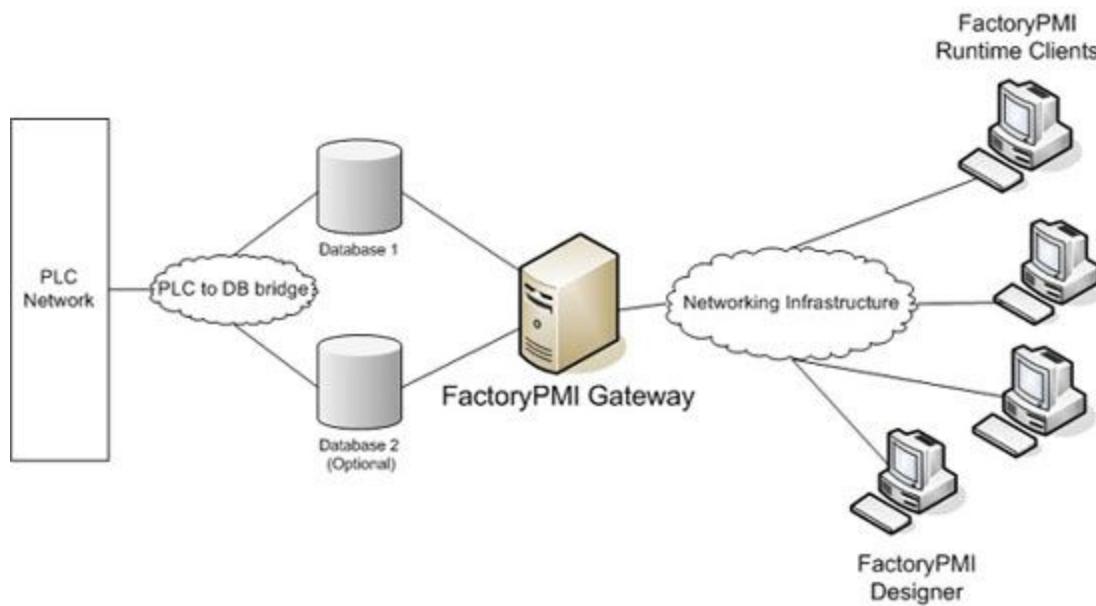




Welcome to FactoryPMI 3 featuring SQLTags™!

FactoryPMI is a powerful and unique platform for creating HMI/SCADA systems, and so much more. The key to FactoryPMI's advantage is its **web-based deployment** and **database-centric architecture**. This allows properly authorized users to launch the client from anywhere, and allows you to incorporate any type of database data into your projects.

There are three main components to the FactoryPMI architecture: The *Gateway*, the *Designer*, and the *Runtime*. The gateway is the central clearing house that allows you to launch clients, the designer, and to communicate with the databases. The following diagram shows an overview of the system:



There are several important areas that are simplified in the image, but warrant discussion (going from left to right):

PLC to Database Bridge: Since FactoryPMI talks to databases, a separate program must be used to communicate with PLCs. For this purpose Inductive Automation offers [FactorySQL](#), a high performance and versatile PLC to DB bridge. In many ways this separation is actually advantageous, as it allows a much finer level of control, not to mention wider range of features, than is normally available in systems that communicate with PLCs directly.

Database(s): There are 2 databases shown to illustrate that FactoryPMI is not restricted to one database, or its own database formats. You can connect FactoryPMI to any number of databases throughout your organization, allowing you to draw together many sources of data in one spot.

Networking Infrastructure: FactoryPMI is *web-deployed*, meaning that the designer and all clients are launched through Internet technologies. This means that you can use your existing network infrastructure to restrict and control access. FactoryPMI is also capable of working with Microsoft ActiveDirectory® to restrict access.

Help File Organization

This help file is organized in 4 main sections:

Introduction

This section provides the basic information needed to get started, including how to install and activate (if necessary). Also, it provides a quick start guide that walks you the steps that would be necessary to get a simple project up and running on a fresh installation.

Gateway

This section provides information on the Gateway, including how to add and configure new projects, how to set up database connections, and authentication profiles.

Designer

This section explains the designer, which is used to build FactoryPMI projects.

Technical Reference

This section provides a more straight-forward technical reference of the advanced technologies that you can use inside of FactoryPMI.

Extended Help

As no manual can fully cover every conceivable situation or topic, it's important that you know where to go for answers. The first and best place is the [Inductive Automation Website](#), where you can peruse the issues and questions that other users have encountered.

From there, you may [E-mail Us](#). We strive to provide a quick turn around on answers - usually within 24 hours. Finally, registered users may call us toll-free at 1-800-266-7798.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



How to Install The Gateway

FactoryPMI's installation is straightforward, much like other windows installations you have performed. Simply run the installation executable, and follow the instructions. Note that this installer installs the FactoryPMI Gateway, which is a Windows service. The FactoryPMI Designer and Runtime aren't installed traditionally, rather they are web-launched by the Gateway. See [Project Launch Page](#) for details.

The main configuration for FactoryPMI will be done after the installation, from the web based [Gateway Configuration Page](#). The FactoryPMI Gateway is a *web server*, so you access it through a web browser like Internet Explorer or Firefox. Once you install the Gateway, you simply point your web browser to: `http://localhost:8080`.



Fig. 1: You must agree to the FactoryPMI license terms before installation.

Got Linux?

FactoryPMI Gateway is also available for Linux! Download the tarball from [Downloads Page](#) and read the README for details.

localhost? 8080?

Computers use the name "localhost" as a simple way to link to themselves. Whenever your computer makes a connection to "localhost", it routes the connection to itself. This is the same thing as the special **ip address** 127.0.0.1.

The ":8080" means to connect over **port** 8080. Most web traffic goes over port 80. For example, if you go to `http://www.amazon.com`, it is equivalent to `http://www.amazon.com:80`. FactoryPMI installs itself on port 8080 by default, to avoid collision with **IIS**, which is often installed on port 80 already. You can change FactoryPMI's port to 80 for convenience after installation. [See the System Settings page](#) for details.

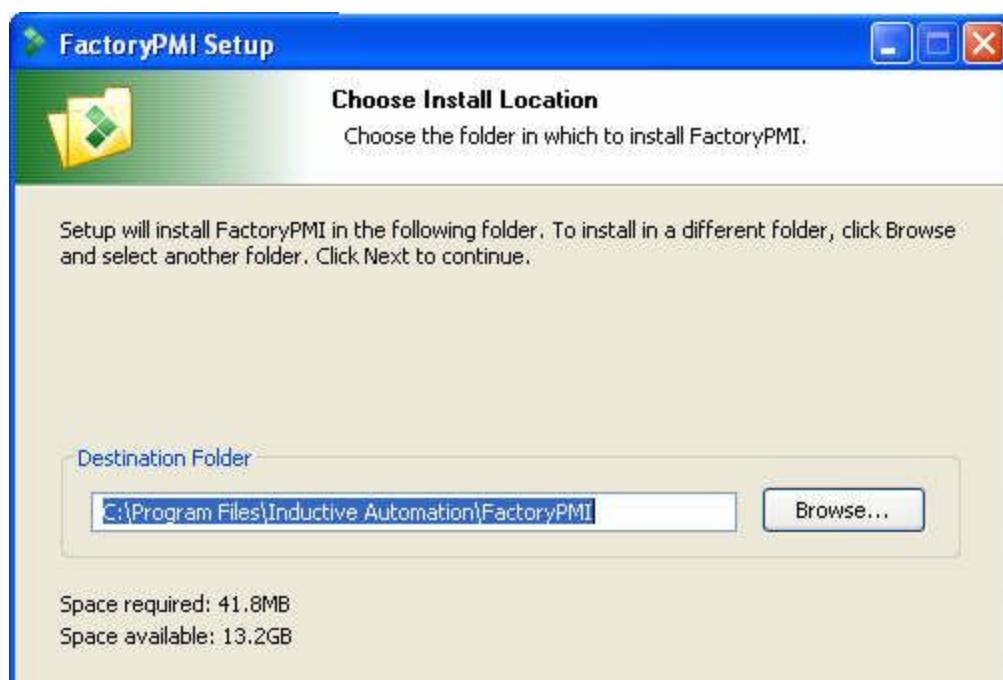




Fig. 2: Location where files will be installed. Requires about 40 MB of disk space.

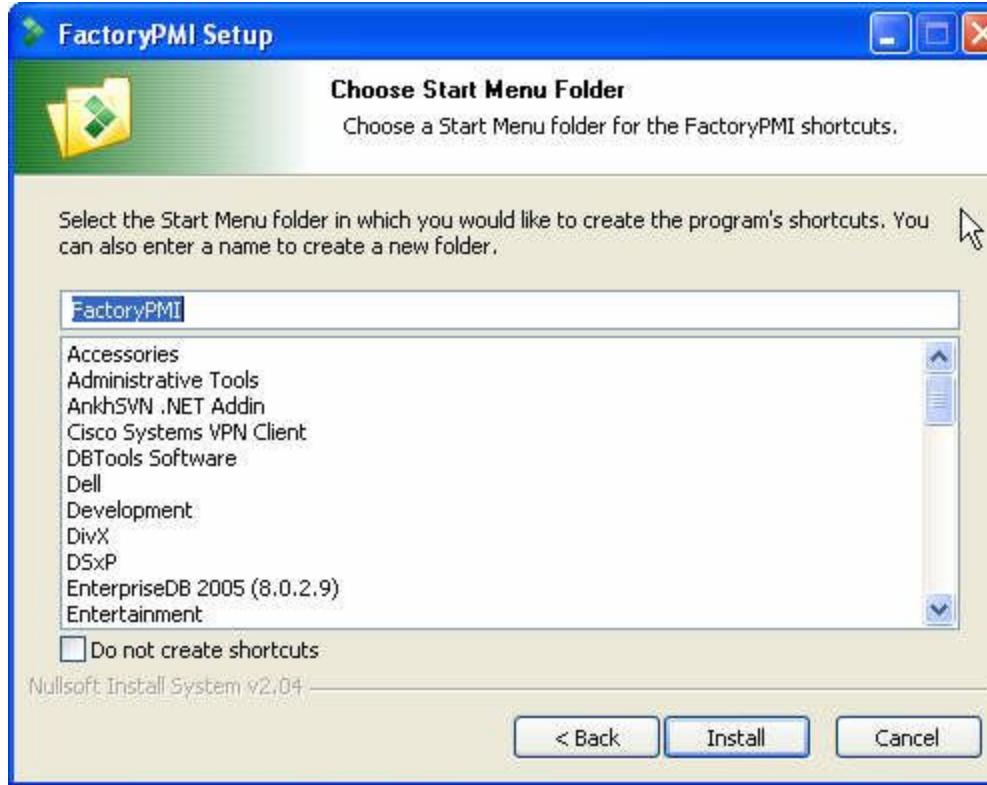


Fig. 3: Start Menu Location

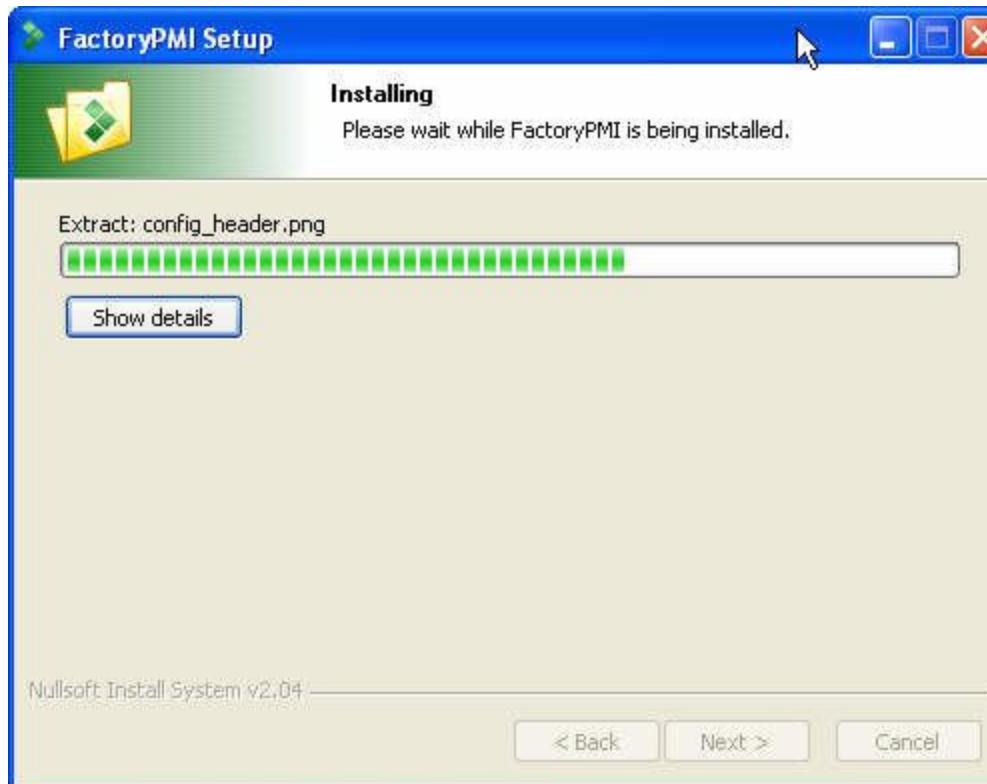
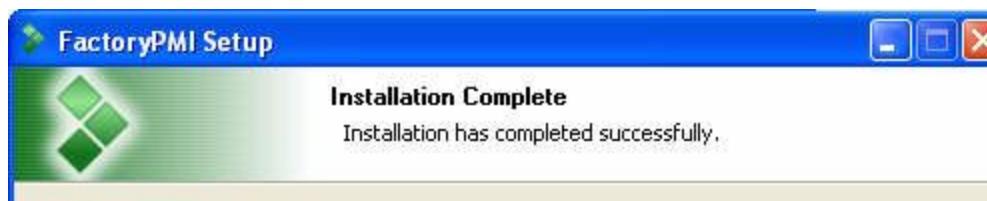


Fig. 4: Install Progress. Install will take a minute or two on most machines.



Thank you for installing FactoryPMI. Installation is now complete. After starting the service, you can access the launch page at <http://localhost:8080>. The default HTTP port is 8080, and the initial username/password for the configuration section is admin/password. Please change the password to something more secure as soon as possible.

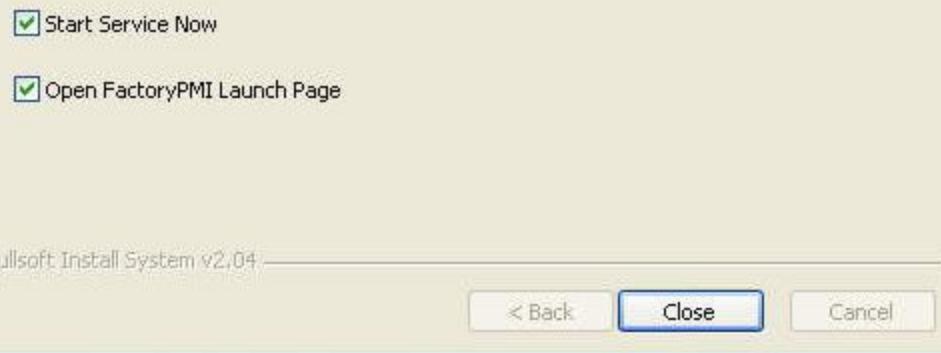


Fig. 5: After installation is complete, you can start the service and launch the FactoryPMI launch/configuration page. The configuration page can only be reached if the service is running. For information on configuring the service, visit the [Gateway section](#).

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

About Trial Mode

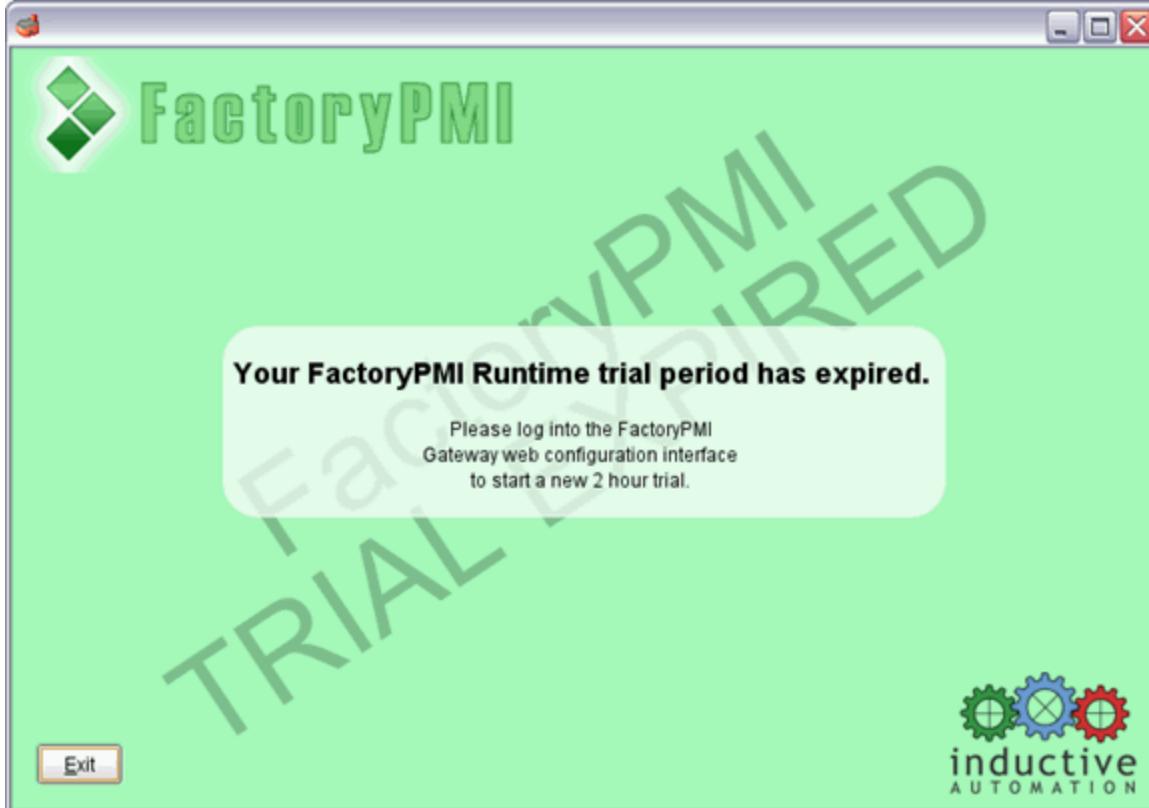
FactoryPMI's trial mode provides a way for you to try our software without any feature restrictions. This allows you to fully evaluate our software to make sure that it's right for you.

The only restriction in the trial mode is that you get two hours of Client runtime at a time. This means that whenever you have a Client running, your two hour trial window is decreasing. When your two hours run out, any running Clients will stop running, and you will have to log into the FactoryPMI Gateway and start a new 2-hour trial. You can do this as many times as you want, which means that you can evaluate it for as long as you want! This system gives you flexibility to evaluate our product, while making it impractical for industrial use.



Running FactoryPMI Designer *does not* cause your trial window to decrease. In fact, the Designer isn't affected by the trial mode at all. This means that you can design an entire project on an un-activated FactoryPMI Gateway.

When your running clients run out of trial time, they look like this:



Visit the FactoryPMI Gateway to start a new 2-hour trial window.

FactoryPMI Gateway Configuration
PLANT MANAGEMENT INTERFACE

Logged in as: admin [logout] [restart] [shutdown]

System Status

System Status

System Status	
Version	1.4.6 (build 617)
Java Version	Sun Microsystems Inc. 1.5.0_04
Memory (used/max)	33.45mb / 254.06 mb

Types
Authentication
Profiles
Types
Auditing
Profiles
Images

Running Since Thu Dec 08 14:10:26 PST 2005

Uptime 1 days, 2 hours, 45 minutes, 12 seconds

 Launch Designer



FactoryPMI v. 1.4.6 is © 2002-2004 by Inductive Automation. All rights reserved. [\[view license\]](#)



Runtime trial period has expired

[Reset Trial Period](#)

Click here to start a new 2-hour trial window



This server is running in Trial Mode. **Activate Now!**

Your 2-hour Runtime trial period has expired. [Log in](#) to reset it. [What does this mean?](#)

For information on how to register, see [Registration and Activation](#).

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Registration and Activation

Registration and Activation is the process by which you activate your purchased copy of FactoryPMI in order to remove the time limitation. **Registration** refers to the act of acquiring a CD-Key, and entering it in the program. **Activation** refers to notifying Inductive Automation that you are using that cd key for an installation, and in most cases can be done instantly over the Internet.

To begin the process, click one of the "Activate Now" links, such as the one on the bottom of the gateway pages:

Mode. [Activate Now!](#)



es. What does this mean?

This will bring you to the following page, where you may choose to purchase FactoryPMI if you do not have a CD-Key (redirects you to our website), or continue with activation.



The following screen asks which method you would like to use for activation: Internet or phone. If the computer you are installing on has an Internet connection, choose Internet, as it is the quickest way. If the installation computer does not have an Internet connection, click Phone. (NOTE: If a different computer near by has an Internet connection, you may complete the Phone activation yourself by going to InductiveAutomation.com/Support/Activate)



Activate Step 1: Choose Method**Activate Over the Internet**

If you have an internet connection, this is the easiest way to activate.

**Activate Over the Phone**

If you don't have an internet connection, you can activate over the phone.

FactoryPMI v. 1.0.0_17 is © 2002-2004 by Inductive Automation. All rights reserved. [view license]

Internet

Irregardless of the method you selected, the following screen will ask you to enter the cd-key. The cd key can be found on the inside insert of the cd case, or if you purchased online, it was displayed at check out and emailed to the address you provided. It is a 6 character string, formatted like XXX-XXX.

FactoryPMI Activation Explained - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites Media Go Links

Address http://192.168.1.129:8080/Activate?step=2&method=phone

FactoryPMI Gateway
PLANT MANAGEMENT INTERFACE

Activate Step 2: CD-Key

Type your CD-Key:

FactoryPMI v. 1.0.0_17 is © 2002-2004 by Inductive Automation. All rights reserved. [view license]

Internet

Phone Activation

The following screen will appear:

FactoryPMI Activation Explained - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites Media Go Links

Address http://192.168.1.129:8080/Activate

FactoryPMI Gateway
PLANT MANAGEMENT INTERFACE

Activate Step 3: Call For Activation

Please call **1-800-266-7798** to speak to an Inductive Automation customer representative. They will need the **Install-Key** listed below. Enter the **Activation-Key** given to you into the box below and hit 'Activate'

Activation Information:	
CD-Key	XXX-XXX
Install-Key	XXXX-XXXX-XXXX
Activation-Key	<input type="text"/>

FactoryPMI v. 1.0.0_17 is © 2002-2004 by Inductive Automation. All rights reserved. [\[view license\]](#)

Done Internet

When you call, or when you go to the support website, you will be asked for the Temporary Install key. This key is different each time you come to this page, so DO NOT CLOSE or refresh the browser window until you have finished, or you will have to start over. After giving this key, you will be given the Activation Key in response. Enter it in the box, and click "Activate".

If everything is successful you will see a thank you screen. Otherwise, you will be given an error message and a hint at what might be wrong.

Internet Activation

Internet activation will try to contact our server and process the request. If everything goes well you will receive a thank-you screen. Otherwise you'll be given an error message.

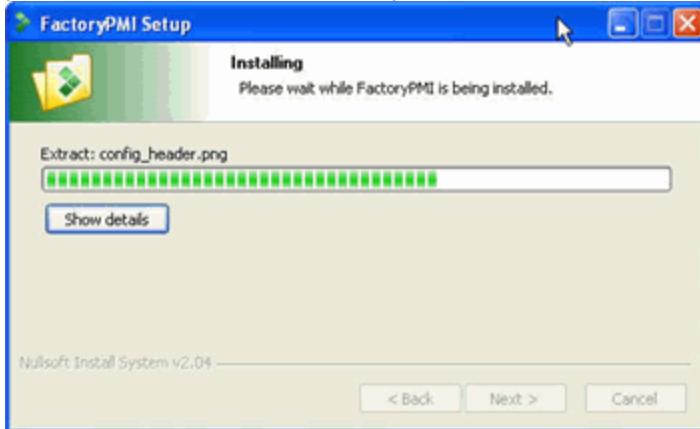
If Internet activation fails several times, try activating over our website at support.inductiveautomation.com

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Quick Start

This guide steps you through the bare minimum to get to designing and launching your first project.

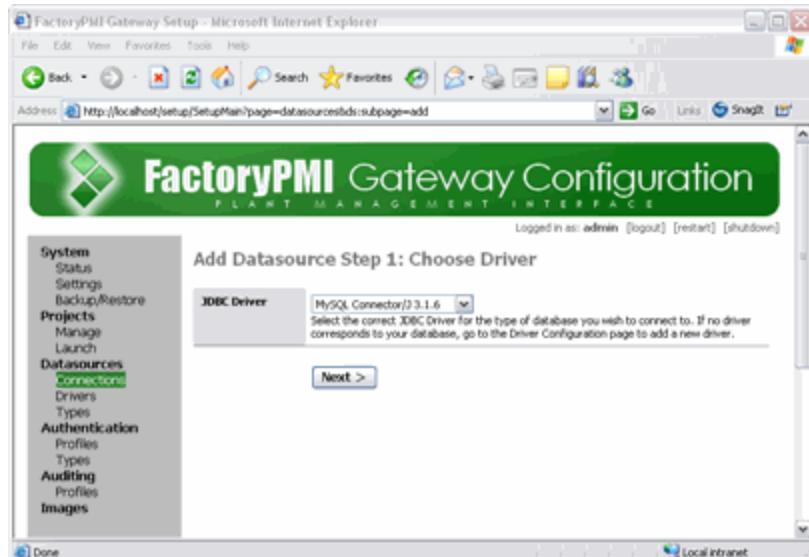
1. **Install FactoryPMI.** Using your Install CD, install FactoryPMI. See [How to Install](#) for more details.



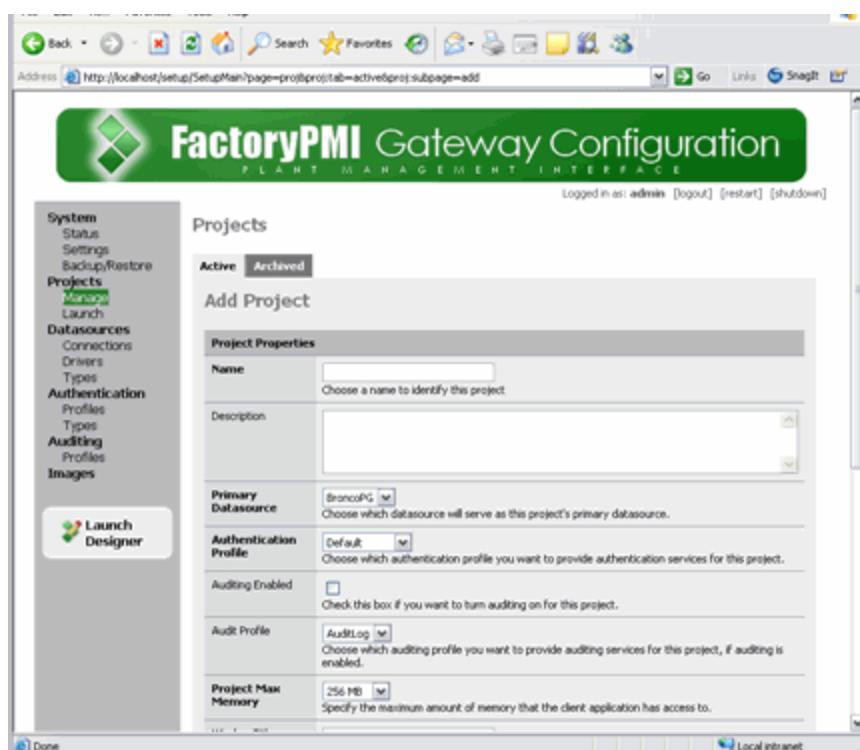
2. **Login to FactoryPMI Gateway.** Browse to your FactoryPMI Gateway Configuration website by pointing your web browser to <http://localhost:8080/setup>. The username/password is **admin / password**



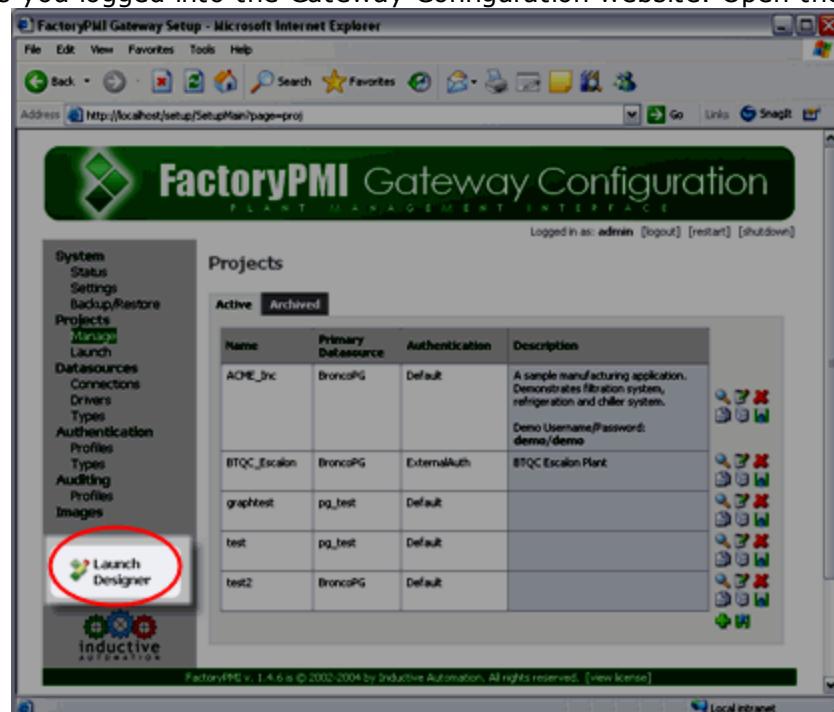
3. **Add Datasource Connection.** Click on the **Datasources / Connections** link. Click the green "add" button. Fill in the parameters to connect to your datasource. You will need to restart FactoryPMI after adding the datasource.



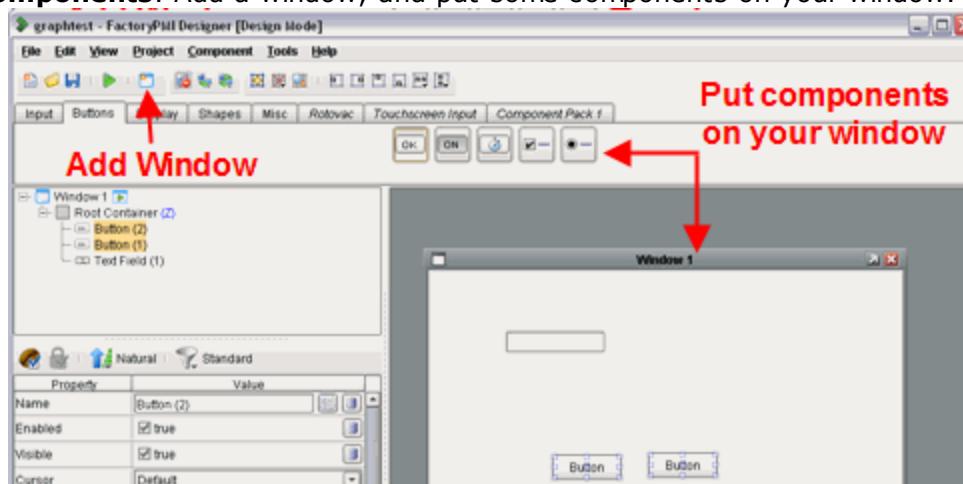
4. **Add Project.** Click on the **Projects / Manage** link. Click the green "add" button. Give your project a name.



5. **Launch Designer.** Click on the Launch Designer link. When it loads, log into the designer with the same username/password as you logged into the Gateway Configuration website. Open the project you created.



6. **Add Window, Components.** Add a window, and put some components on your window.



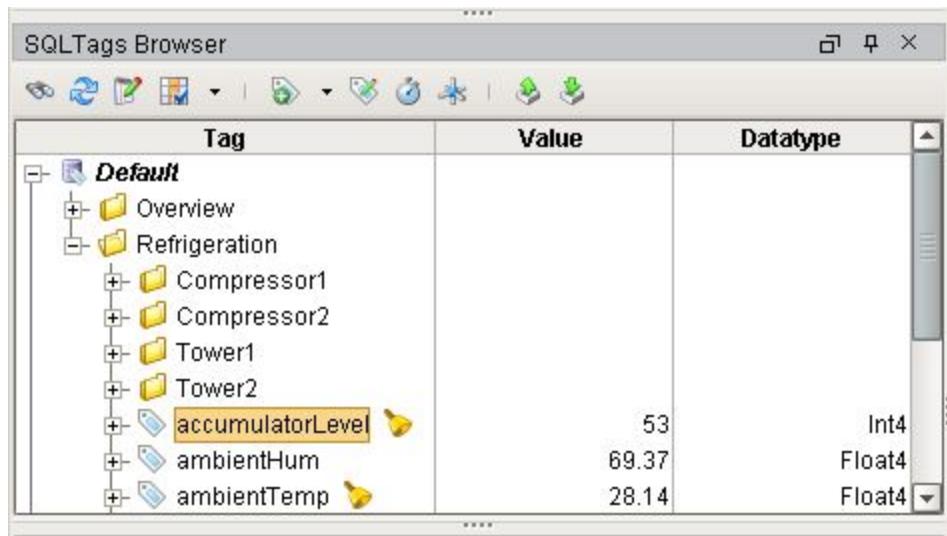


7. **Save Project.** File->Save (Ctrl+S) or click the save icon .
8. **Launch Project.** Go back to the FactoryPMI Gateway Launch Page <http://localhost:8080>. You will see your project listed there. Click on it to launch the client. That's it!

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

SQLTags™ Introduction

SQLTags™ is a technology that simplifies HMI/SCADA development within FactoryPMI making status and control a snap. It turns any SQL database into a high performance industrial tag database. Tags are driven by OPC, or even by expressions or SQL query results.



Did you know...

FactoryPMI 3.0 is fully backward compatible. You don't *have* to use SQLTags™, but they will make life easier!

Ease of use

While your tags reside in your SQL database, giving you state-of-the art performance and reliability, the SQLTags Browser in FactoryPMI gives you a [drag and drop](#) interface to your SQLTag database. Object creation, property binding, and tag editing has never been easier. You don't need to know any SQL in order to use SQLTags!

Performance and Scalability

SQLTags™ creates a well engineered database schema in any SQL database. Adding concurrent FactoryPMI runtime clients will not increase SQL database or OPC load. Several thousand **changing** tags per second is not unreasonable on a modest PC. Unchanging tags have a negligible effect on performance. This means that systems with 500,000 tags are perfectly reasonable, or even more depending on the infrastructure running the database.

Integrated component feedback

SQLTags™ features a *quality* and *overlay* system for FactoryPMI component status. If a tag's data quality is anything but good, a component that depends on it will get a visual overlay. Input components display an animated overlay while write pending requests are being written.

Enhanced functionality

System SQLTags™ provide easy access to status that is more typically obtained with script. Any object can easily be used to display realtime information such as: current date/time, username, IP address, etc. Bi-directional support makes it easy for components to interact with the database or PLCs. Metadata for each tag adds units, tooltips, formatting, value ranges, and other useful information.

How Does it Work?

SQLTags™ creates a tag database on a FactoryPMI [datasource](#). FactorySQL keep this database updated including OPC and expression evaluation and alerting. Multiple FactorySQL installations can drive one SQLTags-enabled database, with each FactorySQL instance driving a subset of the tags.

SQLTags™ features come up in many sections of this user manual:

- [SQLTags Overview](#) - A good starting point

- [SQLTags Browser](#) - Main SQLTags panel
- [SQLTags Editor](#) - Edit SQLTags
- [Scan Classes](#)
- [Component Overlays](#) - Describes component status feedback for users
- [Drag and Drop operations](#)

Also new in FactoryPMI Version 3.0:

[FactoryPMI Client Diagnostics](#)

[Component Quality codes/overlays](#)

Components:

[1-Shot Button](#)

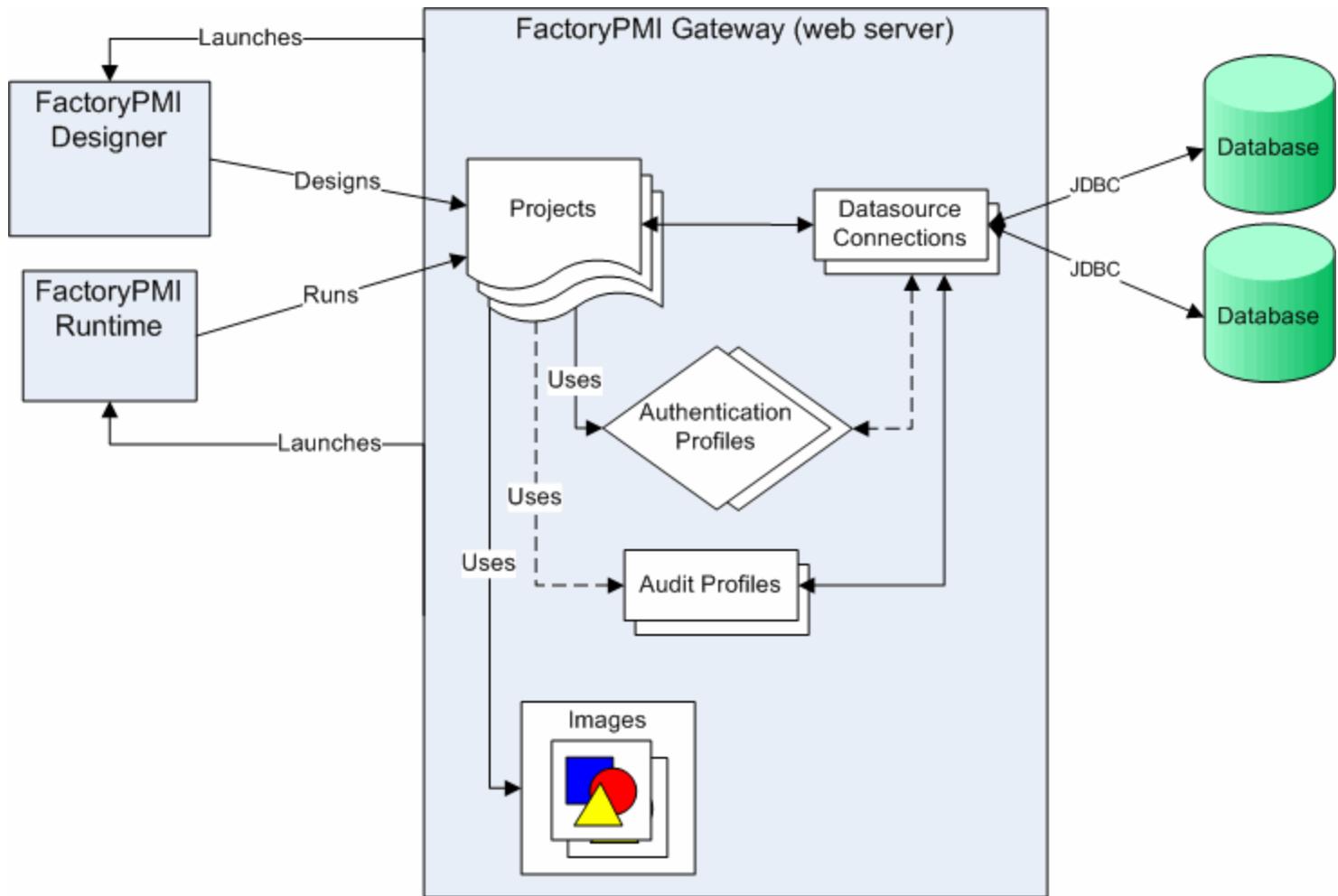
[2 State Toggle Button](#)

[Multi-State Button](#)

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Gateway Overview

The FactoryPMI Gateway (or "Service" as it is sometimes called), is the core of the overall system. It is the central repository that stores the projects, connection settings, authentication settings, and the location from which clients are launched.



