project configured on each service.

Choose one of the following options to get started quickly:

[✓ Do nothing - leave project blank and start from scratch.]
Load existing project from local service.
Load existing project from file.
File:

Note: Source project will not be modified.

There are 3 options: do nothing, load from service, load from file. The first option, "Do nothing", will not load a project, and you will be able to start with a blank slate. The other 2 will load an existing project.
Normally you'll want to load the local service's project:


Choose one of the following options to get started quickly:

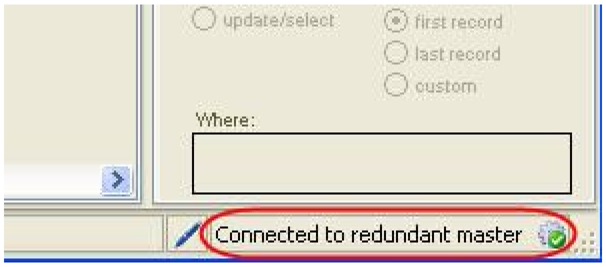
- Do nothing - leave project blank and start from scratch.
- [✓ Load existing project from local service.]
- Load existing project from file.

File:

Note: Source project will not be modified.

Step 3: Done

After the project is loaded, you should see a screen that looks very similar to when you started. In the lower right corner, however, you'll notice that you are now running in a redundant state. Due to your setup and the other nodes on the network, you may or may not be currently connected to the master node.



Connected to redundant master

Redundancy Overview

FactorySQL Redundancy provides a way to boost uptime in critical systems, by running multiple nodes that can execute a shared project if needed. Setting up redundancy is fairly straightforward, but there is a good amount of flexibility in designing and implementing different redundant schemes. A solid understanding of the principals and components of FactorySQL redundancy can help in making these design choice.

Definitions & Terminology

In discussing redundancy, it's useful to have the basic terminology well defined. The following definitions are listed in logical order instead of alphabetical.

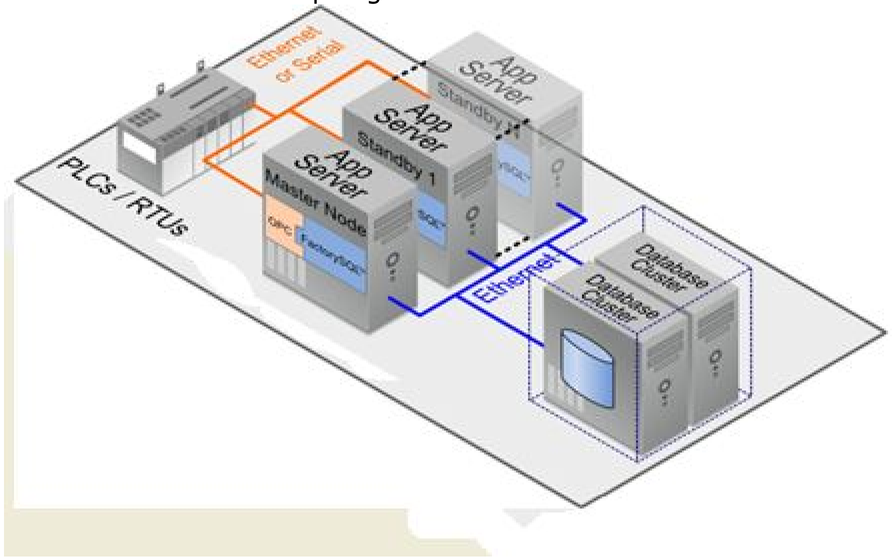
Node: An instance of FactorySQL, running in redundant mode.
Responsibility: What a node should be- master, backup, provisional master, etc.

Shared Project: The project that is read & executed by all redundant nodes. Changes made on any node will be recognized by all of the other nodes.
Redundancy/Shared Database: The database that is used by the nodes to communicated project information, node state, etc.

Master: The instance of FactorySQL that is executing the project.
Provisional Master: An instance of FactorySQL running in a reduced version of master. Caused by an inability to determine with certainty what the responsibilty should be.
Slave/Backup: An instance of FactorySQL that is not currently executing the project, but could start should the master fail.
Warm Backup: A backup node that has all of the OPC items connected and groups started- but is not currently executing them.
First Backup: A backup node that is the next up to run, should the master fail. Usually runs warm.
Node State : The operating state of the node- Running, errored, join_wait, etc.
Project State: The operating state of the project on a specific node: Ready, Warm, Active, etc.

Basic Principals

A basic redundant setup might look as follows:



At the core there is a **shared database** that all of the nodes can see and communicate through. This can be a clustered/replicated/mirrored database for increased reliability. It does not have to be the same database that the project writes to, but that is often the case.
Each node will register itself in the **responsibility table**. This table contains key information about the nodes, and is queried in order to determine who should be master. If a particular node does not update its row within the proper amount of time, it is determined to be dead, and is removed. The next node in line will become master.

When a node is master, it will run the shared project as if it were a normal, local project. If the node is not master, but instead a backup, the project will be loaded but not executed. A **warm backup** will connect all OPC points, but

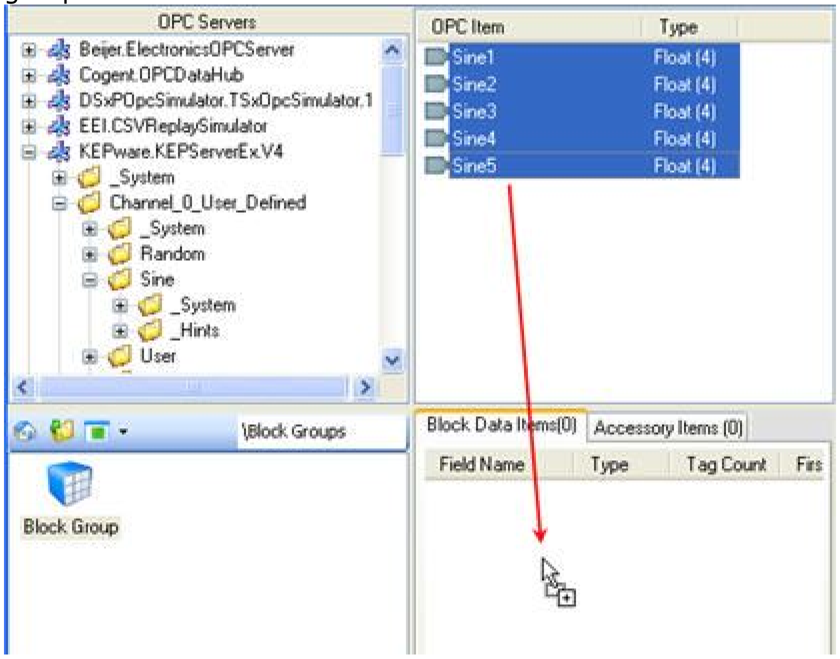
won't execute groups.
If the shared database is not available, there will be no way for the nodes to communicate. In this case, each node will run in **Provisional Master** mode, in which case it will run and cache historical data. When the shared database is available again, the node who becomes master will write the cache to the database, and all other nodes will discard their data.

Block Data Items & Segments

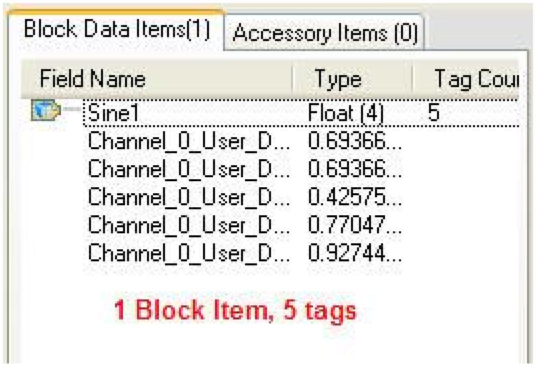
A **Block Data Item** defines a database column to read & write to, a data type, an operation mode, and a set of addresses to use. Addresses are defined as **Segments**. Each block data item contains one or more **Segments**. A segment defines a set of OPC addresses. A block data item can contain any number of segments, for the sake execution they will be compiled down to a single list of addresses. There are currently 2 types of segments: the **Address List** segment, and the **Range** segment.

Creating Block Data Items

Block data items can be created from the item toolbar menu, or by dragging tags from the OPC browser into the group.



Which creates:

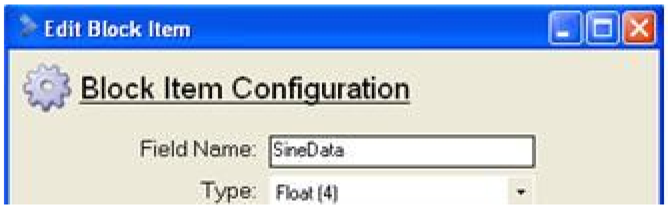


Field Name	Type	Tag Cou
Sine1	Float (4)	5
Channel_0_User_D...	0.69366...	
Channel_0_User_D...	0.69366...	
Channel_0_User_D...	0.42575...	
Channel_0_User_D...	0.77047...	
Channel_0_User_D...	0.92744...	

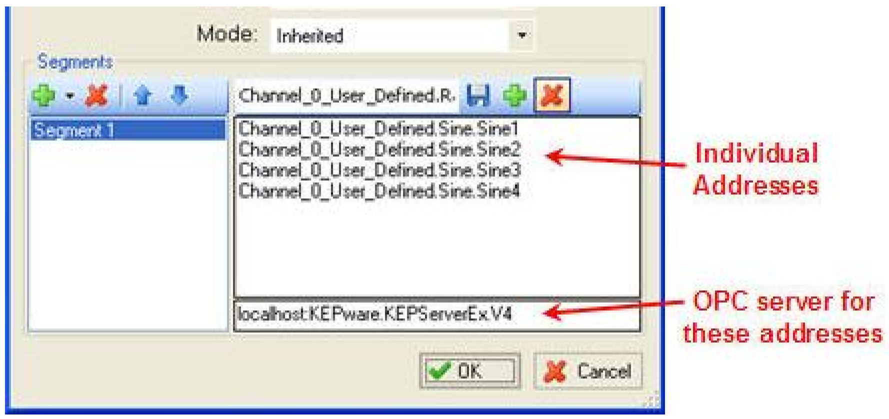
Note: To view addresses contained by an item, and their values, you have to **Expand** the item, by using the right click menu or CTRL-E.

Address List Segment

This type of segment is simply an ordered list of OPC addresses. There is no need for the addresses to be contiguous.



Field Name: SineData
Type: Float (4)



Mode: Inherited

Segments

Segment 1

Channel_0_User_Defined.R

Channel_0_User_Defined.Sine.Sine1
Channel_0_User_Defined.Sine.Sine2
Channel_0_User_Defined.Sine.Sine3
Channel_0_User_Defined.Sine.Sine4

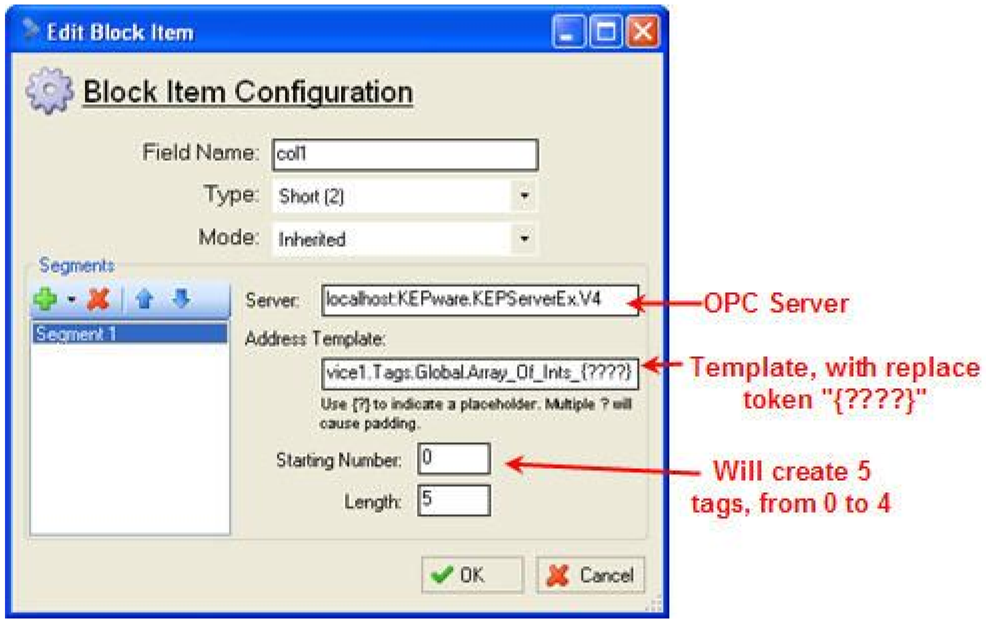
localhost:KEPware.KEPServerEx.V4

Range Segment

The range segment allows you to define an address template, a starting address, and a length. In this way, you can specify a set of contiguous addresses quickly.

Parameters:
Server: The OPC server this segment uses.
Address Template: The template that will be used to create the item paths. Use "{?}" to denote the unique part of the address. Multiple "?" will cause padding.
For example:
TagPath.DataPoint{?} will become TagPath.DataPoint0
TagPath.DataPoint{???} will become TagPath.DataPoint000

Starting number: The address to begin with.
Length: The number of addresses to create.



Edit Block Item

Block Item Configuration

Field Name: col1
Type: Short (2)
Mode: Inherited

Segments

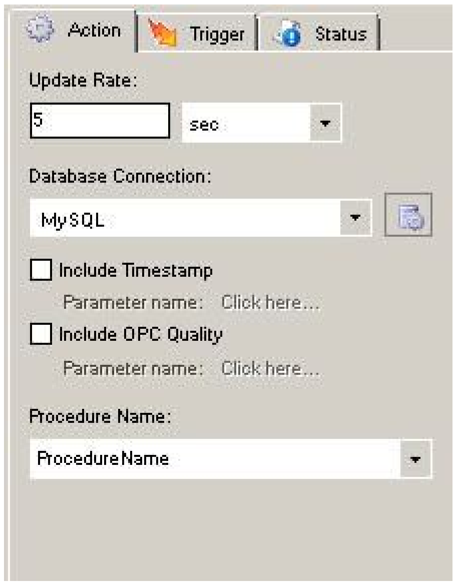
Segment 1

Server: localhost:KEPware.KEPServerEx.V4
Address Template: vice1.Tags.Global.Array_Of_Ints_{????}
Use {?} to indicate a placeholder. Multiple ? will cause padding.
Starting Number: 0
Length: 5

OPC Server
Template, with replace token "{????}"
Will create 5 tags, from 0 to 4
OK
Cancel

Stored Procedure Group Options

The stored procedure group's action tab is relatively simple. There is the standard update rate and database connection, and then several options specific to SP groups.



Update Rate: 5 sec

Database Connection: MySQL

☐ Include Timestamp

Parameter name: Click here...

☐ Include OPC Quality

Parameter name: Click here...

Procedure Name: ProcedureName

Stored Procedure Group Options

Include Timestamp: If selected, the current timestamp will be included as an input parameter to the procedure. The parameter name is selected by clicking the link.

Include Quality: The data quality of the items will be included in the parameter specified.

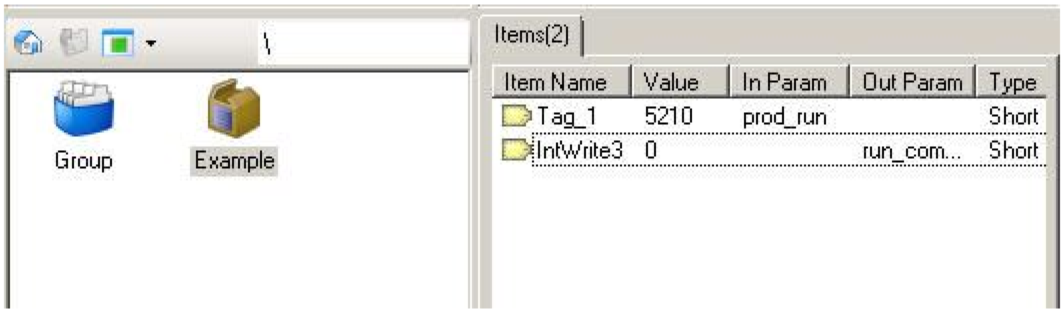
Procedure Name: The name of the procedure to target.

Trigger Tab

The Stored Procedure group shares the same trigger tab as the other groups.

Stored Procedure Groups

Stored Procedure Groups make it quick and easy to map OPC values to and from stored procedure parameters. With the stored procedure group, you can effectively leverage the power of stored procedures.



Item Name	Value	In Param	Out Param	Type
Tag_1	5210	prod_run		Short
IntWrite3	0		run_com...	Short

Features

Stored procedure groups support standard group features like triggering and handshaking, along with the following:

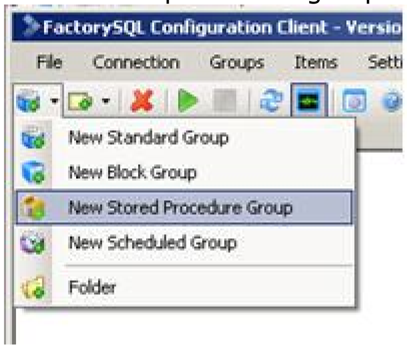
- **Full bi-directional support**: Not only can you map values to inputs, you can also map outputs and return values to OPC items.
- **Action items as inputs**: Map the value of action items to parameter inputs.
- **Include timestamp and quality parameters**

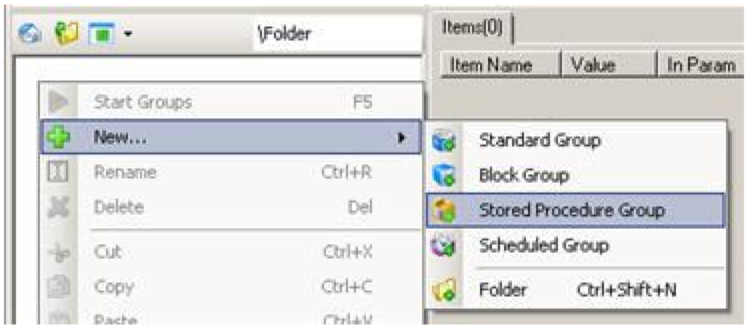
Items

Stored procedure groups support slightly modified **OPC Items** and **Action Items**.

Creating Stored Procedure Groups

New stored procedure groups can be created from the toolbar or by right clicking on the project pane:

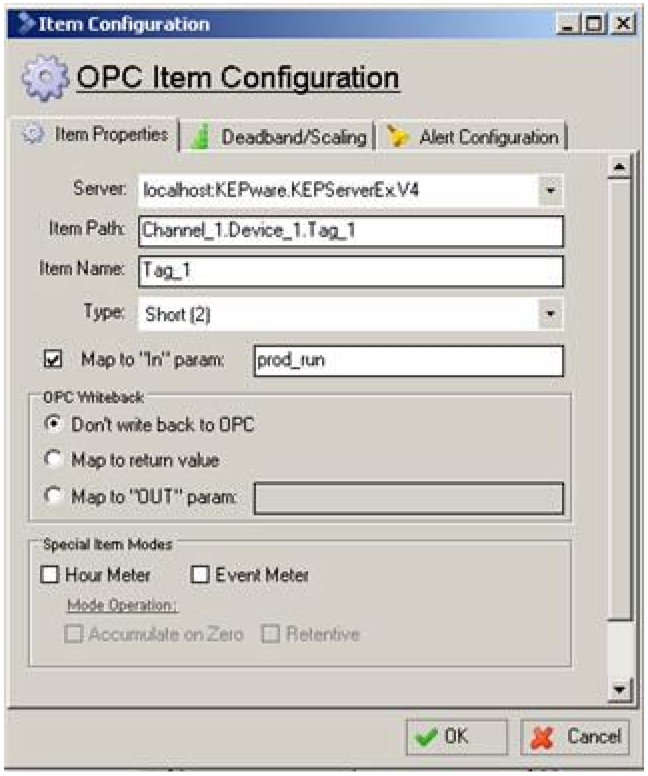




Mapping Parameters

Both OPC items and Action items can be mapped to stored procedure parameters. It is done through the item's configuration dialog.

OPC Items



Server: localhost:KEPware.KEPServerEx.V4
Item Path: Channel_1.Device_1.Tag_1
Item Name: Tag_1
Type: Short (2)
[✓] Map to "In" param: prod_run

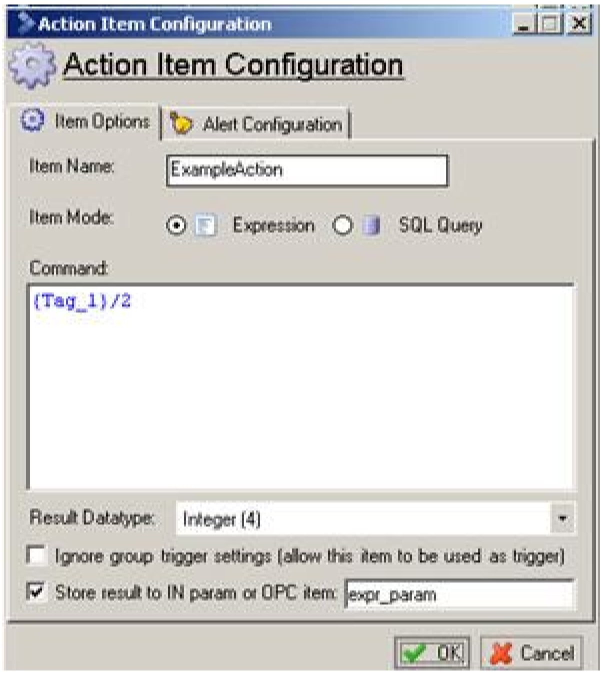
OPC Writeback
[✓ Don't write back to OPC]
Map to return value
Map to "OUT" param:

Special Item Modes
☐ Hour Meter ☐ Event Meter
Mode Operation:
☐ Accumulate on Zero ☐ Retentive

OPC Items can be mapped to and from the stored procedure.
To map the value to an **IN** parameter, simply select the check box for "In Param" and set the parameter name.
NOTE: FactorySQL will automatically add any special parameter name character neccessary.
Coming back from the procedure, the item may be bound to an **OUT** parameter, or to the **Return value**.

You may also bind to an **INOUT** parameter by using the same parameter name on both the IN and OUT fields.

Action Items



Item Name: ExampleAction
Item Mode: [✓ Expression] SQL Query
Command:

```
{Tag_1}/2
```

Result Datatype: Integer (4)
☐ Ignore group trigger settings (allow this item to be used as trigger)
[✓] Store result to IN param or OPC item: expr_param

Action items can be mapped to **IN** parameters by selecting the "Store result..." checkbox and entering the parameter name.

SQLTags^{TM} Overview

SQLTags provides a powerful link between FactorySQL and FactoryPMI, while maintaining all of the benefits of a database-centric architecture.

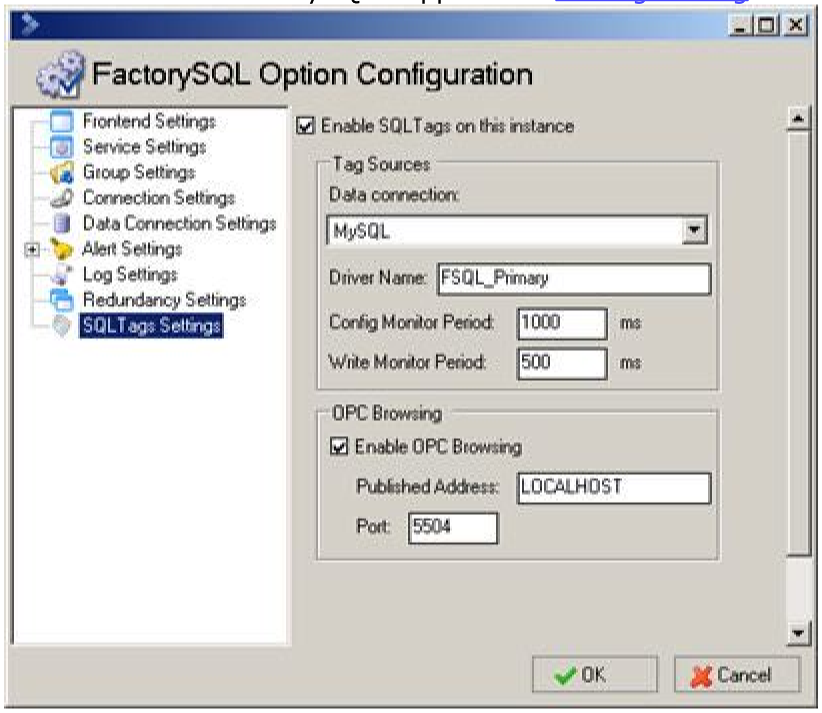
History & Overview

In the past, configuring systems using FactorySQL and FactoryPMI focused heavily on the central SQL database. Tags were mapped in FactorySQL from the OPC server to a particular location in the database, and then it was up to the designer to interpret the database values in a meaningful way in FactoryPMI. This direct approach was very powerful, but had several drawbacks. Primarily, there was a disconnect between addresses in the OPC server and their display/control counterparts in the PMI project, creating significant room for error and difficulty in maintenance. Additionally, there was a significant learning curve for users with little previous SQL experience.

SQLTags aims to address these issues, as well as provide a range of other benefits not previously possible in FactorySQL/PMI. Essentially, SQLTags creates a tag based architecture directly in FactoryPMI, allowing users to develop systems rapidly while conceptually "skipping over" the database and FactorySQL configuration. Users can browse OPC servers, configure expressions and alerts, and otherwise benefit from all of the features they would use in FactorySQL, without leaving the FactoryPMI design environment. Additionally, there is better support for control systems with an improved write response system built in.

Configuration

SQLTags is designed to be easy to use. Therefore, there is very little setup in FactorySQL. The SQLTags settings can be found in FactorySQL's application settings dialog.


FactorySQL Option Configuration

☑ Enable SQLTags on this instance

Tag Sources
Data connection: MySQL
Driver Name: FSQL_Primary
Config Monitor Period: 1000 ms
Write Monitor Period: 500 ms

OPC Browsing
☑ Enable OPC Browsing
Published Address: LOCALHOST
Port: 5504

Enable SQLTags: Turns on/off SQLTags for the current FactorySQL service.
Data connection: The database connection used for SQLTags. The necessary tables will be created if they do not already exist.

Driver Name: FactorySQL will only execute SQLTags whose driver name matches its own driver name. All other tags will run as "external tags"- their values will be monitored, but it they should be executed by a different instance.
Config Monitor Period: How often, in milliseconds, the database should be checked for tag configuration changes.
Write Monitor Period: How often the database should be checked for OPC write requests.

Enable OPC browsing: FactorySQL can provide OPC browsing services to the FactoryPMI gateway. Enabling this feature requires an open port that is visible from the FactoryPMI gateway.
Published Address: The address that FactoryPMI will use to contact the FactorySQL service for OPC browsing. If

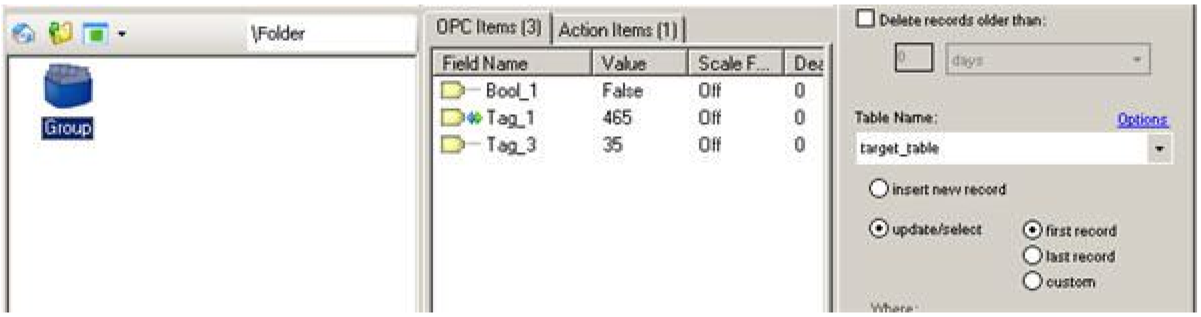
both services are on the same machine, LOCALHOST should work fine.
Port: The port that will be used by FactoryPMI to browse.

SQLTags and Redundancy

SQLTags automatically supports redundancy. When redundancy is enabled, FactorySQL will only execute SQLTags while it is the master. The primary consideration when setting up redundant SQLTags systems is that **every instance's "driver name" should be the same**. Otherwise, some nodes may view tags as external tags instead of properly executing them.

Standard Groups

Standard Groups are the core group type in FactorySQL. They are very versitile, and provide a wide range of features.



Field Name	Value	Scale F...	Dec
Bool_1	False	Off	0
Tag_1	465	Off	0
Tag_3	35	Off	0

Table Name: target_table

Features

- **Easy, flexible mapping between OPC and DB**: Drag and drop items, quickly change their column names, target table & row, and start synchronizing.
- **Historical Logging**: Simply set table mode to "Insert new record" and your instantly logging historical data.
- **Full bi-directional support**: Keep OPC values and database values perfectly synched.
- **Column definable bi-directional mode**: Mix and match bi-directional columns to create complete status and control systems in one group.

Items

- **OPC Items**: Pull in OPC data. Can be mapped to a database field, or used internally for trigger, expressions, where clause lookup, etc.
- **Action Items**: Can be expressions or SQL queries, can reference other item values, and can write their value to the database or OPC server.

License Agreement

IMPORTANT-READ CAREFULLY: This FactorySQL End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Inductive Automation. By installing, copying, or otherwise using FactorySQL, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE.