The gateway is run as a windows service, holding a small internal database, and running its own small web server. By default, the webserver is configured to run over port 8080, so you can reach it by going to "<http://localhost:8080>". The use of port 8080 is to reduce the chances of conflicting with another web server, such as IIS, that might be running on the computer. However, if this port is already in use, the server will not be able to function, and the port will need to be changed. To change the port, use the [FPMI Gateway Utility](#), found under the FactoryPMI start menu folder.

When you open the gateway's address, you'll see the [Gateway Launch Page](#). This page contains links to launch a project, open the designer, or access the gateway configuration page.

The gateway has many, many configurable options, which are covered throughout this section of the help file. In general, configuration "out of the box" should work fine, but you'll want to learn more about all of the full possibilities soon.

At bare minimum, to get the project going quickly, you'll need to [Create a Datasource Connection](#), and [Create a New Project](#)

Project Launch Page

The project launch page is the first page you see when you go to the Gateway's web page. It serves as the central "home" for everything FactoryPMI can do. You open the launch page in a web browser by opening <http://localhost:8080>

By default the *Project Launch Page* listens over all IP addresses on all adapters. The default port is 8080 and may be changed in the [configuration page](#).



Default Launch works by clicking on the application name or logo. The launch method and Optional Launch Method links are set in the [project property](#) of the [configuration page](#). Options are the next 3 launch modes. Default launch is Web Start.

Applet runs inside the current browser window. It's suitable for small applications or embedding in a corporate intranet site.

Web Start (Windowed) is the typical application for a desktop machine. The project will open in a new window. Java Web Start will prompt the user to integrate the project into the user's desktop and start menu, allowing the user to launch the without having to open a web browser.

Full Screen (Web Start) opens the project as if it is the only application running on the computer. Typical windows and borders do not show up. This is the ideal application for touchscreen computers to replace PanelView type control terminals.

Click [here](#) to see an example of different launch methods.

Java Detection is covered [here](#). The links at the top of the page allow easy access to other features:



Launches the FactoryPMI Designer application via Java Web Start. The Designer is the FactoryPMI project GUI creation tool.



Opens [Gateway Status](#) page, where you can find useful statistical and diagnostic data about the Gateway web server.



This link brings you to the Gateway Configuration section of the Gateway's web interface, which is where all of the configuration of things like projects, datasources, authentication, auditing, etc is done.



Opens the web based help file.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Launching Projects

Launching projects is simply a matter of clicking a link on the Project Launch page. It is often best to create a Windows shortcut.

FactoryPMI Launch paths:

- **Java Web Start** - <http://IP:PORT/gateway/launch/PROJECT.jnlp>
- **Full Screen** - <http://IP:PORT/gateway/launchfs/PROJECT.jnlp>
- **Applet** - <http://IP:PORT/gateway/applet/PROJECT>
- **Designer** - <http://IP:PORT/gateway/designer/designer.jnlp>



FactoryPMI Launch Page - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost:8080/Launch

Back Forward Stop Home Search Favorites Refresh Mail Print Find Help SnagIt

FactoryPMI Gateway

PLANT MANAGEMENT INTERFACE

Launch Designer Gateway Status Configuration Help

Projects

Demo Application Inductive Automation Demo Application

Launching a project
Click Here

Full Screen

Applet

Skeleton

TrainingPants Windowed

docs_test

tpp2 Windowed

Local intranet

Launch method examples:



FactoryPMI Launch Page - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Back Forward Stop Home http://localhost:8080/Launch Yahoo

FactoryPMI Gateway

PLANT MANAGEMENT INTERFACE

Launch Designer Gateway Status Configuration Help

Projects



Sample Application

Sample Application

Windowed

FullScreen

Applet

Applet



The correct version of the Java™ Platform has been detected on your computer!



Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Gateway Monitoring/Control Utility

The gateway monitoring tool is a useful utility that allows monitoring and control of the FactoryPMI Gateway service. It allows you to change a few crucial aspects of the server as well. It can **only** be run locally on the FactoryPMI gateway computer.

To launch the utility, click the shortcut under **Start -> Programs -> FactoryPMI -> Monitor FPMI Gateway**.



Gateway Status indicates the status of the service, cluster position, port, and software version.

Gateway Control allows you to start and stop the service. This equivalent to typing *net start/stop factorypmi* in a command prompt.

Gateway Configuration

Menu Item	Function
Launch Main Page	Opens the default web browser to the FactoryPMI Project Launch Page
Launch Config Page	Opens the default web browser to the FactoryPMI Gateway Configuration Page
Reset Password	Resets the default password to user:admin password:password
Set Port	Changes the TCP port that FactoryPMI uses. The default is 8080
Remove Config Page IP Restrictions	Allows the Gateway Configuration Page to be accessed from any IP address



Java, Web Start, and Applets

This page goes into some of the more technical details of how the FactoryPMI launches runtimes and designers. If you are a new user, feel free to skip this page for now. It is meant for users who have some familiarity with the system, to help them overcome some of the issues that arise with web deployment.

As you have assuredly realized by now, FactoryPMI is based on Java. It uses two Java technologies to achieve web-deployment: **Java Web Start** and **Applets**. The main deployment issue that every FactoryPMI user will have to face is: do the client computers that I want to deploy my FactoryPMI project on have Java, and if so, what version? You'll find that this question is a bit more involved than it might seem. It is complicated by issues such as computers with multiple versions of Java installed, computers that have other Java programs running that require versions of Java other than what FactoryPMI requires, etc. FactoryPMI has a number of launching features that aim to ease the headaches involved, which are your first tool in diagnosing launch problems.

The Detection Applet

(Note: This guide is assuming that Windows is being used). **FactoryPMI requires the Java Runtime Environment Version 5.0 (also known as 1.5.x) or 6.0 (also known as 1.6.x)**. It can be downloaded from Sun Microsystems at <http://www.java.com/en/download/manual.jsp>. The installer is also packaged with FactoryPMI Gateway, and can be found at <http://serveraddress/JavaInstall>. The **Java Version Detection** Applet is a very small Applet that will run in most versions of Java. It is displayed on the Gateway's launch page to detect the version of Java that is running on the machine viewing the page. If everything is working correctly, it will look like this:



The correct version of the Java™ Platform has been detected on your computer!

Note that you can click on the image (which is the applet) and more information about your Java version will be shown. If Java is installed, but the wrong version is installed, the Detection Applet will look like this:



If you don't see the Java™ logo with a green check on it, the correct version of Java was not found on your computer. FactoryPMI needs Java in order to launch a project. [Click here](#) for help installing Java.

And if no Java is installed, or a very old version of Java is installed, the Detection Applet will look like this:



If you don't see the Java™ logo with a green check on it, the correct version of Java was not found on your computer. FactoryPMI needs Java in order to launch a project. [Click here](#) for help installing Java.

The detection applet is accurate 95% of the time in **Internet Explorer 6**, however there are some situations that will cause it to be inaccurate:

The applet is displaying the "wrong version" image, but I have Java Version 5.0 installed! This can be caused by having multiple versions of Java installed, especially if you viewed the launch page, installed Version 5.0, and then viewed the launch page again without restarting Internet Explorer. Projects will still launch correctly.

The applet isn't loading (displaying the "no java"), but I have Java Version 5.0 installed! This is caused by having a custom Java installation that didn't install support for the <Applet> tag in Internet Explorer. Projects will still launch correctly using the Web Start method. You can turn on <Applet> support by going to **Start > Control Panel > Java > Advanced > "<APPLET> tag support"** and checking the "Internet Explorer" check box.

If you are deploying in a scenario where the detection applet is giving false negatives too often, you can simply turn it off in the [System Settings](#) page.

Installing Java

If you need to deploy on a computer that doesn't have Java version 5.0 installed, you will need to install it. If you have Auto-install (see next paragraph) turned off, and you go to that computer and try to launch a project or the designer, the launch links will detect that you don't have Java, and let you know. The detection routine in the launch links is *very accurate* on Internet Explorer 6. On other browsers, however, it doesn't work, and always lets you try to launch the project. See [Java Web Start and Applets](#) for more info about what may happen on other browsers.

In order to install Java Version 5.0 on any computer, you need to have administrator privileges on that computer. If you don't, you will need to work with the IT department in order to have them install Java for you. Your main Java installation option is whether to install manually by going to <http://serveraddress/JavaInstall>, or to let FactoryPMI auto-install Java when the user tries to launch a project without the proper version of Java installed. Auto-install is turned off by default; you can turn it on in the [System Settings](#) page. Auto-install only works on Windows, running IE6 with ActiveX controls enabled, and the user has to have Administrator privileges on their computer.

Java Web Start and Applets

FactoryPMI Gateway uses one of two Java technologies to achieve web-deployment: **Java Web Start** or **Applets**. Applets let your FactoryPMI project run within the web browser, at a fixed size. A project is launched in an applet by going to the Launch page, or navigating directly to the URL: <http://serveraddress/gateway/applet/projectname>. Applets are usually useful for small, dashboard-like applications. Java Web Start is the preferred launch method for FactoryPMI.

Java Web Start is a full-fledged application web-deployment technology. It works by invoking the Java runtime environment on a computer whenever a user goes to a *.jnlp file. This is why the direct launch URL for projects using Java Web Start is: <http://serveraddress/gateway/launch/projectname.jnlp>. When Internet Explorer goes to this link, it knows that Java should be used to launch that file, which is how the project is started.

The first time a user loads a Java Web Start project, it will download all of the JAR files that make up the FactoryPMI runtime. These will only be downloaded the first time - then they are stored on the user's hard drive. The user will be asked to trust the Inductive Automation certificate, because FactoryPMI caches information to the hard drive as well. Finally, on the first load, the user will be given the opportunity to integrate the application with their desktop. This is akin to the familiar act of installing a stand-alone program: it will put a link on their desktop and in the Start Menu. From then on, the user doesn't need to go to the Gateway's launch page to start the project. You can even drag one of these shortcuts into the Windows startup folder so that the application opens automatically on startups (useful for dedicated workstations and kiosks)

If for whatever reason you want to make Java Web Start 'forget' that it has ever seen an application, use the **Java Control Panel**. Launch this by going to **Start > Control Panel > Java**. If you want to clear out everything, click on **Delete Files...** under the **General** tab, and hit **OK**. To remove a particular Java Web Start application, click **Settings** and then **View Applications**. This gives you a detailed view of all of the applications that Java Web Start is aware of. It won't hurt anything to remove these - all it means is that next time you launch that application, you will have to re-download the JAR files.



What is Clustering?

The FactoryPMI Gateway can work in a *cluster*. A cluster is any number of Gateways (on separate physical machines) that are on the same network and are members of the same cluster. These Gateways share all configuration data (projects, datasource configurations, plugins, user sessions, etc). There are two reasons to cluster multiple Gateways together:

Redundancy

Normally if the computer running the Gateway crashes or loses network communication, all running Clients will lose their connection, causing downtime! Clustering reduces downtime by introducing redundancy into the system - as long as one node of the cluster is up and reachable from the Clients, the Clients will remain usable. For more redundancy, simply add more nodes!

Scalability

Clients spread their load out evenly amongst all cluster members. This means that in a 3 node cluster, each Gateway node only handles the request traffic for 1/3 the amount of total running clients. This enables very high concurrent client counts. Gateways getting bogged down in traffic? Simply add another node to the cluster to increase capacity. This is called *scale-out*, as opposed to *scale-up*, which involves buying specialized (expensive) computers to increase capacity.

How do I set it up?

Setting up clustering is very simple. Two Gateways will be in a cluster if:

1. **They are on the same subnet.** (Actually, if they can communicate via *UDP multicast*. A good network administrator should be able to enable multicast communication between different subnets)
2. **They share the same cluster name.** To set the cluster name, go to the Gateway's configuration section, and navigate to `System > Cluster`. Set the cluster name. This will cause the Gateway to restart, and when it starts back up, it will join the new cluster.

When a Gateway starts up, it joins the cluster specified by its **cluster name**. If there are no nodes already in this cluster, the Gateway becomes the *MASTER* node of the cluster. If there are already nodes in this cluster, the gateway joins the cluster. This means that the gateway will assume the state of the cluster's *MASTER* node.

When a Gateway joins a cluster upon startup, it assumes the state of the cluster, overwriting any projects, datasources, authentication profiles, etc in that Gateway. Make sure you join empty (newly installed) Gateways to already-configured Gateways, not the other way around.

To be extra clear, lets look at the steps involved to take a standalone, already configured Gateway and turn it into a 2-node cluster. Lets call the Gateway that is already configured with projects, etc, **Gateway A**, and the new Gateway that we just installed on another computer **Gateway B**. Lets say we want to call our new cluster **MyCluster**.

1. Go to the configuration site of **Gateway A**. Set the cluster name to "**MyCluster**" and let it restart.
2. **Gateway A** will restart and become the *MASTER* of cluster **MyCluster**.
3. Go to the configuration site of **Gateway B**. Set the cluster name to "**MyCluster**" and let it reboot.
4. **Gateway B** will join cluster **MyCluster**, first copying the configuration out of **Gateway A**.
5. The two nodes are now in cluster **MyCluster** and are sharing configuration and session information.

If the *MASTER* node of a cluster crashes, another node will become the new *MASTER* node. When the crashed node comes back online, it will become a *MEMBER* node. It isn't important which node is the *MASTER* node. All Designers communicate only with the *MASTER* node. Clients communicate with either the *MASTER* or *MEMBER* nodes.

Did you know...

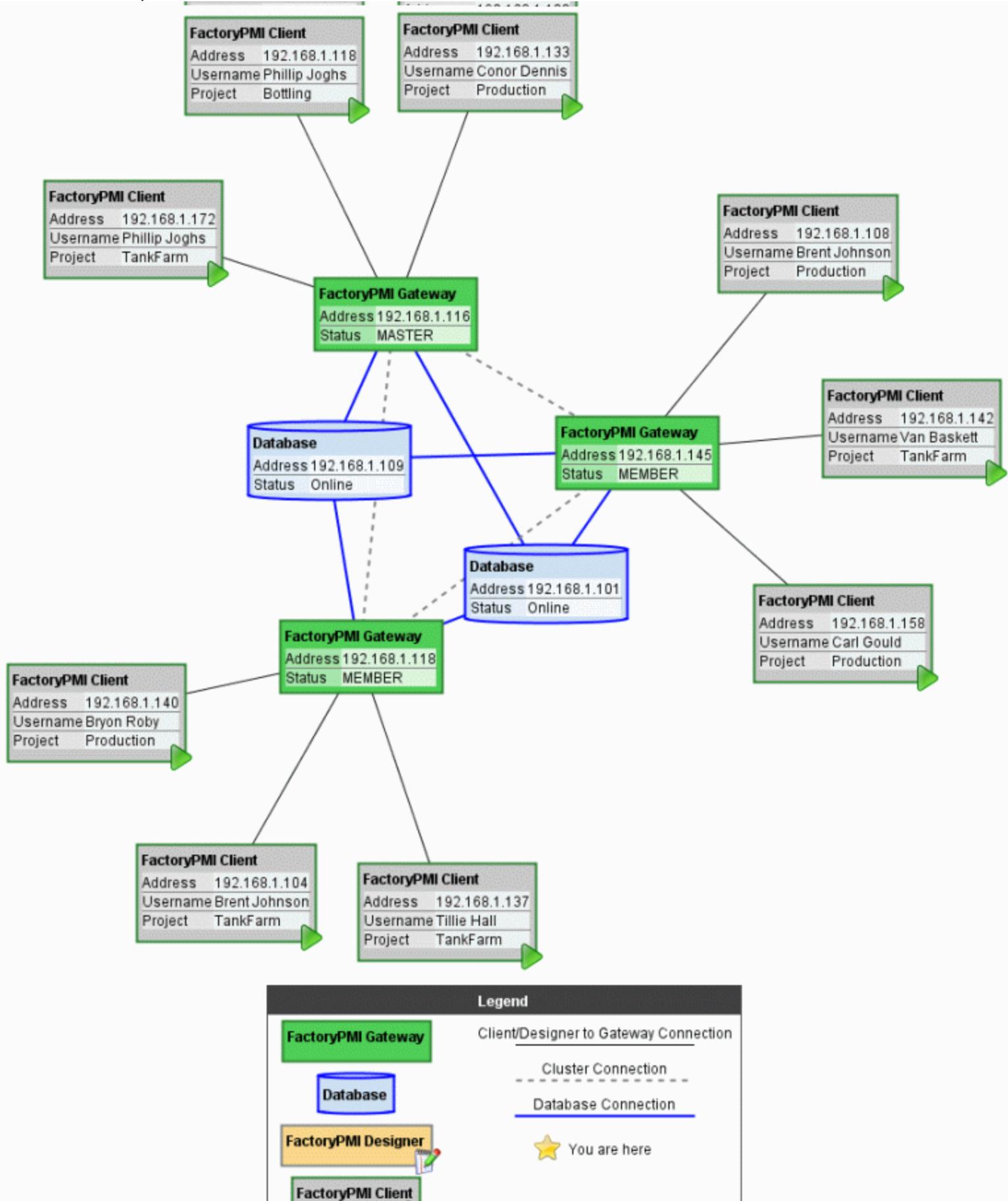
Every FactoryPMI Gateway is a member of a cluster! Most are simply members of 1-node clusters. This allows allows the Gateway to always act in the same manner, simplifying it and improving stability.

When you first install a FactoryPMI Gateway, it is assigned a random cluster name, like `PrivateCluster2819`. When you install another Gateway on the same network, you *could* have it join that cluster, but it is a better idea to assign both Gateways to a cluster name that has more meaning like `TankFarmCluster`. You can have multiple distinct clusters on the same network - they just need different cluster names.

How do I tell what's going on in my cluster?

Having trouble visualizing what is going on in your cluster? Fear not! Simply go to the **Gateway Status** page of any cluster node and click on the **Cluster Map** tab, where you will see an image representing the state of your cluster at that moment in time. Refresh the page to see any changes.

A 3-node FactoryPMI cluster with 9 clients and 2 databases:



Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

FactoryPMI Configuration

The FactoryPMI Configuration Web Page allows you to adjust virtually every aspect of the service. To get to the configuration area, click the icon on the launch page:



Or to go `http://serveraddress/setup`, where `serveraddress` is your server name or IP address including port (in most cases "localhost:8080").

You will be asked to log in. The default username and password are **admin/password**. Change this as soon as possible.

Please Login

Username	<input type="text"/>
Password	<input type="password"/>

Login

After a successful log-in, you will be taken to the configuration homepage, which will look similar to:

The screenshot shows a Microsoft Internet Explorer window with the title bar "FactoryPMI Gateway Setup - Microsoft Internet Explorer". The address bar shows "http://localhost:8080/setup/SetupMain". The main content area has a green header "FactoryPMI Gateway Configuration" and "PLANT MANAGEMENT INTERFACE". Below it, a message says "Logged in as: admin [logout] [restart] [shutdown]". On the left, a sidebar menu includes "System" (Status, Settings, Network, Cluster, Backup/Restore), "Projects" (Manage, Launch), "Datasources" (Connections, Drivers, Types), "Authentication" (Profiles, Types), "Auditing" (Profiles, Plugins, Manage, Images, Manage). At the bottom of the sidebar are buttons for "Launch Designer", "Gateway Status", "Help", and the "inductive AUTOMATION" logo. The central content area displays "FactoryPMI Gateway Server Status" with a table of system status information:

System Status	
Version	2.1.1 (build 1094)
Java Version	Sun Microsystems Inc. 1.6.0_01
CD-Key	(un-activate)
Memory (used/max)	194.42mb / 1,016.12 mb
Running Since	Fri May 04 20:49:28 EDT 2007
Uptime	0 days, 13 hours, 39 minutes, 43 seconds

At the bottom of the page is a footer bar with the text "FactoryPMI v. 2.1.1 is © 2002-2004 by Inductive Automation. All rights reserved. [view license]" and a "Local intranet" button.

This starting point also serves as a Server Status page, showing statistics on how much memory is being used, and how long the service has been running.

The structure of this section of the help file is designed to loosely follow the structure of the menu on the left side of the config page. You should browse from top to bottom and become acquainted with the options that are available.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



System Settings

These settings control some basic options for the gateway, such as which port the service runs on, and who can access the various sections. All of the options are explained on the actual page, as well as below.

System Properties

Port	The TCP/IP port the gateway will communicate on. This port must be open across any firewalls in order for the gateway to be accessible by outside clients. Note that the default port for web pages is 80 , so if you change FactoryPMI Gateway to run on port 80, you will be able to omit the ":80" from the URL of the launch page.
SSL Port	The TCP/IP port that will be used for SSL encrypted traffic. This port must be open across any firewalls in order for the gateway to be accessible by outside clients. Note that the default port for secure web pages is 443 , so if you set this to 443, you will be able to omit the ":8443" from the URL of secure pages.
Authentication Profile	Chooses which authentication profile is used to log into configuration. Must have the role "FPMIAAdministrator". See Authentication
Setup Site: SSL Enabled?	If enabled, the Gateway Configuration portion of the Gateway will be accessed over SSL.
Setup Site: Restrict Connections	If enabled, only the addresses in the Setup Site allowed address list will be able to view <code>http://serveraddress/setup</code> .
Setup site: Allowed address list	Specifies all of the IP address allow to connect to the setup site. Must have the above option enabled. IP list is comma separated, with "*" wildcards allowed.
Gateway:SSL Enabled	If enabled, the entire gateway (except the setup site - see above setting) will be accessed over SSL. This will make all launched projects also communicate over SSL.
Gateway: Restrict Connections	Only allow the addresses in the Gateway allowed address list to access the gateway (computers on the network that don't have access will not be allowed to run projects).
Gateway: Allowed address list	Addresses allowed to connect to gateway, if restriction is enabled above.

Designer

Designer Memory	The maximum amount of memory allocated to the designer when it runs.
-----------------	--

Launch Page

Autoinstall Java?	If checked, the gateway will attempt to automatically install Java upon the launch of a project or the designer. We say attempt, because a variety of factors can cause this autoinstall to fail, including: user doesn't have administrator access, user is not using Internet Explorer 6 or 7, user is using Internet Explorer 6 or 7 and has high security on (disallows ActiveX controls), etc. If this option is turned off, a user without the proper version of Java will simply see a warning popup telling them to install the proper version of Java when trying to launch a project.
Java detection?	Controls whether or not the section on the launch page that lets the user know if they have an acceptable version of Java is enabled or not.
Skip Javascript Detection?	If you turn this on, all javascript checks for detecting proper Java installation will be skipped, and launch links will go directly to the *.jnlp file. This is not recommended for most installations.
Designer Link	Determines whether a link to the designer is shown on the project launch page.
Gateway	Determines whether a link to the gateway status page is shown on the project launch page.

Status Link	
Configuration Link	Determines whether a link to the configuration section is shown on the project launch page.
Help Link	Determines whether a link to the HTML help is shown on the project launch page.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

About SSL (Secure Sockets Layer) for FactoryPMI

As of FactoryPMI version 3.1, SSL is fully supported for both the configuration pages and for client communication. This article will explain what SSL is, how to use it, and let you know if you should turn it on or not.

What is SSL?

SSL is a standard cryptographic protocol that provides secure communication between the FactoryPMI Gateway and anything that communicates with it. Typically, that is web browsers and FactoryPMI clients. This is an extremely strong form of encryption, identical to the technology that secures your online banking, credit card processing, and other high-security tasks you may be used to on the Internet. Because all traffic must be encrypted, you will incur some performance overhead by turning on SSL. But, in many scenarios, this overhead is necessary given the security risks involved.

Web-Techies Only:

FactoryPMI is actually using TLS (Transport Layer Security), the successor to SSL. We, like most people, still uses the term SSL to avoid acronym overload.

Do I need to use SSL?

The quick answer is: probably not. Most installations of FactoryPMI are installed on a **private intranet**. When this is the case, you typically aren't worried about malicious parties snooping on the data going over the wire. Note that FactoryPMI's communication protocols already encrypt sensitive information such as logon credentials, even with SSL turned off.

However, some of our users are allowing access to their FactoryPMI Gateway over the **public internet**. In this case, it is probably wise to turn SSL on. Snooping data isn't the main worry here - its *session hijacking*. You probably don't mind too much if a third party *knows* that your high level setpoint is 87° F, but you wouldn't want them *setting* that setpoint. If they hijacked your session, this would be possible. Turning on SSL will prevent this from happening.

Have it both ways!

You can easily have your local clients use non-encrypted communication for performance, and your remote clients use SSL encrypted communication for security. Simply access the gateway through the non-SSL address `http://myhost:8080` for the local clients, and the SSL address `https://myhost:8443` for the remote clients. You don't need to turn SSL "on" for this.

How do I use SSL?

Using SSL is easy - simply turn on the appropriate settings under **System > Settings**. This will force all access to go through SSL. (Hint: You know that you're using SSL if your web browser's address bar starts with `https` instead of `http`). Note that you don't need to turn SSL on to use it - you can simply type in the "`https`" manually and specify the SSL port in your gateway URL.

Help! My browser is complaining about the certificate!

SSL provides two benefits: identity assurance and communication encryption. The identity assurance part is so that you know that the website is who they claim they are. This involves a certificate that must be purchased from a *Certificate Authority* such as Verisign or Thatwe. The certificate is paired up with the server's address. The communication encryption part is what gives you your security.

FactoryPMI ships with what is called a *self-signed* certificate. This is one of your browsers complaints - that its not a "trusted" certificate. The other complaint is that the certificate doesn't match the gateway's address. This is because we can't know *your* server's address when we create the certificate - only you know that once you install it. Rest assured, if you tell your browser to trust the certificate, you will get the encryption part of SSL that you care about. If you're hosting a FactoryPMI-based service, you will probably want to buy your own certificate from a *Certificate Authority*. If you need to do this, we can help you install it in your FactoryPMI Gateway.



Network Settings

These settings control how the FactoryPMI interacts with and is accessed from the network. Typically you can leave these settings alone. Common situations that require modification of these settings are:

- Multiple NIC Server
- Server accessed through a DNS name
- Port forwarding setup

Note - these settings are not propagated through the cluster, they are set for each gateway

Network Properties

Autodetect HTTP Address	The HTTP Address is the address that Clients and Designers use to communicate with the Gateway. To specify an explicit HTTP address, turn this off. Most users will leave autodetect on, meaning that the gateway's bind address will be used as its HTTP address. You will want to specify an explicit HTTP address when the bind address is not the address that clients use. Port forwarding and DNS are the most common examples of this scenario.
HTTP Address	If you choose to specify an explicit HTTP Address by turning autodetect off, this is where you enter it in (This should be an IP address or DNS name)
HTTP Port	If you choose to specify an explicit HTTP Address by turning autodetect off, this is where you enter the port number.
Autodetect Bind Interface	The Bind interface is the address of the network card that will be used for clustering communication and for client communication if Autodetect HTTP Address is turned on. If you have two NICs in your server, you will probably have to turn this property off so that you can specify which NIC to use.
Bind Interface	If you turned off Autodetect Bind Interface, this is where you specify your explicit bind interface (should be an IP address).

This article on [Exposing a FactoryPMI Gateway behind a closed network](#) describes how you would use FactoryPMI with NAT, port forwarding, firewalls, and a DMZ.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Exposing FactoryPMI to the Internet behind a closed network

The Inductive Automation May 07 newsletter article covers some basic networking principals and settings to expose a local FactoryPMI gateway to a wider network.

Background

We're going to learn about TCP/IP and networking with FactoryPMI by example. Our setup uses the address range 192.168.0.1-254. This is an example of a *non-routable Class C IP network*. **Class C** means that we have 255 addresses to deal with and a 24 bit subnet mask (255.255.255.0). **Non-routable** means that we're using addresses have been reserved for private (non-Internet) use. This means that Internet routers will ignore requests that use these addresses. Make sure that you use non-routable addresses when setting up private control networks! We have a router set up that has a single legal IP address and provides Internet access to our network with Network Address Translation (NAT). This article is relevant to any setup where you use NAT, port forwarding, or a DMZ (Demilitarized zone, a subnetwork that sits between the internal and external network).

- The FactoryPMI gateway uses the static (non-DHCP) address **192.168.0.2** and currently runs over **port 8080**
- The router uses the LAN address **192.168.0.1**
- The router uses the WAN (Internet) address **69.19.188.26**
- Clients' addresses are assigned via DHCP in the range 192.168.0.100-150. They need to access the FactoryPMI project
- We want to be able to access our application over the Internet

Getting Started

Our first step to allow access to the FactoryPMI gateway is by setting up a **port forward** rule in the router. It should specify that TCP traffic directed to 69.19.188.26 over port 8080 be forwarded to 192.168.0.2. You may also need to add an incoming firewall rule to support this with the same settings.

To test, open **http://69.19.188.26:8080** in a web browser. If you see the default FactoryPMI Gateway web site it worked! If not, try loosening up your firewall policy and using 192.168.0.2 as the DMZ host. Keep in mind that a home router *DMZ host* is not a true DMZ in terms of network segmenting - it is a feature that will pass **all** traffic to our Gateway, with the exception of certain attacks. This is much more wide open than a single port forward - more geared toward Internet games that require numerous ports to be open. Incrementally tighten back security as you determine what works.

Next make sure that your firewall doesn't block outbound TCP traffic from your local network over port 8080. In most cases it shouldn't, but our network is very secure so we'll set up an outbound firewall rule to allow TCP traffic from 192.168.0.x to 69.19.188.26 over port 8080. Without this rule, Internet users won't have a problem, but your local clients won't be able to access the system. Your clients should address 69.19.188.26 instead of 192.168.0.2 when using the FactoryPMI runtime. I would then restrict gateway configuration access to either 127.0.0.1 (localhost) or 192.168.0.*.

Launching Projects

Now launch one of your applications via Java Web Start by clicking on a project link. The application will seem to download properly, but fail to launch. What gives? The FactoryPMI Gateway web server can listen over all IP addresses, but the client application needs to know the address of its Gateway - this setting is true for each Gateway in the Cluster. Normally this is automatically detected properly, but our Network Address Translation fools the client. Another error that you may see will say **Error Loading Plugins**, indicating the same problem.

Read this first!

As of FactoryPMI 3.1, anyone *NOT* running a cluster probably doesn't need to read this article. Simply turn on the **Standalone Mode** feature under **System > Settings**, and your multi-homed or port forwarded setup should simply work. This feature works by automatically detecting the incoming launch request's network perspective. If you are running a cluster, however, this isn't possible.





We need to statically tell clients that their gateway address will be the valid Internet IP address. Go to the Gateway Configuration Page -> Network and uncheck **Autodetect HTTP Address**. We then type **69.19.188.26** under HTTP Address.

I'm also going to uncheck **Autodetect Bind Interface** since I have multiple network adapters. I'll then specify **192.168.0.2** as the address for **Bind Interface**. This wasn't necessary, but is good to disambiguate our IP addresses.

The screenshot shows the FactoryPMI Gateway Configuration interface. The top navigation bar includes a logo, the title "FactoryPMI Gateway Configuration", and a sub-header "PLANT MANAGEMENT INTERFACE". It also shows the user is logged in as "admin" with links for [logout], [restart], and [shutdown].

The left sidebar contains a navigation menu with the following items:

- System
- Status
- Settings
- Network** (selected)
- Cluster
- Backup/Restore
- Projects
- Manage
- Launch
- Datasources
- Connections
- Drivers
- Types
- Authentication
- Profiles
- Types
- Auditing
- Profiles
- Plugins
- Manage
- Images
- Manage

The main content area is titled "Network Properties". It contains a table with the following rows:

Network Properties	
Autodetect HTTP Address	<input type="checkbox"/> To specify an explicit HTTP address, turn this off. Most users will leave autodetect on.
HTTP Address	69.19.188.26 If autodetect HTTP address is false, clients will use this address. Uses of explicit HTTP addresses include hostnames or port-forwarding setups.
HTTP Port	8080 If autodetect HTTP address is false, clients will use this port.
Autodetect Bind Interface	<input type="checkbox"/> To specify an explicit bind interface, turn this off. Most users will leave autodetect on.
Bind Interface	192.168.0.2 If autodetect bind interface is false, this interface will be used. This is useful to specify the correct interface for multi-NIC servers.

At the bottom right of the main content area is a "Save Changes" button. The bottom navigation bar includes icons for Launch Designer, Gateway Status, and Help.

Summary

Here's how we setup a FactoryPMI Gateway to work with NAT

- Set up a port forward rule in our router
 - Under the **Network** tab of the FactoryPMI Gateway Configuration page, uncheck **Autodetect HTTP address** and specify
 - Ensure that outbound TCP client traffic is allowed from our network to the WAN address of our router over the FactoryPMI port
 - Clients now reference the FactoryPMI project from the WAN address. So do computers over the Internet.
-

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Cluster Settings

These settings control the FactoryPMI Gateway's clustering settings. See the [Clustering](#) page for more information.

Cluster Properties

Clustering Enabled	<p>This property, which is false by default, affects whether or not clustering is enabled. This property has two distinct effects on the FactoryPMI Gateway:</p> <ul style="list-style-type: none">• Bind Address. When clustering is enabled <i>and</i> Autodetect Bind Address is enabled, the Gateway will attempt to bind to the first real NIC on the server. When clustering is not enabled (the default) <i>and</i> Autodetect Bind Address is enabled, the Gateway will bind to 127.0.0.1, effectively disabling clustering. Binding to 127.0.0.1 also prevents startup problems that are caused by strange network adapters and Windows Media Sense. As of FactoryPMI 3.1.5, binding to 127.0.0.1 is the default. Note that if you turn Autodetect Bind Address <i>off</i>, then the bind address you specify will override this logic.• Client HTTP Address. When clients are launched, they need to be told the address of the Gateway. When clustering is disabled, clients automatically detect the Gateway's address based on how they reached the launch webpage. This makes multi-homed and port-forwarded setups work naturally. When clustering is enabled, clients will be told to connect to the Bind Address by default, or to the HTTP Address if Autodetect HTTP Address is disabled. In this situation, multi-homed and port-forwarded setups can still be achieved with careful DNS management.
Cluster Name	Each cluster has a name. By default FactoryPMI gateway is given a randomly-chosen name. If you put two Gateways on the same subnet (so that they have multicast UDP traffic connection) and give them the same Cluster Name, they will join together into a cluster.
Password	You can optionally specify a password for joining your cluster.
Multicast Address	This is the UDP multicast address that will be listened on for clustering traffic. The default is 239.4.4.44.
Multicast Port	This is the UDP multicast port that will be listened on for clustering traffic. The default is 45566. If you have two FactoryPMI Gateways on the same subnet that are <i>not</i> part of the same cluster, we recommend having them on different ports, so that they are not wasting processing time discarding each other's clustering traffic.



Backup and Restore

The back up and restore section of the gateway configuration provides you with an easy way to download everything pertinent to your FactoryPMI projects, and to load it again if something goes wrong (or to transfer servers).

To backup, click the "Download Server Backup" link under the backup tab:

[Download Server Backup](#)

This will download a file ending in ".fpgb" (FactoryPMI Gateway Backup), which contains everything about your system: projects, data connections, custom data connectors, etc.

To restore, click on the "Restore" tab, and put the path to the file you want to load in the address box. When you click the restore button, the file will be loaded, and a soft reset will be performed. All existing data will be erased, and everything from the file will be loaded.

Backup **Restore**

Choose a FactoryPMI Gateway Backup file (*.fpgb) to restore. This operation forces a soft reset and will need to re-login after restoring.

Note:
Please ensure that all clients and designers are closed before restoring.

 Be aware that *All* of this server's current configuration (except activation keys) will be lost. It is recommended to backup this server before restoring.

[Browse...](#)

Restore

Remember, EVERYTHING is saved in the backup file, so if the file you are loading specifies a different operating port, you won't automatically be redirected to the log in page after the reset.

About Data Source Connections

Data Source connections are the links between FactoryPMI and various databases. You will need to create one for each different database you want to connect to. There is no limit to the number of databases you can connect to. FactoryPMI connects to databases through JDBC, the standard connection technology for Java programs. Virtually all databases provide a JDBC driver. FactoryPMI comes with two JDBC drivers installed - one for PostgreSQL and one for MySQL.

To create a new connection, click on the "New" icon, to the bottom right of the connection list.

Datasources

Name	Type	Connect URL	Description
No datasources configured.			

New Connection 

Configured Connections

First, you will be asked to select a communication driver. Different database systems require their own drivers, so you must choose the one that corresponds to the database you are using. If the correct one isn't in the list, see the [Drivers](#) section for help on finding and installing one.

Add Datasource Step 1: Choose Driver

JDBC Driver Select the correct JDBC Driver for the type of database you wish to connect to. If no driver corresponds to your database, go to the Driver Configuration page to add a new driver.

[Next >](#)

The next page will display a list of options for the selected connection. These options are explained on the page, but the main properties that must be filled out are the Name, Connect URL (see actual page for format help), Username, and Password. In most cases, you can leave the Extra Connection Properties at its driver-specific default (probably blank).

Add Datasource Step 2: Configure Parameters

Main Properties

Name	<input type="text"/> Choose a name to identify this datasource.
Description	<input type="text"/>
Connect URL	<input type="text" value="jdbc:postgresql://localhost:5432/test"/> The Connect URL is JDBC-driver specific. It usually contains the address of the machine that the database is running on. The format of the PostgreSQL connect URL is: <code>jdbc:postgresql://host:port/database</code> With the three parameters (in bold): <ul style="list-style-type: none">• host: The host name or IP address of the database server.• port: The port that the database server is running on. PostgreSQL default port is 5432.• database: The name of the logical database that you are connecting to on the PostgreSQL server.
Username	<input type="text"/> This is the username that will be used to connect to the database.
Password	<input type="text"/> This is the password that will be used to connect to the database.

Extra Connection Properties

The connection properties that will be sent to our JDBC driver when establishing new connections.

Format of the string must be

Examples:

```
property=value;
```

```
propertyOne=valueOne;propertyTwo=valueTwo;
```

NOTE - The `user` and `password` properties will be passed explicitly, so they do not need to be included here.

No extra connection parameters are recommended for PostgreSQL. For possible parameter values, see the documentation at [the PostgreSQL JDBC driver website](#).

Create Datasource

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Connection Settings

The data connections have a number of advanced settings that can be altered. To view the settings, click the "Edit" icon on the Datasource - Connections page.

Name	Type	Connect URL	Description
DemoConnection	PostgreSQL	jdbc:postgresql://localhost:5432/test	  Edit connection 

All of the settings are explained on the configuration page.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Viewing Connection Status

You can check the status of data connections by going to the FactoryPMI Gateway Status page. The status page provides a number of useful pieces of information, and to get to the status page, click the link at the bottom of the connection overview screen, or go to <http://serveraddress/gateway>

Datasources

Name	Type	Connect URL	Description	
DemoConnection	PostgreSQL	jdbc:postgresql://localhost:5432/test		

Note:

To view datasource's status, see the [Gateway Live Status](#) page. Remember, any changes to datasource configuration requires a restart before they will take effect.

The status page will look similar to the following, showing the datasource connection status at the bottom:

The screenshot shows the FactoryPMI Gateway Status page. At the top, there is a green header bar with the FactoryPMI logo and the text "FactoryPMI Gateway Status". Below the header, there are two main sections: "FPMI Gateway Statistics" and "Datasource Status".

FPMI Gateway Statistics

Version	1.0.0_17 (build 504)
Memory (used/max)	7.57mb / 254.06 mb
Running Since	Fri Jun 24 15:07:11 PDT 2005
Uptime	3 days, 1 hours, 22 minutes, 20 seconds
Requests Serviced	0
Average Request Time (ms)	0
Max Request Time (ms)	0
Min Request Time (ms)	0
Most Recent Query Time (ms)	0
Most Recent Formulation Time (ms)	0
Status	Running

Datasource Status

Datasource 'DemoConnection'	OK
--------------------------------	----

Or, if the connection has failed, it will show the error message:

The screenshot shows the FactoryPMI Gateway Status page with an error message for a datasource connection. The "Datasource Status" section shows a row for "Datasource 'DemoConnection'" with the status "FAIL" and the error message: "FAIL (Cannot create PoolableConnectionFactory (Connection rejected: FATAL: password authentication failed for user "postgres").)"



JDBC Drivers

FactoryPMI connects to databases using **JDBC**, which stands for Java Database Connectivity. There are JDBC drivers for virtually every database system ever made. FactoryPMI comes with four JDBC drivers pre-installed:

1. **JDBC-ODBC Bridge.** This driver lets you connect to an ODBC datasource through JDBC. ODBC is another database access method commonly used by Windows applications. See the cautionary note below about the JDBC-ODBC bridge.
2. **MS SQL Server by jTDS.** This driver is an open-source MS SQL server driver. It is very high performance, commonly outperforming the official SQL Server JDBC driver from Microsoft.
3. **MySQL Connector/J.** The official JDBC driver for MySQL.
4. **PostgreSQL JDBC Driver.** The official JDBC driver for PostgreSQL.

NOTE: The JDBC-ODBC driver provided by Sun Microsystems comes with this cautionary note:

If possible, use a Pure Java JDBC driver instead of the Bridge and an ODBC driver. This completely eliminates the client configuration required by ODBC. It also eliminates the potential that the Java VM could be corrupted by an error in the native code brought in by the Bridge (that is, the Bridge native library, the ODBC driver manager library, the ODBC driver library, and the database client library).

The JDBC-ODBC Bridge driver is recommended for use in prototyping efforts and for cases where no other JDBC technology-based driver exists. If a commercial, all-Java JDBC driver is available, we recommend that it be used instead of the Bridge.

Installing a New Driver

If you are using a database whose driver doesn't come pre-installed with FactoryPMI, you will need to add the JDBC Driver to the system. The database vendor should provide a JDBC Driver (it should come as a *.jar file) and some basic instructions.

Once you have your JDBC Driver's jar file, click on the **Drivers** link under **Datasources**. Click on the green plus icon to add a new driver.

Datasources

Name	Type	Connect URL	Description
No datasources configured.			

New Connection

Configured Connections

A JDBC driver needs the following parameters to be filled in:

Name

Give this JDBC driver a name, such as the name and version of the database it connects to.

Description

Optionally describe this JDBC Driver

Driver Type

You must assign your driver a type. Don't worry if the database that you are using isn't listed. Currently the only four types that are listed are MySQL, PostgreSQL, Microsoft SQL Server, and Generic. This setting is used for certain features where FactoryPMI has to generate SQL queries, and those queries tend to change in syntax from database to database. For instance - when you browse the database in the FactoryPMI Designer, it tells the database to cap the number of rows returned in the browser at 1000 to avoid swamping the designer with too much data. The syntax of this cap varies based on database server - in MS SQL Server it is `SELECT TOP 1000 * FROM TableName`, where in PostgreSQL it is `SELECT * FROM TableName LIMIT 1000`. To make a long story short, this setting isn't critical, and it certainly doesn't affect your project's queries. If you are using a database that isn't listed, try using the Generic database type. If you find a feature that isn't working, please let us know, and we will work to add your database's quirks to the list.

Classname

This property is the name of the Java class within the supplied JAR file that represents the JDBC driver. The

documentation for the JAR file should make this very obvious.

JAR File

Here you can upload the actual JAR file for this driver

The remaining properties are provided so that users of this JDBC driver can have a hint about how to use it. This is necessary because each driver's URL format is different, and some drivers let you specify extra properties. See the instructions that came with your JDBC driver to learn how these properties are used, and then enter some hints for the person adding the Datasource connection (See how the existing JDBC drivers are set up for an example)

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Creating a New Project

A "project" is the executable level of PMI. Every project you create will be listed on the project launch page, and will have a number of options that can be set independently of other projects.

Before you can begin editing a project, you must first create it. The "Manage" link under Projects on the left menu brings you to a screen where you can Add, Remove, and [Archive](#) projects.

Projects

The screenshot shows a table titled 'Active' with columns for Name, Primary Datasource, Authentication, and Description. A message 'No projects configured.' is displayed below the table. To the right, there are two buttons: 'Create New Project' (with a green plus icon) and 'Load project from file' (with a blue file icon). Red annotations include a bracket under the table labeled 'All projects on this server' and arrows pointing to each button labeled 'Create New Project' and 'Load project from file'.

Name	Primary Datasource	Authentication	Description
No projects configured.			

Create New Project

Load project from file

All projects on this server

This will bring you to the project properties page for your new project. After editing the required properties, you may click the "Add Project" button to save them, and actually create the project on the server. You can now open the project in the FactoryPMI Designer and edit it. Note that you can create a new project from within the Designer as well.

New Project Properties	
Name	The name that will identify your project.
Description	An optional description for your project.
Primary Datasource	This is the datasource that will be used by default for your project. See Datasource Connections for more information.
Authentication Profile	The profile that will be used to authenticate access to your project. See Authentication for more information.
Auditing Enabled	Turn on auditing for this project. See Auditing for more information.
Audit Profile	Chooses which auditing profile to use, if auditing is enabled.
Project Memory	Specifies the amount of memory the client has access to.
Window Title	The title of the application, when launched.
Welcome Message	The message displayed on the log-in screen for the project

Project Properties

There are a number of properties, most of which have to do with launching, that you configure from the Gateway.

Project Properties	
Name	The name that will identify your project.
Description	An optional description for your project.
Primary Datasource	This is the datasource that will be used by default for your project. See Datasource Connections for more information. Note that a project can use any of the datasources configured on a Gateway. Most projects primarily use one datasource, however. The point of the primary datasource is so that it is really easy to point a project from one database to another, and do it from one spot. For instance, it is common to develop a project using a 'test' database, and then when the project is done, you switch the primary datasource to the 'real live' database.
Authentication Profile	The profile that will be used to authenticate access to your project. See Authentication for more information.
Auditing Enabled	Turn on auditing for this project. See Auditing for more information.
Audit Profile	Chooses which auditing profile to use, if auditing is enabled.
Project Max Memory	Specifies the amount of memory the client has access to.
Project Initial Memory	Specify the initial amount of memory that the client application uses.
Touchscreen Enabled	If checked, a button will appear on the login screen that will enable touchscreen-mode for that instance of the application.
Publish Mode	The default publish mode for a project determines how updates are handled by running clients, either notify client or automatically push update to client.
Window Title	The title of the application, when launched.
Auto-login username	If not blank, the project will try to automatically log in with this username. This is useful in cases where security isn't important, or on executive dashboard applets that are embedded in already secured webpages.
Auto-login password	The password to use for auto-login. See above.

These properties affect how the project is shown on the launch page of the gateway.

Launch Screen Properties	
Launch Title	The title of the project to be displayed on the launch page. If blank, the project name will be used. Use this if you want spaces and punctuation to be displayed on the launch page.
Default Launch Mode	<p>The default launch mode is the method that will be used to launch the project if a user clicks on the name of the project on the launch page. Choose between these three modes:</p> <ul style="list-style-type: none"> Web Start Web start will launch the project as a full-fledged application on the user's computer. The application will be in windowed mode, meaning they will be able to use it alongside other applications. Note, this is not to be confused with the "JWS Start Maximized" property explained below. Web Start Full Screen Like Web Start, except the application will be run in full-screen-exclusive mode. On Windows, this means that the Start bar won't be accessible, and the application won't have a title bar or border. This mode is best for HMI-exclusive terminals, like panel-mounted touchscreen computers. Applet Applet mode means the application will run embedded inside the user's web browser.

	This works best with Internet Explorer. This is useful for smaller projects, that should be embedded in a webpage.
Show 'Windowed' Button	If true, a button will appear next to this project on the launch page that will launch the project in standard Web Start mode, regardless of the project's default launch mode setting.
Show 'Full Screen' Button	If true, a button will appear next to this project on the launch page that will launch the project in Web Start Full Screen mode, regardless of the project's default launch mode setting.
Show 'Applet' Button	If true, a button will appear next to this project on the launch page that will launch the project in standard Applet mode, regardless of the project's default launch mode setting.
Hide from Launch Page	If checked, the project will be hidden from the launch page. You can still launch this project directly from its launch URLs: http://serveraddress/gateway/launch/projectname.jnlp (Web Start) http://serveraddress/gateway/launchfs/projectname.jnlp (Web Start Full Screen) http://serveraddress/gateway/applet/projectname (Applet)

These properties affect how the login screen of the project looks.

Login Screen Properties	
Welcome Message	The message displayed on the log-in screen for the project. Note that HTML formatting is allowed, just like in most FactoryPMI components.
Welcome Image	If blank, the default image will be displayed next to your welcome message. If you'd like to override this image, specify a gateway image path here.
Username Text	The text displayed to identify the username field on the login screen.
Password Text	The text displayed to identify the password field on the login screen.
Login Button Text	The text that will appear on the login button.

These properties affect how the application is launched if using the Web Start launching mode.

Java Web Start Properties	
JWS Title	This will be the title of the application in the Java Webstart loading splashscreen and the title that Java Web Start uses to integrate the application into a user's desktop, if chosen.
JWS Vendor	This field is used by Java Web Start to show who made the application. for example, "ACME Integration"
JWS Width	This is the width of the application window, if the application doesn't start up maximized.
JWS Height	This is the height of the application window, if the application doesn't start up maximized
JWS Start Maximized	If this is checked, the application will start up as a maximized window, instead of the width and height specified above.
JWS Homepage	An optional homepage for the application. The URL to the Gateway would be a logical choice.
JWS Description	A description of the application.
JWS Icon	An icon that will be used in the Java Web Start splash screen
JWS	If present, this image will be the Java Web Start loading splash screen.

These properties affect how the application is launched if using the Applet launching mode.

Applet Properties

Applet Width	The width (in pixels) of the applet in the web page.
Applet Height	The height (in pixels) of the applet in the web page.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



What's in a project?

From the gateway, you deal with projects from a management point of view - creating, deleting, archiving, exporting, importing, copying, and changing parameters. Before we talk about how to do all of those things, it is important that you understand exactly *what* you're coping, exporting, etc.

A project in a FactoryPMI Gateway is a collection of parameters used to launch the project, and the configuration of all of the project's windows, special scripts, etc. What is important to realize is that the datasource connections, Authentication profile setup, and perhaps most importantly, images, are *not* contained within the project. The reason for this separation is so that multiple projects can use the same images, datasources, etc. The downside is that if you want to move just one project to another Gateway, you have to think about moving all of the other resources that it is dependent on.

Remember, it is easy to backup your entire server (see the [System Backup / Restore](#) page) to get a complete copy of all of your settings. Likewise, it is easy to backup all of the images that a project uses, as long as you keep your images organized by project (FactoryPMI automatically creates an image folder for each project for this purpose).

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

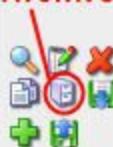
Project Archiving

Archiving allows you to create "snapshots" of your projects at any time, saving them off as copies. Later, if you decide that a previous version was better, you can roll back easily.

To archive a project, click the "Archive" icon on the **Projects - Manage** screen.

Projects

Active Archived

Name	Primary Datasource	Authentication	Description	Archive
DemoProject	DemoConnection	Default		

You will be presented with a screen that explains what archiving is, and allows you to enter a note for this particular archive, so that you'll be able to distinguish it from the other ones later.


Logged in as: admin [logout] [restart] [shutdown]

System
Status
Settings
Backup
Projects
Manage
Launch
Datasources
Connections
Drivers
Types
Authentication
Profiles
Types
Auditing
Profiles
Images
Plugins

Projects

Active Archived

Create Archive Snapshot

An archive snapshot is a backup copy of your project in its current state. This will not affect the active copy of the project. You can restore this archived snapshot later.

Note:
This snapshot archives just the project itself, and does **not** create a backup of your current:

- Images
- Datasources
- Authentication Profiles
- Audit Profiles

As such, if you restore this snapshot later, you may have to reconnect this project to a datasource, authentication profile, and audit profile. You will also have to verify that all of the images that it used still exist, and have not changed paths.

Archive Properties

Archive Notes

These notes give you a chance to describe the state of the project as of this snapshot.

Archive

FactoryPMI v. 1.0.0_17 is © 2002-2004 by Inductive Automation. All rights reserved. [view license]

To view your archived projects, click the "Archive" tab on the project management screen.

Active Archived

Name	Primary Datasource
DemoProject	DemoConne

The archive list shows all of your saved "snapshots" (previously archived projects), sorted by time. From here you can delete them, or restore them to current status. When you restore, your current project will be erased, and the saved snapshot will be reinstated.

Active Archived

Delete Snapshot

Snapshots of 'DemoProject'

Name	Archive Date	Archive Notes	Restore
------	--------------	---------------	---------

NAME	CREATE DATE	PROJECT NOTES	REVISIONS
DemoProject	06/28/2005 11:40 AM	All data connections complete, first production version.	 
DemoProject	06/25/2005 10:39 AM	First revision. Main screens complete.	 

Delete all snapshots for project → 

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Copying Projects

You can easily copy a project by clicking on the "Copy" icon under the Project Management screen.

Active **Archived**

Name	Primary Datasource	Authentication	Description	
DemoProject	DemoConnection	Default		 Copy project 

You will be asked to enter a name for the project. After it has been copied, you can edit and run it like any other project.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Importing and Exporting Projects

You can import and export projects (meaning saving them to files and loading them, without all of the other PMI server settings) easily from the Project Management page. The two icons at the bottom right of the project list let you quickly accomplish this task:



Exporting will provide you with a file download of your project. Select save and then select the location to save to.

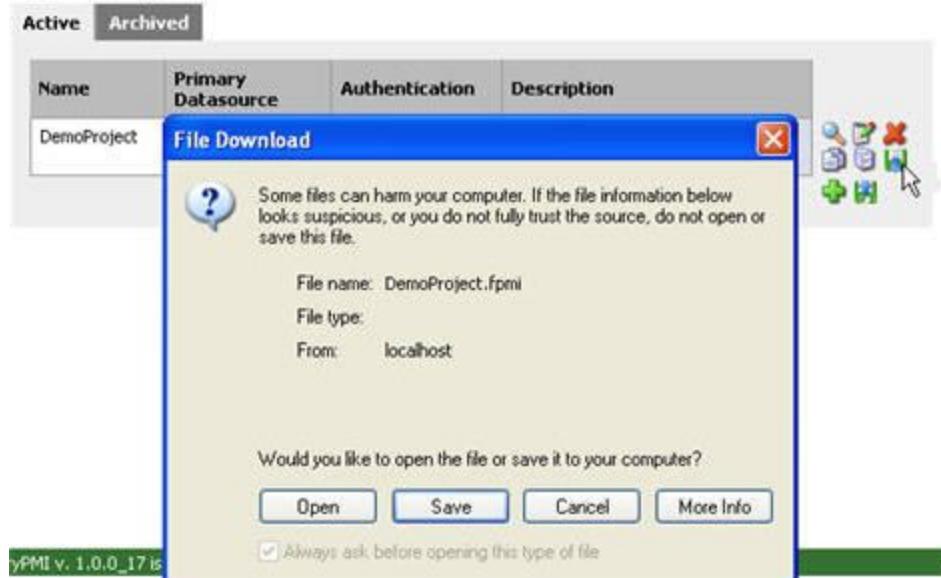


Fig. 2: Exporting a project

When you import a project, you will be able to specify a new project name, primary datasource, and authentication profile:

Active Archived

Upload Project

If you have a backup copy of a project, you can upload it here. You can assign this project a new name, or leave it blank to use the name of the exported project. You can also assign the project a default datasource and authentication profile, in case the original project's datasource and authentication profile have been removed.

Upload Parameters	
Name	NewProjectImport Choose a name to identify this project, or leave blank to use the backup copy's name.
Primary Datasource	[Use Backup] Choose which datasource will serve as this project's primary datasource, or leave blank to use the backup copy's datasource.
Authentication Profile	[Use Backup] Choose which authentication profile you want to provide authentication services for this project, or leave blank to use the backup copy's authentication profile.
Project Backup File	C:\PMI\Exports\DemoProject.fpmi Choose a *.fpmi file to upload as a new project.

Upload Project

Authentication Profiles

Authentication profiles define the source of user authentication. Each project specifies an authentication profile that will be used when users log into that project. Likewise, access to the configuration website is also controlled by an authentication profile. You can use one profile for all of your authentication needs, or a mix of profiles.

You can access the authentication profile configuration screen by clicking on **Profiles** under **Authentication** in the configuration page menu.



The screenshot shows the FactoryPMI Gateway Configuration interface. The top navigation bar includes the FactoryPMI logo, the title "FactoryPMI Gateway Configuration", and a status message "Logged in as: admin [logout] [restart] [shutdown]". On the left, a sidebar menu lists various system components: System (Status, Settings, Backup), Projects (Manage, Launch), Datasources (Connections, Drivers, Types), Authentication (Profiles, Types), Auditing (Profiles), Images, and Plugins. The "Authentication Profiles" option is currently selected, highlighted with a green background. Below the sidebar is a "Launch Designer" button. The main content area is titled "Authentication Profiles" and contains a brief description: "An authentication profile defines a source of user authentication. Each project uses an authentication profile that will be used when users log into that project. Likewise, access to this website is also controlled by an authentication profile. You can use one profile for all of your authentication needs, or a mix of profiles." A table displays the current profile configuration:

Profile Name	Source	Description
Default	Internal	Default authentication. Internally managed.

To the right of the table are several icons: a pencil for edit, a trash can for delete, a person icon, and a gear icon. A large green plus sign icon is positioned below the table, indicating the ability to add new profiles.

There are three types of authentication profiles (you may have a number of each):

Authentication Profile Types	
Internal	All user/role information is stored in the FactoryPMI internal database (the same place projects are stored). You may edit user information from the PMI configuration site.
Datasource	User information is stored in a separate database. It can be administered from anything that can communicate with the database (including from within a FactoryPMI project!).
Active Directory	User/Role information configured through Microsoft's Active Directory, commonly used in medium to large corporate settings..
Active Directory / Internal Hybrid	Microsoft Active Directory is used for user authentication, but then FactoryPMI internal role management is used for roles.

An authentication profile, of any type, is responsible for doing three things:

1. Authenticate a **user**. This means accept or reject a given username/password pair.
2. List a User's Roles. Provide a list of **roles** for a given username/password pair
3. List All Roles. Provide a list of all possible roles.

For more on how security on FactoryPMI works, see [Designer > Concepts > Security](#)

By default there is one profile, using an Internal source. You can configure it using the icons to the right:

Edit options →   ← **Edit Users**

Test profile

You cannot delete the Default authentication profile. For more on the Default profile, see [Default Authentication](#)

To create a new profile, click the "New" icon 

TIP

If you do point the Gateway Configuration site to use another Authentication profile, make sure that there are users with the roles "FPMIAdministrator" and "FPMIDesigner". These are the required roles to open the Gateway Configuration site, and launch the Designer, respectively.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Default Authentication Profile

The FactoryPMI Gateway comes with one internal authentication profile configured - the *default authentication profile*. The default authentication profile is an *internal profile*. It comes with one user defined, with the username/password of **admin / password**, and the roles **FPMIAAdministrator** and **FPMIDesigner**. This is needed, otherwise you wouldn't be able to log into the Gateway or launch the Designer!

You can change which profile the Gateway Configuration site and the Designer uses in the [System Settings](#) page, but you cannot delete the Default profile. This is a safeguard, in case you lose your password, or somehow lock yourself out of the Gateway Configuration site. If this happens, use the [Gateway Monitor Utility](#) to [reset your admin password](#).

Tip

If you do point the Gateway Configuration site to use another Authentication profile, make sure that there are users with the roles "FPMIAAdministrator" and "FPMIDesigner". These are the required roles to open the Gateway Configuration site, and launch the Designer, respectively.

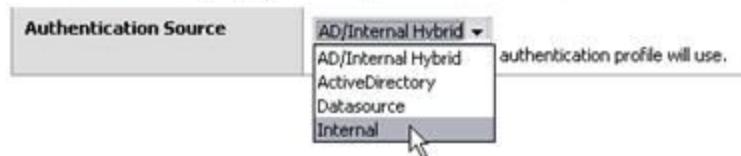


An Internal authentication profile stores its usernames and role information internally within the FactoryPMI Gateway, similar to how your projects are stored. You can configure the users and passwords through the FactoryPMI Gateway's Configuration page, by clicking on the Manage Users button () next to the Internal profile.

To add a new Internal profile, click the "New Profile" icon from the profiles list (), and select "Internal". Internal profiles don't have much configuration beyond the profile name, description, and optional failover profile. After you add your profile, you can add [users](#) and [roles](#) through the Manage Users button () that appears next to the profile.

Add Authentication Profile Step 1: Choose Source

See the [Authentication Types](#) page for a description of the different authentication sources.



The screenshot shows the FactoryPMI Gateway Configuration interface. The left sidebar contains navigation links for System, Projects, Datasources, Authentication (Profiles selected), Auditing, Plugins, Images, Launch Designer, Gateway Status, Help, and Inductive Automation logo. The main content area has a title 'Add Authentication Profile Step 2: Configure Properties'. A 'General Properties' panel includes fields for Name (placeholder: Choose a name to identify this authentication profile.), Description (text area), and Failover Profile (dropdown set to 'none'). A large 'Add Profile' button is at the bottom of the panel. At the bottom of the page is a footer bar with the text 'FactoryPMI v. (Dev Version) is © 2002-2004 by Inductive Automation. All rights reserved. [view license]'

Fig. 1: The Internal profile config page



User Admin

Users represent a single username/password combination. They can be placed into different Roles, or can be specified individually when restricting access later on. To administer users for an internal profile, click the "Manage Users" icon () from the Profiles page.

You may add, edit, and delete users using the icons on the right.

To add a user, you must specify a name and password. Beyond that, you may type a description, and assign the user to different roles.

Users Roles

User Properties

Username	<input type="text"/>
Choose the username for this user.	
Password	<input type="password"/>
Choose a password for this user.	
Confirm Password	<input type="password"/>
Re-type the password.	
Description	<input type="text"/>

Roles

Role Membership	<input type="checkbox"/> FPMIAdministrator <input type="checkbox"/> FPMIDesigner
Check all roles that this user is a member of.	

Add User

Fig. 1: Adding a user

Authentication Roles

Roles are a way to group users, so that you can restrict access to a number of people easily. To edit the roles for an internal authentication profile, click the "Manage Users" icon on the Authentication Profiles page () and then on the "Roles" tab:



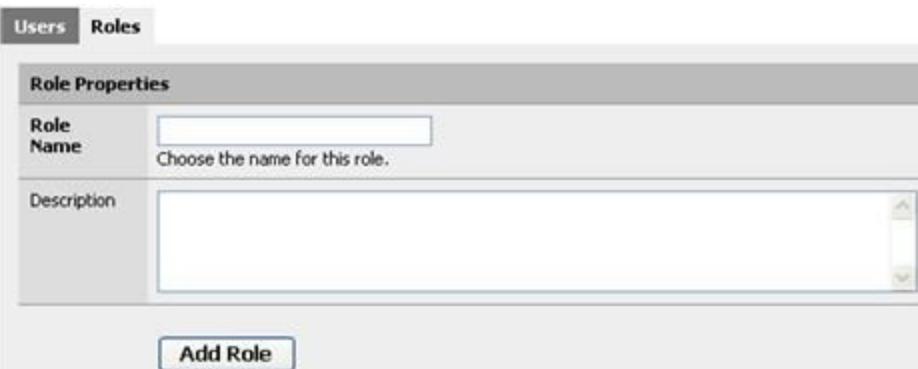
Username	Roles	Description
admin	FPMIAdministrator, FPMIDesigner	

You may add more roles by clicking on the "New" icon.



Role Name	Members	Description
FPMIAdministrator	admin	Users with this role have access to the Factory PMI Gateway Server configuration website.
FPMIDesigner	admin	Users with this role have access to the Factory PMI Designer.

Fig. 1: The roles list



Role Properties

Role Name: Choose the name for this role.

Description:

Add Role

Fig. 2: Adding a new role

To add users to the role, see [User Admin](#)

Datasource Based Profile

A datasource based profile is one that authenticates users against a table in a remote database. That way user authentication can be managed outside of the FactoryPMI Gateway Configuration site. There are a number of options available for configuring Datasource profiles.

To add a datasource profile, click the "New Profile" icon from the profiles list (+), and select "Datasource".

Add Authentication Profile Step 1: Choose Source

See the [Authentication Types](#) page for a description of the different authentication sources.

Authentication Source	<select data-bbox="230 473 388 502">Datasource</select> <input type="button" value="ActiveDirectory"/> <input checked="" type="button" value="Datasource"/> <input type="button" value="Internal"/>	This authentication profile will use.
		<input type="button" value="Next >"/>

The profile configuration page outlines many configuration options, which are all explained fully on the page. You will need to choose an existing [Datasource](#), and then choose whether the authentication table will be created automatically, or manually. REMEMBER: If you choose to create the table automatically, all other options on the page will be ignored.

 **FactoryPMI** Gateway Configuration
PLANT MANAGEMENT INTERFACE

Logged in as: admin [logout] [restart] [shutdown]

System
Status
Settings
Backup

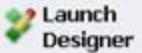
Projects
Manage
Launch

Datasources
Connections
Drivers
Types

Authentication
Profiles
Types

Auditing
Profiles

Images
Plugins

 Launch Designer

Add Authentication Profile Step 2: Configure Properties

General Properties

Name
Choose a name to identify this authentication profile.

Description

Datasource Properties

Datasource DemoConnection
Choose the datasource this authentication profile will use.

Table Creation Auto-Create
 Manual Configuration

If you choose 'Auto-Create', all other fields below will be ignored, and the user, role, and mapping tables will be created with default names in the datasource that you choose. If you choose 'Manual Configuration', you will need to enter the correct values that correspond to the table names and column names in the database tables that you have created.

Fill in this section if you chose 'Manual Configuration' above and you have created the 3 necessary tables in your database.

Table Parameters

User Table
The name of the table that the User Id(optionally), User Name, and Password columns can be found.

User Id Column
The name of the column in the User Table that stores the user id. If left blank, then the User Name is treated as an Id (i.e. it must be unique)

User Name Column
The name of the column in the User Table that stores the user name.

User Password Column

Column	The name of the column in the User Table that stores the users password.
Hash Function	<input type="text"/> If the passwords are stored in a hashed format (i.e. <i>not</i> plain text), enter the name of the hashing function to use here. This name is dependent on what hashing functions are available in the database you are using. Examples: "MD5" or "SHA1".
Role Table	<input type="text"/> ROLES The name of the table that the roles are stored in.
Role Id Column	<input type="text"/> The name of the column in the Role Table that stores the role id. If left blank, then the Role Name is treated as an Id (i.e. it must be unique)
Role Name Column	<input type="text"/> Rolename The name of the column that the role name is stored in.
Mapping Table	<input type="text"/> USER_ROLE_MAPPING The name of the table that stores the mapping between users and roles. That is, for each role that a user has membership in, there will be an entry in this table with that user's id and the role's id (or names, if names are being used as ids)
Mapping User Id Column	<input type="text"/> Username The name of the column in the Mapping Table that stores the user's id or name.
Mapping Role Id Column	<input type="text"/> Rolename The name of the column in the Mapping Table that stores the role's id or name.

Fill in this section if you chose 'Manual Configuration' above and you have your own user/role management system in your database. If these queries are provided, they will override the Table Parameters above. This section is for experts only.

Query Configuration (expert)	
Use Expert Configuration	<input type="checkbox"/> Check this box if you want to use this section, instead of the Table Parameters above.
Authentication Query	<input type="text"/> A query that will return one or more rows if the given username and password combination provided is valid. The username and password will be inserted into this query whenever "\$\$username\$\$" and "\$\$password\$\$" are found. Example: <code>SELECT Username FROM USERS WHERE Username = '\$\$username\$\$' AND Password = MD5(\$\$password\$\$)</code>
User's Roles Query	<input type="text"/> A query that returns all of the roles that the provided user belongs to. The roles must be strings (i.e. the role names), and must be in the first column of the query's results. The username will be inserted into this query whenever "\$\$username\$\$" is found. Example: <code>SELECT Rolename FROM USER_ROLE_MAPPING WHERE Username = '\$\$username\$\$'</code>
Role List Query	<input type="text"/> A query that returns all possible roles that any user could have. The role names must be returned in the first column of the query's results. Example: <code>SELECT Rolename FROM ROLES</code>

Add Profile

FactoryPMI v. 1.0.0_17 is © 2002-2004 by Inductive Automation. All rights reserved. [\[view license\]](#)

Fig. 1: The Datasource profile config page

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



ActiveDirectory Based Profile

An ActiveDirectory® based profile lets you interface FactoryPMI with your existing corporate authentication architecture centered on Microsoft ActiveDirectory.

To add an ActiveDirectory based profile, click the "New Profile" icon from the profiles list (+), and select "Active Directory".

Add Authentication Profile Step 1: Choose Source

See the [Authentication Types](#) page for a description of the different authentication sources.

Authentication Source

ActiveDirectory

ActiveDirectory This authentication profile will use.

Datasource

Internal

Next >

Directory".

On the profile configuration you will have to specify a number of settings that outline how FactoryPMI will connect to ActiveDirectory.

Domain

This is Windows domain that your server is a member of. Typically this should be something like MyCompany.com or MegaWidgets.local

Gateway Username/Password

This username and password will be used to connect to Active Directory to retrieve the list of all possible roles. This is used to support the Designer's security settings windows. An incorrect setting here will not prohibit the profile from working correctly.

Primary Domain Controller Host/Port

The IP address or hostname and port of your primary domain controller. If you do not know this information, please consult your network administrator. This is crucial for an Active Directory authentication profile to function properly.

Secondary Domain Controller Host/Port

The IP address or hostname and port of your secondary domain controller, if any. This will be used in case the primary domain controller cannot be reached.



Add Authentication Profile Step 2: Configure Properties

General Properties

Name	<input type="text"/>	Choose a name to identify this authentication profile.
Description	<input type="text"/>	
Failover Profile	- none - <input type="button" value="▼"/>	
If the authentication source is unreachable, this failover profile will be used instead.		

Active Directory Properties

Domain	<input type="text" value="MyCompany.com"/>	The Windows domain for this Active Directory server. Examples: "MyCompany.com" or "SuperCorp.local". If you aren't sure of your domain, ask your network administrator.
Gateway Username	<input type="text"/>	The login name for the gateway to use when querying Active Directory. Only used for retrieving the entire list of roles for the Designer. Not strictly necessary for authentication.
Gateway Password	<input type="text"/>	The password for the above username.
<input type="button" value="Cancel"/>		



Help



Primary Domain Controller Host	<input type="text" value="x.x.x.x"/> The IP address or hostname of your primary domain controller. Example: "192.168.1.4" or "MainServer"
Primary Domain Controller Port	<input type="text" value="389"/> The port number for the primary domain controller's LDAP interface. The default is 389.
Secondary Domain Controller Host	<input type="text" value="y.y.y.y"/> The IP address or hostname of your secondary domain controller (optional). Example: "192.168.1.4" or "MainServer"
Secondary Domain Controller Port	<input type="text" value="389"/> The port number for the secondary domain controller's LDAP interface. The default is 389.

Add Profile

FactoryPMI v. (Dev Version) is © 2002-2004 by Inductive Automation. All rights reserved. [view license]

Fig. 1: The ActiveDirectory® profile config page

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.
Microsoft ActiveDirectory® is a registered trademark of the Microsoft Corporation.



Active Directory / Internal Hybrid Profile

This authentication profile type is a hybrid between the [Active Directory profile](#) and the [Internal profile](#). To add an Active Directory / Internal Hyrbid based profile, click the "New Profile" icon from the profiles list (+), and select "AD/Internal Hybrid".

Add Authentication Profile Step 1: Choose Source

See the [Authentication Types](#) page for a description of the different authentication sources.

The screenshot shows a dropdown menu titled 'Authentication Source' with the option 'AD/Internal Hybrid' highlighted. A tooltip indicates that this selection will use 'ActiveDirectory' and 'Datasource' authentication methods. Other options in the menu are 'Internal' and 'External'.

This hybrid authentication source uses Active Directory to authenticate a user (that is, provide a yes/no answer for whether or not a given username/password combination is valid), but uses the Internal authentication source's style of role management. This internal role structure can be configured in the same way that an Internal Authentication Profile's users and roles are administered, except that it won't ask you for any user's password. It is simply a role mapping.

This separation allows IT to manager usernames and passwords, but allows the FactoryPMI designer to administer their role mappings, without having to bother IT to add new roles to the corporate Active Directory setup. This is a win-win, as you get the security and central user management of Active Directory, but without the sometimes cumbersome cooperation of role definition between engineering and IT.

For example, suppose that you had a user named **Joe**. When **Joe** logged on, this authentication profile would ask Active Directory if **Joe** and has password are a valid user. If Joe authenticated against Active Directory successfully, the profile will then look for a user named **Joe** (case insensitive) in the internal configuration. If one is there, the roles assigned to the *internal* "Joe" user will be used for Joe's roles.

On the profile configuration you will have to specify a number of settings that outline how FactoryPMI will connect to ActiveDirectory.

Domain

This is Windows domain that your server is a member of. Typically this should be something like MyCompany.com or MegaWidgets.local

Primary Domain Controller Host/Port

The IP address or hostname and port of your primary domain controller. If you do not know this information, please consult your network administrator. This is crucial for an Active Directory authentication profile to function properly.

Secondary Domain Controller Host/Port

The IP address or hostname and port of your secondary domain controller, if any. This will be used in case the primary domain controller cannot be reached.



Logged in as: admin [logout] [restart] [shutdown]

System
Status
Settings
Network
Cluster
Backup/Restore
Projects
Manage
Launch
Datasources
Connections
Drivers
Tunes

Add Authentication Profile Step 2: Configure Properties

General Properties	
Name	<input type="text"/> Choose a name to identify this authentication profile.
Description	<input type="text"/>

Authentication**Profiles**

Types

Auditing

Profiles

Plugins

Manage

Images

Manage

**Failover Profile** - none -

If the authentication source is unreachable, this failover profile will be used instead.

Active Directory Properties

Domain

MyCompany.com

The Windows domain for this Active Directory server. Examples: "MyCompany.com" or "SuperCorp.local". If you aren't sure of your domain, ask your network administrator.

Primary Domain Controller Host

x.x.x.x

The IP address or hostname of your primary domain controller. Example: "192.168.1.4" or "MainServer"

Primary Domain Controller Port

389

The port number for the primary domain controller's LDAP interface. The default is 389.

Secondary Domain Controller Host

y.y.y.y

The IP address or hostname of your secondary domain controller (optional). Example: "192.168.1.4" or "MainServer"

Secondary Domain Controller Port

389

The port number for the secondary domain controller's LDAP interface. The default is 389.

 Add Profile

FactoryPMI v. (Dev Version) is © 2002-2004 by Inductive Automation. All rights reserved. [view license]

Fig. 1: The Hybrid profile config page

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.
Microsoft ActiveDirectory® is a registered trademark of the Microsoft Corporation.



Configuring Audit Profiles

Auditing is a way to keep track of what data changes to your datasources are made through FactoryPMI. It will store a log of all important events that happen inside FactoryPMI, including log-ins, log-outs, and all UPDATE and DELETE queries executed.

To use auditing, you simply need to create an *Audit Profile*, and then assign the profile to your project. To create an audit profile, click on **Auditing - Profiles**, and then click on the New Profile icon.

Profile Name	Datasource	Table	Description
No audit profiles configured.			

Configured Profiles   **New Profile → **

The audit profile configuration page is rather straight-forward and self documenting. You must provide a name and a datasource at minimum, and the FactoryPMI gateway will take care of the rest. You may also select "Manual Configuration", and enter all of the parameters yourself, if the audit table already exists.

Once the audit profile is created, you must assign it to one or more projects. To do this, simply go to **Projects - Manage**, edit project, select "Auditing Enabled", and select your profile from the list. Once you save your project, auditing will be enabled, and events will begin being recorded.

Image Management

All images for FactoryPMI projects are stored in the server. The Image section of the configuration gateway lets you manage them in a rudimentary way. The FactoryPMI Designer provides a much more powerful, drag-drop style image management tool. See [Designer - User Interface - Image Tool](#) for details.



Creating Folders

You can create organization folders easily by entering a name and clicking "Create Folder".

Path: root /

Create New Folder

Name	MyImageFolder
<input type="button" value="Create Folder"/>	

You may then click on that folder to make it the current working folder.

Path: root / 

Sub folders → 
MyImageFolder

Create New Folder

Name	<input type="text"/>
<input type="button" value="Create Folder"/>	

Uploading Images

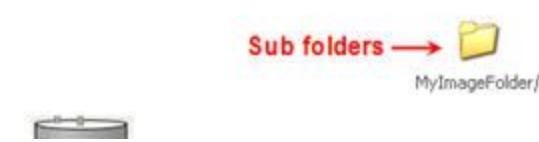
To upload an image, you must give it a name, and optionally a description. Then just browse for it, and click "Upload".

Upload New Image

Image Name	StaticTank Choose a descriptive name to identify this image
Image Description	A static tank image. <input type="text"/>
Image File	c:\tank.jpg <input type="button" value="Browse..."/> Image must be a PNG, GIF, or JPEG image.
<input type="button" value="Upload"/>	

After the image is uploaded, the top part of the page will show the image, located in the current directory.

Path: root / 





← Images in current folder

StaticTank [64x78] (1k)

Create New Folder

Name

Create Folder

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



How to Launch the Designer

To launch, go to the gateway configuration page (by default <http://localhost:8080>), and click the link that says "Launch Designer".

The screenshot shows the FactoryPMI Gateway interface. At the top, there's a green header bar with the title "FactoryPMI Gateway" and "PLANT MANAGEMENT INTERFACE". Below the header, there are three buttons: "Launch Designer" (highlighted with a red oval), "Gateway Status", and "Configuration". Underneath the header, there are two tabs: "Web Start" and "Applet", with "Applet" being the active tab. A large green banner below the tabs displays the word "Projects" in white. Inside this banner, there's a green icon followed by the text "DemoProject". At the bottom of the screen, a green footer bar contains the text "FactoryPMI v. 1.0.0_17 is © 2002-2004 by Inductive Automation. All rights reserved. [view license]".