Copyright © 2002-2007 Inductive Automation. All rights reserved.
www.factorysql.com
support@factorysql.com

1. **BY DOWNLOADING, INSTALLING AND/OR IMPLEMENTING THIS SOFTWARE YOU AGREE TO THE FOLLOWING LICENSE:**
2. **DEFINITIONS.** "You" and "Licensee" refers to the person, entity or organization which is using the Software known as "FactorySQL", and any successor or assignee of same. "Inductive Automation" refers to Hechtman Enterprises, Inc. dba Inductive Automation and its successors, or manufacturer and owner of this Software.
3. **AGREEMENT.** After reading this agreement carefully, if you ("Customer") do not agree to all of the terms of this agreement, you may not use this Software. Unless you have a different license agreement signed by Inductive Automation that covers this copy of the Software, your use of this Software indicates your acceptance of this license agreement and warranty. All updates to the Software shall be considered part of the Software and subject to the terms of this Agreement. Changes to this Agreement may accompany updates to the Software, in which case by installing such update Customer accepts the terms of the Agreement as changed. The Agreement is not otherwise subject to addition, amendment, modification, or exception unless in writing signed by an officer of both Customer and Inductive Automation. If you do not wish to agree to the terms of this Agreement, do not install or use this Software.
4. **GRANT OF LICENSE.**
    i. **Evaluation Copy.** You may use FactorySQL without charge on an evaluation basis. In the unregistered version you have these limitations: **two hour runtime.** You must pay the license fee and register your copy if you wish to use FactorySQL without any limitation.
    ii. **Redistribution of Evaluation Copy.** If you are using FactorySQL on an evaluation basis you may make copies of the evaluation FactorySQL as you wish; give exact copies of the original evaluation FactorySQL to anyone; and distribute the evaluation FactorySQL in its unmodified form via electronic means (Internet, BBS's, Shareware distribution libraries, CD-ROMs, etc.). You may not charge any fee for the copy or use of the evaluation FactorySQL itself, but you may charge a distribution fee that is reasonably related to any cost you incur distributing the evaluation FactorySQL (e.g. packaging). You must not represent in any way that you are selling FactorySQL itself. Your distribution of the evaluation FactorySQL will not entitle you to any compensation from Inductive Automation. You must distribute a copy of this EULA with any copy of FactorySQL and anyone to whom you distribute FactorySQL is subject to this EULA.
    iii. **Registered Copy.** After you have purchased the license for FactorySQL, and have received the serial number enabling the registered copy, you are licensed to copy FactorySQL only into the memory of the number of computers corresponding to the number of licenses purchased. Under no other circumstances may FactorySQL be operated at the same time on more than the number of computers for which you have paid a separate license fee. You may terminate this license at any time by destroying the original and all copies of FactorySQL in whatever form. You may permanently transfer all of your rights under this EULA provided you transfer all copies of FactorySQL(including copies of all prior versions if FactorySQL is an upgrade) and retain none, and the recipient agrees to the terms of this EULA.
5. **RESTRICTIONS.** You may not reverse engineer, de-compile, or disassemble FactorySQL, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation. You may not rent, lease, or lend the SOFTWARE. You may permanently transfer all of your rights under this EULA, provided the recipient agrees to the terms of this EULA. You may not publish or publicly distribute any serial numbers, access codes, unlock-codes, passwords, or other end-user-specific registration information that would allow a third party to activate FactorySQL without a valid license.
6. **OWNERSHIP OF SOFTWARE AND COPYRIGHTS.** The Software is copyrighted and protected by the laws of the United States and other countries, and international treaty provisions. You may not remove any copyright notices from the Software. Inductive Automation may make changes to the Software at any

time without notice, but is not obligated to support or update the Software. Except as otherwise expressly provided, Inductive Automation grants no express or implied right under Inductive Automation patents, copyrights, trademarks, or other intellectual property rights. You may transfer the Software only if the recipient agrees to be fully bound by these terms and if you retain no copies of the Software.

7. **GRANT OF LICENSE AND PROHIBITIONS.** Title to all copies of the Software remains with Inductive Automation. This Software is licensed to you. You are not obtaining title to the Software or any copyrights. You may not sublicense, rent, lease, convey, translate, decompile, or disassemble the Software for any purpose. The license may be transferred to another Licensee if you keep no copies of the Software. Permission must be obtained before mirroring or redistributing the evaluation copies of the Software. You may not convert this Software or its parts to a different computer language or environment, either manually, or using an automated conversion tool, such that this Software or any modification thereof will run under any language, software, or program other than implemented by Inductive Automation. You agree that any modifications made to this Software belong to Inductive Automation and are permitted for your exclusive use during the period of this License Agreement, and may not be transferred, sold or licensed to another entity.
8. **LIMITED WARRANTY.** THE SOFTWARE IS PROVIDED AS IS AND INDUCTIVE AUTOMATION DISCLAIMS ALL WARRANTIES RELATING TO THIS SOFTWARE, WHETHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE.
9. **LIMITATION ON CONSEQUENTIAL DAMAGES.** NEITHER INDUCTIVE AUTOMATION NOR ANYONE INVOLVED IN THE PRODUCTION, OR DELIVERY OF THIS SOFTWARE SHALL BE LIABLE FOR ANY INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH SOFTWARE EVEN IF INDUCTIVE AUTOMATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR CLAIMS. IN NO EVENT SHALL INDUCTIVE AUTOMATION LIABILITY FOR ANY DAMAGES EXCEED THE PRICE PAID FOR THE LICENSE TO USE THE SOFTWARE, REGARDLESS OF THE FORM OF CLAIM. THE PERSON USING THE SOFTWARE BEARS ALL RISK AS TO THE QUALITY AND PERFORMANCE OF THE SOFTWARE. IN NO EVENT WILL INDUCTIVE AUTOMATION BE LIABLE FOR ANY AMOUNT GREATER THAN WHAT YOU ACTUALLY PAID FOR THE SOFTWARE.
10. **TERMINATION.** This Agreement is effective until terminated. This Agreement terminates on the date of the first occurrence of either of the following events: (1) The expiration of one month from written notice of termination from Customer to Inductive Automation; or (2) At any time if you violate the terms of this Agreement. Upon termination you shall destroy all copies of the Software, including modified copies, if any. You agree that monetary damages alone is not an adequate and just relief resulting from any breach of this License, that a court order prohibiting any further breach of this License is necessary to prevent further damages, and that you will not oppose any reasonable request for a temporary restraining order, preliminary injunction, or other relief sought by Inductive Automation in the event of a breach of this License. Inductive Automation shall not be required to notify you of any breach, nor make any demand or claim against you resulting from any such breach, or for a demand to stop any use or distribution in violation of the terms of this License, and you agree that any breach of this License and damages resulting therefrom shall relate back to the first and earliest breach thereof. Failure of Inductive Automation to enforce its rights pursuant to this License shall not constitute a waiver of such rights, and shall not prejudice Inductive Automation in any later enforcement of its rights or rights to seek damages therefrom.
11. **UPGRADES.** If you acquired this Software as an upgrade of a previous version, this Agreement replaces and supercedes any prior Agreements. You may continue to use the previous version of the Software, provided that both the previous version and the upgrade are installed on the same computer at all times. You may not have a previous version and the related upgrade version installed on separate computers at any time.
12. **ENTIRE AGREEMENT.** This End-User Agreement is the entire agreement between you and Inductive Automation relating to the Licensed Software, and supercedes all prior written or oral statements, promises, representations and agreements.
13. **GOVERNING LAW.** The agreement shall be governed by the laws of the State of California. Any action or proceeding brought by either party against the other arising out of or related to this agreement shall be brought only in a state or federal court of competent jurisdiction located in Sacramento County, California. The parties hereby consent to the jurisdiction of such courts.

Main Features of FactorySQL

Log from any PLC to any SQL database.

FactorySQL complies with the OPC and ODBC standards making it fully compatible with virtually all major PLCs and all major SQL databases. It supports native .NET database connectors as well, available for nearly all brands of database.

Write from any SQL database to any PLC

FactorySQL supports writing down to PLCs just as easily as reading up from them. This is very useful for batch downloads, recipe management and a host of other tasks.

Bi-directionally synchronize any SQL database to any PLC

FactorySQL can bi-directionally synchronize SQL databased and PLC data. A change on either end can trigger sending the changed data to the other side instantly. This establishes a whole new concept in SCADA system design. This is the concept of the SQL database as the "center of the world". SQL databases are fast, multi-user, secure and robust - perfectly suited as tag databases.

Trigger on value change, value range, value or time interval

FactorySQL allows you to set up triggers for when to transfer data. This can be based on value change, a value range, any value or time interval - or a combination of several of these. All this is easily configured in intuitive configuration windows.

Handshaking for data integrity and confirmation

Handshaking is provided for in several different ways. This feature can be used to guaurantee or acknowledge the actual transfer of data for data integrity, confirmation and timing purposes.

Browse PLC addresses, drag, drop and start logging

FactorySQL is configured with simple drag and drop configuration. Drag an entire PLC file from the OPC browse window and drop it into a logging group. Select a database connection and press "GO". Instantly your database table and fields corresponding to each PLC address are created and logging begins. Of course, you can customize these settings for your exact needs! Drag items from group to group. Drag groups between folders and lots more. The point is, you can set up data transfers and logging between an SQL database and multiple PLCs in just minutes!

Log thousands of tags efficiently with block groups.

Have a large number of tags that you want to mirror into the database? Block groups can efficently handle many thousands of tags easily and efficiently. Additionally, they support nearly all of the features of standard groups: bi-directional synchronization, triggers, handshakes, and indirect block addressing.

Leverage the power of stored procedures

The Store Procedure group can let you easily map OPC tags to and from stored procedures. Additionally, you can leverage the power of action items and include their values as input parameters. The stored procedure group supports the same trigger and handshake capabilities as the other groups.

Audit Log records changes written to PLC which initiated by a change in the database

You can elect to set up an audit log which will record to the SQL database anytime a new value, initiated from the database, has been written to the PLC. The date, time, old value and new value are stored as well as a field for the username of the person initiating the change. The audit log table and all its fields are created for you automatically when you elect to use audit logging.

Log history, store status

With FactorySQL, setting up historical logging and real-time status are equally easy. You can choose to insert data, update existing data, or update data based on a value from an expression, query, or OPC tag!

Hour meter mode turns database fields into timer fields

This powerful feature is very useful in downtime tracking applications - greatly reducing PLC programming. Accumulate downtime for multiple sections of a line and at the end of the hour or shift trigger the insertion of a new record - leaving the last record with the accumulated values for the last shift. From here you can read the database values from you web application and report on line efficiency, causes of downtime and more.

Event counter mode turns database fields into counter fields

Similar to the hour meter feature above, you can elect to turn any field being logged to into a counter. Each time the tag goes true, the value of the item will be incremented. Track product and production rates and lots more with this feature of FactorySQL.

Comprehensive alert server - alert logging and alerting on any PLC item

FactorySQL is a complete alarm server. Any logged item may also be configured to trigger alarm logging and email alerts. Any number of alarm states are supported. Once set for a single item you can copy your settings to other items. Additionally, the current alarm state is reflected in the alarm status table. These can then be reflected back down to the PLC eliminating a tremendous amounts of PLC programming for alarms. The intuitive and rapid development environment of FactorySQL can speed alarm development enormously.

Dynamic Alert Setpoints

Alert setpoints can be mapped from PLC or SQL database items. Therefore, setpoints can be changed from the plant floor or the front office.

SQL commands can run on any trigger event

SQL queries can be set up to run with the normal update interval of logging groups. In this way SQL database data can be merged with PLC item data in expressions and then be written back up to the database.

Action Items

Action items can be added to any group to evaluate expressions or queries you create every time a group is updated. These can combine numerous PLC items, SQL query results and the results of other Action items to yield values which can be written back to the database or an OPC item. This can save extensive amounts of PLC programming time.
Additionally, Action Items can be enhanced with your own programming with the drop-in function plugin architecture.

Runs as Windows service

FactorySQL runs as a Windows service. The FactorySQL frontend configuration client can connect to the service locally or remotely. One frontend can configure many FactorySQL service applications in an enterprise remotely. The FactorySQL service can be configured to be password protected. By running as a service, FactorySQL runs anytime the host PC is turned on.

Local Data Caching

Feel free to log data to a remote database- should the connection go down, FactorySQL will cache the data until it is back up. You won't lose any historical information.

Free download is fully functional. Only limitation is 2 hour runtime. Develop and don't purchase until deployed.

The FactorySQL service and frontend configuration client is free to download and use with the only limitation being that the logging groups will stop running after two hours. But they can be restarted every two hours to continue developing and testing with FactorySQL.

Entire projects can be developed without ever buying FactorySQL. When your project is finally deployed, it will be necessary to register FactorySQL to remove the two hour limit. Otherwise, FactorySQL is fully functional. Unlimited data points. Unlimited PLCs and SQL database connections. Any project you develop before you register FactorySQL will still be good after your register.

System Requirements

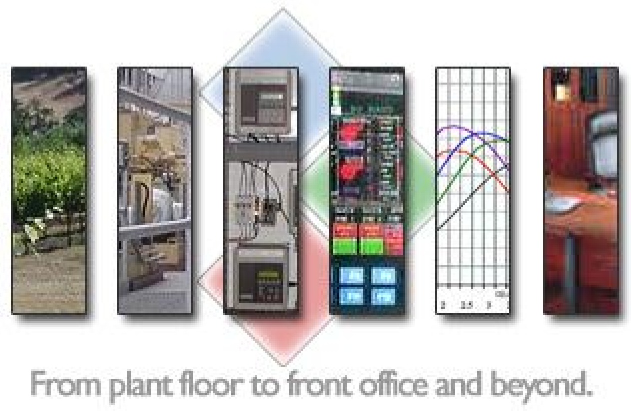
Required Hardware:

- Windows 2000/XP/2003
- 128 MB RAM or more
- 20 MB free disk space
- At least 800*600 screen resolution

Required Software:

- Any OPC server such as Keware's KEPServerEX, Rockwell's RSLinx (OEM or above), etc.
- A supported database: Microsoft SQL Server, Oracle, MySQL, Postgres, IBM DB2, EnterpriseDB, or any database with and ODBC driver.
- .NET framework v2.0

Welcome to FactorySQL



Welcome to FactorySQL!