The link is also available on the left menu, after you log into the gateway.

The screenshot shows the FactoryPMI Gateway Configuration interface. At the top, there's a green header bar with the title "FactoryPMI Gateway Configuration" and "PLANT MANAGEMENT INTERFACE". Below the header, it says "Logged in as: admin [logout] [restart] [shutdown]". On the left, there's a sidebar with a "System" section containing "Status", "Settings", "Backup", and "Launch" (which is bolded). Below that are sections for "Projects", "Datasources", "Authentication", "Auditing", and "Images/Plugins". At the bottom of the sidebar, there's a red oval highlighting the "Launch Designer" link. The main content area is titled "FactoryPMI Gateway Server Status" and contains a table with system status information:

System Status	
Version	1.0.0_17 (build 504)
CD-Key	L55-FG1
Memory (used/max)	6.85mb / 254.06 mb
Running Since	Wed Jun 29 14:24:30 PDT 2005
Uptime	0 days, 4 hours, 10 minutes, 54 seconds

At the bottom of the screen, a green footer bar contains the text "FactoryPMI v. 1.0.0_17 is © 2002-2004 by Inductive Automation. All rights reserved. [view license]".

Once the application has loaded, you will be presented with the splash screen. Log in with a valid username/password (default: admin/password), and the designer will open.



Designer

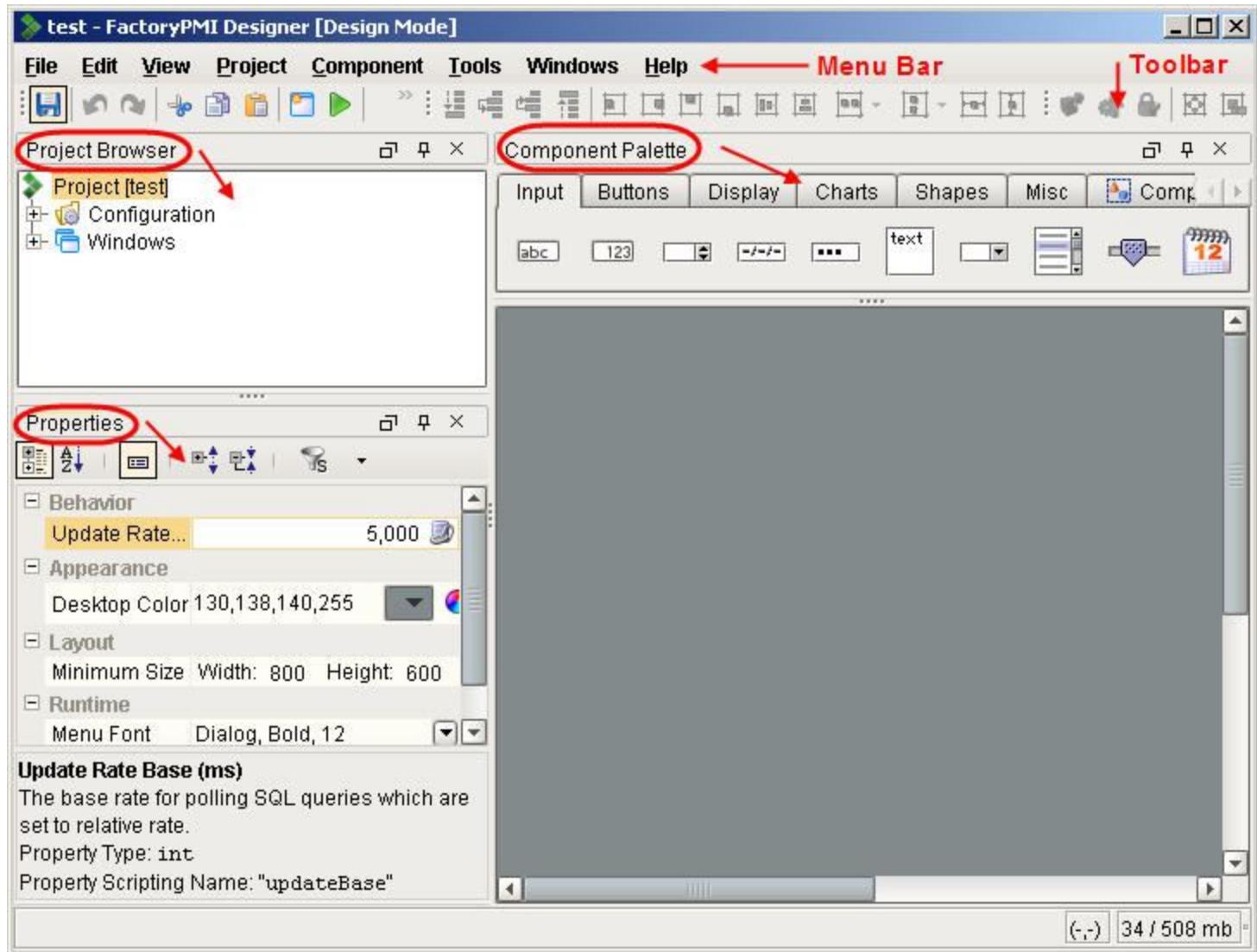
Use of this application is subject to the acceptance of the terms and conditions set forth in the accompanying license agreement.

Copyright © 2004 Inductive Automation. All rights reserved.

Username:

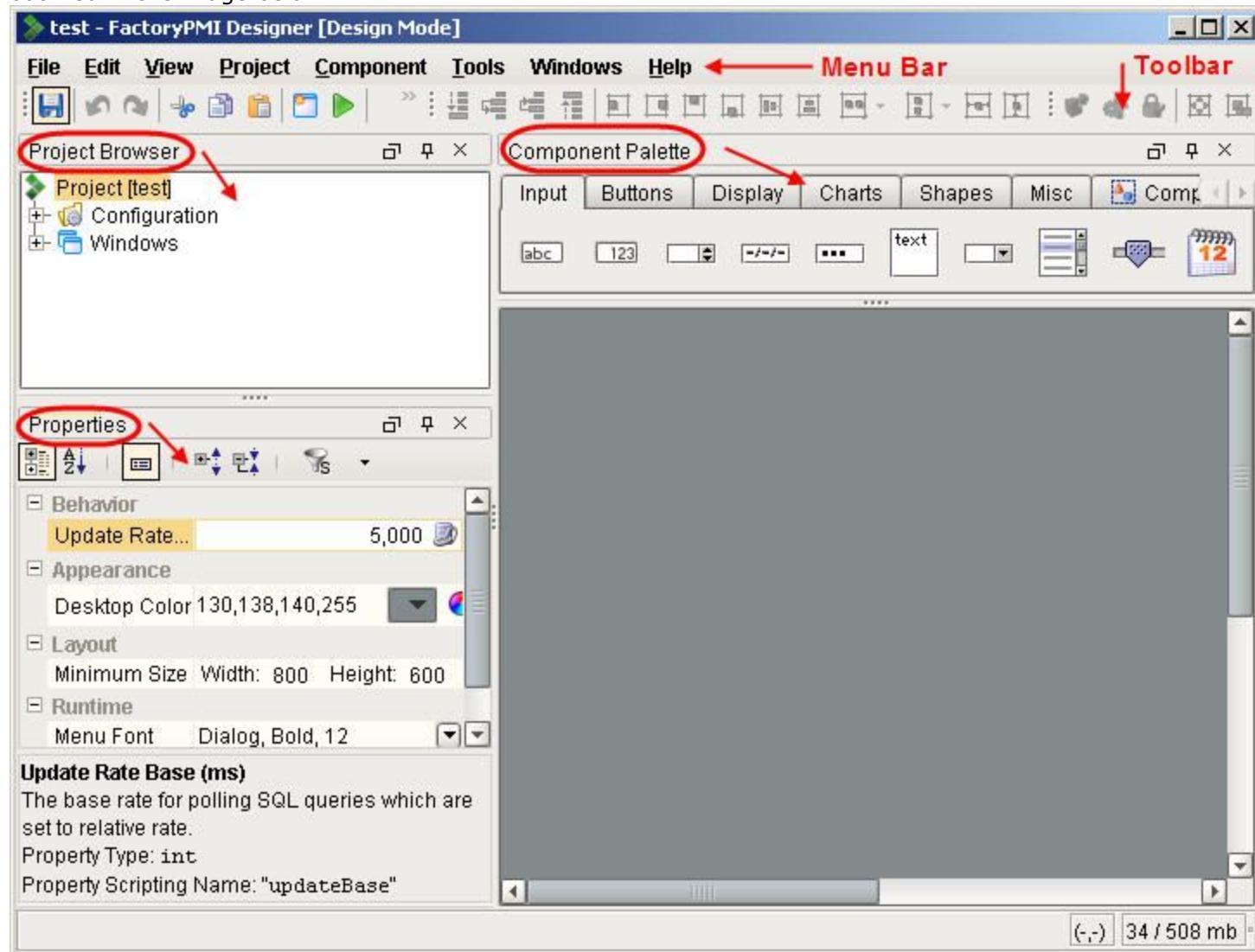
Password:

After logging in, you will be asked to choose a project to work on. If you do not have a project created, you can make one by clicking on the "Create New" tab, or by [adding a project in the configuration gateway](#).



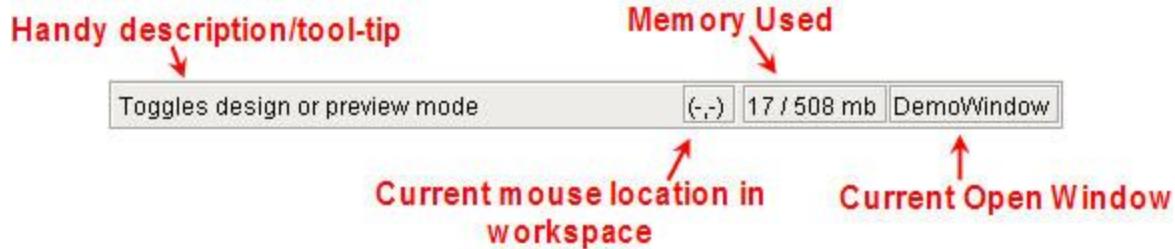
User Interface Overview

The FactoryPMI designer is the application used to create projects. There are several main areas of the window, outlined in the image below:



The [Project Workspace](#) is where the graphical elements of your project appear, and is a working representation of your project. That is, everything will appear the same in the running program as it does in the project workspace. In fact, you can turn the workspace into the running project by clicking the "Preview Mode" button (▶) on the toolbar.

At the very bottom of the window there is an information bar, that shows various pieces of useful data. On the right, it shows the current mouse location, how much memory the designer is currently using (out of the available amount), and the current active window. On the left side, various pieces of information will appear (often a description of the button or item currently under the mouse):



The rest of this section will outline each particular area, button, and menu item. This is meant to be a straightforward documentation of functionality. For help on specific features, see [Concepts](#).



Menu Bar

File Edit View Project Component Tools Windows Help

The menu bar provides quick access to many common functions, as well as program wide options and settings. It is divided into eight sections:

- [File](#)
- [Edit](#)
- [View](#)
- [Project](#)
- [Component](#)
- [Tools](#)
- [Windows](#)
- [Help](#)

File

The file menu provides access to common functions for dealing with projects.

Menu Item	Function
New	Create a new project.
Open	Open a different project.
Save	Commits open windows and saves the current project back to the service.
Save As	Commits open windows and saves the current project back to the service as a new project.
Update Project	Downloads recent version of project. Useful for multiple concurrent designers.
Export Backup	Commits open windows and saves the project to a backup file. These files are then imported in the Gateway project page .
Exit	Close the designer.

Edit

The edit menu provides functions like cut, copy and paste.

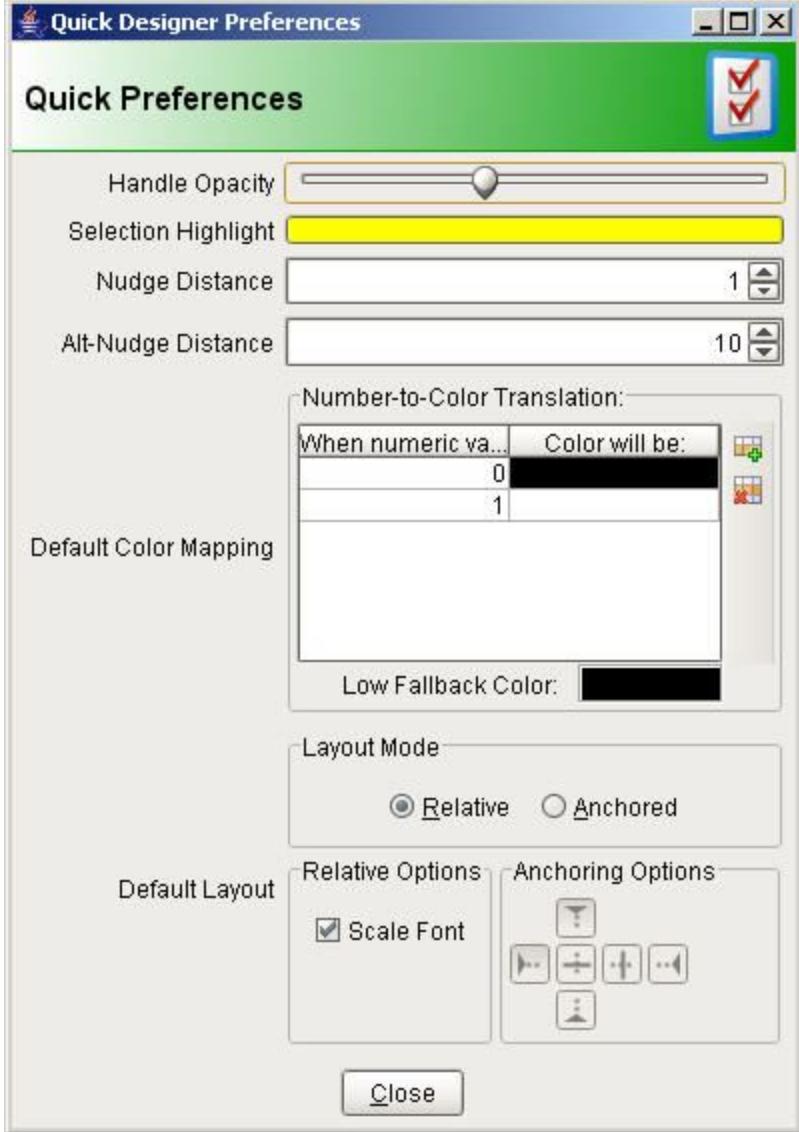
Menu Item	Function
Undo	Undoes the last action
Redo	Undoes the last undo action
Cut	Remove the item from the current window, and place it on the clipboard.
Copy	Put the item on the clipboard, but leave the current object alone.
Duplicate	Copy and paste immediate without going to the clipboard. Works with components and windows.
Paste	Begin the paste procedure. The current mouse icon will change, indicating that you are pasting. Simply click on the window where you want the object to be placed, and the item will be created there.

Cancel Paste	Breaks out of the paste routine after it has been started.
Paste Immediate	Paste the item on the clipboard to its original location.
Select All	Selects all components that are siblings of the selected component.
Select Same Type	Selects all components that are siblings of the selected component and the same component type.
Select Same Type in Window	Selects all components in a window that are of the same type as the selected component.
Delete	Remove the selected object from the window.
Group Rename	Allows you to rename multiple items at once. Select a number of components, choose this menu item, and then pick a prefix for the naming. The items will now be named sequentially.

View

The view menu contains a number of features that affect how the UI is displayed, and how the Project Workspace acts.

Menu Item	Function												
Emulate Touchscreen	If touchscreen mode is enabled for the project, this will emulate a touch screen.												
Toggle Relative Scaling	Toggles whether relative scaling is enabled during this designer session. Typically relative scaling should be off in the designer.												
Reset Layout	Resets toolbars and panels to their default positions.												
Toolbars	<table border="1"> <thead> <tr> <th>Toolbar Item</th> <th>Buttons</th> </tr> </thead> <tbody> <tr> <td>Component</td><td>Customizer, Jython Actions, Security, Layout, and Size/Position.</td></tr> <tr> <td>Layout</td><td>Various layout, orientation, and Z-ordering buttons.</td></tr> <tr> <td>Main</td><td>Save, Undo, Redo, Cut, Copy, Paste, New Window, Preview, SQL Datasources Off/Read Only/Read-Write.</td></tr> </tbody> </table>	Toolbar Item	Buttons	Component	Customizer, Jython Actions, Security, Layout, and Size/Position.	Layout	Various layout, orientation, and Z-ordering buttons.	Main	Save, Undo, Redo, Cut, Copy, Paste, New Window, Preview, SQL Datasources Off/Read Only/Read-Write.				
Toolbar Item	Buttons												
Component	Customizer, Jython Actions, Security, Layout, and Size/Position.												
Layout	Various layout, orientation, and Z-ordering buttons.												
Main	Save, Undo, Redo, Cut, Copy, Paste, New Window, Preview, SQL Datasources Off/Read Only/Read-Write.												
Panels	<table border="1"> <thead> <tr> <th>Panel</th> <th>Function</th> </tr> </thead> <tbody> <tr> <td>Collapsible Palette</td><td>FactoryPMI object palette that works well in a vertical orientation.</td></tr> <tr> <td>Output Console</td><td>Output Console for debugging.</td></tr> <tr> <td>Project Browser</td><td>Hierarchical tree used for editing Windows, objects, and scripts.</td></tr> <tr> <td>Properties</td><td>Panel where object properties and bindings are edited.</td></tr> <tr> <td>Tabbed Palette</td><td>Horizontally oriented panel that contains FactoryPMI objects.</td></tr> </tbody> </table>	Panel	Function	Collapsible Palette	FactoryPMI object palette that works well in a vertical orientation.	Output Console	Output Console for debugging.	Project Browser	Hierarchical tree used for editing Windows, objects, and scripts.	Properties	Panel where object properties and bindings are edited.	Tabbed Palette	Horizontally oriented panel that contains FactoryPMI objects.
Panel	Function												
Collapsible Palette	FactoryPMI object palette that works well in a vertical orientation.												
Output Console	Output Console for debugging.												
Project Browser	Hierarchical tree used for editing Windows, objects, and scripts.												
Properties	Panel where object properties and bindings are edited.												
Tabbed Palette	Horizontally oriented panel that contains FactoryPMI objects.												
Show Grid	Determines whether the grid is displayed over the top of the project workspace.												
Snap to Grid	Toggles whether moving or resizing components snaps to the grid.												
Grid Off	Sets grid size as off or in 5 or 10 pixel squares.												
Grid Size 5													
Grid Size 10													

Dependencies	Determines how dependencies between components are shown. See also Databinding - To Components
Quick Preferences	Brings up the quick preference menu, which lets you specify certain display settings such as handle opacity, selection highlight, and nudge distance.
 <p>Quick Preferences Dialog</p>	

Project

This menu lets you access project wide script modules and events

Menu Item	Function
Preview Mode 	Turns "preview mode" on and off. That is, determines whether the project workspace is in design mode (normal), or acts like the running program (preview mode).
New Window 	Adds a new window to the project
Runtime Menu 	Edits the runtime application's menu items.
Global Event Scripts 	Lets you edit the global events. See Events - Global Events
Script Modules 	This lets you edit and manage custom script modules that can be called from any other script in the system under the <code>app.*</code> package. See Events for more info.

Component

This menu lets you modify the selected FactoryPMI component

Menu Item	Function
Group Container 	Sets container to be treated as a group. Children of the container can no longer be selected in the designer (except via the Project Browser). Convenient once objects are set up
Lock 	Locks or unlocks components' size and location.
Layout	Configure layout constraints for selected components.
Size & Position 	Type exact size and position for component or window.
Move to Front 	Moves components to the front of the Z-order.
Move to Back 	Moves components to the back of the Z-order.
Move Forward 	Moves components forward in Z-order.
Move Backward 	Moves components backward in Z-order.
Component Customizers 	Shows the customizers for selected component. Used for editing complex properties.
Configure Actions 	Configure actions (Jython event handlers) for the selected component.
Component Security 	Shows the security settings for selected component.

Tools

The tools menu houses several utilities that can help you develop and test.

Menu Item	Function								
Image Management 	This tool lets you upload, download, rename, and delete images from the Gateway. See Image Tool for more information.								
Designtime Security Tool 	This tool lets you specify what roles are assigned to you during the designer session. This lets you test the functionality of your project from the view of a specific user.								
Advanced	<table border="1"><thead><tr><th>Menu Item</th><th>Function</th></tr></thead><tbody><tr><td>Console </td><td>Opens the system console. Scripts' <code>print</code> output is printed to this console. Also, if you discover a bug in FactoryPMI, you can find information about what went wrong on the console</td></tr><tr><td>Script Playgroud </td><td>Lets you run any Jython script. This is a good way to play with the Jython language or to test a piece of Jython code.</td></tr><tr><td>Reserialize Windows </td><td>Opens and commits (a.k.a. serializes) each window. Sometimes when you upgrade to a new version of FactoryPMI you will be instructed to reserialize all of your windows. This is the easiest way to do so.</td></tr></tbody></table>	Menu Item	Function	Console 	Opens the system console. Scripts' <code>print</code> output is printed to this console. Also, if you discover a bug in FactoryPMI, you can find information about what went wrong on the console	Script Playgroud 	Lets you run any Jython script. This is a good way to play with the Jython language or to test a piece of Jython code.	Reserialize Windows 	Opens and commits (a.k.a. serializes) each window. Sometimes when you upgrade to a new version of FactoryPMI you will be instructed to reserialize all of your windows. This is the easiest way to do so.
Menu Item	Function								
Console 	Opens the system console. Scripts' <code>print</code> output is printed to this console. Also, if you discover a bug in FactoryPMI, you can find information about what went wrong on the console								
Script Playgroud 	Lets you run any Jython script. This is a good way to play with the Jython language or to test a piece of Jython code.								
Reserialize Windows 	Opens and commits (a.k.a. serializes) each window. Sometimes when you upgrade to a new version of FactoryPMI you will be instructed to reserialize all of your windows. This is the easiest way to do so.								

Windows

The Windows menu is a familiar menu that shows the currently opened windows and lets you switch between them.

Help

Menu Item	Function
Help 	Launches this documentation.
About 	Shows you the about screen, which has various pieces of data that are useful for technical support. You can also launch the Console and view FactoryPMI's license here.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Toolbars

The FactoryPMI Designer has 3 toolbars that provide quick access to useful and important features. Each can be enabled or disabled by the View - Toolbars menu.

Main 

Icon	Name	Description
	Save Project	Saves the project to the gateway.
	Undo	Undoes the last action
	Redo	Undoes the last undo action
	Cut	Remove the item from the current window, and place it on the clipboard.
	Copy	Put the item on the clipboard, but leave the current object alone.
	Paste	Begin the paste procedure. The current mouse icon will change, indicating that you are pasting. Simply click on the window where you want the object to be placed, and the item will be created there.
	New Window	Adds a new window to the project.
	Start / Stop Preview Mode	Previews project in designer as would be seen by runtime.
	Gateway Off	Disconnects database communication with the gateway.
	Gateway Read-Only	Allows read-only database communication with the gateway.
	Gateway Read-Write	Allows full database access through the gateway.

Layout 

Icon	Name	Description
	Move to Back	Moves components to the back of the Z-order.
	Move Backward	Moves components backward in Z-order.
	Move Forward	Moves components forward in Z-order.
	Move to Front	Moves components to the front of the Z-order.
	Align Left	Aligns all of the selected components with the left edge of the leftmost component.
	Align Right	Aligns all of the selected components with the right edge of the rightmost component.
	Align Top	Aligns the top edges of the selected components to the top of the highest component.
	Align Bottom	Aligns the bottom edges of the selected components to the bottom of the lowest component.
	Align Center Horizontally	Aligns 2 or more components center points horizontally.
	Align Center Vertically	Aligns 2 or more components center points vertically.

	Align as Row	Aligns the selected components in a row. A pop-up dialog allows you to specify horizontal spacing.
	Align as Stack	Aligns the selected components in a column. A pop-up dialog allows you to specify vertical spacing.
	Center Horizontally	Centers the selected components horizontally with respect to their parent container.
	Center Vertically	Centers the selected components vertically with respect to their parent container.

Component

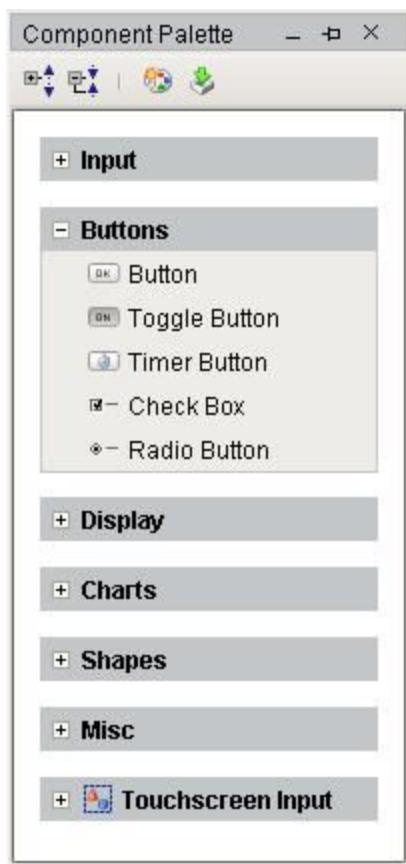


Icon	Name	Description
	Customizers	Shows the customizers for selected component. Used for editing complex properties.
	Configure Actions	Configure actions (Jython event handlers) for the selected component.
	Configure Security	Shows the security settings for selected component.
	Layout	Configure layout constraints for selected components.
	Size & Position	Type exact size and position for component or window.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Component Palettes

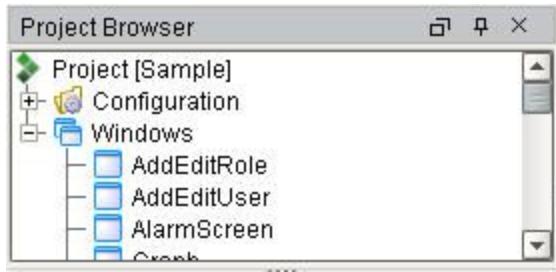
The component palettes hold all of the components available for you to use in your projects. [Custom Palettes](#) can be added to the list which house components built from collections of other components. See [Component Placement](#) for more info on how to place a component in your window.



See the [Component Technical Reference](#) for more information on individual components.

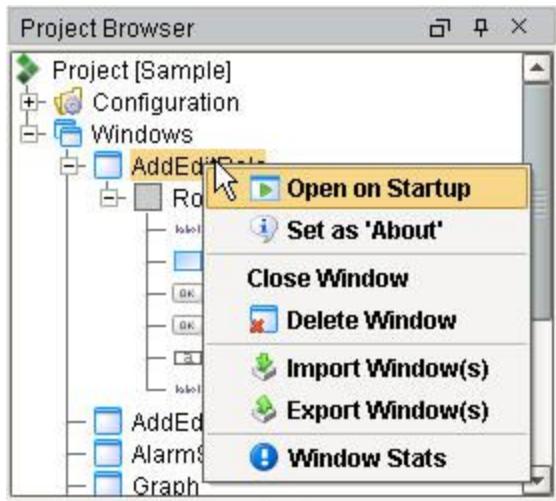
Project Browser

The project browser is a tree that shows the components of your project. On the top level of the tree are windows. Below them are all of the components they contain (a window only displays its contents in the project browser if it is open).



The following right click menus are available in the Project Browser Window:

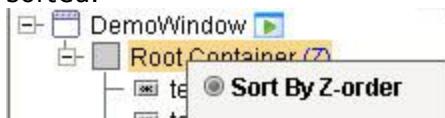
Window Right-Click Menu



Menu Item	Function
Open/Don't Open on Startup	Sets the window to open (or not) when the project is launched. Windows that are set to startup automatically have the startup icon (next to them in the project browser).
Set as 'About'	Sets the selected window to be the "About" window, when users click About in the running project.
Close Window	Closes open window.
Delete Window	Deletes the window from the project (window must be closed in the workspace).
Import Window(s)	Load one or more windows from a file.
Export Window(s)	Saves windows to a file.
Window Stats	Compressed size of the window.

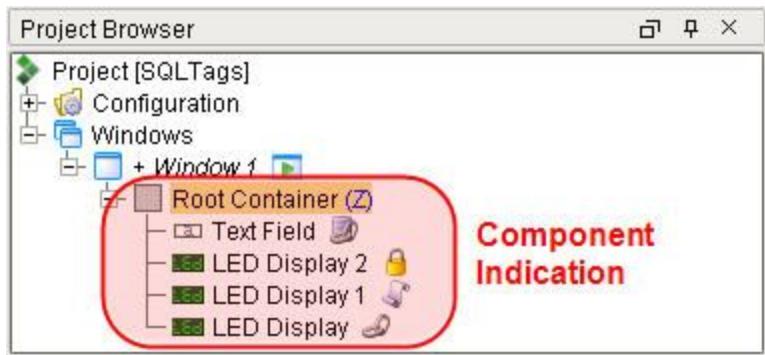
Container Right-Click Menu

Right clicking on a container shows the following menu, which lets you specify how components beneath it are sorted.



- Sort By Name
- Sort By Type

Component indication



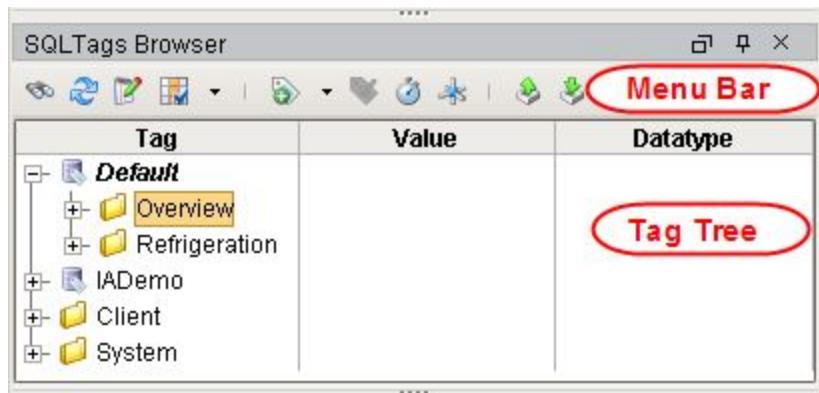
Components are displayed in the Project Browser under windows. They may be selected in the tree as if they were selected in the [project workspace](#). Components with binding, action scripts, or security are indicated with icons on the right side.

Icon	Meaning
Security	Component has been configured using builtin security .
Jython	Component has at least one Jython Action Script defined on it.
Property Binding	At least one component property is bound to another property or expression .
SQL binding	At least one property is bound to the database or an SQL query.
SQL and property binding	Component has the 2 above bindings.

Components may indicate security, Jython, and one of the 3 binding icons together.

SQLTags™ Browser

The SQLTags Browser is the panel that is your main interface to the SQLTags system. It is where you create and edit SQLTags. There are many [drag and drop](#) applications. A common one is creating OPC SQLTags by dragging OPC tags from the [OPC Browser Tree](#) to the SQLTags Browser Tag tree.



Menu Bar

Menu Item	Function
Search	Find tags by name. Hides/shows <i>Find</i> search dialog.
Refresh	Refresh Tag Tree.
Edit Values	Toggles ability to edit SQLTags values.
Column Selection	Select columns to view in Tag Tree.
Create Tag	Dropdown list to create new tags or folders.
Edit Tag	Edit Tag Properties .
Scan Class editor	Edit SQLTags scan classes .
OPC Browser	Open OPC Browser .
Import/Export CSV	Import and Export SQLTags to a .CSV file. The recommended usage is to export the file, edit it with your favorite spreadsheet, then import the file.

Tag Tree

The Tag Tree displays SQLTags, their properties, and values. The root of each tree is a **SQLTags Provider**, which is really a SQLTags enabled [Data Source connection](#). Use the Column Selection dropdown to choose displayed columns. Items in the Tag Tree can be [dragged and dropped](#) onto components and properties. See [SQLTags types](#) for more information on the types of SQLTags. The following tag icons will be displayed.

Tag Icon	Meaning
Folder	A SQLTags folder. It contains SQLTags and folders.
Client Tag (green)	Client tags are variables of any type that can be accessed from anywhere in a FactoryPMI client. They can be thought of as global dynamic properties that operate within an instance of a FactoryPMI client. Separate FactoryPMI clients will have separate values of Client tags. Client Tags are not persistent , meaning tag values will re-initialize if the user logs out of, or closes, the application.
DB Tag (yellow)	A DB Tag is stored in the SQL database. This value is shared between

	FactoryPMI clients and is persistent (value doesn't change from logging off or closing a FactoryPMI client). The left icon indicates that the value is being updated, where the (red tinted) right icon indicates that the value is old.
OPC Tag (blue)  	OPC Tags read and write their values from an OPC path via FactorySQL.
System Tag (purple) 	Tags whose values are system properties such as: CurrentDateTime, Username, IPAddress, etc. System Tags are typically read only.
Tag forbidden/disabled  	User does not have sufficient permission or tag has been set to disabled, respectively.
Expression/SQL 	Indicates SQLTag is bound to an expression or SQL Query. Applicable for DB and Client SQLTags.
Security 	Indicates tag has permissions configured. Applicable for all SQLTags.
Alert 	Indicates tag has an alert configured. Applicable for DB and OPC SQLTags.

Search Dialog

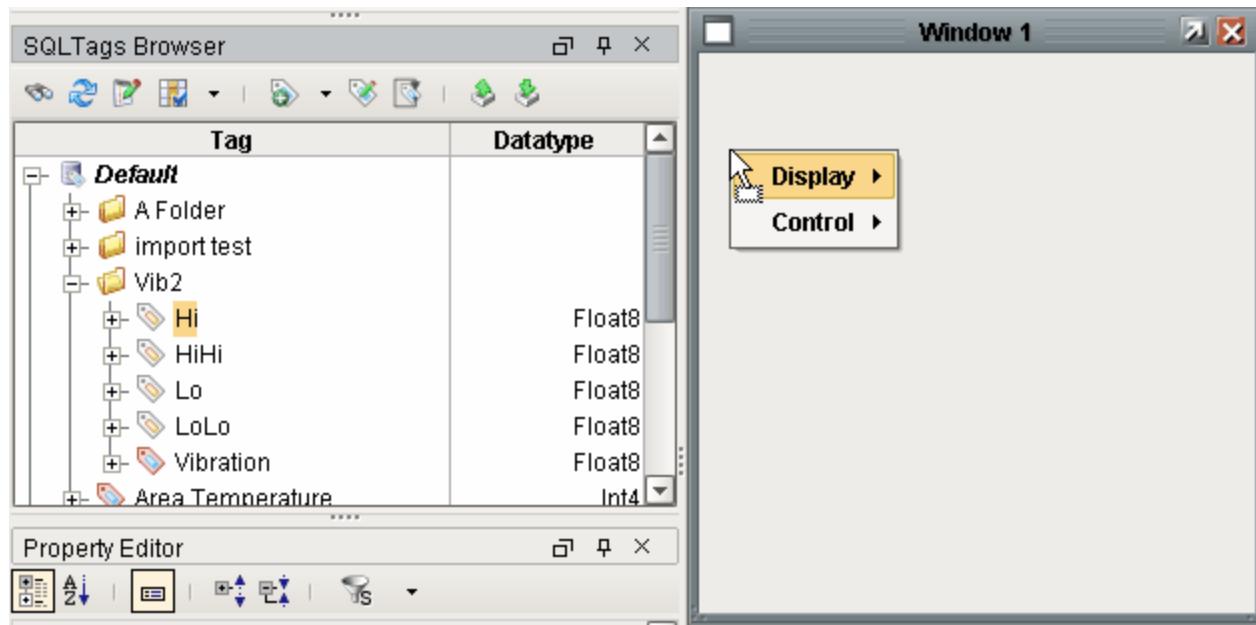
Search  opens the [Tag Search](#) dialog for the selected SQLTags provider.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Overview

SQLTags™ supports a variety of drag and drop operations. Below is an example of dragging a DB SQLTag onto the Root Container of a Window to create display and control components.

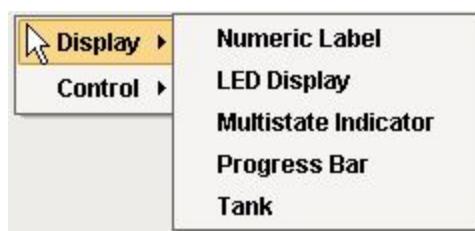


Components and Properties

Components understand how to receive a SQLTag drag and drop request. They behave differently depending on what they are. Here are a few examples:

Containers

Dragging a SQLTag onto a container prompts a menu for what kind of component to create. Dragging a property or the tag itself will have the same effect. There are a number of display and control components to choose from. The new component will have multiple properties bound to different values on the SQLTag. For example, a [numeric label](#) created from a SQLTag will have the following properties bound: **Mouseover Text**, **Value**, **Units**, and **Number Format Pattern**.



Components

Dragging a SQLTag onto a component will bind as many properties as that component is set up to. It tends to be similar to creating a new component by dragging the tag onto a container.



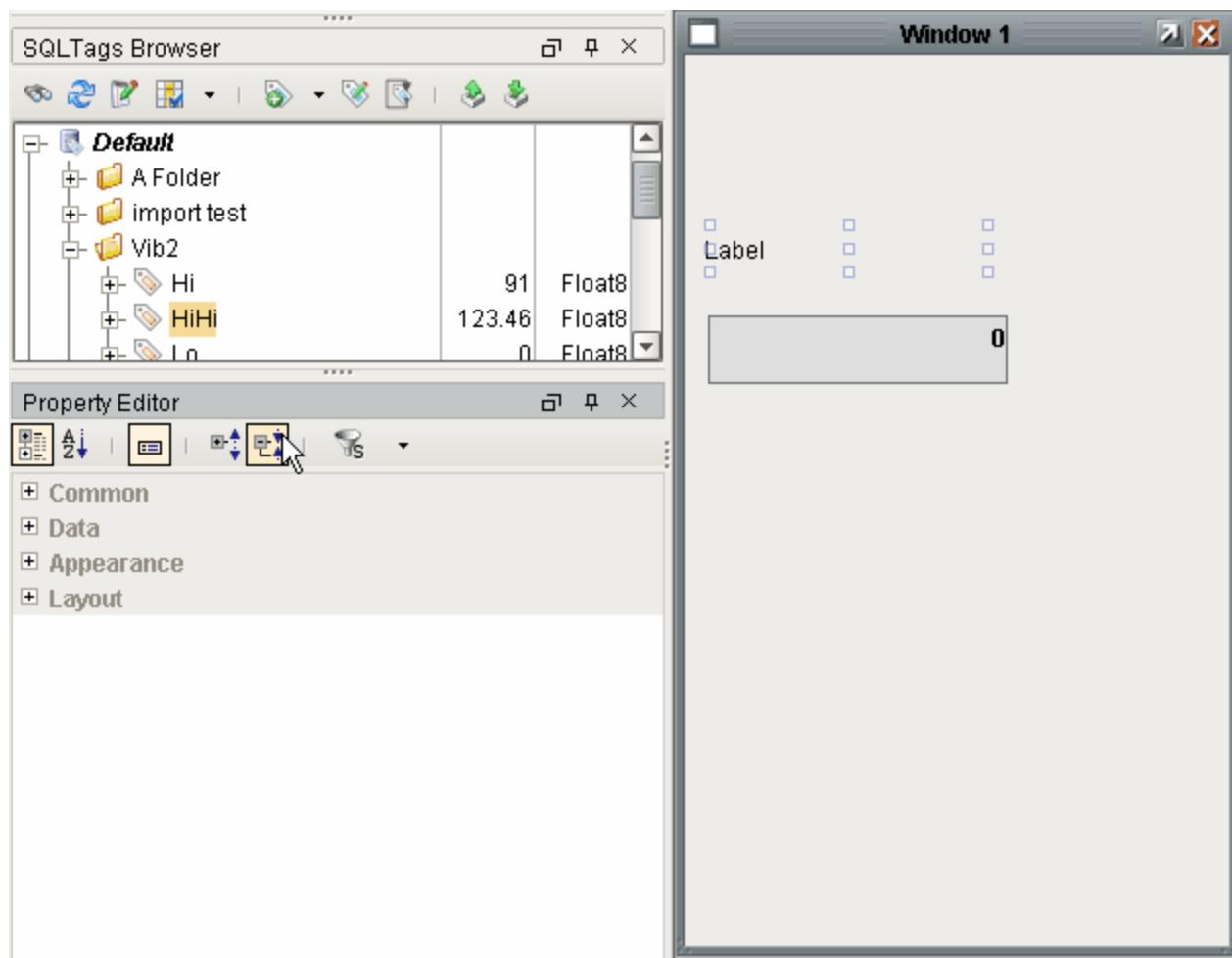
Properties

Dragging a SQLTag property onto a component property is a shortcut to binding the component's property. Dragging a SQLTag itself (instead of a property) will bind to the SQLTag, which is the same as the **Value** property. Below indicates dragging a property onto the **Text** property of a [label](#).



Example

The example below illustrates dragging a SQLTag onto a property of a [label](#) and then onto a [numeric label](#).

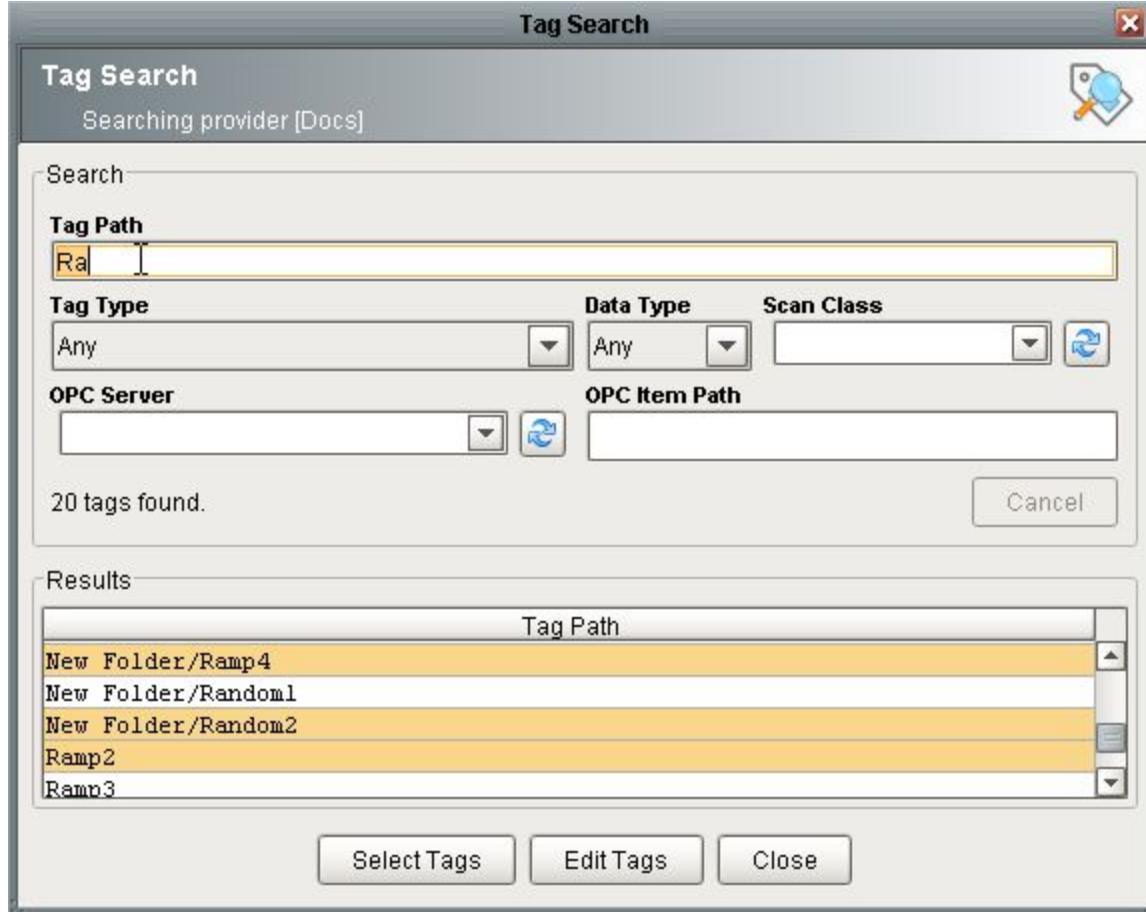


SQLTags™ Tag Search

The Tag Search Window allows you to find SQLTags based on their properties. Select a tag provider (SQLTags enabled Data Source) in the [Tag Browser](#) and click search .

Tip

You may select and edit multiple tags at once!



Search

Menu Item	Function
Tag Path	Searches for string in Tag Path including Tag Name and Folders.
Tag Type	Any , or filter by SQLTag Type .
Data Type	Any , or filter by any data type .
Scan Class	Filter by any scan class . Leave blank to ignore
OPC Server	Filter by OPC server. Leave blank to ignore
OPC Item Path	Filter by OPC item path. Great for finding what is connected to actual PLC registers. Leave blank to ignore

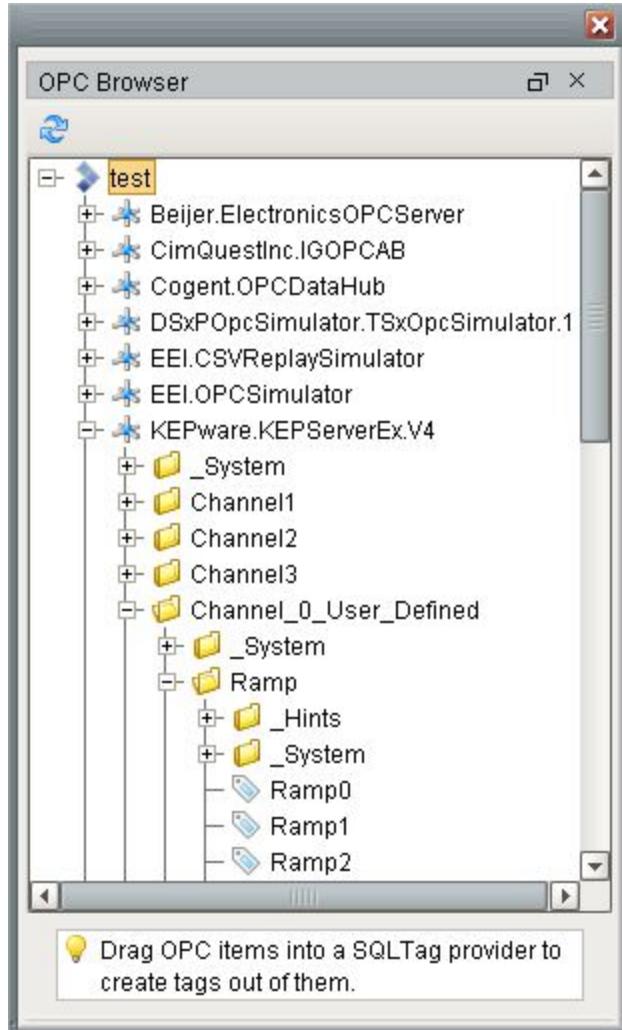
The **Select Tags** button will close the **Tag Search** window and select the tags in the [Tag Browser](#). The **Edit Tags** button will close the **Tag Search** window and open the [Tag Editing Window](#) for the selected tags.



OPC Browser

The SQLTags **OPC Browser** is the panel that allows you to browse OPC items from FactorySQL. You may browse multiple FactorySQL nodes. Each datasource that has SQLTags defined is contacted for browsing. Browsing occurs directly between a FactoryPMI designer and a FactorySQL service. This is a case where the SQL database is not involved.

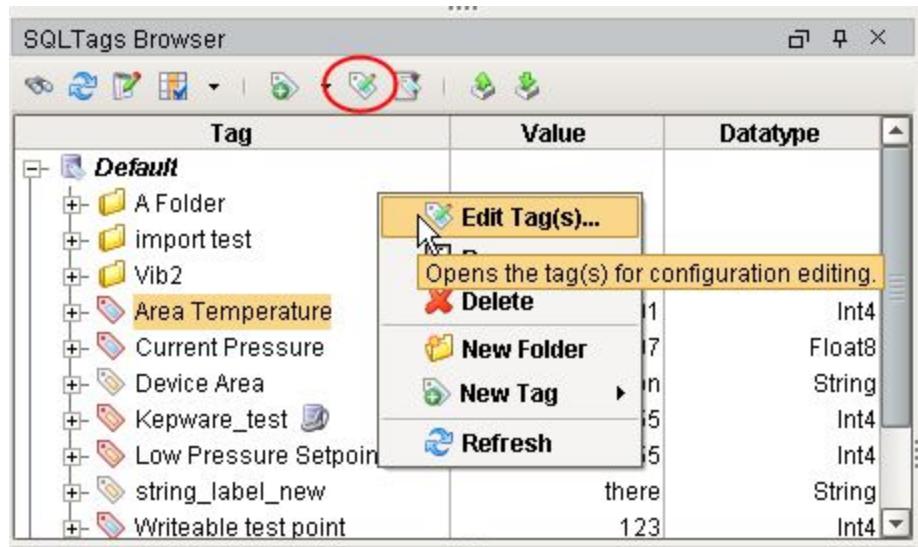
You can create [OPC tags](#) by dragging OPC Items onto SQLTags providers on the [SQLTags Browser](#)



Drag OPC items into a SQLTag provider to create tags out of them.



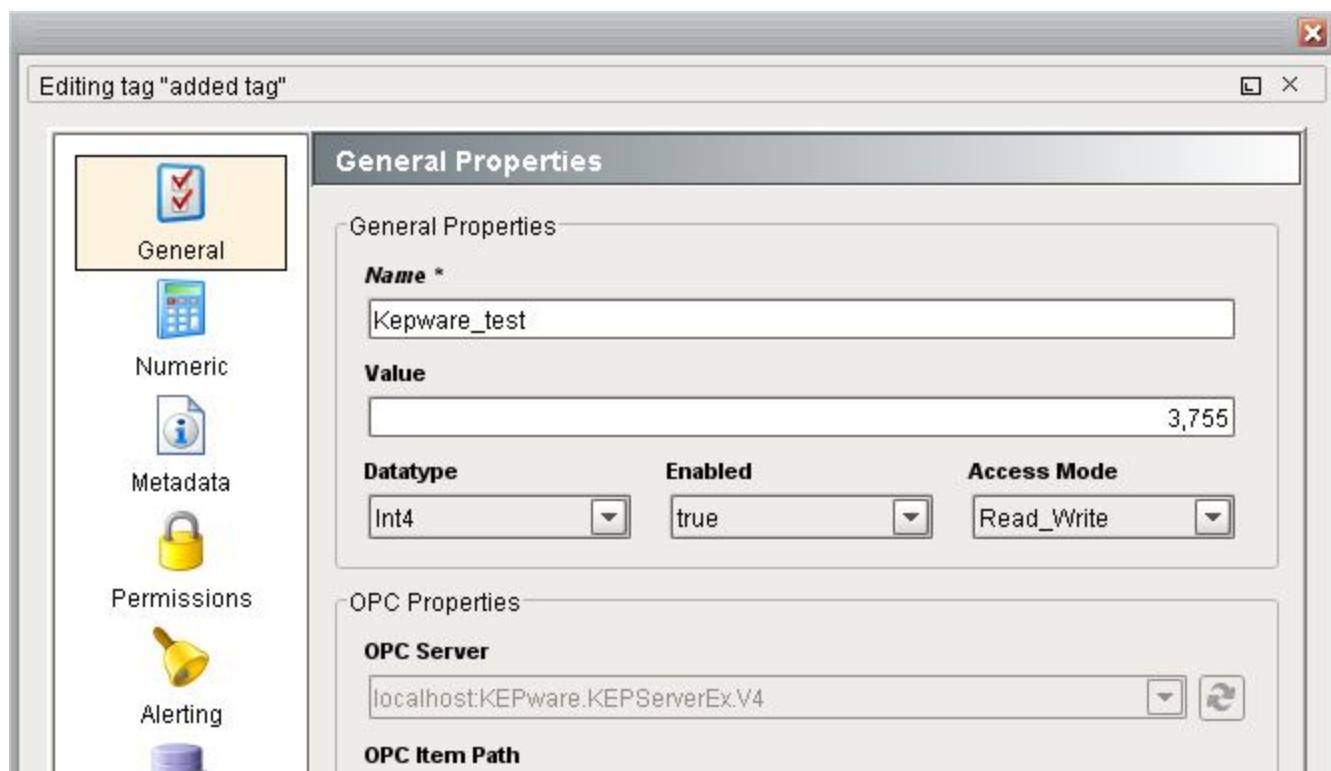
The SQLTags™ Editor is a panel that allows you to edit SQLTags. You get to it by selecting a tag and clicking on Edit tag button of the [SQLTags Browser](#) Menu Bar or right click menu.

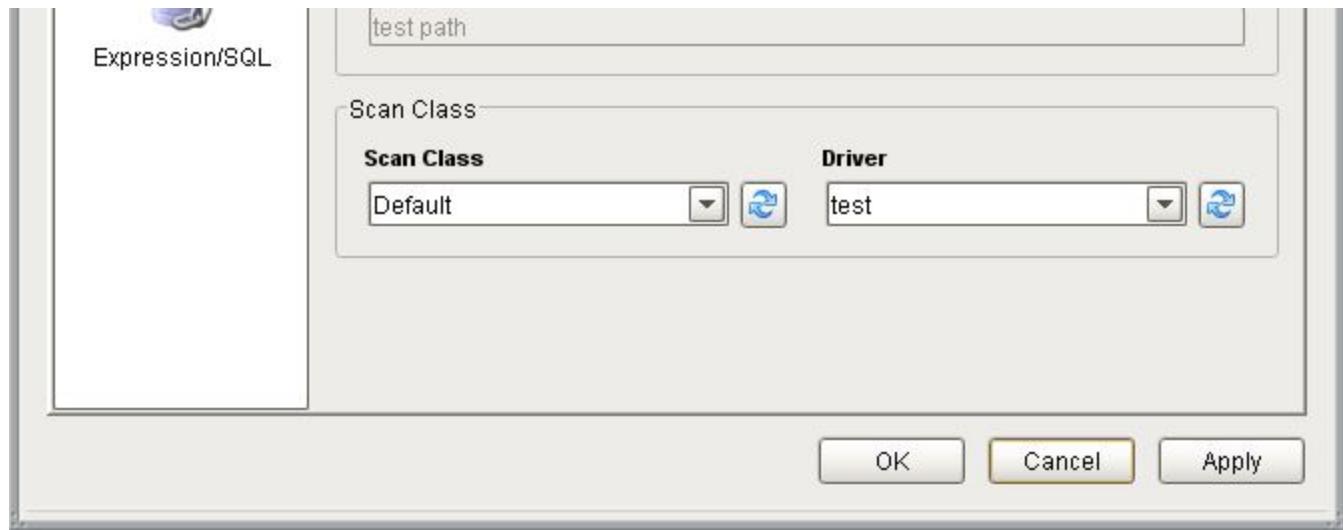


Sections

- [General](#) - applicable to all properties
- [Numeric](#) - applicable to numeric properties of OPC tags
- [Metadata](#) - applicable to all properties
- [Permissions](#) - applicable to all properties
- [Alerting](#) - applicable to OPC and DB SQLTags
- [Expression/SQL](#) - applicable to Client and DB SQLTags

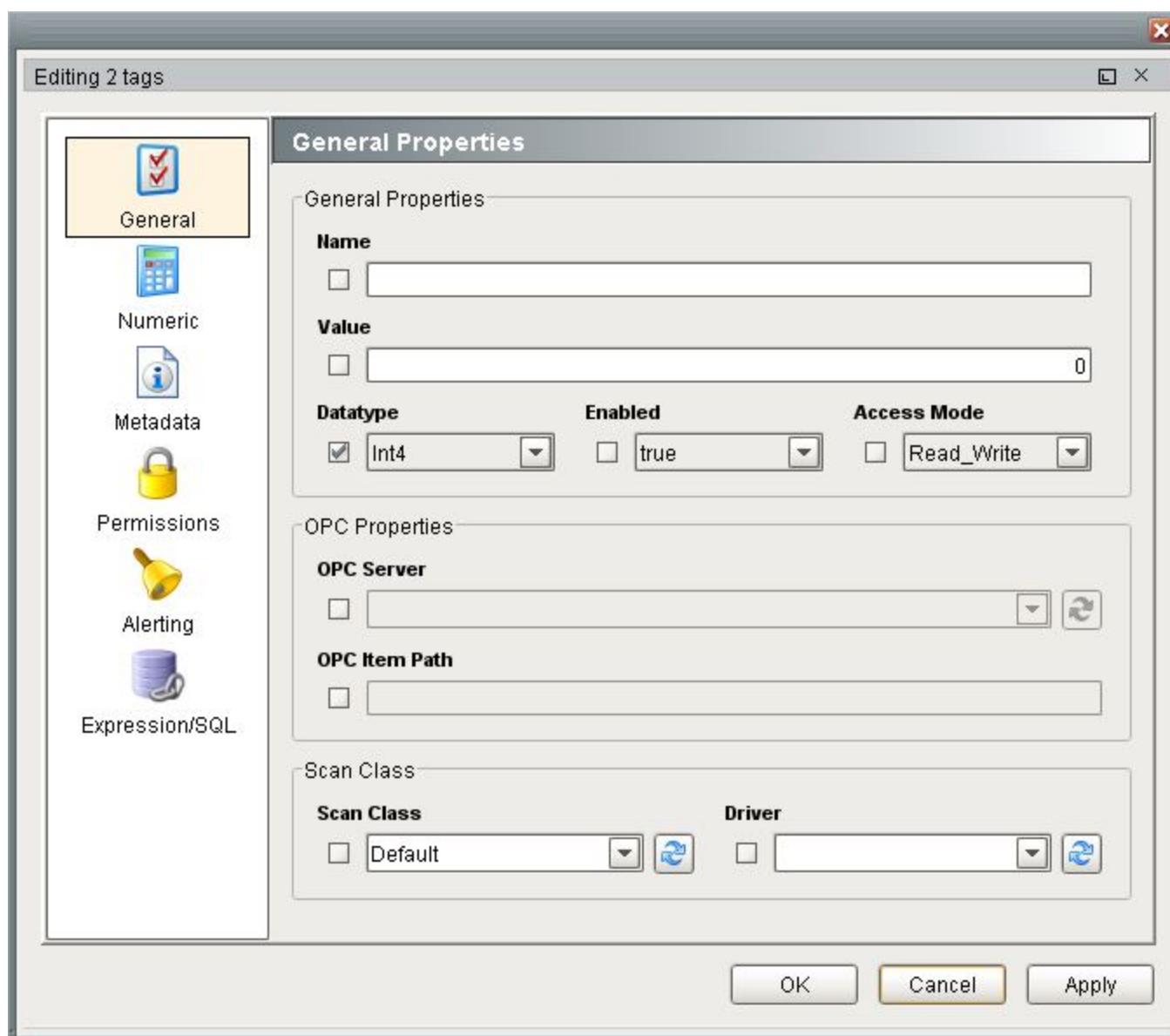
The Editor will display an asterisk (*) on properties that you have changed prior to committing with the **OK** or **Cancel** button. The example below shows the name changing from *added tag* to *Kepware_test*.





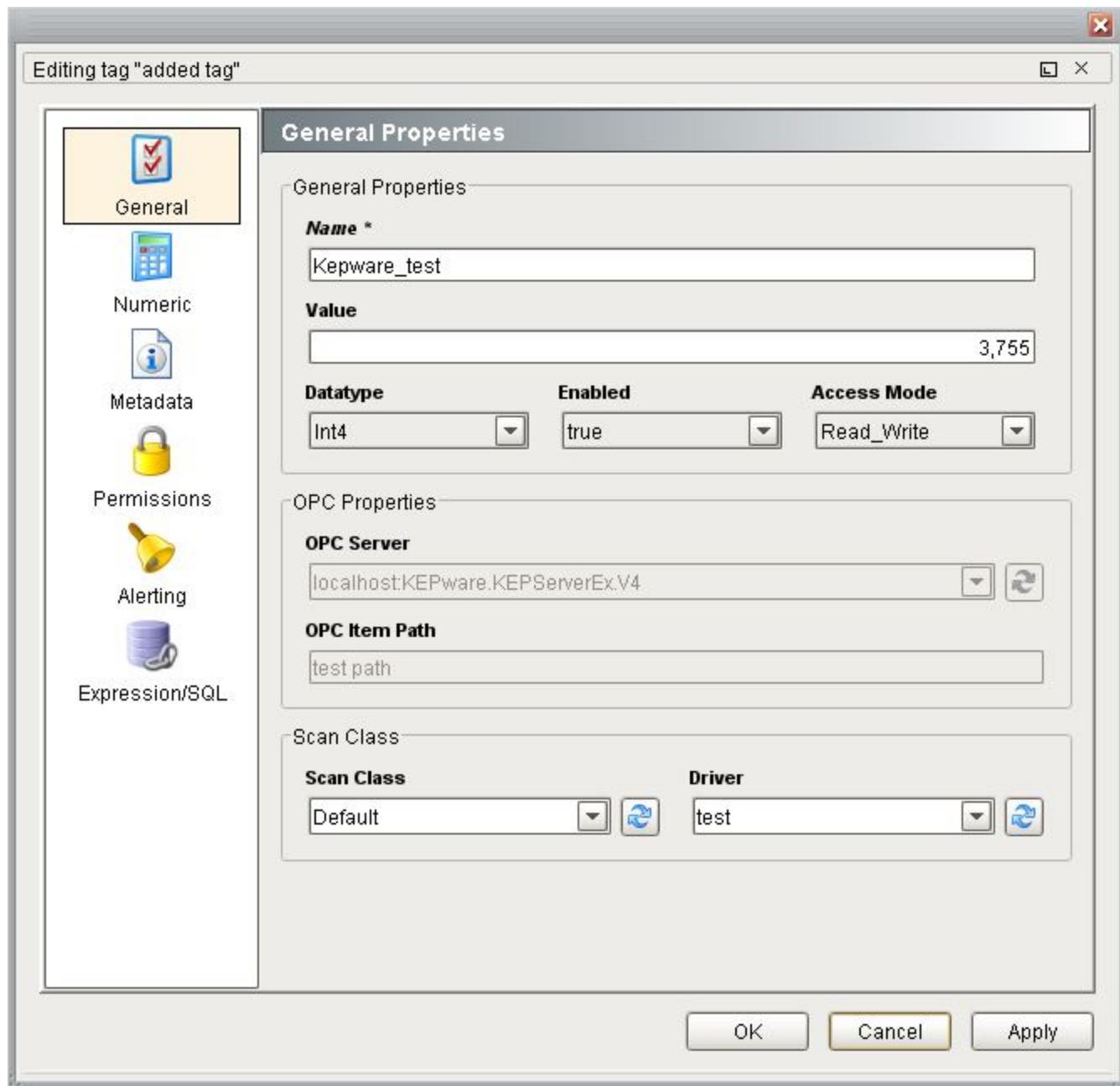
Multiple SQLTag editing

You can edit multiple SQLTags at once! Choose multiple SQLTags in the **Tag Tree** by holding Ctrl or Shift while clicking, then click **edit** . Each property now includes a check box. Selected properties will be written to the set of SQLTags when you click *Apply* or *OK*.



SQLTags™ Editor - General Properties

All SQLTags have **General Properties** to configure tag basics.



General Properties

Property Name	Function
Name	The name of the SQLTag. It must be unique in its folder. Be sure to get the name right because tag bindings are based on names and paths (which can be relative).
Value	The current value of the SQLTag. Equivalent to SQLTag.Value.
Datatype	The data type of the SQLTag, which can be any FactoryPMI data type .
Enabled	Enables/Disables tag. A disabled tag will maintain a fixed value.

OPC Properties

OPC Properties are only applicable to OPC type SQLTags.

Property Name	Function
OPC Server	Specifies OPC server for SQLTag
OPC Item Path	Specified OPC path. This will typically be generated for you from dragging OPC tags from the OPC Browser to the SQLTags Project Browser . OPC Servers often allow some amount of formatting, indexing, etc here.

Scan Class

[Scan Classes](#) deal with how SQLTags update.

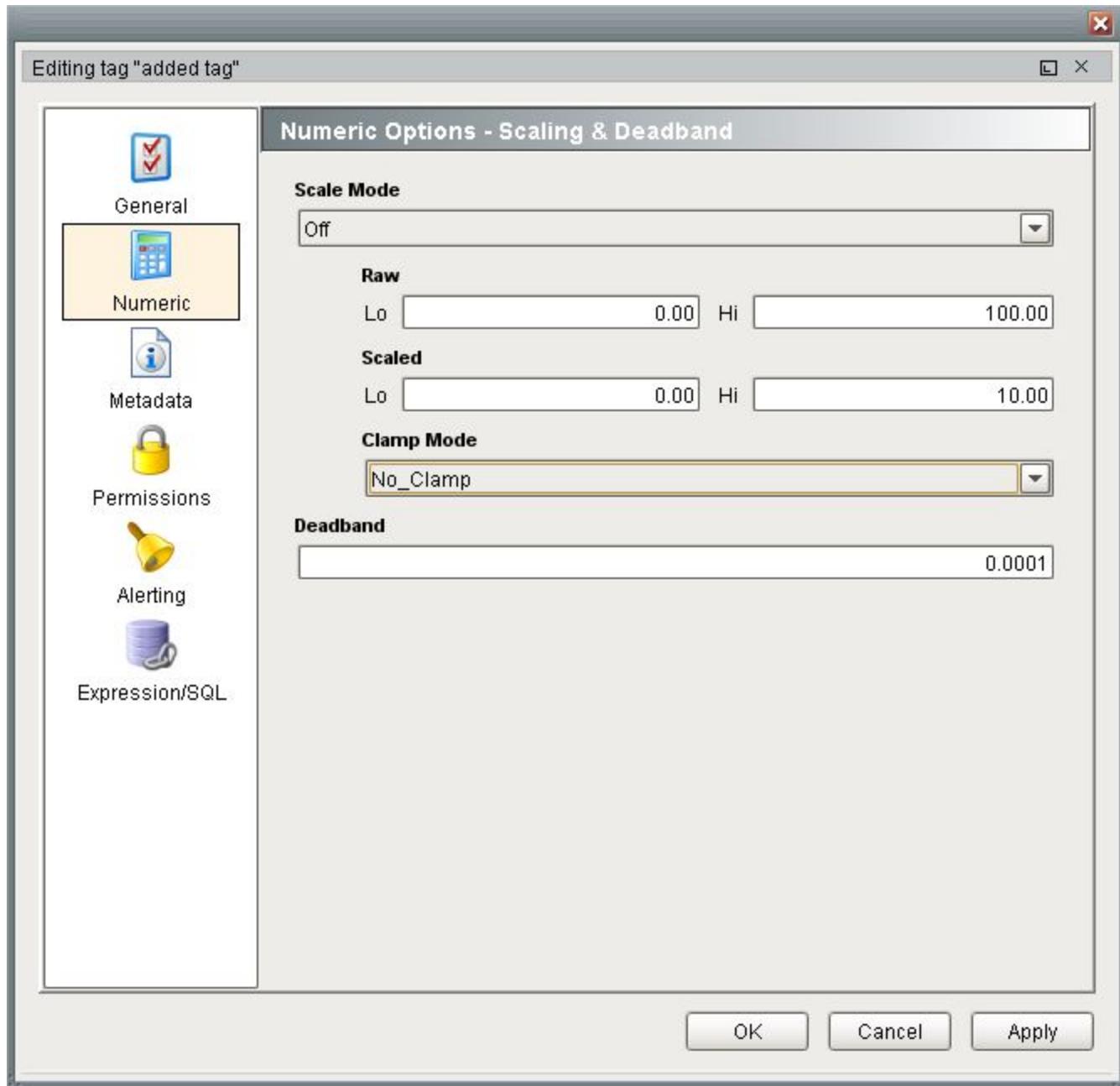
Property Name	Function
Scan Class	Selects a scan class that you have defined.
Driver	Indicates which FactorySQL node is updating the scan class, evaluating expressions, etc.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



SQLTags™ Editor - Numeric Options

Numeric options are available on OPC SQLTags of a **number** [data type](#). These options are handled through FactorySQL, so make sure that you're not looking at stale data!



Numeric Options

Property Name	Function
Scale Mode	Scales tag between raw and scaled values. Options are <i>Off</i> , <i>Linear</i> , and <i>Square Root</i> .
Raw	Raw value input range. The domain for scaling or clamping.
Scaled	The output value range if Scale Mode is not set to <i>Off</i> .
Clamp Mode	Restricts the tag value to lo or hi depending on the clamp mode. <i>No_Clamp</i> does not restrict the tag value. <i>Clamp_Low</i> will never allow the tag value to get below the appropriate lo value. <i>Clamp_High</i> will never allow the tag value to go above the appropriate Hi value. <i>Clamp_Both</i> is a combination of the two.

Deadband

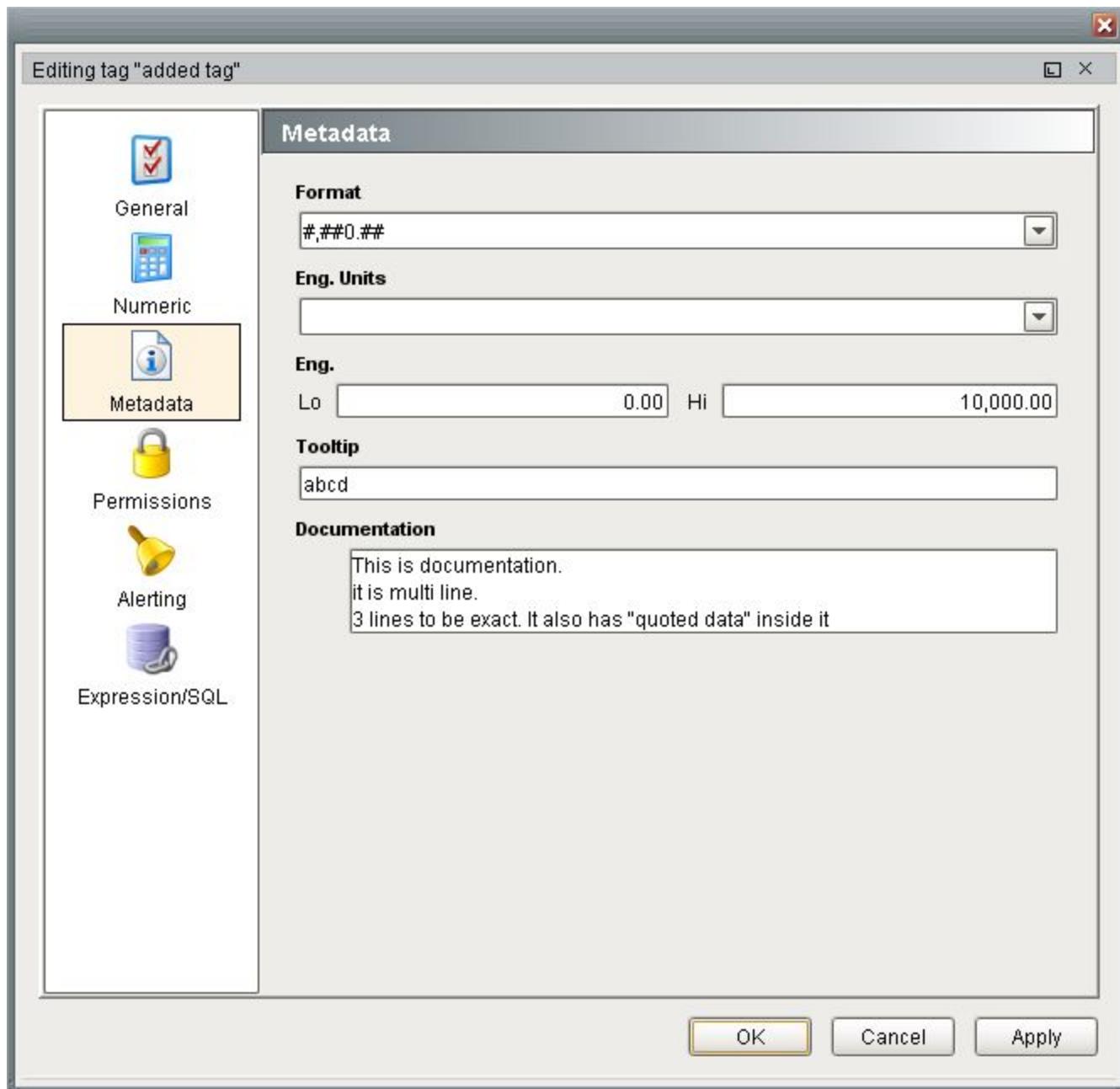
Tag value changes will only fire after exceeding the deadband amount since the last change. The example above requires a change of *.0001*, regardless of update cycles, before the tag will change value.

SQLTags™ Editor - Metadata

Metadata provides a SQLTag additional information. These values often come from the OPC Server when creating OPC SQLTags by dragging OPC tags from the [OPC Browser Tree](#) to the [SQLTags Browser](#) Tag tree.

Did you know...

Using Metadata is purely optional, but recommended. Often times it will allow you to make changes in fewer places to things like formatting that can affect many components.



Metadata

Property Name	Function
Format	Choose format from the dropdown list or create your own. Numbers recognize

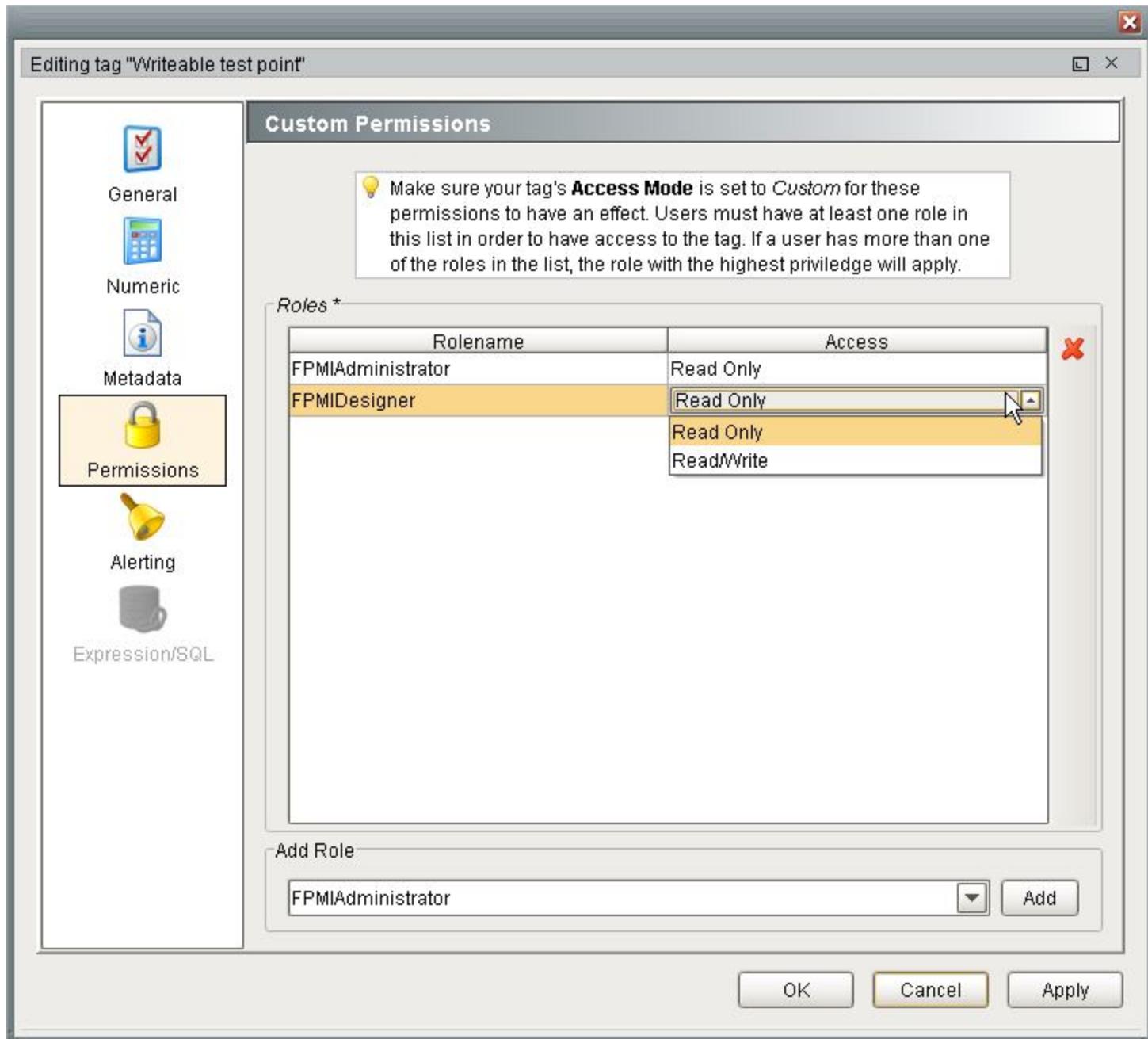
(# 0 , .) while dates use the standard FactoryPMI [dateFormat patterns](#).