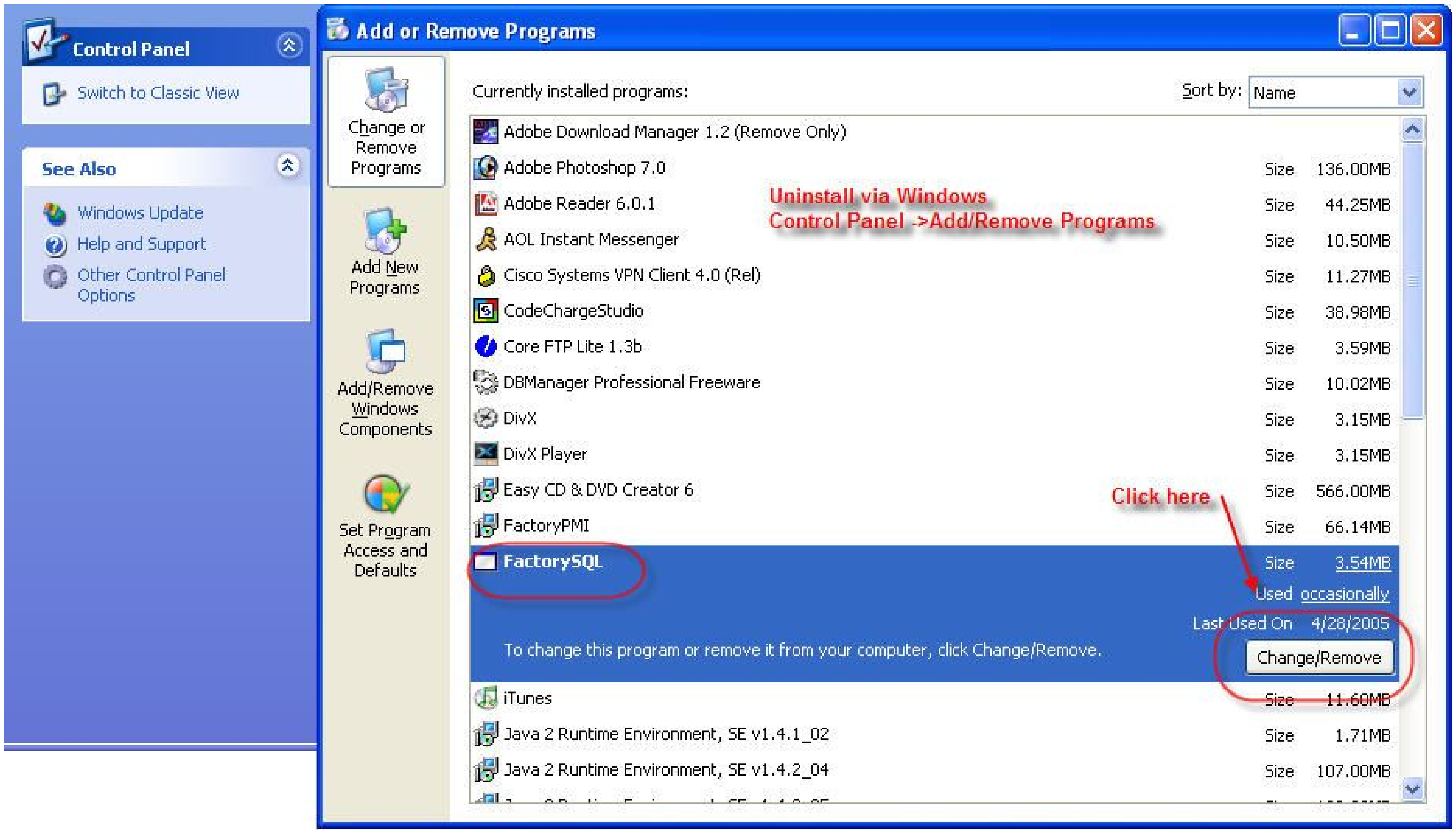
FactorySQL is the fastest, easiest and most reliable way to link your PLC data to an SQL database. FactorySQL is an OPC client and database connector program that makes bi-directional transfer of data a snap.   It is the most powerful and affordable solution on the market today.

FactorySQL allows you to:

- Communicates with nearly any SQL database system, and any OPC compliant data server.
- Browse PLC addresses, drag them, drop them and start logging them.
- Set your own logging rules - OPC to DB, DB to OPC, or bi-directional.
- Create your own trigger rules - periodic, on value change and others.
- Use built-in handshaking to insure data integrity.
- Automatically create and maintain database tables.
- Call and retrieve values from stored procedures.
- Use built-in alarm server to log alarms and send email alerts based on rules you create.
- Keep an audit log to track changes made to PLC memory from database for accountability.
- Run hundreds of logging or transaction processes simultaneously.
- Log or transact into one or many different databases at the same time.
- Run as a Windows service for reliability.

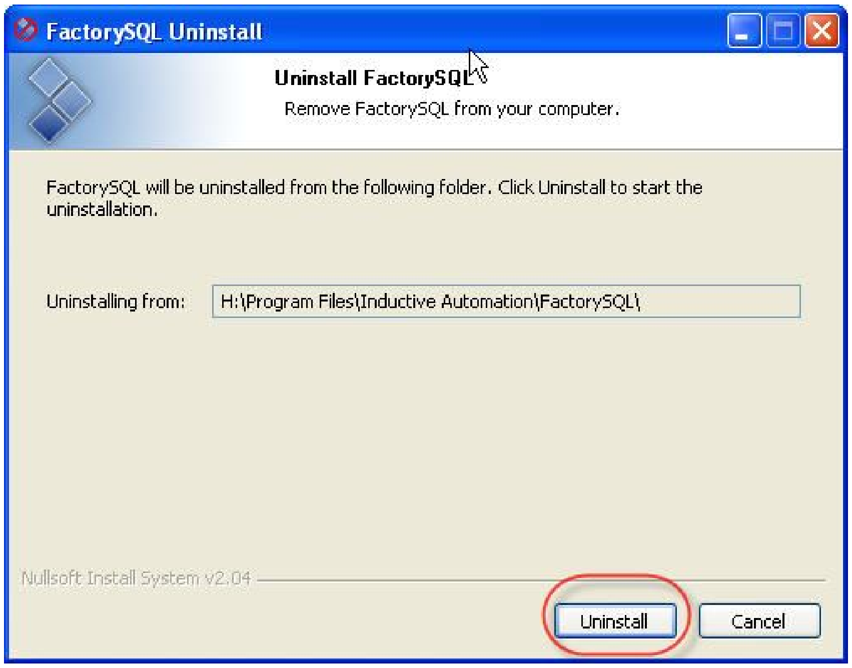
Uninstalling FactorySQL

To uninstall FactorySQL simply follow the standard procedure to uninstall a program in Windows. Go to the Control Panel. Open "Add or Remove Programs". Select FactorySQL. Click "Change/Remove". Follow uninstallation instructions.


Control Panel
Switch to Classic View
See Also
Windows Update
Help and Support
Other Control Panel Options
Add or Remove Programs
Change or Remove Programs
Add New Programs
Add/Remove Windows Components
Set Program Access and Defaults
Uninstall via Windows Control Panel ->Add/Remove Programs
Click here
Change/Remove

Currently installed programs: Sort by: Name

Adobe Download Manager 1.2 (Remove Only)		
Adobe Photoshop 7.0	Size	136.00MB
Adobe Reader 6.0.1	Size	44.25MB
AOL Instant Messenger	Size	10.50MB
Cisco Systems VPN Client 4.0 (Rel)	Size	11.27MB
CodeChargeStudio	Size	38.98MB
Core FTP Lite 1.3b	Size	3.59MB
DBManager Professional Freeware	Size	10.02MB
DivX	Size	3.15MB
DivX Player	Size	3.15MB
Easy CD & DVD Creator 6	Size	566.00MB
FactoryPMI	Size	66.14MB
FactorySQL	Size	3.54MB
	Used	occasionally
	Last Used On	4/28/2005
To change this program or remove it from your computer, click Change/Remove.		
iTunes	Size	11.60MB
Java 2 Runtime Environment, SE v1.4.1_02	Size	1.71MB
Java 2 Runtime Environment, SE v1.4.2_04	Size	107.00MB



Uninstall FactorySQL
Remove FactorySQL from your computer.

FactorySQL will be uninstalled from the following folder. Click Uninstall to start the uninstallation.

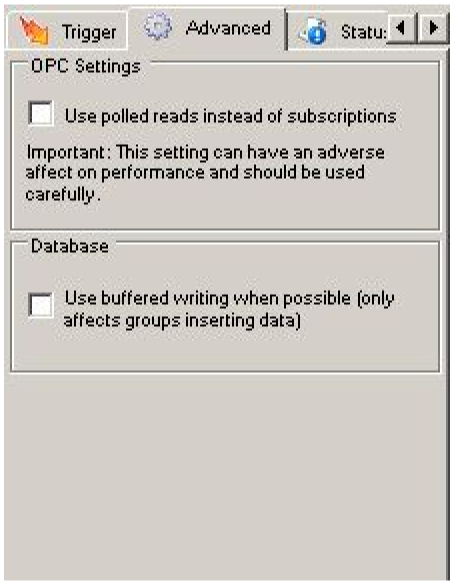
Uninstalling from: H:\Program Files\Inductive Automation\FactorySQL\

Nullsoft Install System v2.04

Group Options - Advanced Tab

IMPORTANT: The Advanced Options tab is not displayed by default. It must first be enabled in Frontend Settings.

The advanced tab houses a few additional settings that affect group execution. They are not necessarily advanced in what they do, but are designated as such in order to reduce confusion among new users.



OPC Settings

Use polled reads instead of subscriptions

Important: This setting can have an adverse affect on performance and should be used carefully.

Database

Use buffered writing when possible (only affects groups inserting data)

OPC Settings

Use polled reads instead of subscriptions: Normally FactorySQL acquires data from the OPC server on a subscription basis. That is, FactorySQL provides the server with a list of items to monitor (the items in a group, for instance) and the server provides notifications whenever the value or quality of an item changes.
By selecting this option, FactorySQL will instead call a Read function on the server each execution in order to retrieve the values. This behavior can result in much more work for both FactorySQL and the server, and is generally discouraged by the OPC Foundation. However, there are certain situations in which this option is very useful. In particular, it is useful for batch operations, which are usually triggered. When the trigger goes high, FactorySQL will read the values, and you can know for certain that the values retrieved are the most recent. Using subscriptions there is the slight possibility that the trigger may arrive before the other data, and thus the older data would be logged.

Even with this option selected, certain items **will still be subscribed**:

- The trigger
- Any item that is referenced by a run-alway action item (assuming the group is triggered).

This way, FactorySQL is able to monitor the trigger or any items necessary for the trigger, and only call read when the group actually needs to execute.

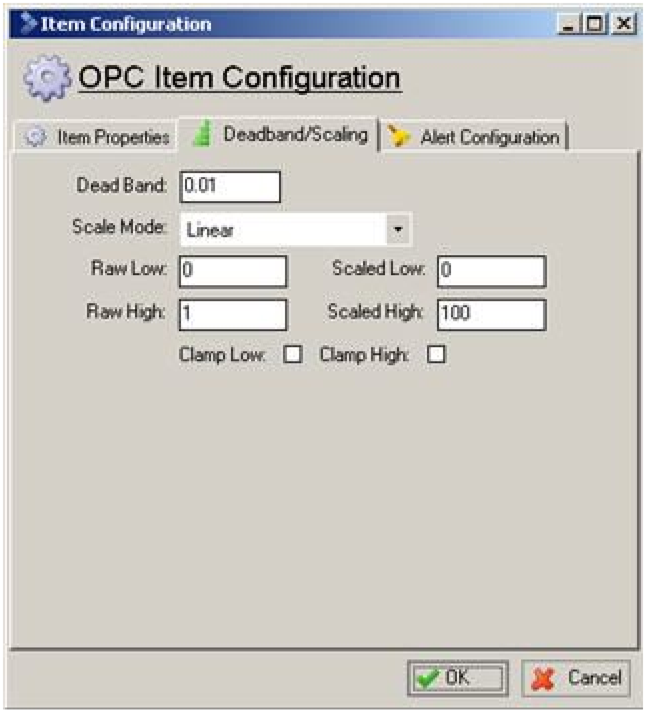
Database Settings (Standard group only)

Use buffered writing when possible: If selected, FactorySQL will write historical data to a buffer instead of directly to the database. By doing this, it is less likely that data will be lost when the database connection goes down. Even with data caching enabled, there is a certain period of time required to determine connection failure. Without buffering, the group cannot execute during this time. With buffering enabled, however, the data will be queued until the switch over to the data cache is complete.

On the downside, the buffer executors currently only use 1 thread per connection, so concurancy is more limited than with normal unbuffered execution.

Deadband & Scaling

OPC items allow you to specify deadband and scaling settings.



Dead Band: 0.01
Scale Mode: Linear
Raw Low: 0 Scaled Low: 0
Raw High: 1 Scaled High: 100
Clamp Low: ☐ Clamp High: ☐

Deadband: Prevents data changes from occuring until the value has changed by the specified amount.

Scalemode: Off, Linear, Square Root
Raw Low, High, Scaled Low, High: The values to scale between, from the perspective of OPC to DB. For instance, in the screenshot above, the raw goes from 0 to 1, and scaled from 0 to 100, in linear mode. Essentially, FactorySQL will multiple each value coming from the OPC server by 100, and will divide any value coming from the database by the same amount.

Clamp: Prevents the scaled value from moving outside of the specified bounds. Continuing the example from above, if we clamped both Low and High, the value would always be between 0-100.

Table Column Assignment

The **Table Column Assignment** screen allows you to change the columns that FactorySQL uses for certain functionality.

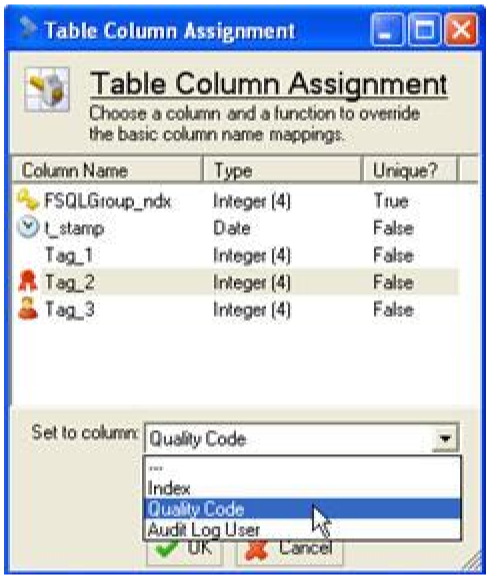


Table Column Assignment
Table Column Assignment
Choose a column and a function to override the basic column name mappings.

Column Name	Type	Unique?
FSQLGroup_ndx	Integer (4)	True
t_stamp	Date	False
Tag_1	Integer (4)	False
Tag_2	Integer (4)	False
Tag_3	Integer (4)	False

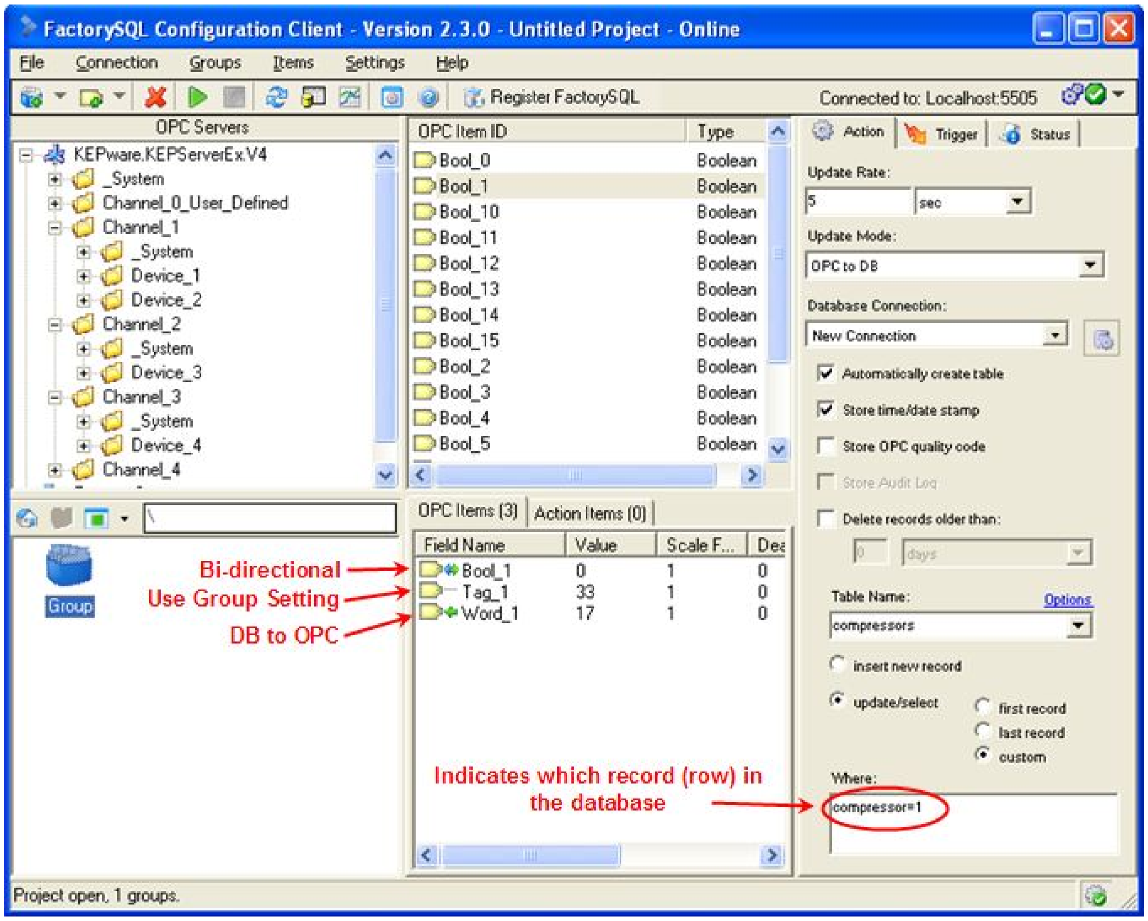
Set to column: Quality Code

This tool is particularly useful when trying to log to an existing table that cannot be altered. In this case, you can re-assign necessary columns, such as the index, to the existing columns of the table.
The only column that is required for logging is the Index column. The other 3 columns (timestamp, quality code, audit log user) depend on the settings of the group.

Custom Where Clause Example

Suppose we have 4 identical compressors. We're interested in making a web page that displays: "pressure" and "temperature" (OPC->DB), allows either the PLC or user to change "hiPressSP" with a PLC change having priority (Bi-directional, OPC wins), and having a "reset" bit that sends a signal from the web page to the PLC (DB->OPC).

To set up our example, we'll choose OPC->DB as the default group mode. For the first compressor, we'll choose "update custom (record)" and indicate WHERE "compressors_ndx = 1". Notice that each OPC item in our transactional group corresponds to the database column of the same name in the record indicated by the WHERE clause.



OPC Item ID	Type
Bool_0	Boolean
Bool_1	Boolean
Bool_10	Boolean
Bool_11	Boolean
Bool_12	Boolean
Bool_13	Boolean
Bool_14	Boolean
Bool_15	Boolean
Bool_2	Boolean
Bool_3	Boolean
Bool_4	Boolean
Bool_5	Boolean

Field Name	Value	Scale F...	Dea
Bool_1	0	1	0
Tag_1	33	1	0
Word_1	17	1	0

Update Rate: 5 sec

Update Mode: OPC to DB

Database Connection: New Connection

[✓] Automatically create table

[✓] Store time/date stamp

[ ] Store OPC quality code

[ ] Store Audit Log

[ ] Delete records older than: 0 days

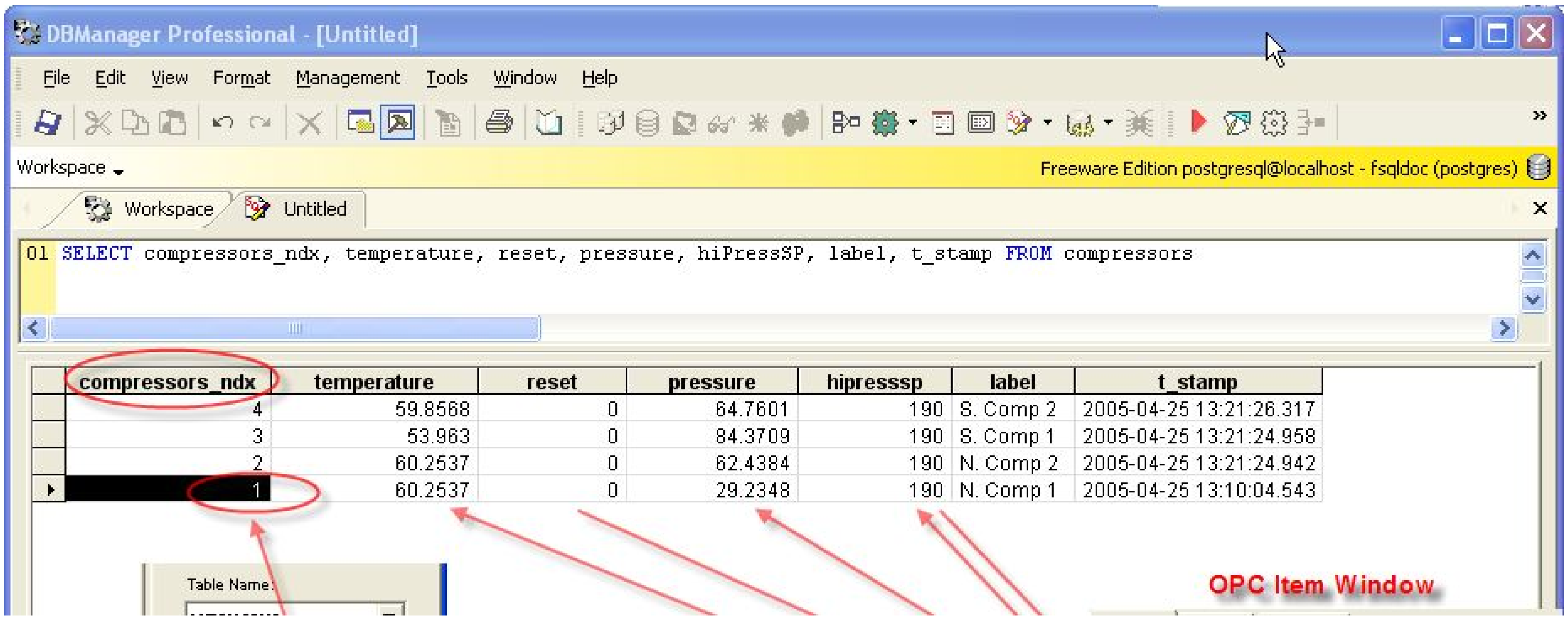
Table Name: compressors Options

insert new record [✓ update/select] first record last record [✓ custom]

Where: compressor=1

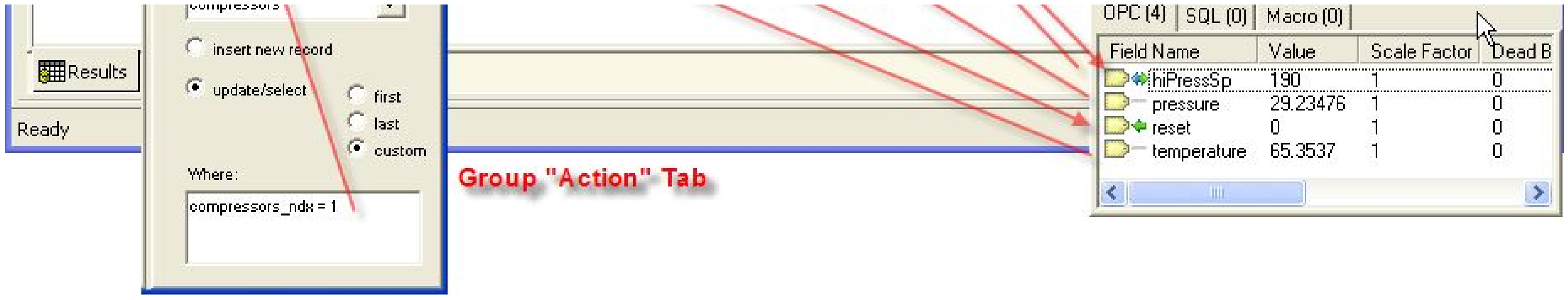
We now set "hiPressSP" to bi-directional (OPC wins) and "reset" to (DB->OPC). Recall that this overrides the group settings for these OPC items. more on OPC tab.

Now we add 3 more identical compressors. The index column, "compressors_ndx" (*tablename*_ndx by convention in FactorySQL) will indicate which record to use. We then insert 3 more records with the "compressors_ndx" ranging from 1 to 4. Now each of the 4 rows represents a different compressor, and the columns are our status/control variables.



```
SELECT compressors_ndx, temperature, reset, pressure, hiPressSP, label, t_stamp FROM compressors
```

compressors_ndx	temperature	reset	pressure	hipresssp	label	t_stamp
4	59.8568	0	64.7601	190	S. Comp 2	2005-04-25 13:21:26.317
3	53.963	0	84.3709	190	S. Comp 1	2005-04-25 13:21:24.958
2	60.2537	0	62.4384	190	N. Comp 2	2005-04-25 13:21:24.942
1	60.2537	0	29.2348	190	N. Comp 1	2005-04-25 13:10:04.543



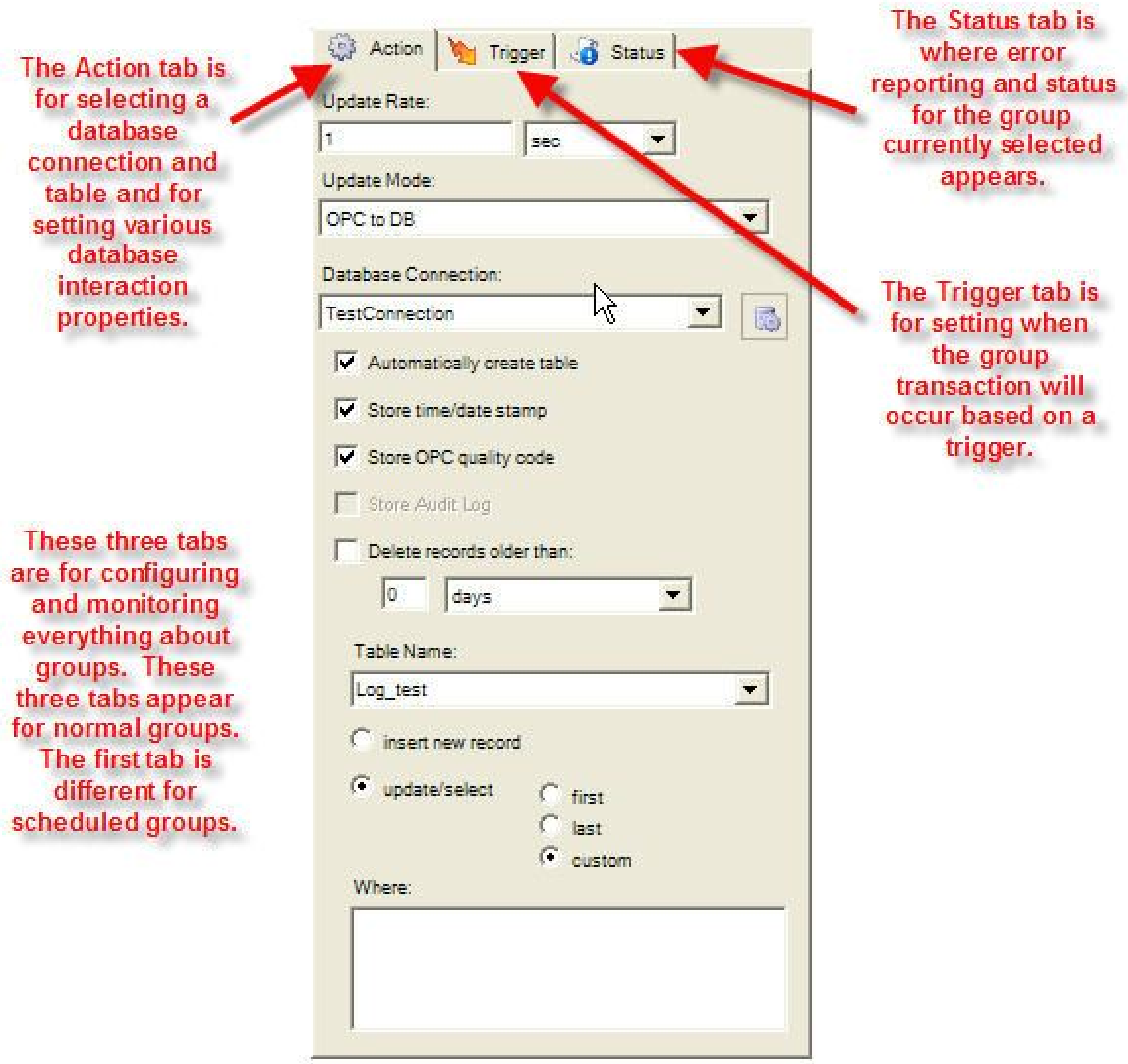
compressors_ndx = 1

Field Name	Value	Scale Factor	Dead B
hiPressSp	190	1	0
pressure	29.23476	1	0
reset	0	1	0
temperature	65.3537	1	0

The **Group Settings Tabs** (on the far right) are for setting properties and displaying the status of the *currently selected group only*.

Normal (Transaction) Groups

The **Group Settings Tabs** are comprised of the **Action Tab**, **Trigger Tab**, and **Status Tab**



The Action tab is for selecting a database connection and table and for setting various database interaction properties.

The Status tab is where error reporting and status for the group currently selected appears.

The Trigger tab is for setting when the group transaction will occur based on a trigger.

These three tabs are for configuring and monitoring everything about groups. These three tabs appear for normal groups. The first tab is different for scheduled groups.