Eng. Units	String based unit description. Choose from the list or type your own. Convenient because numeric labels and other components know how to bind to units.
Eng. Lo, Hi.	Expected low and high values. Often useful.
Tooltip	Used to create mouseover effects.
Documentation	Stores tag development notes.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

SQLTags™ Editor - Permissions

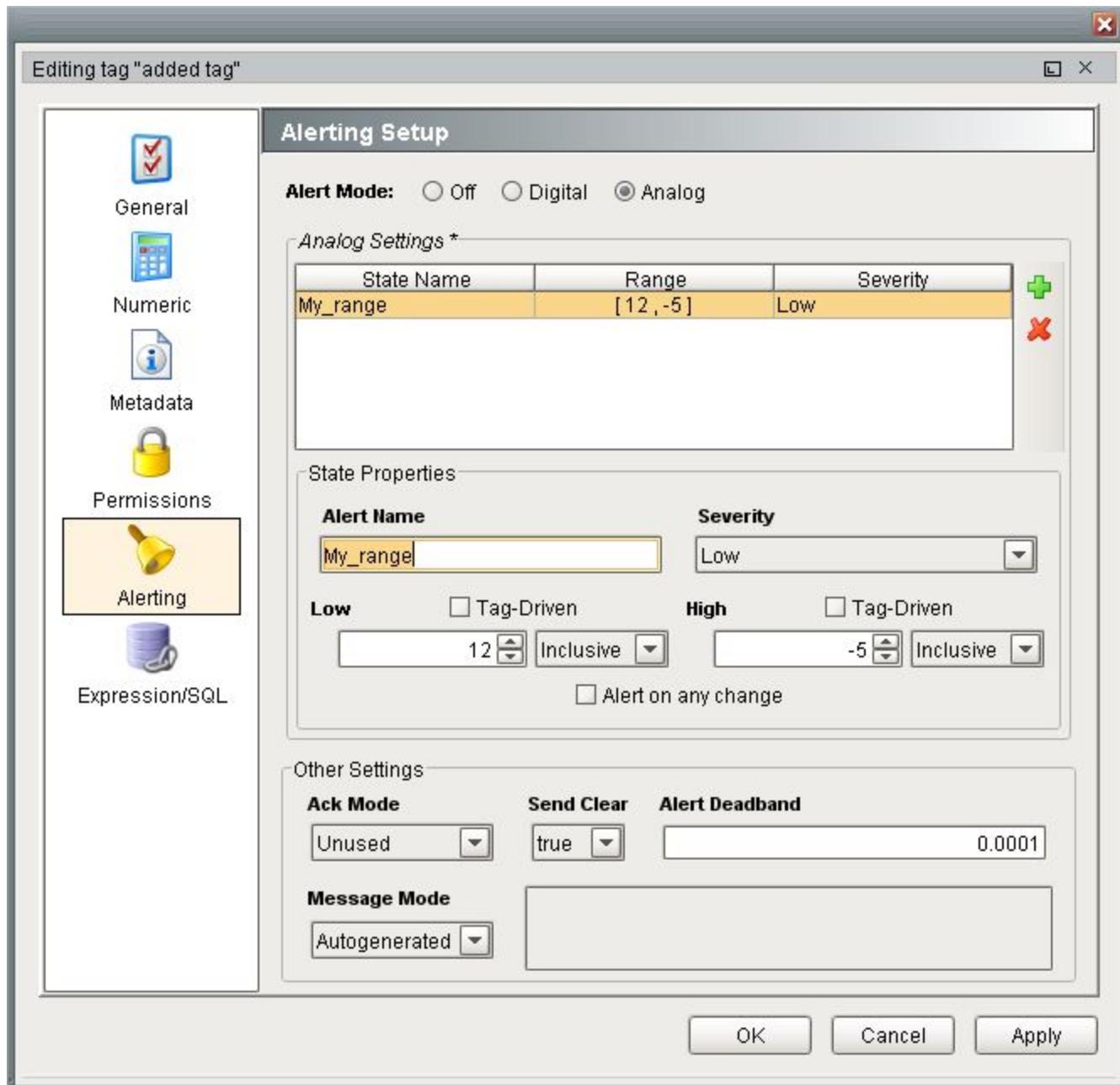
Permissions allow designers to restrict access to tags. The tag's **Access Mode** must be set to *Custom* in the [General](#) section of the SQLTags Editor for permissions to apply.



Simply choose a [security](#) Role and click the **Add** button. The **Rolename** will be added to the **Roles** table. You can then click on **Access** to choose the appropriate level of access. Options include: *Read Only* and *Read/Write*. The highest clearance takes precedence.

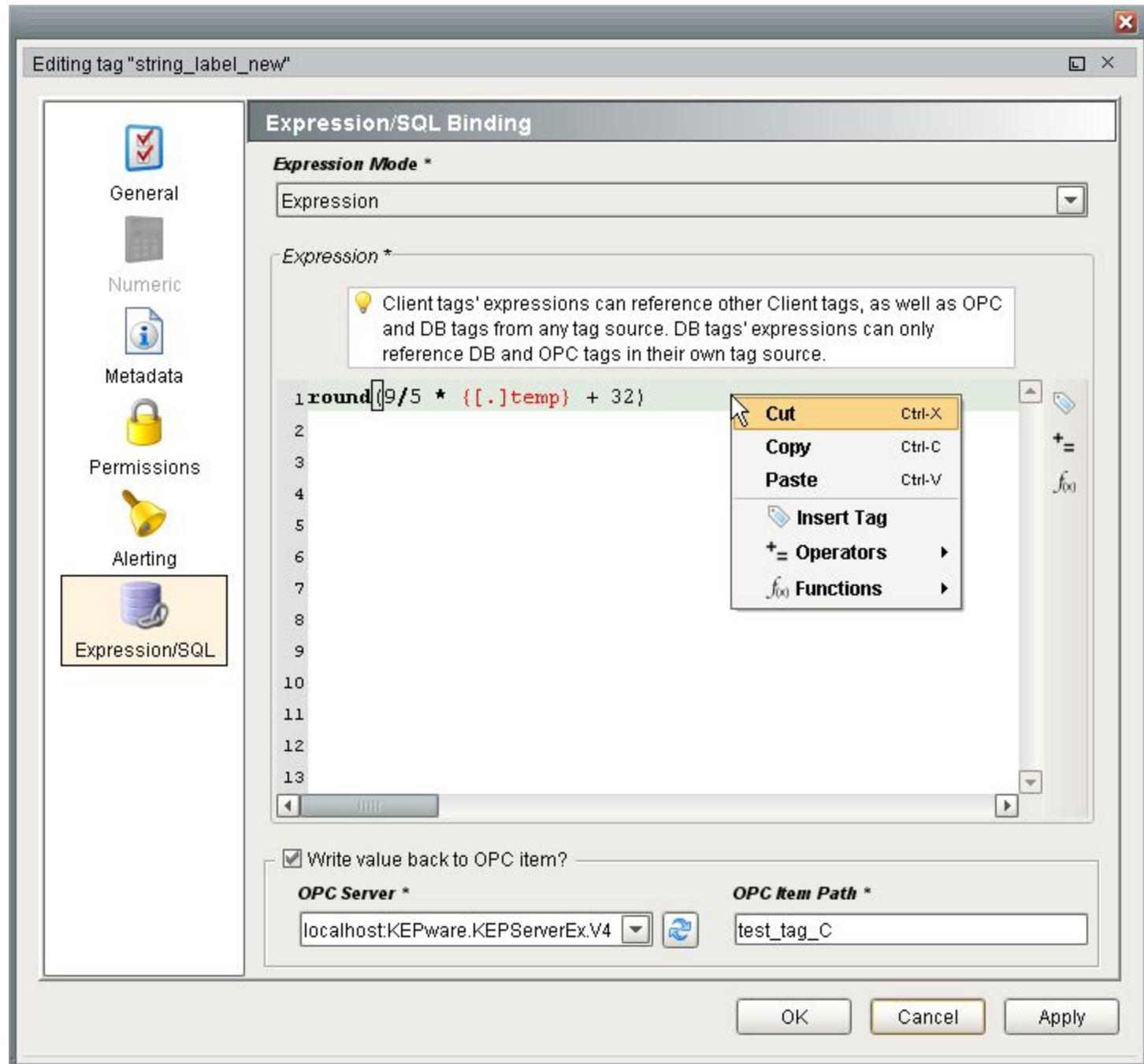
SQLTags™ Editor - Alerting

Alerting works with FactorySQL to log alarms to the database and notify users via email, alphanumeric page, or text message. It may be configured on OPC and DB SQLTags. Click for more information on [alerting](#). The FactorySQL help manual covers more options including acknowledgement, notifying groups, etc.



SQLTags™ Editor - Expressions and SQL Binding

The SQLTags™ Editor is a panel that allows you to edit SQLTags. You get to it by selecting a tag and clicking on Edit tag button of the [SQLTags Browser](#) Menu Bar or right click menu.



Expression/SQL Binding

Property Name	Function
Expression Mode	Options include: <i>None</i> , Expression , and SQL Query .
Expression	Text input area for Expression or SQL Query. Note the right click menu and buttons to insert tags  , operators  , and (expression language) functions  for either.

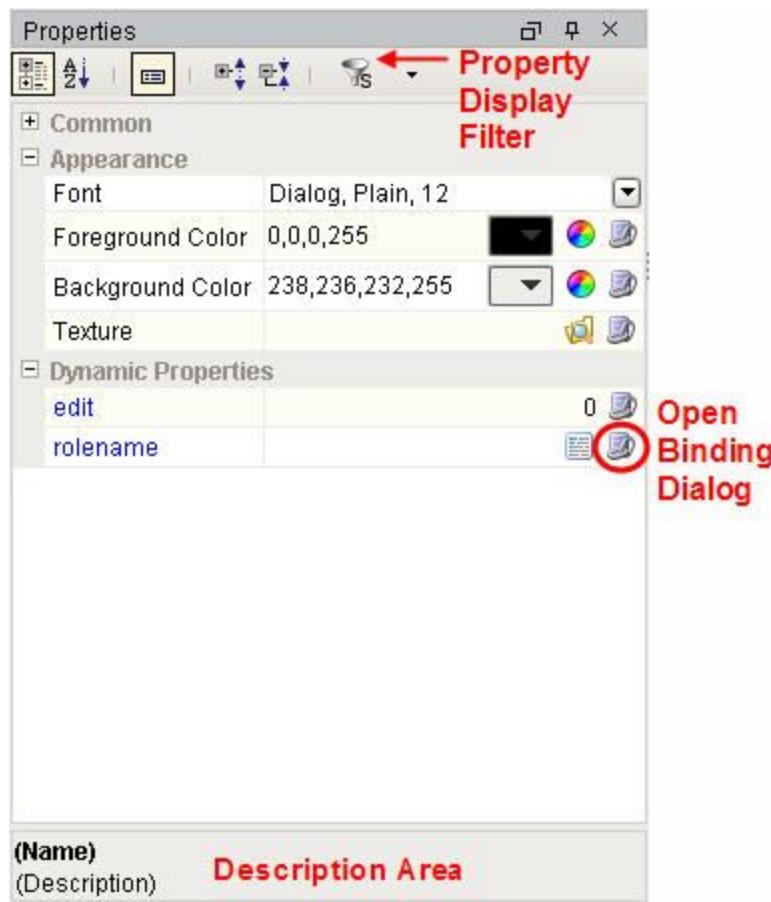
OPC Write back

Property Name	Function
Write value back to OPC item?	Enables OPC writeback on value change
OPC Server	Select OPC Server from list
OPC Item Path	Choose OPC tag path to write to

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Property Panel

The property panel is where you can set the various properties for the component(s) you have selected. The available options will change depending on what type of component you select.



Property Panel Buttons

Categorized	Sort and group properties by category
Alphabetize	Sort properties alphabetically
Expand/Collapse Properties	Expand or collapse all property groups. Does nothing when set on Alphabetize.
Property Display Filter	The property display filter limits the properties that are shown in the property panel. If you want to modify a property that was listed in the Component Reference with the Expert icon use the Expert filter. It is always possible to view everything with the All filter. 
Property Binding Button	Nearly every property has this button, which allows you to "Bind" its value to the value of something else. Whenever that other value changes, the value of this property will change as well. For more information on binding, see the Concepts - Databinding section.

Tip: Selecting multiple components will display their common properties. Changing the value of any property will affect all selected components.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Project Workspace

The project workspace ("workspace") is where you visually design your project. Think of it as the same space that will show up when you launch your project.

The only items that can be added to the workspace are windows. All other items must be placed inside a window.

The actual workspace does not have many options/properties. The most important is the **Update Rate Base**, which is the default update rate for all SQL queries (in milliseconds). For more information on update rates, see [Databinding - SQL Queries](#). You may also change the background color of the workspace.

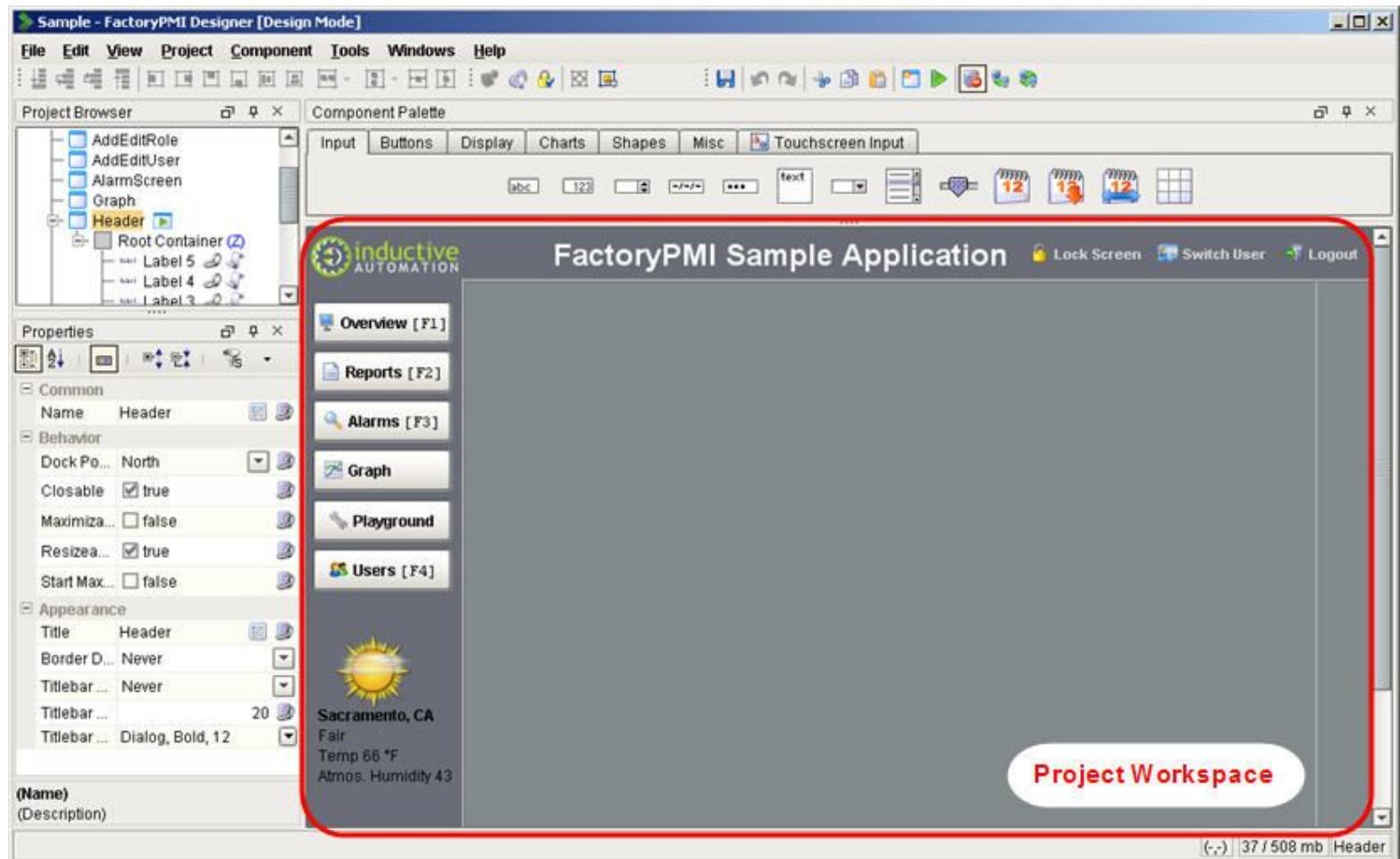
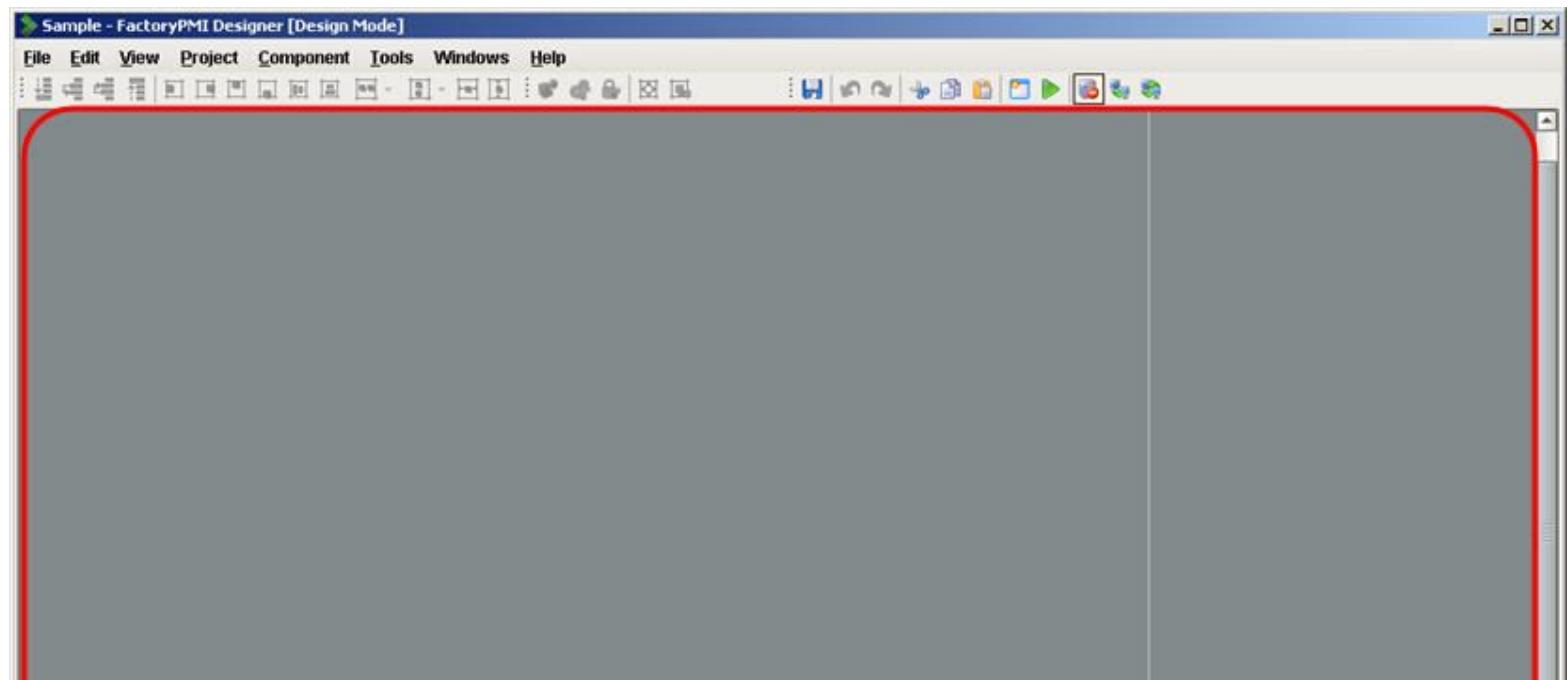


Fig. 1: Busy Workspace



Project Workspace

Minimum Size:

Project selected.

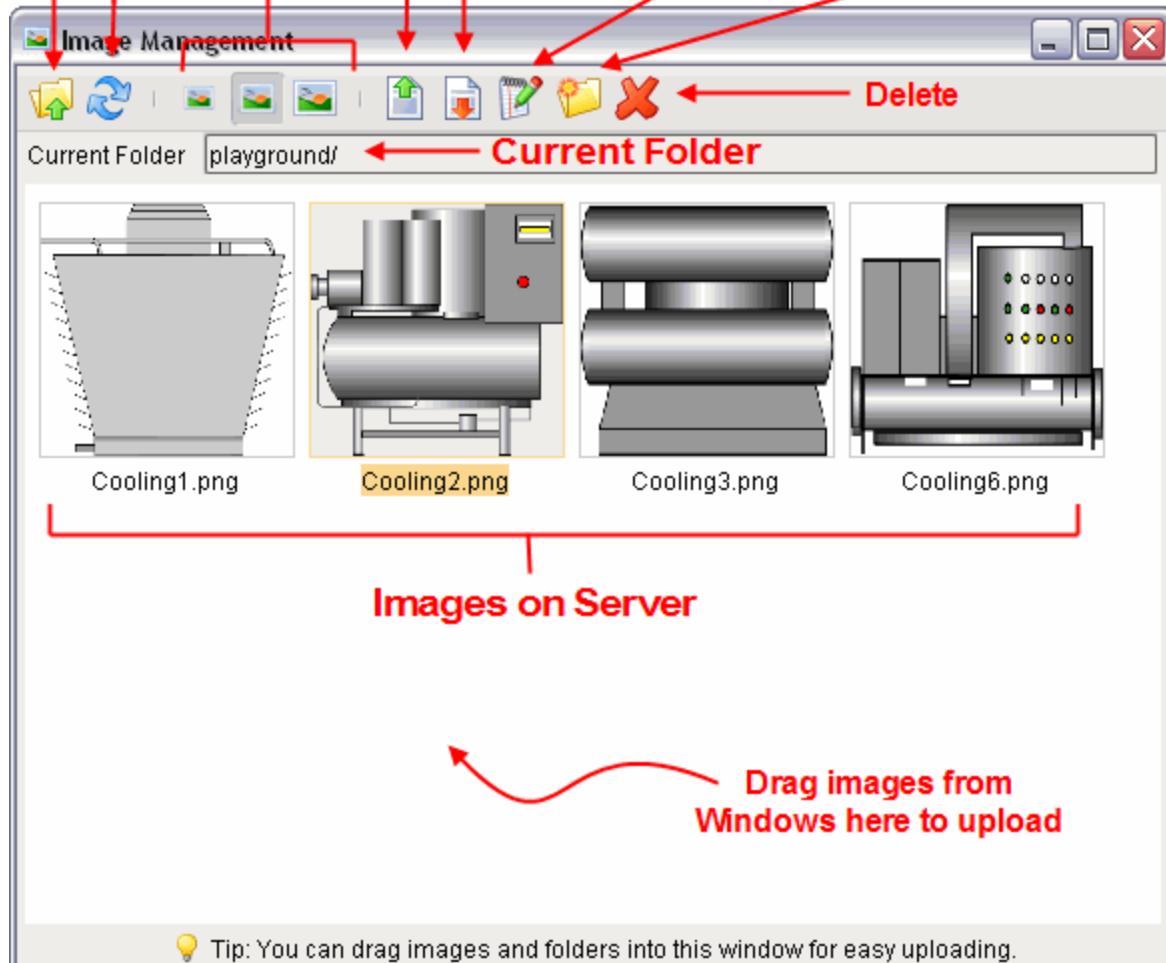
(-.) 37 / 508 mb **NONE**

Fig. 2: Empty Workspace

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Navigation



The image tool allows easy management of the images on the Gateway. FactoryPMI creates a folder for each project's images, and this tool opens up into the appropriate folder for the project you are working on. You don't need to maintain this separation, however, it makes it easier to back things up if you are working on projects for different customers on one Gateway server.

You can drag images, and even entire folders of images into this window to upload them to the server. You can also drag images around into other folders within this window to move them. Tip: Open up two Image Tools if you are doing a lot of moving, and drag between the two.

Within this tool, it is easy to do common image tasks, such as rename, upload (add to Gateway), download (to your hard drive), delete, etc.



Interactive Script Tester

The **Interactive Script Tester** allows a developer to run Jython script in a command line environment. Modules **are not** automatically imported as they are with [component events](#).

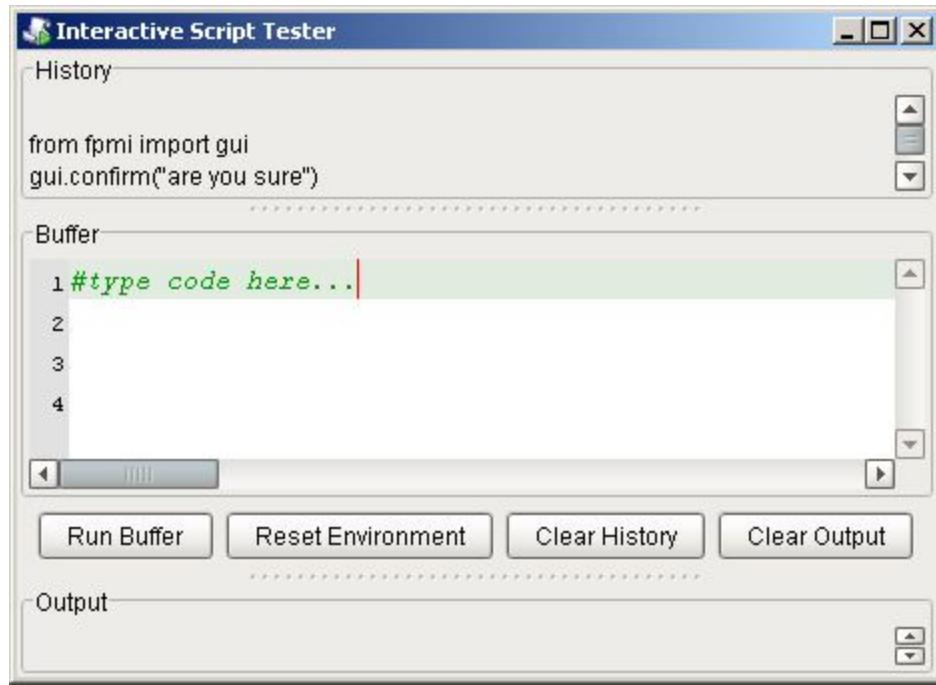


Fig 1: Interactive Script Tester

Tip...

The more you can do without scripting the better! Many beginners make the mistake of trying to do everything with Jython. Don't be *that guy*. Read the docs and learn the easy way!



Concepts - Overview

This section provides a technical reference to the designer, broken up into 5 broad themes. These themes encompass most of what you will need to do in FactoryPMI, and are roughly positioned in the order you might encounter a need for them.

The themes are:

Windows

This section discusses windows, the primary unit of organization within a project.

Components

This section discusses placing components on windows, how to organize them, size them, and set their properties. Components are fundamental to FactoryPMI, and a solid understanding of them is essential.

Databinding

Databinding is another essential topic. This section covers how to bind components to various data sources in order to present and control information.

Events

Events are a way to specify a response to some action, such as a mouse click or key press. This section covers how to specify and configure Events.

Security

This section involves securing the FactoryPMI frontend, from hiding certain components from under privileged users, all the way to blocking them from a certain project.



Windows Overview

A FactoryPMI project consists mainly of a collection of windows. Most of your effort in FactoryPMI will be arranging and configuring [components](#) inside of your windows.

Basics

- **Adding.** To add a new window to your project, press the "New Window" (toolbar button.
- **Deleting.** To delete a window, first close it, and then right click on it in the project browser and choose "Delete".
- **Renaming.** To rename a window, open it and make sure the window is selected by clicking on its title bar. Then change its name property.
- **Startup.** When a project starts, some of its windows can open up automatically. To make a window startup automatically, right click on it in the project browser and choose "Open on Startup".

The rest of this section covers more window-related issues.

- [**Navigation.**](#) Covers the details involved in opening and closing windows.
- [**Parameters.**](#) Make windows that show different things based on parameters to save yourself lots of repetitive work..
- [**Docking.**](#) Dock windows that you always want shown.
- [**Advanced Topics.**](#) Covers some more advanced topics regarding windows.

Window Navigation

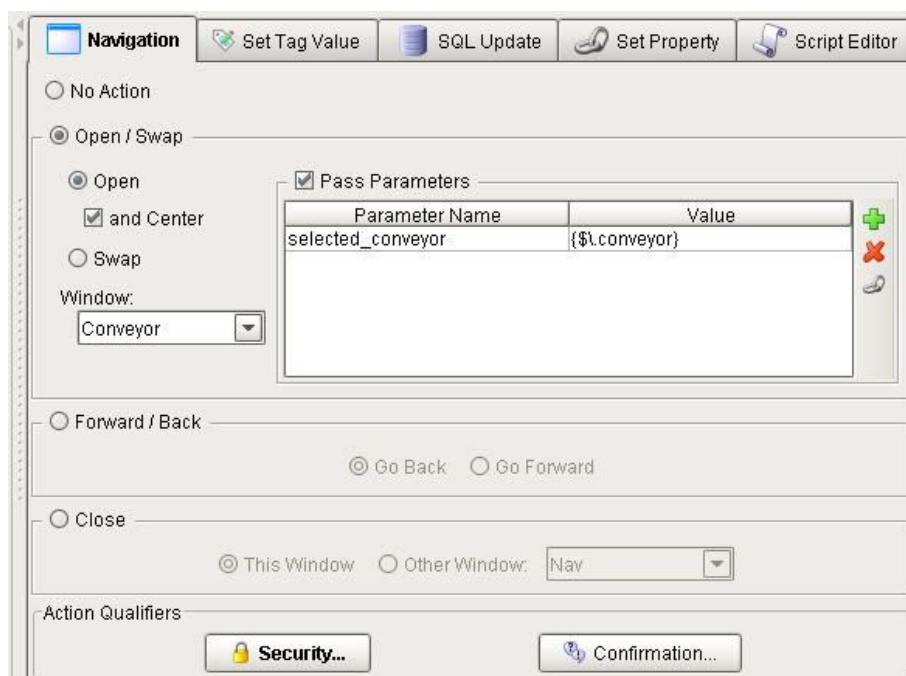
Window navigation is done with the [Navigation Builder](#), which autogenerated Jython scripts based on the [fpmi.nav](#) module.

Navigation

Introduction to Navigation

This video is meant to show you how to do Window Navigation based on a button press with the Navigation Builder!

Navigation Functions



Menu Item	Function
Open	Opens the specified window as a popup. Checking " and Center " centers the window after opening it.
Swap	Swaps the "main" window (usually the maximized one) by calling fpmi.nav.swapTo .

Pass Parameters	This list sets Dynamic Properties of your target window to either static or bound values (Object properties from the calling window). When the window is opened/swapped, the dynamic properties on the Root Container of the window will be set based on the Parameter Name and Value .
Forward/Back	Takes the "main" window forward or back if the user has been Swapping the window.
Close	Closes the specified window.

Navigation Techniques

It is common for beginners of FactoryPMI to use one maximized window at a time, swapping into other windows using navigation buttons on the window. This gives the effect of having 'screens' similar to other HMI packages they may be used to. While this would work fine, it has the drawback that your navigation buttons are most likely duplicated on every window that you have. This means that if you do something like add or remove window, you will have to update every screen's navigation buttons. We recommend one of these two techniques instead:

Docked Navigation Pane. Dock a narrow window and put your navigation buttons on it. You can still have your maximized main window, because maximized windows don't obscure docked windows. This has the added benefit of having a small piece of screen real estate that is always showing, and is good for other things like an E-Stop, or alarm summary.

Navigation Menu. Use the customizable [runtime menu](#) to provide navigation. This has the benefit of having all of your navigation logic in one centralized place, but doesn't take up any additional room on the screen!



Window Parameters

It is often the case that you will want to create one window that changes its functionality based on the context from which it was opened. For instance, suppose you have 30 valves displayed on a screen. When the user clicks on a valve, you want to display a HAND-OFF-AUTO control to control that valve. Instead of making 30 HOA popup windows, you can simply make one and pass it a *parameter*: the valve number.

The window parameterization feature of FactoryPMI piggy-backs on the [dynamic properties](#) feature. A window's parameters are simply its root container's dynamic properties! When you open a window, you pass it a list of name-value pairs that will be used to set the root container's dynamic properties.

So, if your Valve HOA popup window has a dynamic property on its root container called "ValveNum", you could open it with the following command:

```
fpmi .gui .openWindow("ValveHOAPopup", {"ValveNum":3})
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Window Docking

Windows can be "Docked" to the four sides of the project workspace. To dock a window, change its "Dock Position" property to one of the four sides. To un-dock it, change the dock position to "Floating". When a window is docked, it can only be resized in one direction. For example, a window that is docked West can be resized to be wider or thinner, but its height is always the height of the project workspace itself.

Other windows cannot be moved on top of a docked window. Maximizing a window will maximize it into the space not occupied by docked windows.

It is often useful to modify a docked window's [Border Display Policy](#) and [Titlebar Display Policy](#) to hide the window's titlebar and border.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Serializing / Committing

When you close a window that you have been editing in the FactoryPMI Designer it will ask you if you want to *commit* the window. If you choose yes, the window is *serialized*. Serializing is the process of converting the open window into the format in which it is stored in a FactoryPMI project. If you choose not to commit the window, all of your changes since you last opened the window will be lost. Note that all open windows are committed when you save.

Sometimes when you upgrade to a new version of FactoryPMI, the format that windows are stored in has changed. When this happens, your windows will be converted on the fly to the new format every time they are opened. This can cause your project to seem slower because every time a window is opened it is being converted. When this happens, it is a good idea to reserialize all of your windows. You can use the "Reserialize Windows" function under **Tools - Advanced** in the designer to do this quickly. This is functionally equivalent to opening and then closing all of your windows.

Caching

In the FactoryPMI Runtime, windows are *cached*. This means that when a user opens a window, and then closes it, it will remain in memory for a while, waiting to be re-opened. If the user opens that window again while it is still in the cache, the time it takes to open the window will be significantly decreased.

When a window is opened from the cache, all of its component will be in the state they were in when the window was closed, *not* the state they were in when the window was serialized. Sometimes, this isn't desirable. Most notably, a window where the user enters textual information. When opened from the cache, the text boxes will have the text that was previously entered, unless you add special scripting code to clear out the text boxes. Instead, you can simply exclude that window from the cache by setting its cache policy setting to **Never**. For more on caching policies, see [Window properties](#) page.



Components Overview

A component is an item that you place in a window, to achieve some sort of functionality in your PMI projects. By and large, a FactoryPMI project consists of little more than [windows](#) full of components and the logic that connects them and drives them.

Components can be both visible and invisible, and can be static or interactive. Some components are used only for organization, others for graphical display, and others for user feedback. Despite the differences between them, all components share the same fundamental base, and so they have several properties that are shared. For instance, all components are alike in that they are placed and manipulated on the screen. Also, all components have several properties that are displayed in the [Property Panel](#).

The real power behind FactoryPMI project is in [Databinding](#). This is the ability to *bind* any property of any component to any other component's properties, or to a value in a database. These bindings come in a variety of flavors, allowing you to hook up your component's in creative ways, easily and quickly creating a rich, interactive HMI / SCADA application.

All of the available components are listed in the [Component Palettes](#). See [Component Placement](#) for information about placing components in windows.

See the [Component Reference](#) for a help with each individual component.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Placing Components on Screens

Placing components on a screen is a simple click and drag operation. Afterwards, you can adjust its size and position using the keyboard or mouse to make it exactly like you want.

Adding Components

To select a component for placement either:

click on it in the component palette:  The arrow will turn into a cross hair: 

Drag the mouse from the top left to the bottom right on the window to draw the component:



And release the mouse:

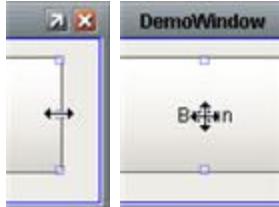
Or drag the component onto your screen:



Manipulating Components

The new component will be selected by default. As you can see, a selected component has 9 handles: 4 corners (for resizing diagonally), 4 mid-points (for resizing vertically or horizontally), and a center handle, for moving the component.

When the mouse is over a handle, the icon will change appropriately:



There are many [keyboard shortcuts](#) that save a lot of time in FactoryPMI



When pasting a control (either through Edit-Paste, or Ctrl-V), the mouse icon will change to the paste icon: . Click on the location where you want to place the component in order to paste it.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Keyboard Shortcuts

Keyboard shortcuts can save you LOTS of time when it comes to component manipulation. Here is a list of all the shortcuts and mouse modification keys available:

General Keyboard Shortcuts

	Nudge. Move selected component(s) the direction of the arrow key by the default nudge distance.
Alt + ()	Super Nudge. Move selected component(s) the direction of the arrow key by the super-nudge distance. (See Quick Preferences under the View menu for details)
Shift + ()	Resize Right. Moves the right edge of the selected component(s) to the left or to the right by the default nudge distance. Add Alt to use the super-nudge distance.
Shift + ()	Resize Bottom. Moves the bottom edge of the selected component(s) up or down by the default nudge distance. Add Alt to use the super-nudge distance.
Ctrl + Shift + ()	Resize Left. Moves the left edge of the selected component(s) to the left or to the right by the default nudge distance. Add Alt to use the super-nudge distance.
Ctrl + Shift + ()	Resize Top. Moves the top edge of the selected component(s) up or down by the default nudge distance. Add Alt to use the super-nudge distance.
Ctrl +	Move Forward. Move selected component(s) forwards in the Z-order.
Ctrl +	Move Backward. Move selected component(s) backwards in the Z-order.
Ctrl + Alt +	Move To Front. Move selected component(s) to the front of the Z-order.
Ctrl + Alt +	Move To Back. Move selected component(s) to the back of the Z-order.
Ctrl +	Copy-Move. Hold down Ctrl while moving a component with the mouse to copy it to the new location.
Shift +	Vertical or Horizontal Move. Hold down Shift while moving a component to only move it up, down, left, or right.
Ctrl + Shift +	Copy Vertical or Horizontal Move. Hold down Shift and Control while moving a component to combine the above two effects.
Shift +	Proportional Resize. Hold down Shift while resizing a component to maintain its proportions.

Component Keyboard Shortcuts

Ctrl + L	Layout Edit component(s) layout
Ctrl + P	Size & Position Exact size and position for selected component or window
Ctrl + U	Customizer Opens Customizer to configure complex component properties

+

Jython Configure Actions for component or window.

+

Security Configures security for Component(s) or window.

File and Project Keyboard Shortcuts

	Help opens HTML based help system
	Preview Mode Toggles preview / design mode
+	New Window Creates a new window
+	Save Saves project
+	Open Opens a different project
+	Global Event Scripts Opens Global Event Script window for Jython scripts
+	Script Modules Opens Script Module window for reuseable Jython functions
+	Quick Preferences specify handle properties, nudge distance, default colors, and other default properties.

Edit Menu Shortcuts

+	Undo last action
+	Redo Gets rid of the last undo action
+	Copy Selected Component(s) or Window
+	Duplicate Selected Component(s) or Window
+	Cut Selected Component(s) or Window
+	Paste Components(s) or Window in Clipboard. Pasted component(s) wait for position before pasting.
+	Immediately Paste Component(s) in Clipboard. Pasted component(s) are placed at the same location where they were copied/cut.
	Cancel Paste Cancels a pending paste operation
+	Select All selects all components that are siblings of the selected component
+ +	Select Same Type selects all components that are siblings of the selected component and the same component type

Ctrl + **Alt** + **Shift** + **A**

 **Select Same Type in Window** selects all components in a window that are the same type as the selected component

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Component Properties

All components have properties that can be edited. These properties appear in the [Properties Panel](#), and can be changed there.

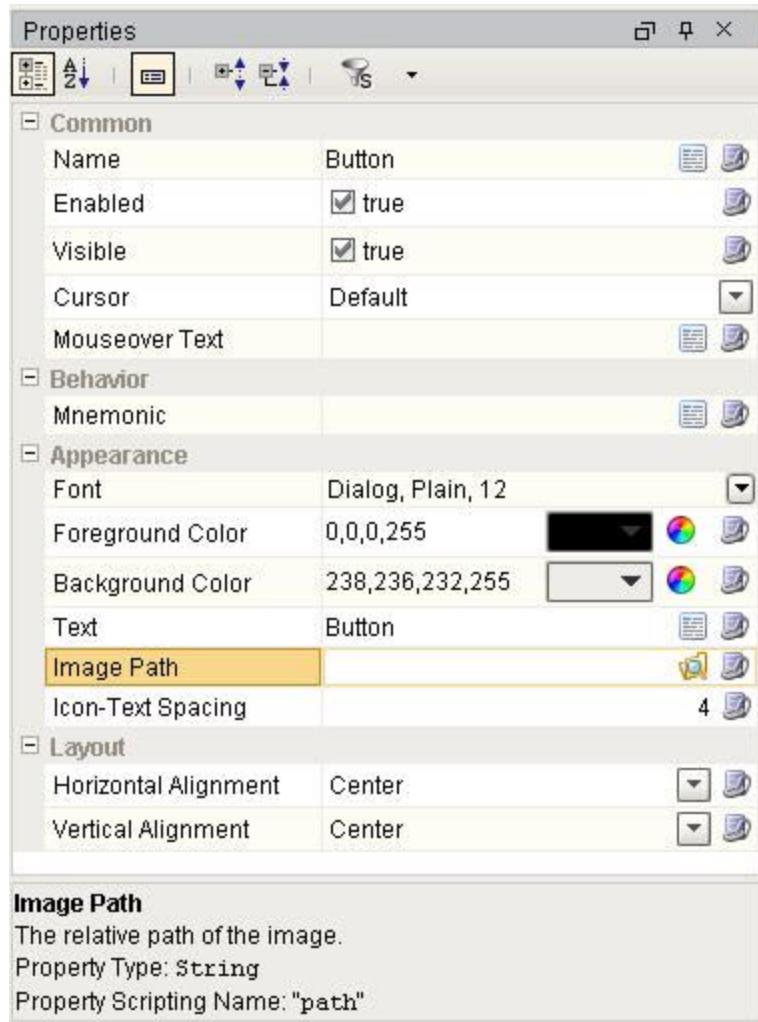


Fig 1: A button's properties

Properties control some aspect of the control, such as the text, color, or size. Their display in the property window depends on what type of property they are. Properties such as color, font, and width will all have different types, and different displays. *Types* are an important concept in FactoryPMI, see [Concepts - Databinding - Types](#) for a description of all property types available in FactoryPMI.

Almost every property can be "bound". To be bound means to have that particular value linked to a value somewhere else... so that when the other value changes, the local one changes to reflect it. The other value can be a database query, an expression, or another property's value. A bindable property has a bind icon (Bind icon). For more information on binding, see [Concepts - Databinding](#).

User defined properties are called [Dynamic Properties](#). They can be of any [data type](#) and are displayed in blue.

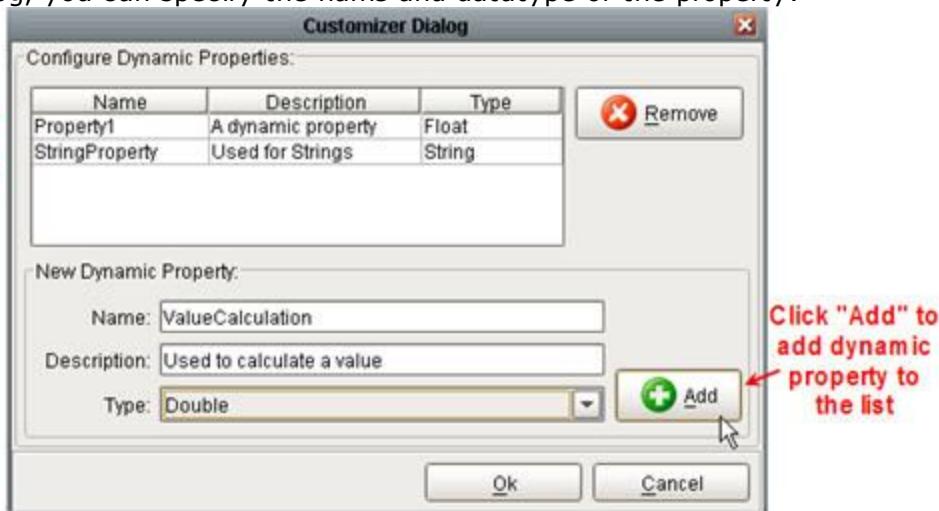
Dynamic Properties

Dynamic properties provide a way to define extra fields that can be used for any purpose. The main use is for [databinding](#), where the property is bound to other fields or expressions, thus acting as a variable or intermediate value calculation. You can create dynamic properties on most of the common components, such as [Containers](#), [Labels](#), [Buttons](#), [Images](#), etc. Each dynamic property has a type; see the [Types](#) page for an explanation of all of the types of dynamic variables that you can create. Dynamic Properties are displayed in blue.

To add a dynamic property to a container, click the "Component Customizers" dropdown button and select "Dynamic Properties":



In the dialog, you can specify the name and datatype of the property:



After clicking "OK" you will see your newly defined properties at the bottom of the [Property Panel](#):

A screenshot of the application interface. On the left is the 'Project Browser' showing a tree with 'DemoWindow' and 'Root Container'. In the center is the 'Properties' panel. Under 'Common', 'Root Container' is selected with 'Enabled' and 'Visible' checked. Under 'Dynamic Properties', 'Property1' and 'StringProperty' are listed. On the right is a window titled 'DemoWindow' containing a single 'Button'. A red arrow points from the text 'New properties show up in the Property Panel' to the 'Dynamic Properties' section of the Properties panel.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Component Styles

Component styles is a powerful feature that you want to use often! It is considerably simpler than it may look.

This feature lets you to create a dynamic visual style for various components without having to setup any [databinding](#). Typically, with most components you have a driving property, such as a numeric **state** or **status**. Based on that state, you want to change various property values, such as the background color, the text that is displayed, the visibility, etc. The Component Styles feature lets you create that style easily on most of the common components, such as Containers, Labels, Buttons, Images, Shapes, etc. You configure the styles via the *Style Customizer* for the selected component(s). The styles you configure are saved in a DataSet [property](#) on the component called "Styles" (this is an expert property).

A style is made up of:

- The *driving property*. This is the property whose value will determine what the component should look like. You will often then bind this property to a [SQLTag](#) or [dynamic property](#) on the *Root Container*.
- One or more *styled properties*. These properties are the properties that will change based on the *driving property*.
- One or more *values*. Each value defines what the *styled properties* should be for a given value of the *driving property*.

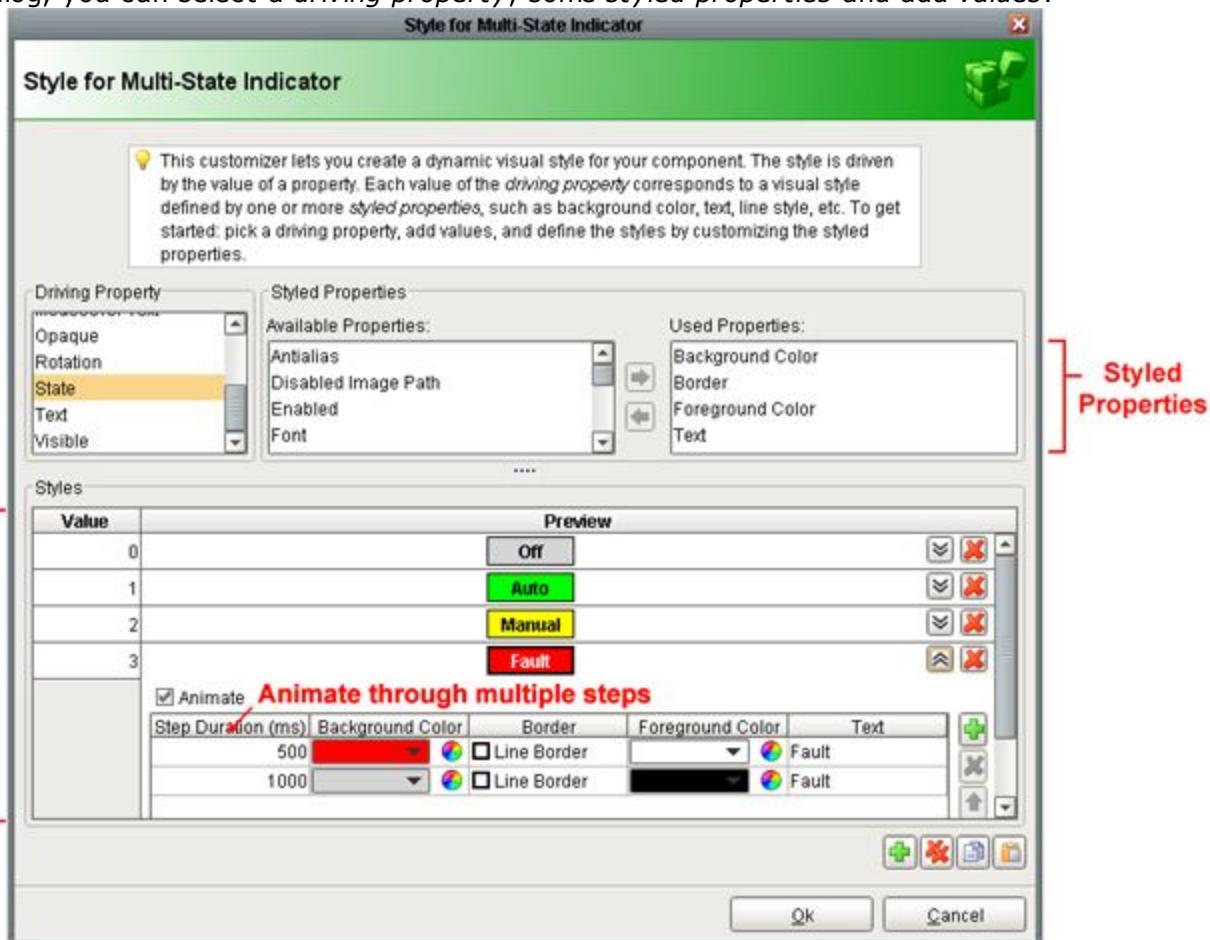
To create or modify a style for the selected component, click the "Component Customizers" dropdown button and select "Style Customizer":



Shared Styles

 The styles you define for a component are stored in a DataSet property on the component called *Styles* (view the expert properties to see it). This means that you could store some common styles used throughout your application in the database. That way, if you ever decide that *Fault* mode should be blinking purple instead of red - you'd only have to change one spot!

In the dialog, you can select a *driving property*, some *styled properties* and add *values*:



Each row in the table shows a preview of what the component will look like for that value. Press the expand button to customize the styled properties for that value. You can animate the styled properties for that value by pressing the animation checkbox, and then add animation steps.

After clicking "OK" your styles will be saved in the "Styles" dataset:

Copying Styles

You can copy and paste styles across components with the copy and paste buttons near the bottom of the Styles Customizer. Even better - just select multiple components and edit their styles together!

The screenshot shows the Styles Customizer interface. The Project Browser on the left lists a project named 'test' with a configuration and a window named 'DemoWindow'. Inside 'DemoWindow' is a 'Root Container' which contains a 'Multi-State Indicator' component. The 'Multi-State Indicator' is currently selected, as indicated by its orange selection bar in the Project Browser and its highlighted status in the preview window. The Properties panel on the right displays various properties for the selected component. A red arrow points from the text 'Driving Property' to the 'State' dropdown, which is set to 'Off'. Another red arrow points from the text 'Styles Dataset' to the 'Styles' dropdown, which is set to 'DataSet [5R x 7C]'. The preview window shows a simple gray rectangle with the word 'Off' centered inside it.

Property	Value	Actions
Name	Multi-State Indicator	
Enabled	<input checked="" type="checkbox"/> true	
Visible	<input checked="" type="checkbox"/> true	
Border	<input type="checkbox"/> Line Border	
Cursor	Default	
Mouseover Text		
Opaque	<input checked="" type="checkbox"/> true	
State	Off	0
Text		
Font	Dialog, Bold, 12	
Foreground Color	0,0,0,255	
Background Color	213,213,213,255	
Image Path		
Disabled Image Path		
Icon-Text Spacing	4	
Rotation	0° Custom...	
Antialias	<input type="checkbox"/> false	
Styles	DataSet [5R x 7C]	
Horizontal Alignment	Center	
Horizontal Text Position	Trailing	
Vertical Alignment	Center	
Vertical Text Position	Center	

When the driving property changes, the styled properties will change based on the style you created.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Component Overlays and Quality Codes

Component Feedback

Most components support overlays to clearly indicate state conditions to the end user, which is based on the component's *quality code*. These represent such things as: stale data, insufficient permissions, incorrect configuration, write pending, etc.

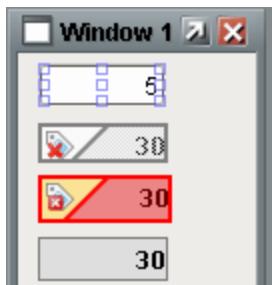


Fig 1: Component feedback

Tip...

Component overlays are driven by the Expert **Data Quality** Property. Feel free to bind this property or externally control it to manipulate overlays.

Overlays

Overlays are defined separately for each component. Each overlay often corresponds to many different quality codes.

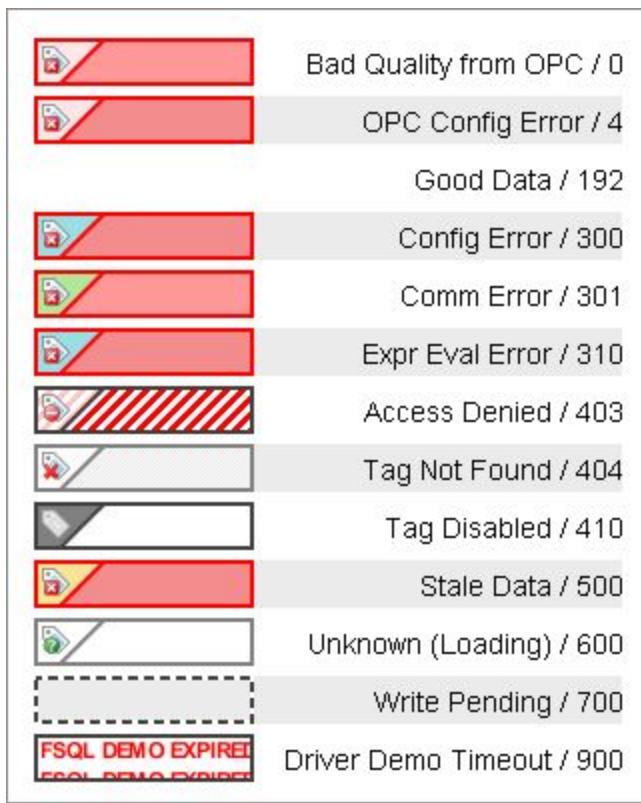


Fig 2: LCD component overlays

Quality Codes

Quality codes indicate states of reliability of data. Many codes match up to OPC specification.

A component's *quality* is determined by the quality of the worst bound property. This value is stored in the (Expert) **Data Quality** property. **-1** means no overlay.

Quality Code	State
--------------	-------

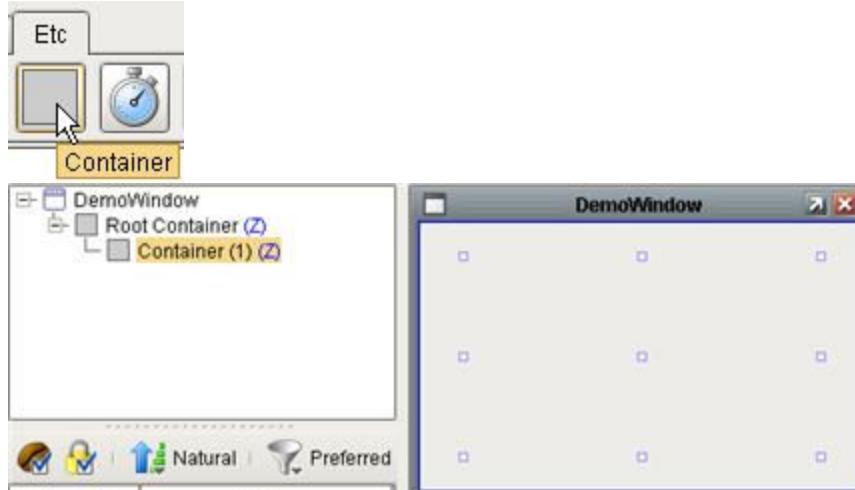
0	Bad Data from OPC
4	CONFIG_ERROR
8	NOT_CONNECTED
12	DEVICE_FAILURE
16	SENSOR_FAILURE
20	Bad, showing last value
24	COMM_FAIL
28	OUT_OF_SERVICE, SERVER_DOWN
32	WAITING
64	UNCERTAIN
68	UNCERTAIN, showing last
80	SENSOR_BAD
84	LIMIT_EXCEEDED
88	SUB_NORMAL
192	Good Data
216	Good, with local override
256	OPC_UNKNOWN
300	Config Error
301	Comm Error
310	Expr Eval Error
330	Tag exec error (FSQL)
340	Type Conversion Error
403	Access Denied
404	Not Found
410	Disabled
500	Stale
600	Unknown (loading)
700	Write Pending

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

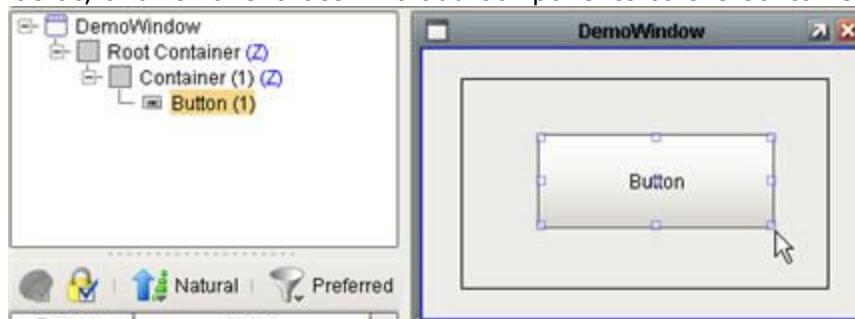
Containers

[Containers](#) are special components that can contain other components. All windows contain at least one Container - the *Root Container*. The components in a container are called '*siblings*'.

To place a container on the window, select it from the [Component Palette](#) (under "Etc."), and draw it on the window like any other component.

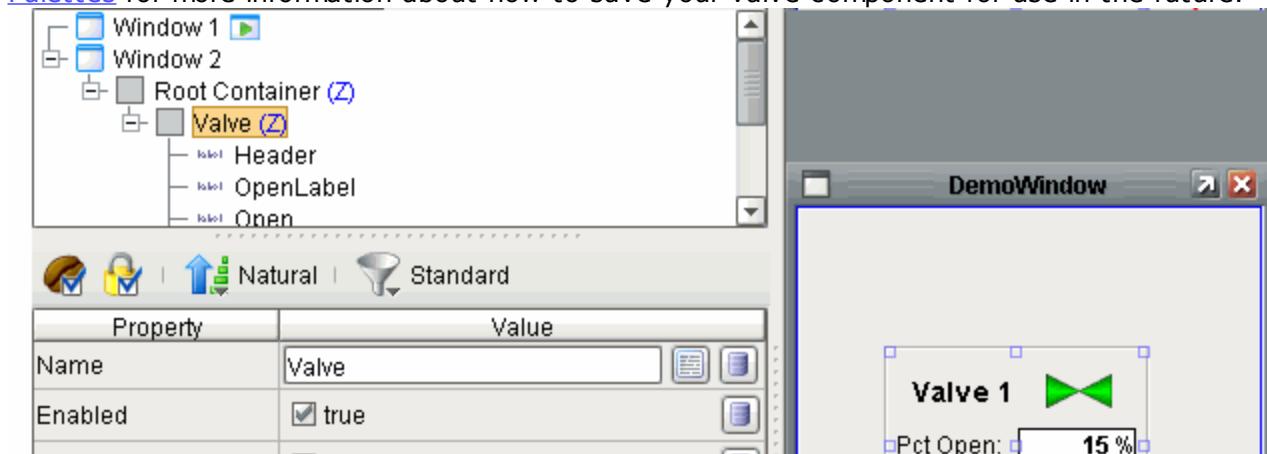


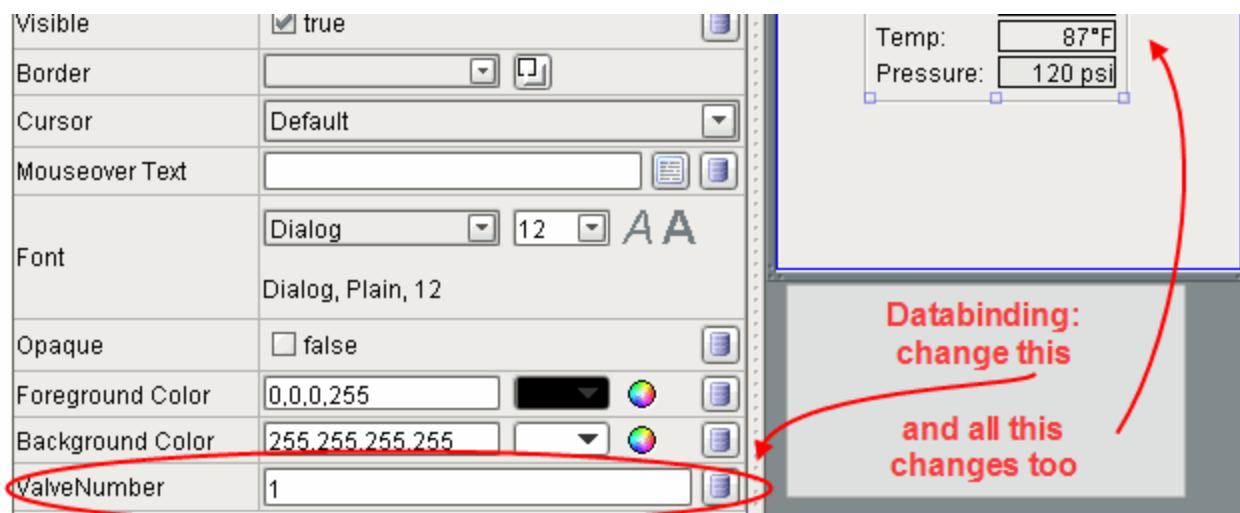
For development, it is usually easiest to set the border property of the container so that you can see exactly where it's at, and remove it later. To add components to the container, simply draw them on top of it, or paste them on it.



Using Containers to make custom components

Containers are really useful for creating groups of components that act as one component. For instance, suppose you have many valves in the process that you are working on, and each of these valves have the same 3 pieces of information that you want to display: % Open, Temperature, and Pressure. You could create a Container that had an Image of a valve, and two Labels: one for the temperature, and one for the pressure. Now, put an integer [dynamic property](#) on your Container that represents the valve number, and make all of the data bindings for your 3 pieces of information use that valve number. Now you can copy that container as many times as you want, just change the valve number to represent a different valve. You've just created a custom component out of existing components! This is a very common idiom in FactoryPMI, and mastering this will save you hours of time. See [Custom Palettes](#) for more information about how to save your valve component for use in the future.





Grouping

The objects within a container may be **grouped** by selecting the container and choosing *Group Container* from the **Component** or **right click** menu. This makes the set of components "feel" like one by preventing the selection of child objects in the designer. They can still be selected in the [Project Browser](#). Ungrouping is done just like grouping.

Grouping is useful when you have large numbers of small components on a screen. It is also useful to do prior to saving components on a [Custom Palette](#).

The Root Container

Every window has a special container called the *Root Container*, which is anchored to stretch the entire size of the window. It contains all window objects including containers and cannot be removed or renamed.

The Root Container has a special job for passing parameters in window navigation. Parameters are passed by setting Root Container [dynamic properties](#) after opening a window. More detail can be found [here](#).

Custom Palettes

Custom palettes provide a way for you to create reusable modules that can be used in any of your FactoryPMI projects. A custom palette item can be created from any set of components, and by using [Containers](#) and Dynamic Properties you can create highly generic and customizable components that can be used over and over.

To start with, you must create a custom palette tab in the [Palettes Area](#). To do so, right-click on any existing tab, and select "New Custom Palette".



You will now see a new tab, which you may rename by right-clicking on it and selecting "rename". This tab will have



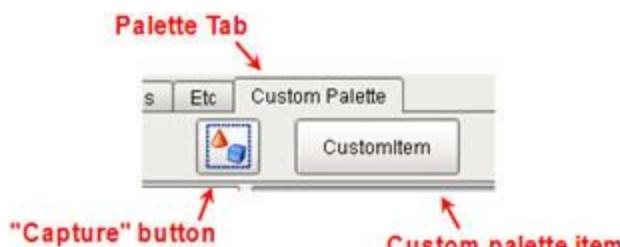
one item, the "Capture" button:

This button will take whatever you have selected in the [Project workspace](#) and create a custom component out of it. You may select a container, a number of components, or even a complete window.

When you click "Capture", the "New Palette Item" window will pop up, where you can edit the properties of your new item, giving it a name, description, and icon.



Once you click "OK", a new button will be placed on the palette, providing access to your item. To place a custom item on the form, click the button on the palette, and then click on the form where you want it to be placed, exactly as if you were pasting an object.



A custom palette with 1 item

Importing/Exporting Palettes

You can import and export custom palettes to files easily by right-clicking on the palette tab:

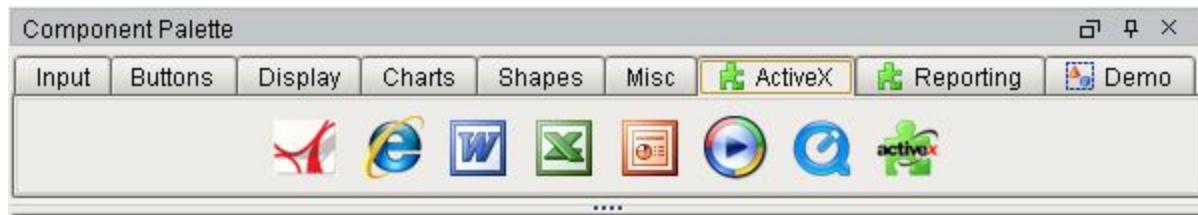


Important: Remember, custom palettes are shared across the *entire server*, not just one project. Therefore, it is not advisable to have multiple designers editing the palettes at one time, as they will overwrite each other.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Plugins

Plugins provide additional functionality in the form of FactoryPMI components. We sell some like the [Reporting Plugin](#), and others, such as the **ActiveX plugin**, are free.



First add plugins in the [FactoryPMI Gateway Configuration](#) page under **Plugins -> Manage**. Click the Add New Plugin icon.