Action | Trigger | Status

Update Rate: 1 sec

Update Mode: OPC to DB

Database Connection: TestConnection

[✓] Automatically create table

[✓] Store time/date stamp

[✓] Store OPC quality code

[ ] Store Audit Log

[ ] Delete records older than: 0 days

Table Name: Log_test

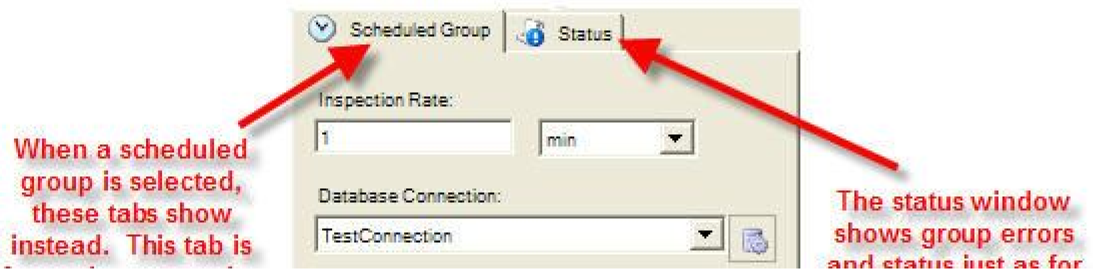
( ) insert new record

(•) update/select — ( ) first ( ) last (•) custom

Where:

Scheduled Groups

The **Scheduled Items Tab** and **Status Tab** make up the **Group Settings Tab** for **Scheduled Groups**



When a scheduled group is selected, these tabs show instead. This tab is

The status window shows group errors and status just as for

Scheduled Group | Status

Inspection Rate: 1 min

Database Connection: TestConnection

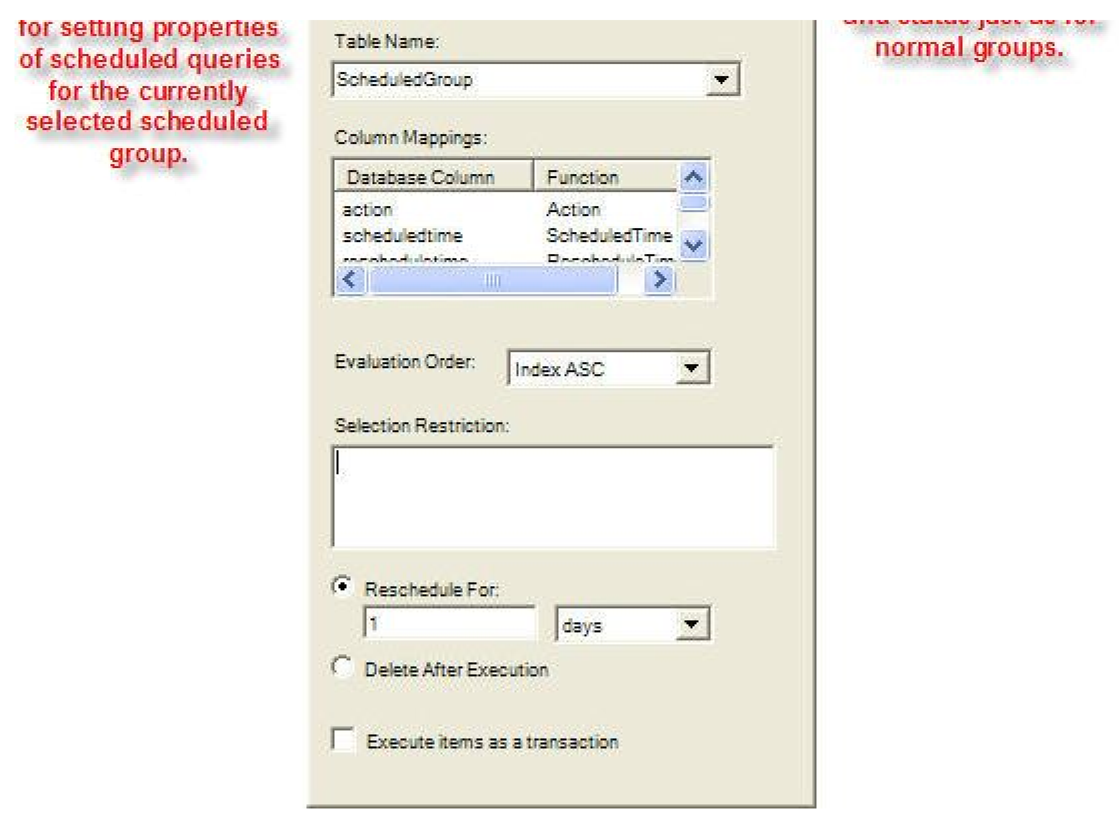

for setting properties of scheduled queries for the currently selected scheduled group.
normal groups.

Table Name: ScheduledGroup

Column Mappings:

Database Column	Function
action	Action
scheduledtime	ScheduledTime

Evaluation Order: Index ASC

Selection Restriction:

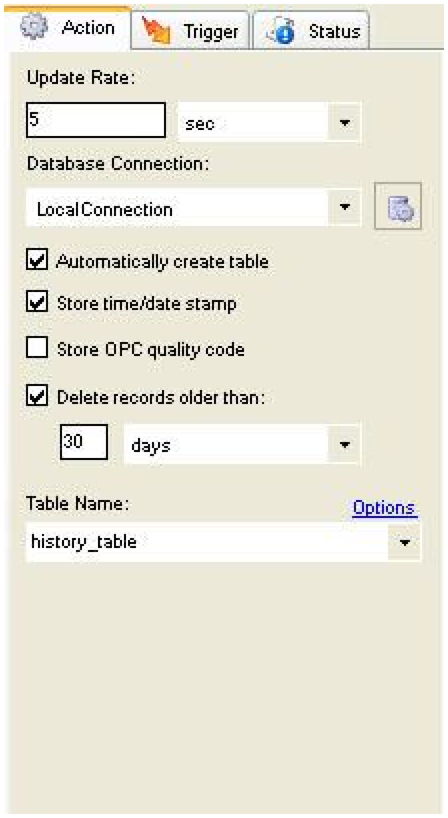
[✓ Reschedule For:] 1 days

Delete After Execution

Execute items as a transaction [ ]

Historical Group Options - Action Tab

The Action tab is where the core settings for the group's execution are configured. Primarily you will set how often the data is logged, at to which table in a specific database.



Update Rate: 5 sec

Database Connection: LocalConnection

☑ Automatically create table

☑ Store time/date stamp

☐ Store OPC quality code

☑ Delete records older than: 30 days

Table Name: history_table

Update Rate: Defines a number and interval unit for the timing of a historical group. When running, the group will be checked at this interval. If the trigger condition is valid, or if a trigger is not defined, data will be logged.

Database Connection: Which database connection to log data to.

Table Name: The name of the table that the data will log to.

Additional Options:

Automatically create table: When you start a group FactorySQL will create a table in the database if it doesn't exist. It will also automatically create columns in the database for each item that doesn't exist. If the table already exists, FactorySQL will ask the user if they want to add the columns.

Store time/date stamp: FactorySQL will maintain a "datetime" that indicates the last time FactorySQL logged data. This is usually very important for data logging situations.

Store OPC quality code: Every value that comes from the OPC server has a "quality code" associated with it, that defines if it is a vaild value, or if an error occured along the way. If you select this box, FactorySQL will maintain the code in the database. This can be useful for troubleshooting problems between the OPC server and device, or for monitoring a connection to a PLC. The codes from all items will be logically ANDed together, so if one item is bad, the value in the database will reflect a bad quality. This is important to keep in mind when mixing items from different OPC servers in the same group, when some items may be good and others bad at the same time. **Note:** To store individual qualities, copy and paste the items, and then select the "Quality" property under their configuration windows.

The OPC tab contains all the **OPC Items** in the selected **Transaction Group**. It indicates the item type, status indications, and settings.

Display Columns

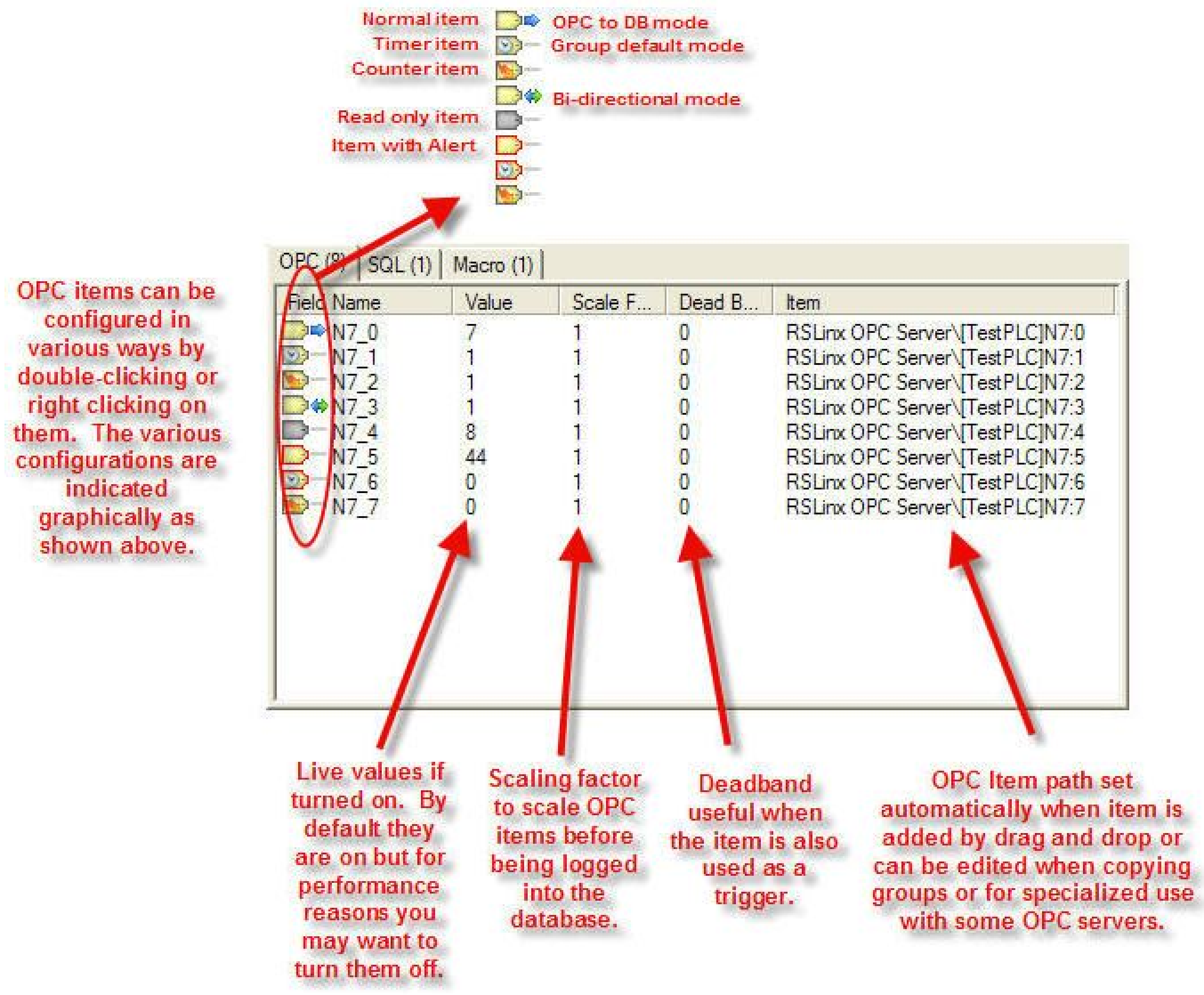
Field Name: is the name of the column in the database that FactorySQL will use for that OPC item. Duplicate names within the same group are not allowed.

Value: indicates the value of the OPC address (PLC register) of the item. **Live Values** must be enabled to see data.

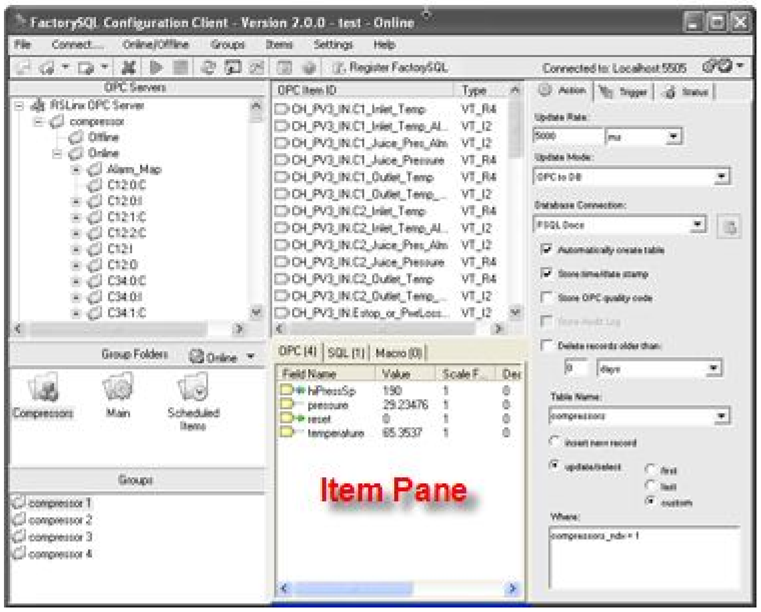
Scale Factor: multiplies the OPC value by the scale factor when writing to the database and divides the database value by the scale factor when writing to the PLC. For example, a scale factor of .1 will cause an "implied decimal" with respect to the PLC. A PLC value of 255 will show 25.5 in the database, and a database value of 28.9 will write 289 to the PLC.

Deadband: requires that OPC item change value by greater than the value of the deadband before it registers as a change. This is useful when using analog values as a trigger.

Item: Shows the full path of the OPC item including; OPC Server, Topic, and path.


Normal item
Timer item
Counter item
Read only item
Item with Alert
OPC to DB mode
Group default mode
Bi-directional mode
OPC items can be configured in various ways by double-clicking or right clicking on them. The various configurations are indicated graphically as shown above.
OPC (8)
SQL (1)
Macro (1)
Live values if turned on. By default they are on but for performance reasons you may want to turn them off.
Scaling factor to scale OPC items before being logged into the database.
Deadband useful when the item is also used as a trigger.
OPC Item path set automatically when item is added by drag and drop or can be edited when copying groups or for specialized use with some OPC servers.

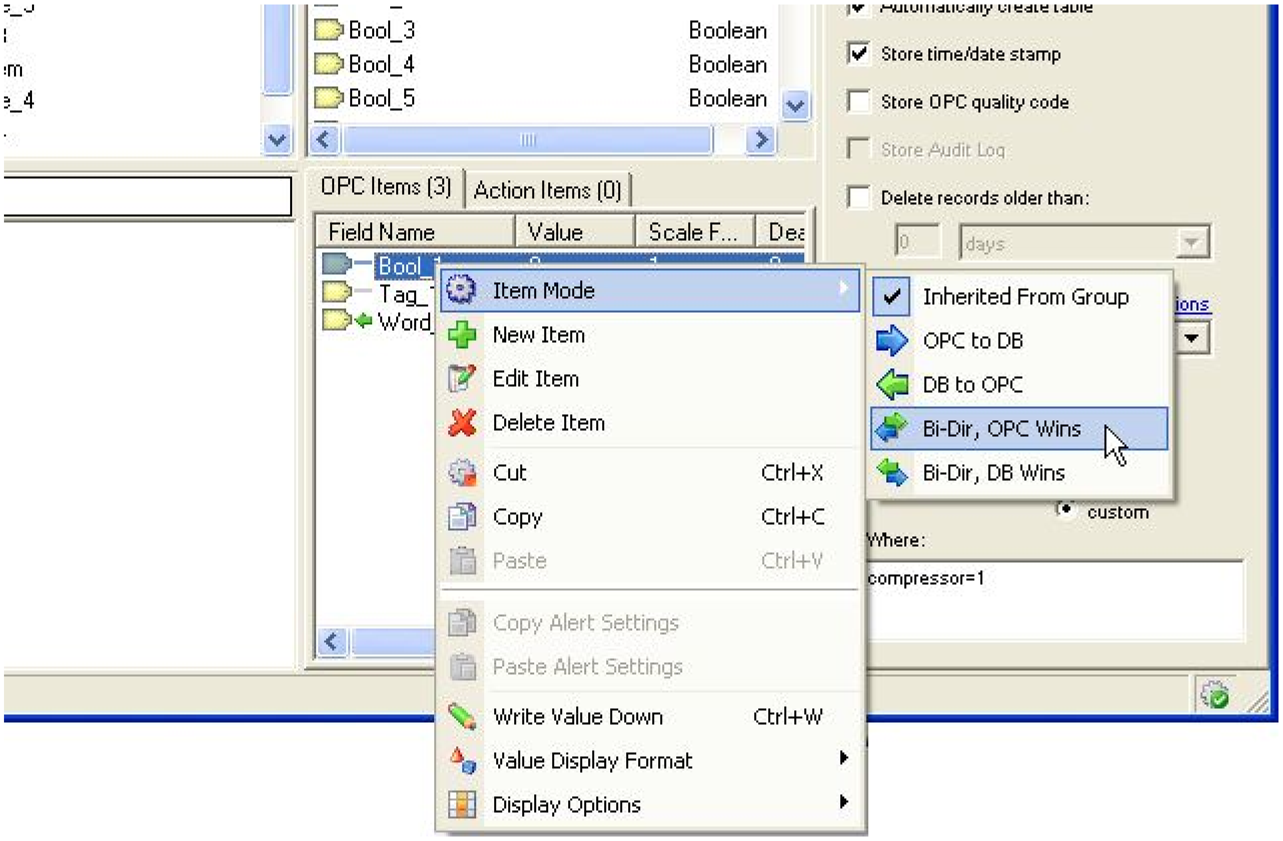
Field Name	Value	Scale F...	Dead B...	Item
N7_0	7	1	0	RSLinx OPC Server\[TestPLC]N7:0
N7_1	1	1	0	RSLinx OPC Server\[TestPLC]N7:1
N7_2	1	1	0	RSLinx OPC Server\[TestPLC]N7:2
N7_3	1	1	0	RSLinx OPC Server\[TestPLC]N7:3
N7_4	8	1	0	RSLinx OPC Server\[TestPLC]N7:4
N7_5	44	1	0	RSLinx OPC Server\[TestPLC]N7:5
N7_6	0	1	0	RSLinx OPC Server\[TestPLC]N7:6
N7_7	0	1	0	RSLinx OPC Server\[TestPLC]N7:7



Change Item Update Mode

Change Item Update Mode forces an OPC item to behave in a specified update mode regardless of the group update mode setting. The default **Item Update Mode** is to use that of the group. See Action Tab Update Mode for a description of update modes.

To change the **Item Update Mode** simply right click on the OPC item, choose "Item Mode", and select the desired update mode.

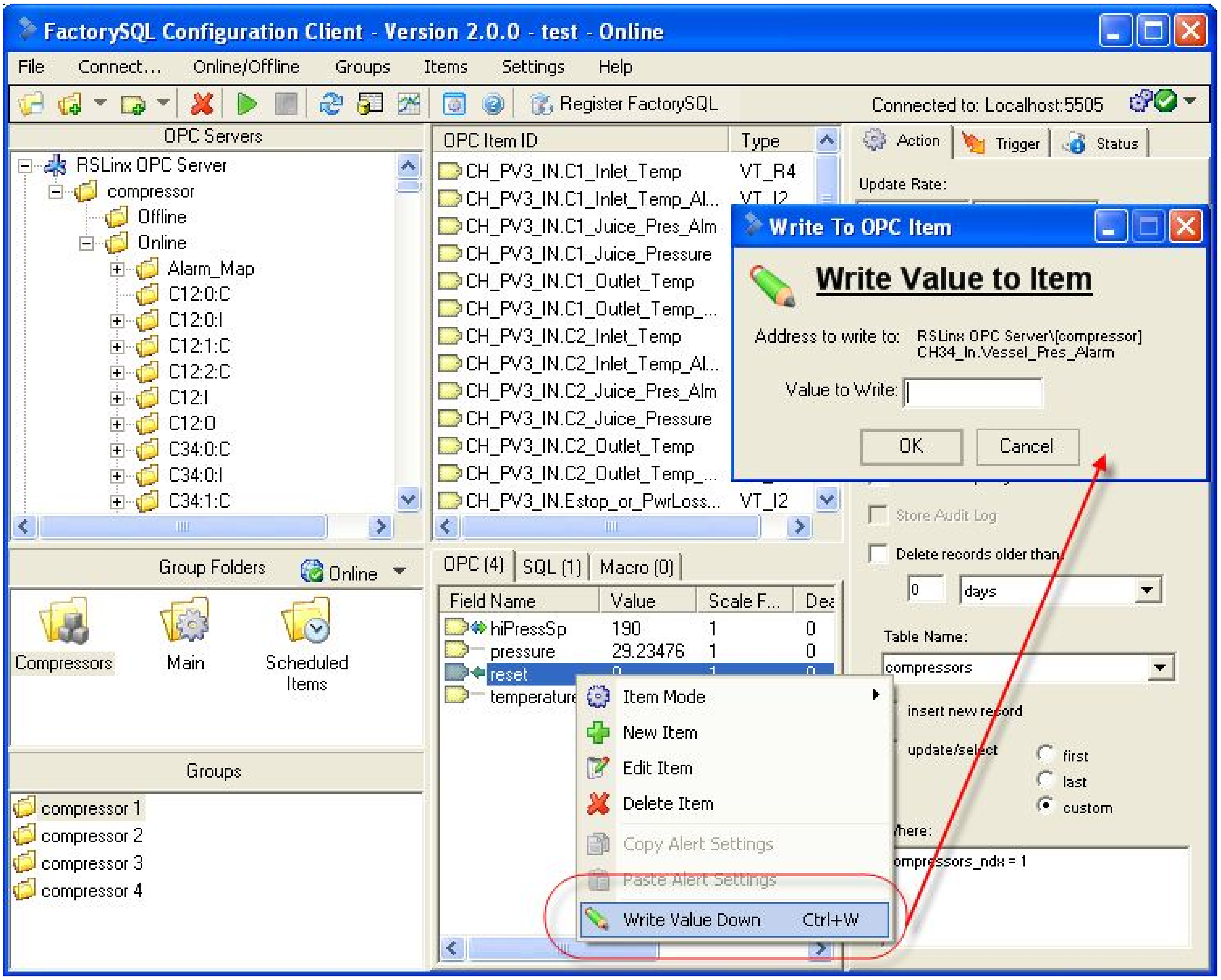


The following example illustrates why you might want to do this.

Write Values Down

Write Values Down allows the user to write an immediate value to the PLC register of the OPC item.

To **Write Value Down** simply right click on the OPC item and select "Write Value Down" (or press Cntl+W). Enter the value to be written to the PLC in the dialog window and click "ok".



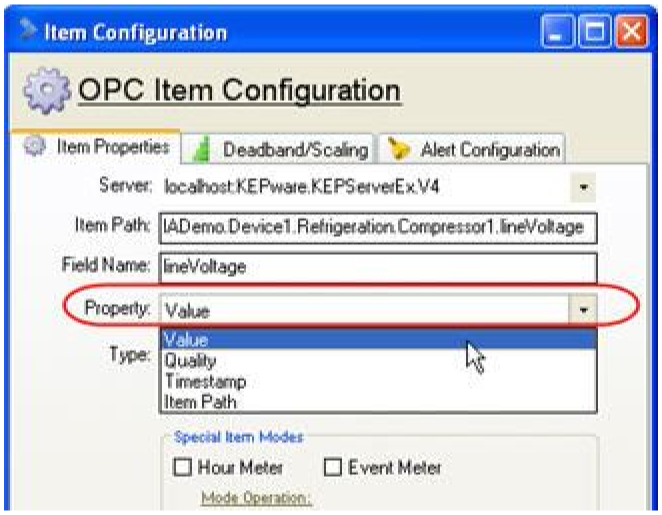
Address to write to: RSLinx OPC Server\[compressor]CH34_In.Vessel_Pres_Alarm

Value to Write:

Field Name	Value	Scale F...	Dea
hiPressSp	190	1	0
pressure	29.23476	1	0
reset	0	1	0
temperature			

OPC Item Property

The **Property** setting allows you to retrieve useful information about a tag.

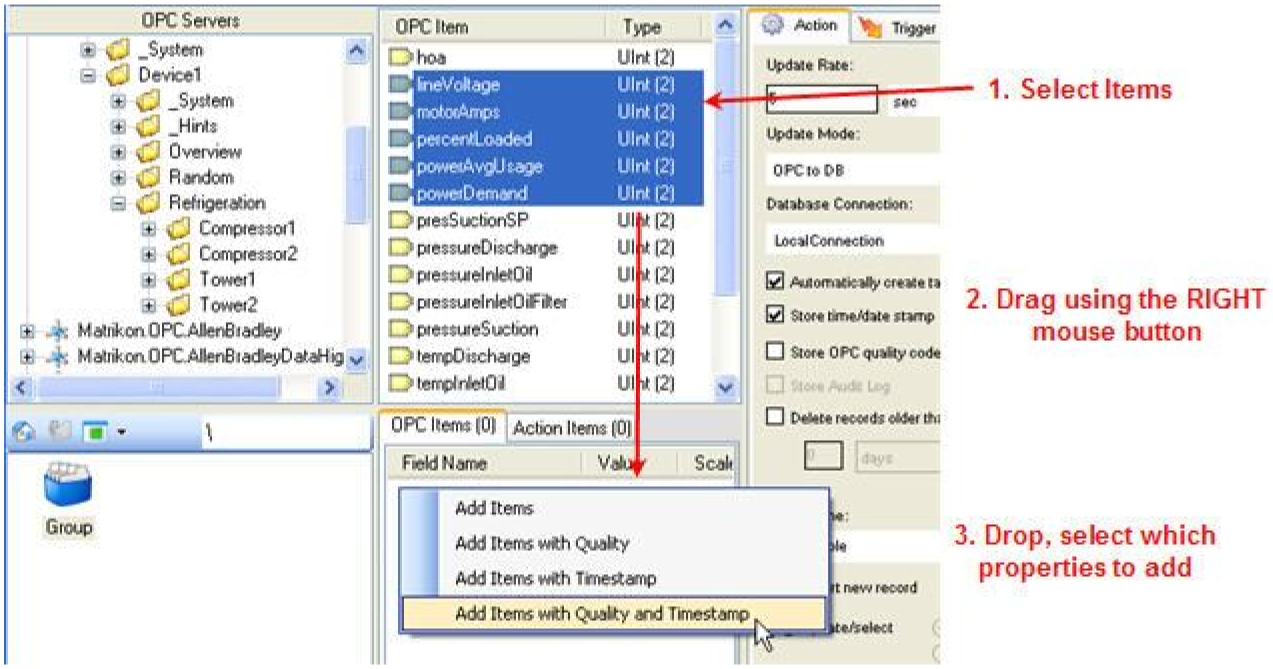


It is important to remember a few points about the properties:

- **Datatype should be set:** It is important that the datatype is correct for the **type of the property** (not the source tag). The datatype is automatically set for the value property when the item is added to the group, but may need to be changed if the property is changed. It should be as follows: Quality - Int, Timestamp - Date, Item Path - String.
- **Timestamp is different than group timestamp:** The group timestamp indicates when the value was written, whereas the item's timestamp property comes from the OPC server and is an indication of when the value last changed.
- **It is possible to have the same item multiple times in a group:** Therefore, you can mix multiple item properties in a group. Simply copy and paste items, and modify them to fit your needs.

Adding Item Properties While Browsing

It is possible to quickly add item properties while browsing by dragging & dropping items with the **right mouse button**.

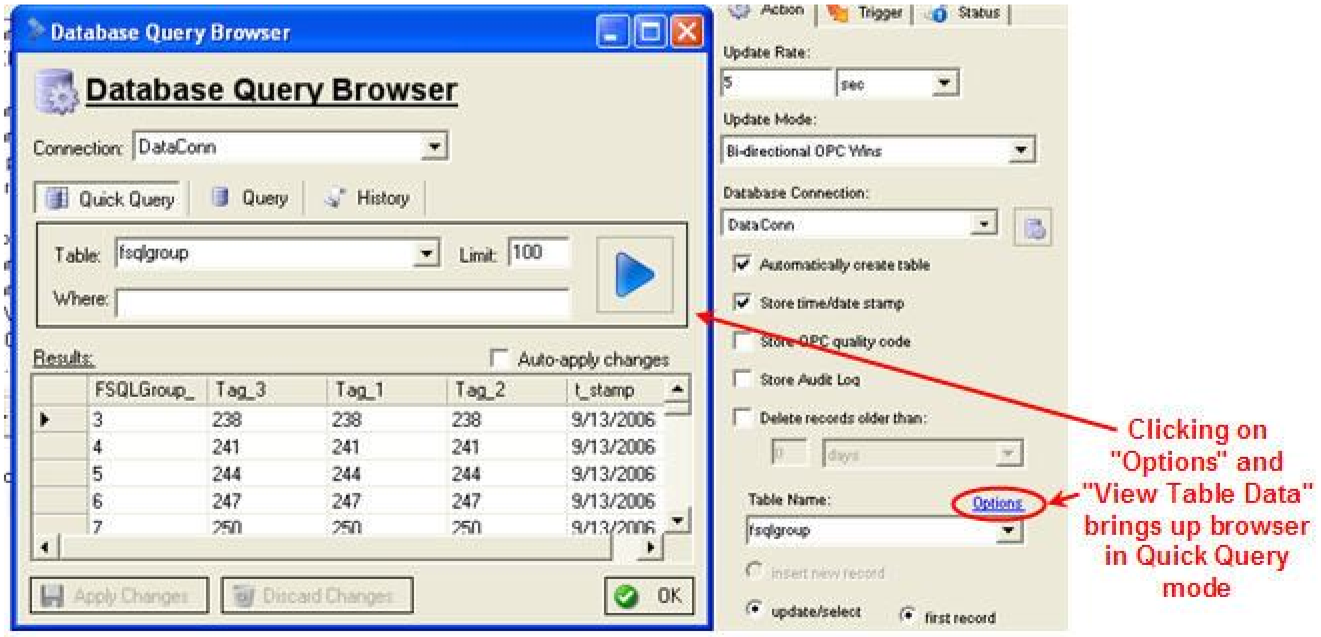


When you drop the items, a menu asks which properties to add. Duplicate copies of the item with the appropriate property and type settings will be created.

Query Browser/View Table Data

The **FactorySQL Query Brower** allows you to execute database queries, and view/edit table data.

You may get to it by clicking on Table Options, and then "View Table Data":



	FSQLGroup_	Tag_3	Tag_1	Tag_2	t_stamp
▶	3	238	238	238	9/13/2006
	4	241	241	241	9/13/2006
	5	244	244	244	9/13/2006
	6	247	247	247	9/13/2006
	7	250	250	250	9/13/2006

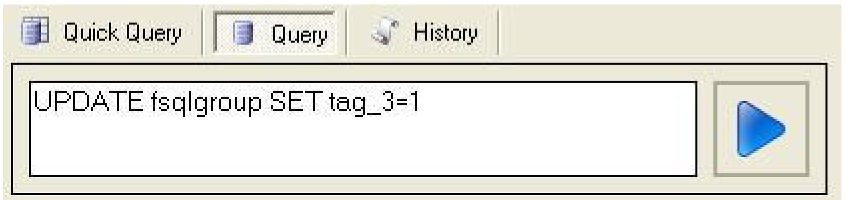
As the image shows, the browser will be opened in **Quick Query** mode, with the base parameters filled in according to the table settings.

Quick Query

The Quick Query tab allows you to exeute a simple query quickly. You can select the table, the number of rows returned, and build a simple where clause.
Note that data editing is only enabled when using the quick query tab.

Editing Data
If the table selected in the quick query tab is a FactorySQL compatible table (that is, has a valid index row specified) it is editable in the results table. To edit the data, double click on a cell and change the value as desired. After changing cells, or hitting enter, the **Apply Changes** and **Discard Changes** buttons will become active. At this point, the changes have not yet been applied to the database. Click the Apply Changes button to commit the values.
Alternatively, you may select the **Auto-apply** check box in order to automatically write changes to the database as they occur.

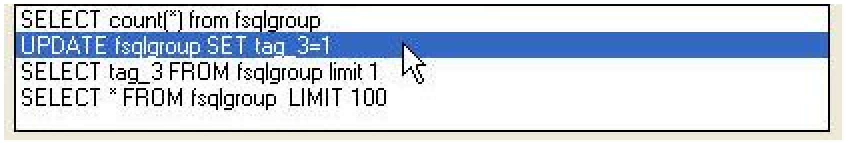
Query



```
UPDATE fsqlgroup SET tag_3=1
```

The Query tab allows you to enter a free form query against the selected connection. This query does not need to be a SELECT statement, it can be any statement that is valid for the target database.

History

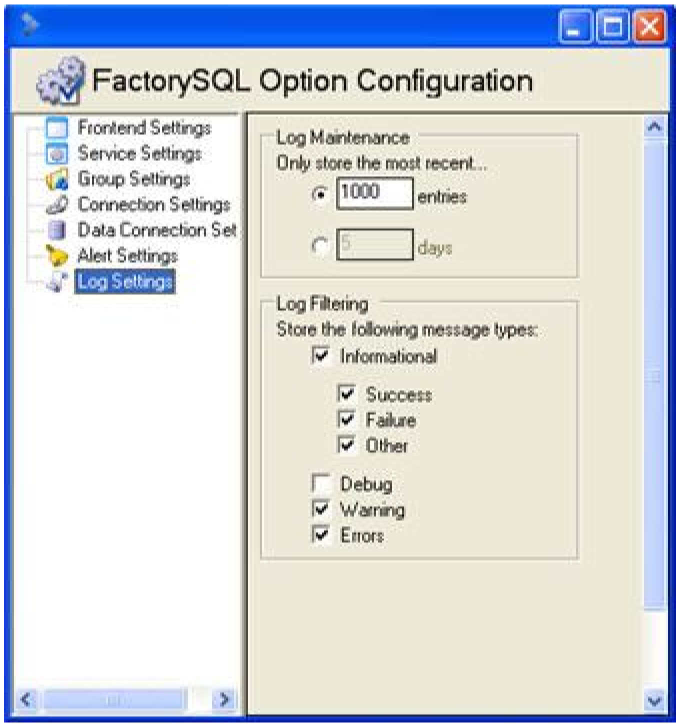


```
SELECT count(*) from fsqlgroup
UPDATE fsqlgroup SET tag_3=1
SELECT tag_3 FROM fsqlgroup limit 1
SELECT * FROM fsqlgroup  LIMIT 100
```

This tab shows a list of the most recently run queries, including those generated by the Quick Query tab. You may double-click on an entry to load it into the query tab so it may be run again.

Log Settings

The **Log Settings Page** allows you to configure the various settings for the FactorySQL Error/Event logging system. The logging system log errors and messages to the internal database, here you can specify what to log, and how long to keep it.



Log Maintenance
Only store the most recent...
(●) 1000 entries
( ) 5 days

Log Filtering
Store the following message types:
☑ Informational
☑ Success
☑ Failure
☑ Other
☐ Debug
☑ Warning
☑ Errors

Log Maintenance

Store X entries: Will only store the specified number of messages.

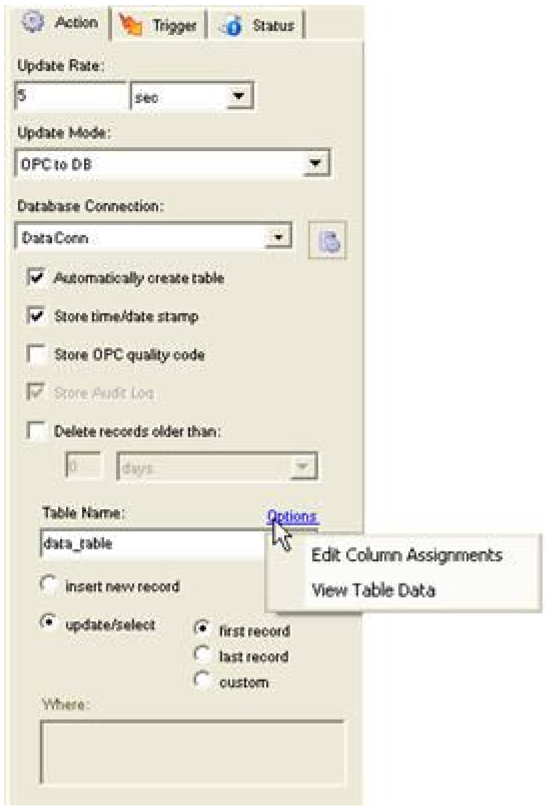
Store X days: Will store the specified number of days worth of events, no matter how many there are.

Log Filtering

Specifies what type of messages to store. The default set includes everything except for Debug messages, as there can be a great number of debug messages generated.

Table Options

The **Table Options** link provides access to 2 useful functions: viewing table data, and editing column assignments.



Update Rate: 5 sec

Update Mode: OPC to DB

Database Connection: DataConn

[✓] Automatically create table

[✓] Store time/date stamp

[ ] Store OPC quality code

[✓] Store Audit Log

[ ] Delete records older than: 0 days

Table Name: data_table

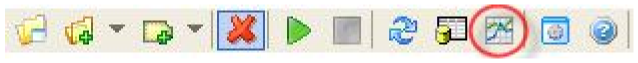
( ) insert new record

(●) update/select — (●) first record ( ) last record ( ) custom

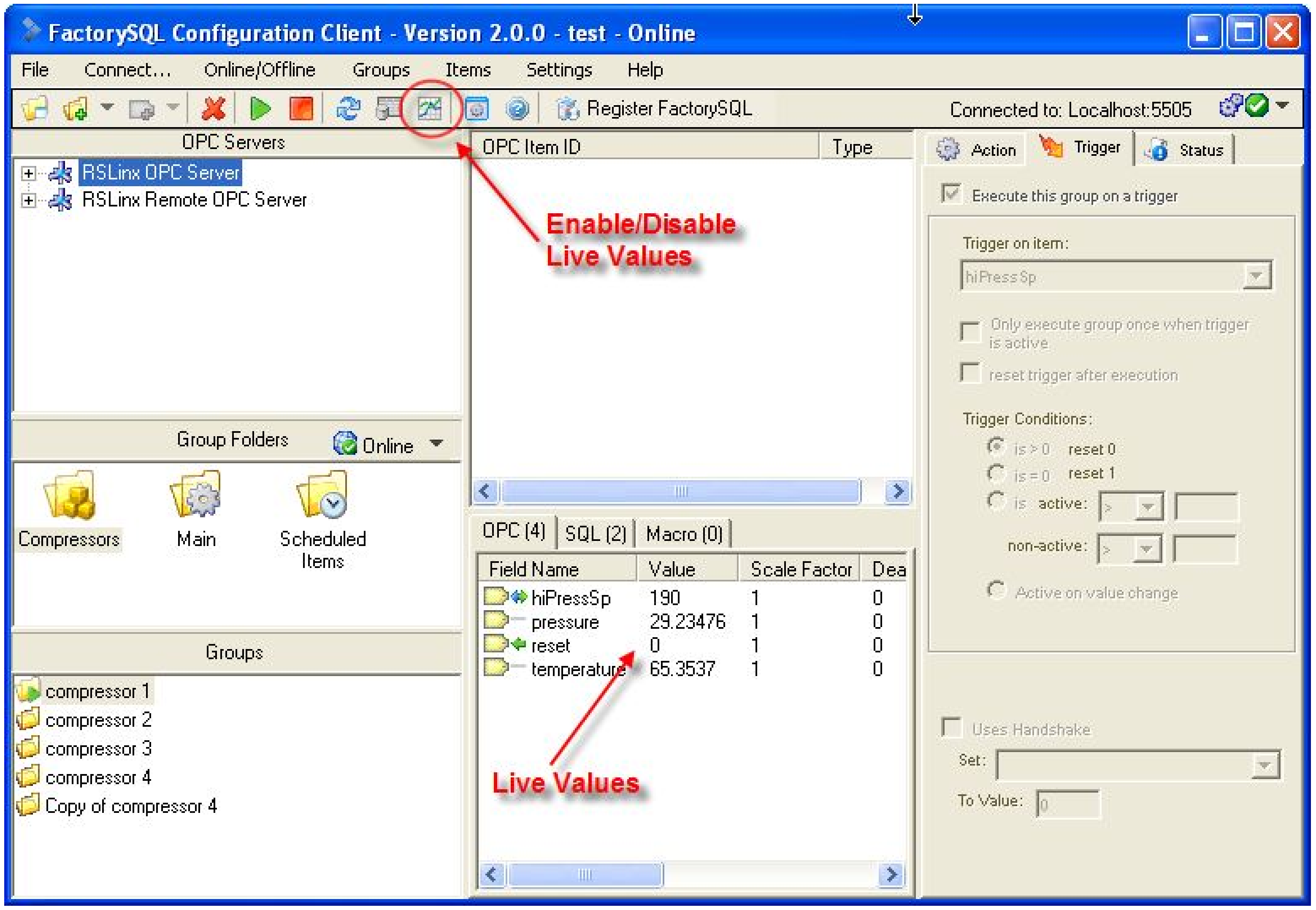
Where:

Edit Column Assignments: Allows you to change the default column for several FactorySQL functions. There are currently 4 columns that FactorySQL looks for by default: The table index, the timestamp, the quality code, and the audit log column. The necessity of these columns depend on the group settings (for instance, you cannot set the audit log user column if auditing is not selected in the group). See Table Column Assignments for more information.

View Table Data: This will bring up the FactorySQL query browser, allowing you to view the data in the currently selected table. From there, you may run other queries or adjust the data. See Table Data for more information.



The **Show Live Values** Icon will display the current value of PLC items in transactional groups.



Field Name	Value	Scale Factor	Dea
hiPressSp	190	1	0
pressure	29.23476	1	0
reset	0	1	0
temperature	65.3537	1	0

[✓] Execute this group on a trigger

Trigger on item: hiPressSp

[ ] Only execute group once when trigger is active

[ ] reset trigger after execution

Trigger Conditions:

[● is > 0 reset 0]

○ is = 0 reset 1

○ is active: > non-active: >

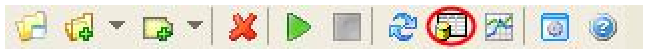
○ Active on value change

[ ] Uses Handshake

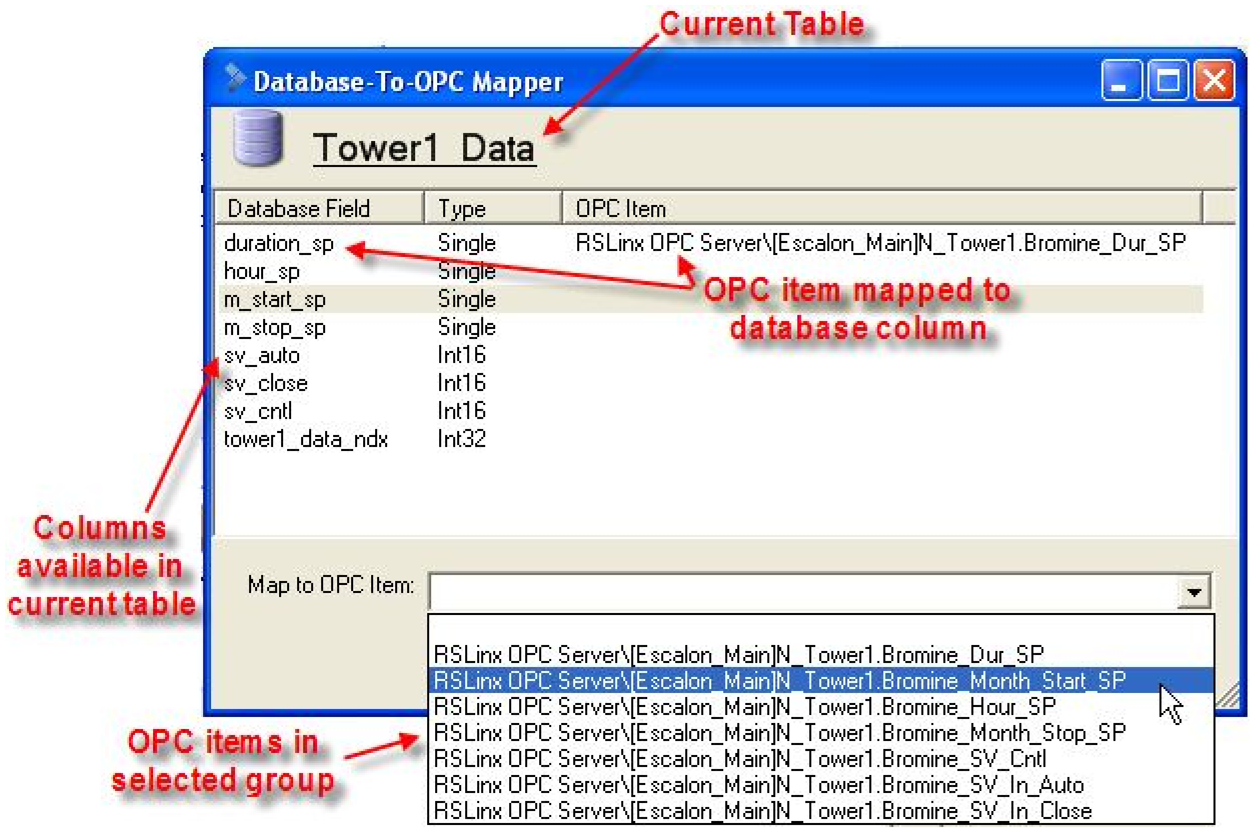
Set:

To Value: 0

Note: The **Live Value Update Rate** is set under frontend settings.



The **DB to OPC Mapper Tool** allows you to quickly map an existing table to your group's OPC items.
When you click the button, the mapper will read the database table specified in your Group Settings, and create a list of available columns.
The left side of the window shows the column name. The right side shows the OPC address of the item associated with that column. To change a mapping, simply select the column you want, and then choose the OPC address to map it to from the list below. When you click the OK button, the window will close and the Field Names of your items will change to reflect the new mapping.


Current Table
Database-To-OPC Mapper
Tower1_Data
OPC item mapped to database column
Columns available in current table
OPC items in selected group
Map to OPC Item:

Database Field	Type	OPC Item
duration_sp	Single	RSLinx OPC Server\[Escalon_Main]N_Tower1.Bromine_Dur_SP
hour_sp	Single	
m_start_sp	Single	
m_stop_sp	Single	
sv_auto	Int16	
sv_close	Int16	
sv_cntl	Int16	
tower1_data_ndx	Int32	

RSLinx OPC Server\[Escalon_Main]N_Tower1.Bromine_Dur_SP
RSLinx OPC Server\[Escalon_Main]N_Tower1.Bromine_Month_Start_SP
RSLinx OPC Server\[Escalon_Main]N_Tower1.Bromine_Hour_SP
RSLinx OPC Server\[Escalon_Main]N_Tower1.Bromine_Month_Stop_SP
RSLinx OPC Server\[Escalon_Main]N_Tower1.Bromine_SV_Cntl
RSLinx OPC Server\[Escalon_Main]N_Tower1.Bromine_SV_In_Auto
RSLinx OPC Server\[Escalon_Main]N_Tower1.Bromine_SV_In_Close