The next time you open the designer there will be a tab on the [Component Palette](#) with the objects in the new plugin .

The screenshot shows the FactoryPMI Gateway Configuration interface. The top navigation bar has the title "FactoryPMI GATEWAY CONFIGURATION" and the subtitle "PLANT MANAGEMENT INTERFACE". It also shows the user is logged in as "admin" with links for [logout], [restart], and [shutdown].

The left sidebar contains a navigation menu with the following items:

- System: Status, Settings, Network, Cluster, Backup/Restore
- Projects: Manage, Launch
- Datasources: Connections, Drivers, Types
- Authentication: Profiles, Types
- Plugins: Manage (highlighted)
- Images: Manage

The main content area is titled "Plugins". A message box displays: "Plugin 'ActiveX Plugin' installed successfully." Below this is a table listing the installed plugins:

Plugin Name	Version	Activation Status	Description
ActiveX Plugin	1.0.1	Free Plugin	This plugin provides components used to embed ActiveX controls into FactoryPMI. Click Here for documentation.
Reporting Plugin	1.1.6	Trial Expired. Click to reset.	This plugin provides components used for advanced reporting functionality. Click Here for documentation.

On the right side of the table, there are several icons: a red X, a blue gear, a yellow lock, a red X, and a green plus sign. At the bottom of the sidebar are three buttons: "Launch Designer" (with a green icon), "Gateway Status" (with a green icon), and "Help" (with a green icon).

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Layout Control

One of FactoryPMI's great features is its ease of application deployment. This inevitably leads to your HMI/SCADA project not only being loaded on the operator's terminal, but on managers desktops, IT's computers, maintenance's laptops, etc. While it is great to get the data in front of as many people's eyes as possible, this introduces new challenges.

One of most obvious practical challenges is the fact that all of these different computers have different sized monitors. This means that you as an application designer need to have a way to ensure that your application is functional on a small monitor, while allowing users with large monitors to take advantage of their impressive screen real estate.

FactoryPMI gives you precise component layout control with two layout modes, controlled on a per-component basis.

- **Relative Mode.** The default layout mode for a new component. This mode simply ensures that a component stays in the same relative size and location (compared to its parent). As an added bonus, this mode scales the component's font size as well so that it stays in proportion. So, design your window at one size, and if a user opens it up on a large monitor, it will look proportionately the same. Likewise for a small monitor. This mode lets FactoryPMI take care of the screen size problem for you.
- **Anchored Mode.** This mode lets you independently anchor each side of a component to be some fixed distance away from the sides of its parent. It also lets you center a component vertically and/or horizontally, without having that component's size change as its parent changes size.

To change the layout mode for a component, right click on it and choose "Layout" (Cntl+L) .

NOTE. The relative layout engine is disabled by default in the Designer. You can enable it with the Relative Layout toggle button on the toolbar. It is disabled by default because usually in the designer when you resize windows/containers, want everything to stay put. Relative Layout is enabled always in the runtime.

Examples

Text centering [example](#) using anchored layout.

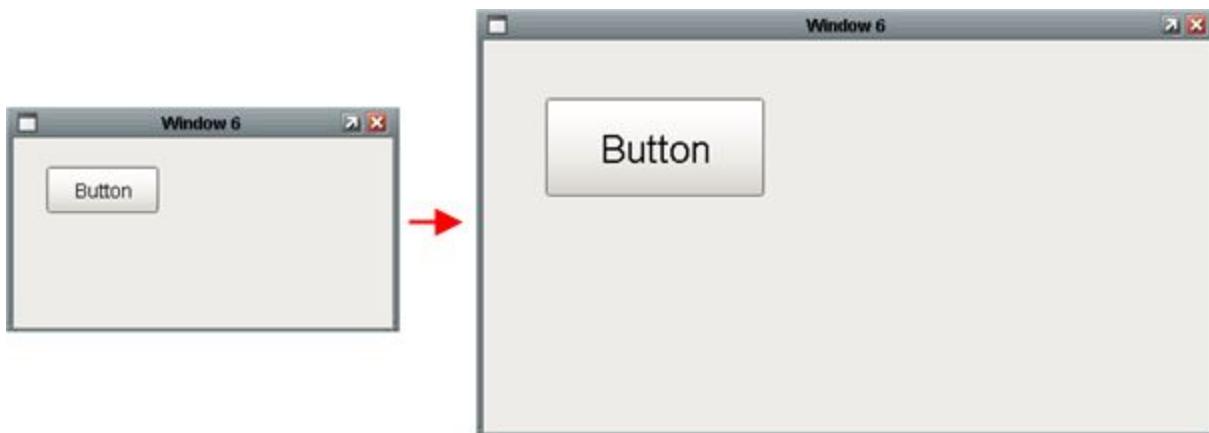
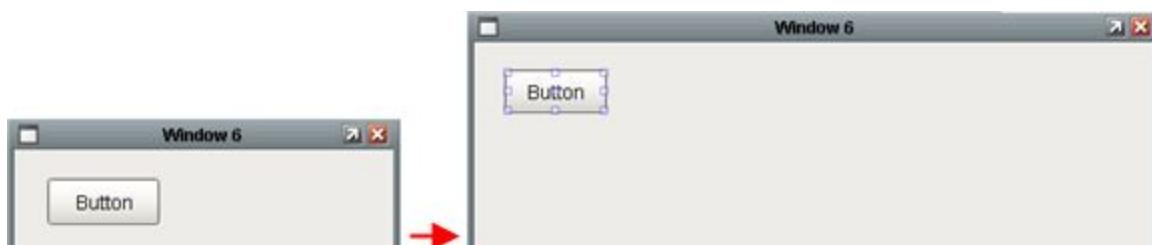


Fig 1. Relative Mode



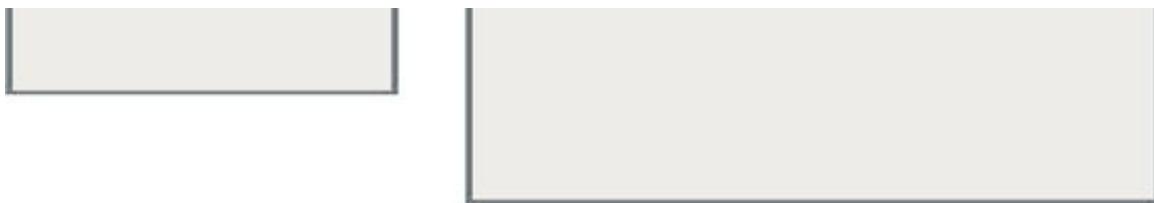


Fig 2. Anchored Mode (North-West anchor)

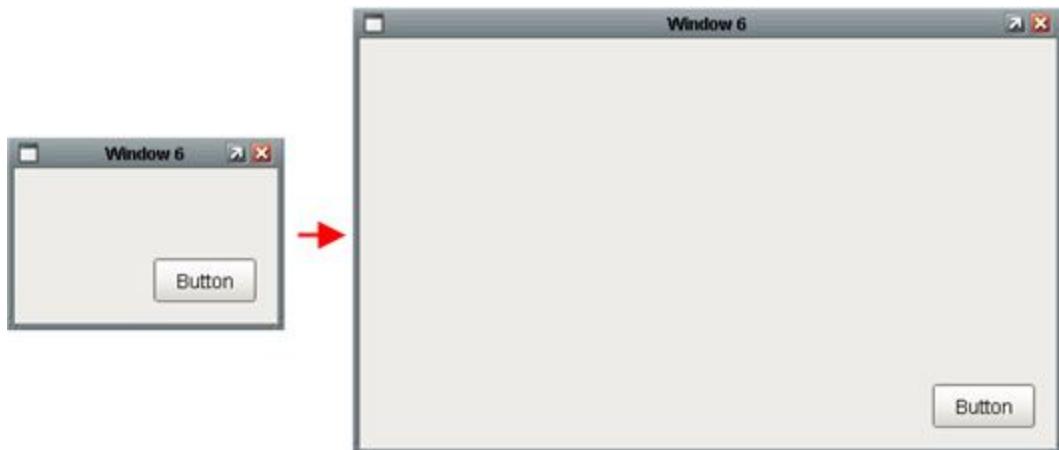


Fig 3: Anchored Mode (South-East anchor)

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



SQLTags™ Overview

SQLTags™ simplifies HMI/SCADA realtime status and control. You will typically not use the technology when dealing directly with dataSets, such as [charts](#), [tables](#), and reports. Nearly everything else in FactoryPMI, specifically when displaying or editing discreet points of any [data type](#), should be implemented with SQLTags™. Not only will development be easier, but performance and scalability will be greatly enhanced.

Things to know about SQLTags™

Tags Have Properties

SQLTags have numerous usable properties. Properties are stored with the tags in the SQL database and modified directly or with the [SQLTags editor](#). Often property values, such as the value of **EngUnit**, will come from the OPC Server directly. The default property of a tag is it's **.Value** property.

Tag	Value	Datatype
Compressor2		
Tower1		
Tower2		
accumulatorLevel	83	Int4
ambientHum	5.4	Float4
ambientTemp	149.25	Float4
dischargePressure	57.37	Float4
dischargeTemp	49.0	Float4
LoLo	24.7	Float4
Value	24.7	Float4
Documentation		String

Did you know...

Using SQLTags properties allows you to make universal changes. For example, changing the **FormatString** on a **dateTime** SQLTag could affect the format of all components bound to it.

Tag Types

There are different [types of SQLTags](#). These are **DB**, **OPC**, **Client**, and **System**. **Client**, and **System** are local to a FactoryPMI Client, while **DB** and **OPC** are stored in an SQL database.

SQLTags™ uses an SQL Database

SQLTags™ creates multiple tables in the SQL database on a [Datasource Connection](#). You will see a listing for each datasource that has SQLTags enabled. A FactorySQL node will create these tables and update your SQLTags datasource.

Tip

In most cases you will want to use SQLTags under **Default**, which corresponds to your [Default/Primary Datasource](#). Carefully consider your situation before using SQLTags on multiple datasources in one project!

Tag	Value	Datatype
Default		
North Area		
Area_temp	97	Float4
Temp High SP	99.4	Float4
Overview		
Refrigeration		

Scan Classes

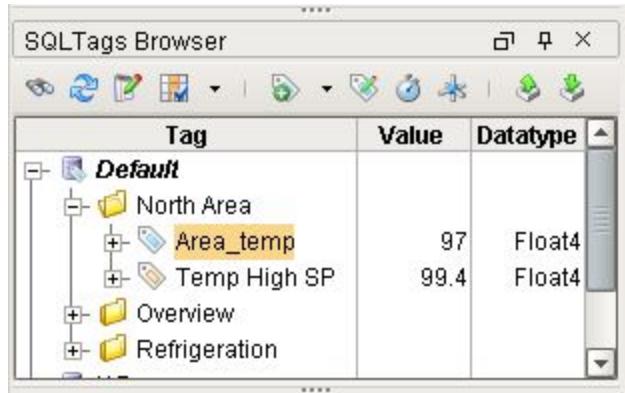
Tags are updated at a rate defined by their [Scan Class](#). This allows changing of timing in one place. There are different update modes that a scan class may use.

Component Overlays

Components now support [overlays](#) to indicate status (quality codes).

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

There are four types of SQLTags. **DB**  and **OPC**  SQLTags are stored in an SQL Database via a [Datasource Connection](#) . **Client**  and **System**  SQLTags work within FactoryPMI and are stored under the builtin *Client* and *System* Tag folders, respectively.



OPC Tags (blue)

OPC Tags point to a specific OPC server and item path and are stored in the SQL database. **OPC Tags** are accessed via a Datasource Connection . **OPC Tags** are the most common type of tag for an HMI/SCADA application. They are usually created by [drag and drop](#) between the [OPC Browser](#) and [SQLTags Browser](#).

DB Tags (yellow)

A **DB Tag** is stored in the SQL database. It is similar to a **Client tag**, except it is shared between FactoryPMI clients and is persistent. **DB Tag** may be bound to [Expressions or SQL Queries](#). **DB Tags** are accessed via a Datasource Connection .

Suppose a manager wanted to be able to edit a daily message that was displayed on the top of all users screens. This could be accomplished with a label bound to a **DB Tag**. The manager would be given access to a window where the DB Tag could be changed.

Client Tags (green)

Client tags are variables of any [type](#) that can be accessed from anywhere in a FactoryPMI client. They can be thought of as global [dynamic properties](#) that operate within an instance of a FactoryPMI client. Separate FactoryPMI clients will have separate values of **Client tags**. **Client tags** are not **persistent**, meaning tag values will re-initialize if the user logs out of, or closes, the application. **Client tags** are stored under the *Client* tag folder.

Client tags are useful when you need a variable across screens and would rather not pass parameters. A system with 3 identical production lines could force users to work on one line at a time, regardless of windows or popups on. They could use a **Client tags** to specify which line is being used.

System Tags (purple)

System Tags come from the FactorySQL client or Gateway. They provide values that are system properties such as: CurrentDateTime, Username, IPAddress, etc. System Tags are typically read only. **System Tags** are stored under the *System* tag folder.

A label whose text property is bound to the *Username* System Tag will display the current user.



Tag Paths

Tags and their properties can be referenced by a string based path. Each has a unique *absolute path*, and will often have many equivalent *relative paths* when referenced from other tags (described below). You will most often generate these by browsing or with [drag and drop](#) property binding. It's a good idea to understand how tag paths work, particularly if you get into [indirect tag binding](#) or scripting.

A tag path will look something like this: **[Source]**folder/path/tag.property

The *italicized* portion of the path may contain the following:

- A tag
- Any number of nested folders followed by a tag, separated by forward slashes (/).
- A period (.) followed by a *propertyName* after the tag. Omitting this is equivalent to using the *.Value* property.

Now consider the **[Source]** (portion surrounded by square braces)

Source Option	Meaning	Applicability
[DataSource_Name]	The name of the FactoryPMI Datasource where the tag is stored	OPC, DB
[]	The Default or Primary Datasource of the current project	OPC, DB
[.]	The current folder of the tag whose property is being bound	DB and Client
[~]	The source of the tag whose property is being bound	DB and Client
[Client]	Stores Client SQLTags	Client
[System]	Stores System SQLTags SQLTags	System

Paths that begin with **[.]** or **[~]** are *relative* whereas all others are *absolute*.

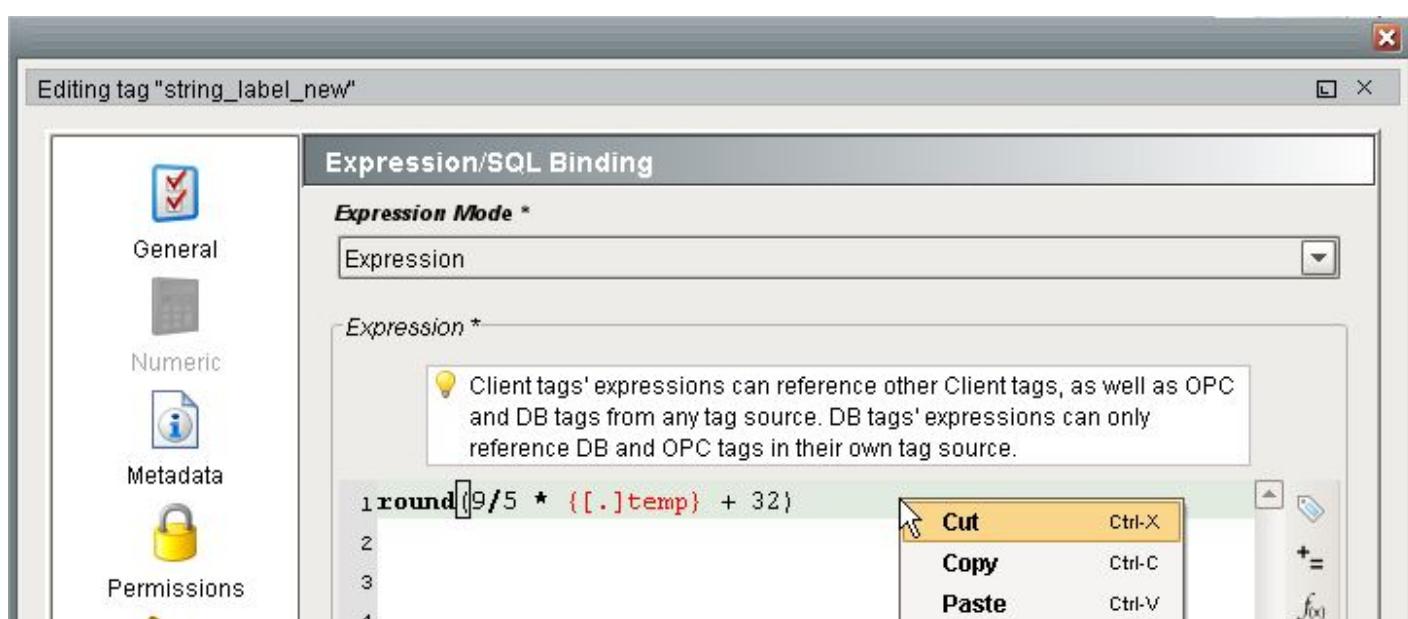
Tag Expression Binding

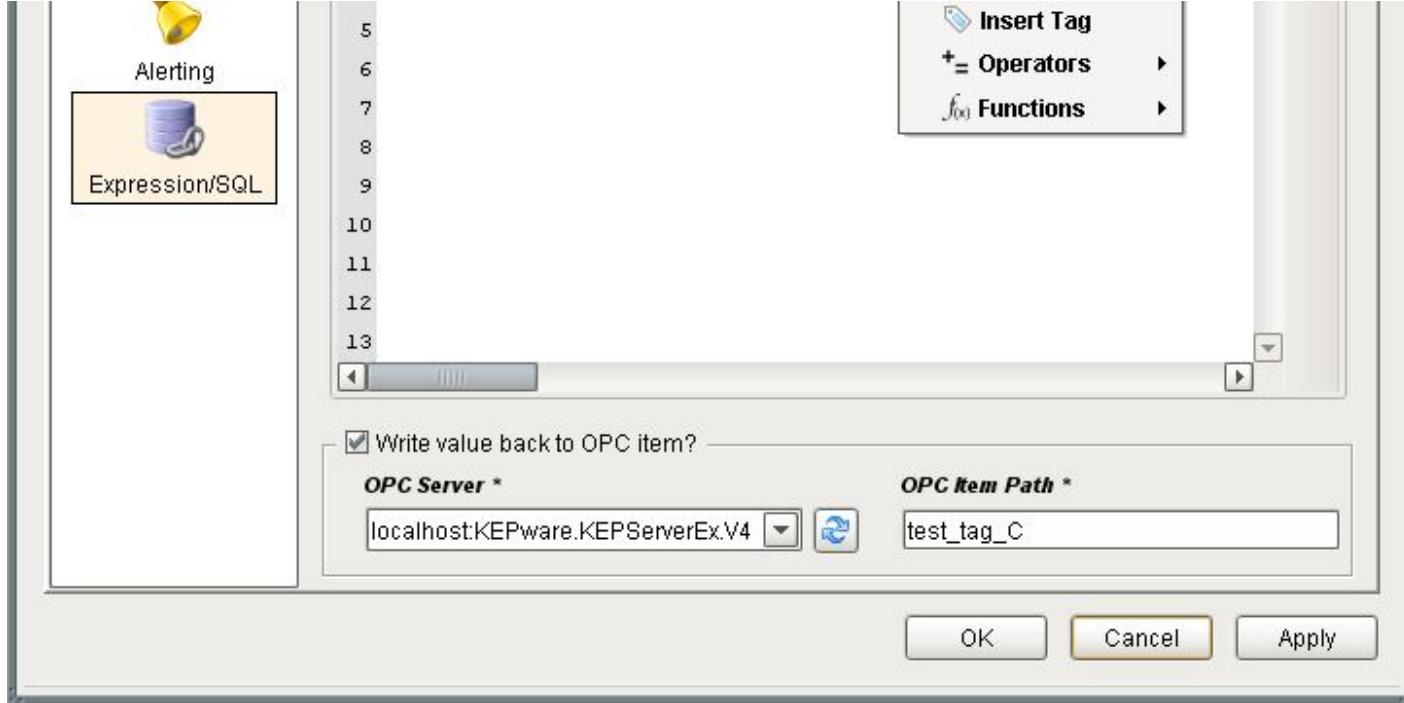
Client and **DB** tags can be [bound](#) to [expressions](#) containing other tags with the [edit tag](#) dialog. DB Tags may be bound to expressions containing other DB Tags. Client tags may be bound to expressions containing Client and DB Tags.

Most often, binding is done through GUI windows. Make use of the Insert: Tag , operators **+=**, and (expression language) functions **f(x)** buttons.

Did you know...

[.] bindings will work with tags that are moved between **folders** and **datasources**, while **[~]** bindings will work with tags being moved between **datasources**.





Component Property Binding

Any [bindable](#) component property may be bound to a SQLTag. The path will always be *absolute* since components exist outside of the [SQLTag tree](#). Selecting a tag or tag property in the **Tag Browser** will automatically fill in the **Tag Path**.

Property Binding: Sample.Root Container.Temp_SP.doubleValue

Property Binding: Sample.Root Container.Temp_SP.doubleValue

Binding Types

- Tag**
- Tag
- Indirect Tag

Property

- Property
- Expression

Database

- DB Browse
- SQL Query

No Binding

Choose Tag:

Default

- + A Folder
- + import test
- + Local_3_I
- + Path
- + Ramp
- + Area Temperature
- + Current Pressure
- + Device Area
- + DXInteger
- + DXInteger 2
- + Low Pressure Setpoint
- + PsFloat1
- + PSFloat1 (Fake)
- + PsFloat2
- + PsInteger1
- + string_label_new
- + Word_1
- + Writeable test point
- + mysql_localhost

Tag Browser

Tag Path

Options

Bidirectional Overlay Opt-Out

OK **Cancel**

Options	Function
Bidirectional	The property will update itself based on the bound tag and write to the tag if the property changes
Overlay Opt-Out	Ignore the quality of this tag when calculating the components Data Quality .

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Tag Paths

Indirect Tag Binding is a form of [component property binding](#) that allows you to build up a tag path with variables (**references**). The reference can be the entire tag path, or any part of it.

The Tag Path is most simply generated by browsing . The substituted text is a number enclosed by braces ({}, {}). The references table maps the dynamic property to the dynamic portion of the path ({}, {}). References are most easily

generated by browsing . They are also created for you if you type in curly braces enclosing a number in the **Indirect Tag Path**.

In the example below the value of the dynamic property **number** on the *Root Container* gets plugged into the **{1}** in the **Indirect Tag Path**. Changing the value of **number** at runtime will re-point all of the indirect tag bindings in the window that reference it.

Example

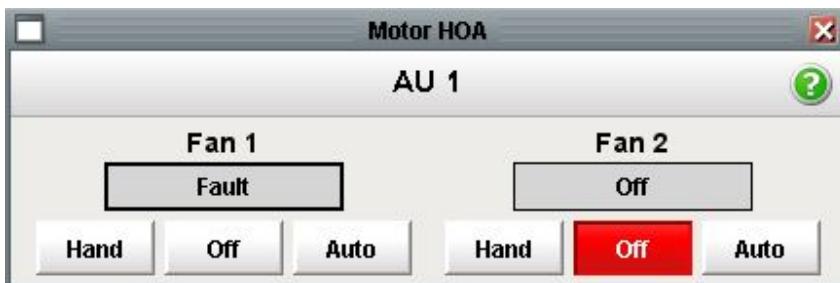
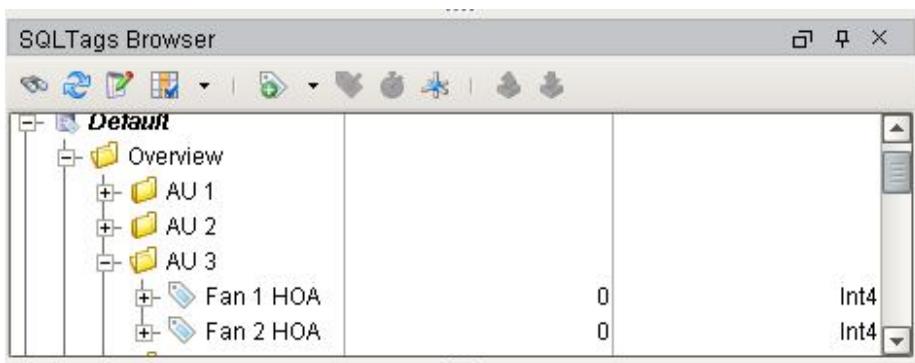
The following example templates Air Units by creating a numbered SQLTags folder for each Air Unit. The designer can create a single window, that constructs the tag path based on consistently named Air Unit folders. This allows the designer to add Air Units with minimal changes in FactoryPMI! Each Fan, or any other component, may get its tag path from an Indirect Tag Binding.

Did you know...

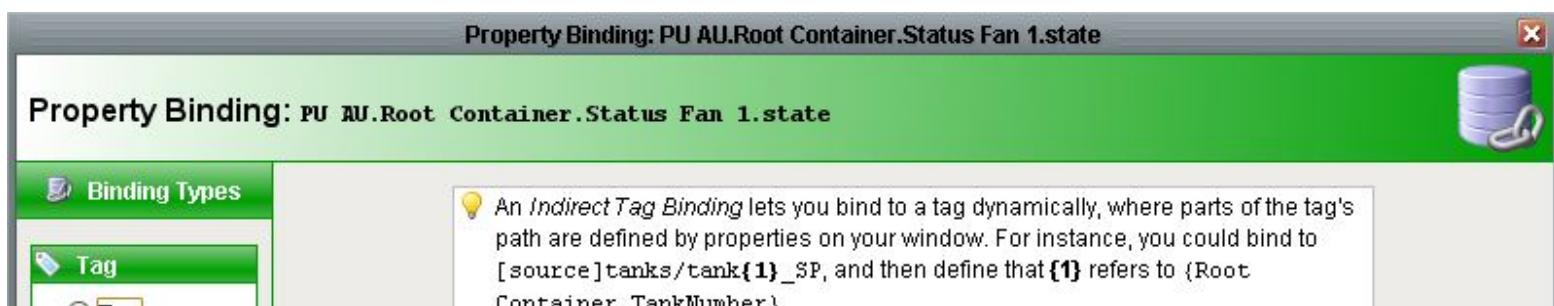
Displaying indirect tag binding can be equivalently done with expressions, but it is very useful for cases where you need bi-directional support. Binding to expressions would require scripting to write to tags!

Tip...

Indirect Tag Binding works great for parameterized popup windows. Don't overuse it in your project as it introduces complexity!



Below is an example of the binding of the *state* of **Fan 1**. Notice that the *Root Container* [Dynamic Property](#) **number** tells the window which AU to refer to, and gets [passed](#) in when the window is opened.





Indirect Tag

Property

- Property
- Expression

Database

- DB Browse
- SQL Query

No Binding

Indirect Tag Path

[] Overview/AU {1}/Fan 1 HOA



References

Ref. #	Property Path
1	Root Container.number



Options

Bidirectional Overlay Opt-Out

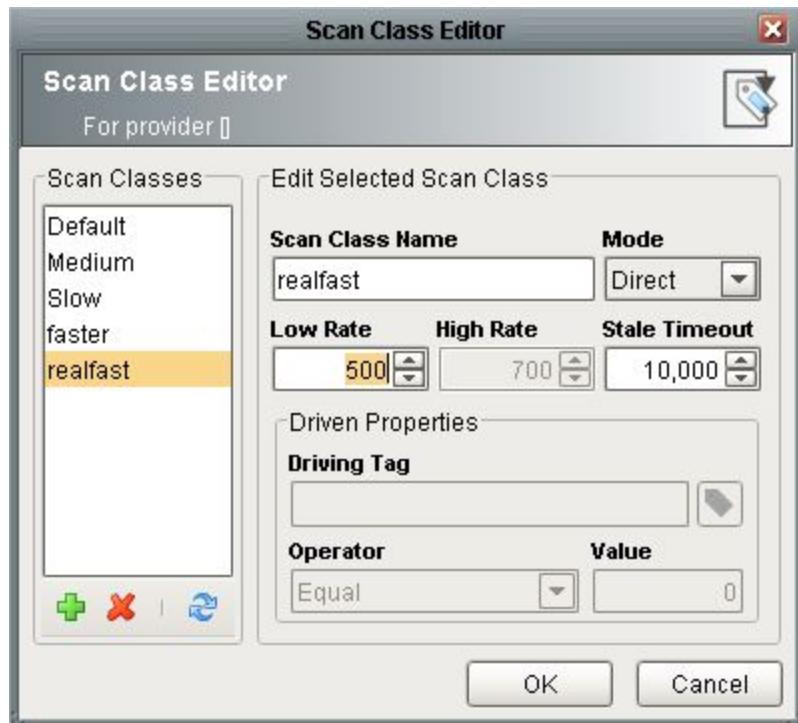
OK

Cancel

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

SQLTags™ - Scan Classes

A **Scan Class**  defines the update rate and associated settings for SQLTags that use it. Each tag necessarily uses a single scan class. Using scan classes instead of having individual update rates for tags is done for simplicity and performance.



Tip...

Don't try to pre-optimize your scan classes. Begin with a **direct mode**. It takes no time to change this if necessary. You can also change the scan class of [multiple tags at once](#).

SQLTags is very efficient. It won't transfer data that isn't changing - be smart with your [deadbands](#)!

You may add an arbitrary number of **Scan Classes**. Each has the following properties:

Property	Function
Scan Class Name	Unique name of the Scan Class
Mode	Described Below
Low Rate	Base update rate. How many milliseconds between polling.
High Rate	Faster update rate for <i>Driven</i> and <i>Leased Modes</i>
Stale Timeout	Time before data is considered to go "stale"

Scan Class Modes

Direct - Always updates at the **Low Rate** (ms).

Leased - Updates at the **Low Rate**, but speeds up to the **High Rate** when tags are "Leased". Opening a window in the runtime that has an object bound to a SQLTag will cause a "lease". Many other conditions "lease" a SQLTag.

Driven - Update at the **High Rate** when the **Driven tag Operator Value** (tag=1, for example) evaluates to *true*, and at **Low Rate** otherwise. This is often used if you want a very fast update while something is running.

Tip...

You can use a driven tag that is based on a complex expression involving other tags.

Property	Function
Driven Tag 	Tag whose value will be compared against the value

Operator

Numeric Operators (>, <, =, !=, etc)

Value

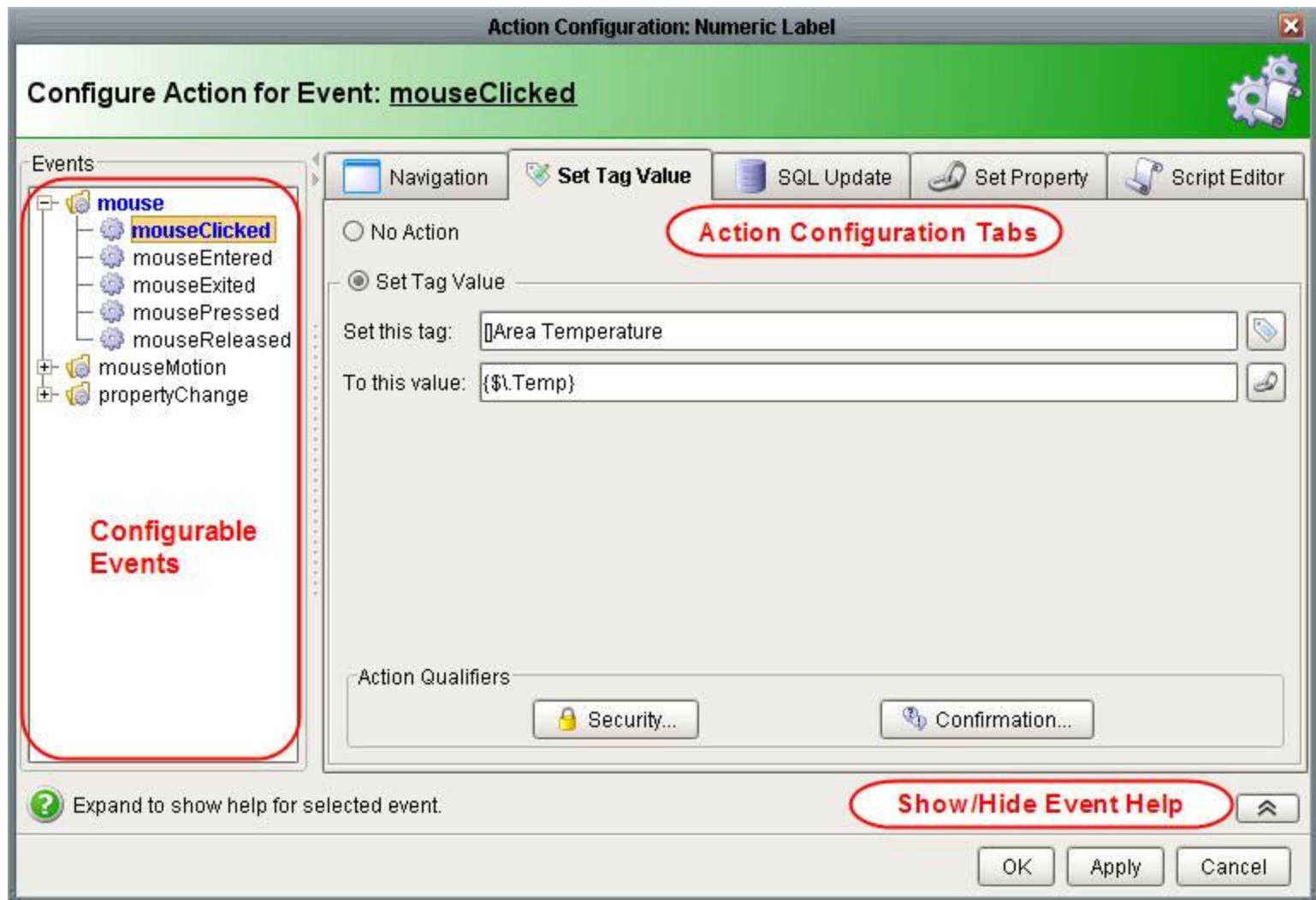
Static value to compare the Driven Tag to. See tip to deal with complexity.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

SQLTags™ - Configure Actions

You may configure Actions that respond to Component Events. The **Set Tag Value** builder generates a script for you to write to SQLTags. You browse a tag path  and set that value to a component property value . This works much like the [Set Property](#) Action Tab.

This is applicable wherever you want user interaction to change the value of a SQLTag, but can not accomplish the task with [binding](#).

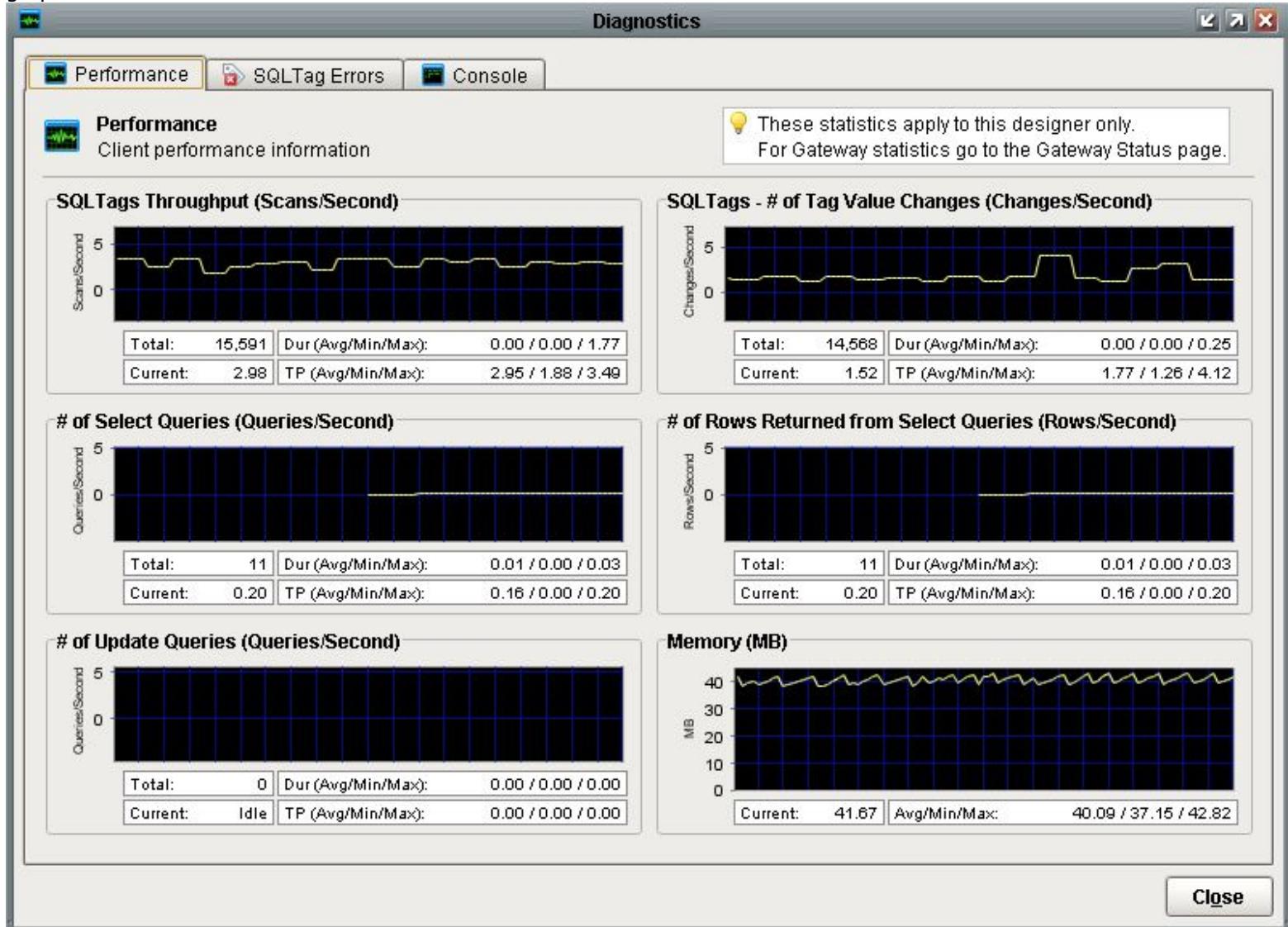


SQLTags - Diagnostics

Client diagnostics give the user a metric of system load and performance for that client. They may be run on a Designer or Client. For Gateway statistics go to the Gateway Status page.

Performance

Client performance statistics including Average, Minimum, and Maximum Duration and Throughput. You may click on the graphs to zoom.



The following metrics are available

Metric	Meaning
SQLTags Throughput	SQLTags scans per second on the FactoryPMI gateway. With the default update delay of 250ms, the ideal throughput is 4.0 per datasource. Useful to determine that the SQLTags system is working from client to gateway.
SQLTags - # Tag Value Changes	Number of Tag values changing per second. This ultimately affects load on the SQL database. Useful for adjusting deadbands on the FactorySQL side.
# SELECT Queries	Number of SELECT queries per second run by the client. This number should be low if you're using SQLTags.
# rows returned from SELECT Queries	Number of rows returned from SELECT queries per second. Useful for analyzing dataSet traffic from large charts , tables , and reports.

UPDATE Queries

Number of UPDATE queries per second run by the client. This metric should usually be low.

Memory

Current memory usage in megabytes. Your client plus whatever you have in memory.

SQLTag Errors

The SQLTag Errors tab provides the ability to monitor errors in SQLTag execution. Errors may be filtered by Tag Path, Lifecycle, or Severity.

Diagnostics

Performance SQLTag Errors Console

SQLTag Errors
SQLTag Errors

Filters

Tag Path: [Carl_Dev]Device Area Lifecycle: Any Severity: Debug

Path	Message	When	Stage
[Carl_Dev]Device Area	Error evaluating expression.	Sep 18, 2007 3:33 PM	FSQL_Exec...		
[Carl_Dev]Local_3_I/CSTTimestamp...	A config error occurred while subscri...	Sep 16, 2007 5:43 PM	FSQL_Loadi...		
[Carl_Dev]Local_3_I/Data	A config error occurred while subscri...	Sep 16, 2007 5:43 PM	FSQL_Loadi...		
[Carl_Dev]Local_3_I/Fault	A config error occurred while subscri...	Sep 16, 2007 5:43 PM	FSQL_Loadi...		

Error Details

Path: [Carl_Dev]Device Area Type: Tag Lifecycle: FSQL_Executing Severity: Error

Message:
Error evaluating expression.

Stack Trace:
Referenced item '[sqltags_test]bigtag1003' does not exist.
FactorySQL.GroupExecutionException
at FactorySQL.Scripting.ScriptExecutor.Execute()
at FactorySQL.SQLTags.SQLTagRuntimeExecutor.Execute()
at FactorySQL.SQLTags.Tags.DBExpressionSQLTag.InternalEvaluate(TagEvaluationToken EvaluationToken)

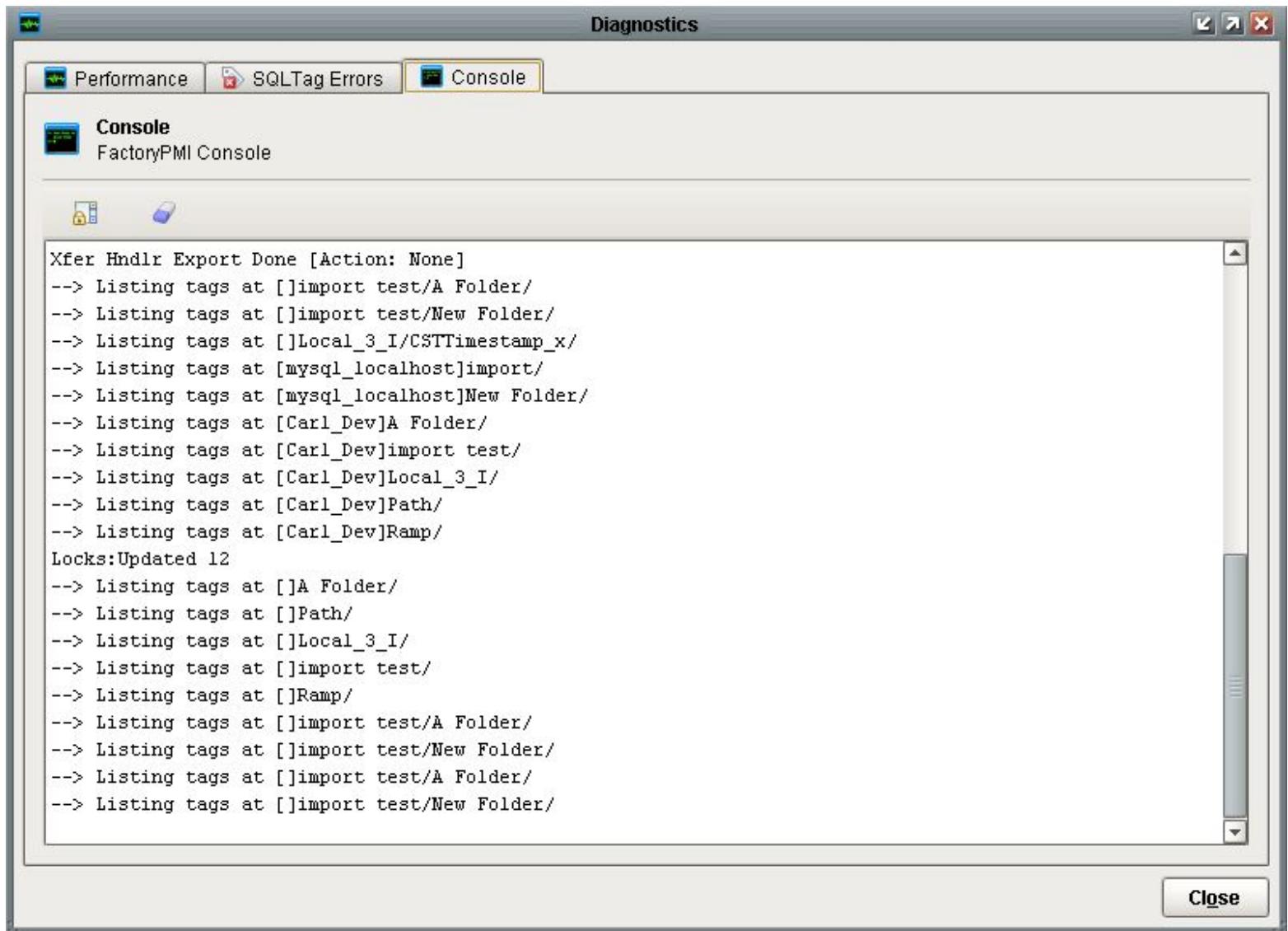
Buttons: Copy to Clipboard Send to Inductive Automation

Close

Property	Meaning
Tag Path	Tag_path
Lifecycle/Stage	Point of error in the SQLTags system. Options include: <i>FSQL>Loading</i> , <i>FSQL>Executing</i> , <i>FSQL>Unloading</i> , <i>Gateway>Polling</i> , <i>Gateway>Responding</i> , and <i>Client>Polling</i>
Severity	Error Severity. Options include: <i>Debug</i> , <i>Info</i> , <i>Warn</i> , and <i>Error</i>
Message	Description of the error
Stack Trace	Full stack trace of the error. This could come from FactorySQL or FactoryPMI. It can be incredibly verbose, but usually tells you what the problem is. This is often useful to provide to tech support if the problem is difficult.

Console

Opens the system console. Scripts' **print** output is printed to this console. Also, if you discover a bug in FactoryPMI, you can find information about what went wrong on the console



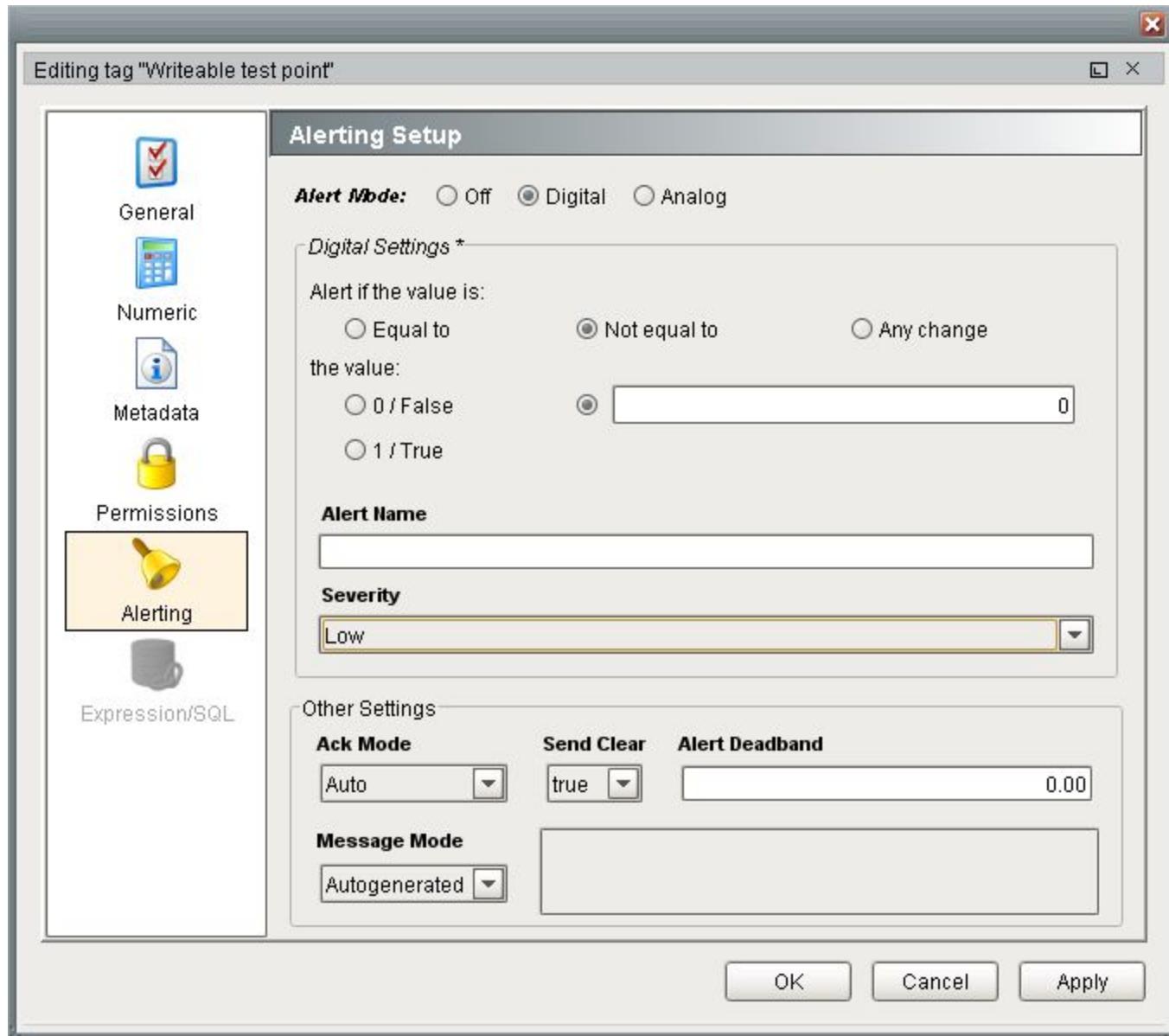
Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

SQLTags™ - Alerting

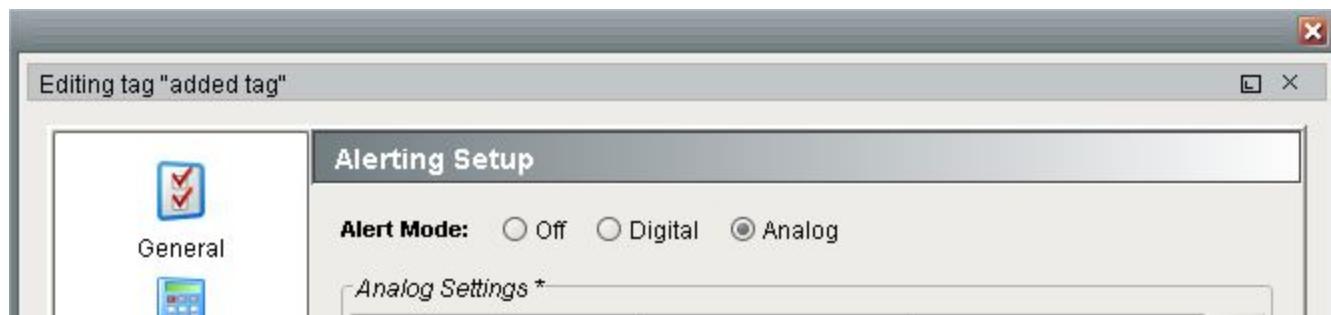
Alerting works with FactorySQL to log alarms to the database and notify users via email, alphanumeric page, or text message. It may be configured on OPC and DB SQLTags. The FactorySQL help manual covers more options including acknowledgement, notifying groups, etc.

Digital Alerting

Digital Alert settings are straightforward. Choose **Equal to** or **Not Equal to** and a value, or Alert on **Any change**. **Other Settings** are identical to *Analog Alert Mode* (below).



Analog Alerting



State Name | Range | Severity

My_range	[12, -5]	Low
----------	----------	-----

State Properties

Alert Name	Severity		
My_range	Low		
Low	<input type="checkbox"/> Tag-Driven	High	<input type="checkbox"/> Tag-Driven
12	Inclusive	-5	Inclusive
<input type="checkbox"/> Alert on any change			

Other Settings

Ack Mode	Send Clear	Alert Deadband
Unused	true	0.0001
Message Mode		
Autogenerated		

OK Cancel Apply

The **Analog Alert Mode** defines an arbitrary number of **Analog Settings** entries. Each entry defines the following:

Property Name	Function
Alert Name	Gives the alert a name.
Severity	5 Different levels to distinguish between alerts. FactorySQL can be set to notify based on severity. FactoryPMI alarm screens often sort by severity.
Low value	The low value of an analog range
High value	The high value of an analog range
Alert on any change	Alert on value change.

Low values and high values may be set **inclusive** (exact border condition triggers alert), **exclusive** (not inclusive), or may use positive or negative infinity in the range. Clicking **Tag-Driven** will use the value of a SQLTag for the alert. This can be very useful for customers who wish to enter their own alert setpoints.

Other Settings - applicable to tag, not each state.

Property Name	Function
Ack Mode	How you want to acknowledge alarms. <i>Unused</i> always acknowledges alarms - for if you're not using an acknowledgement system. <i>Auto</i> acknowledges alarms as soon as they clear. <i>Manual</i> leaves it up to the user to acknowledge the alarm.
Send Clear	<i>true</i> or <i>false</i> indicates whether user will be notified when value goes out of alarm state.
Alert Deadband	Tag must change at least this much to register as a change
Message Mode	<i>Autogenerated</i> uses the FactorySQL Alert system. <i>Custom</i> allows the user to type in a message.

Databinding - Overview

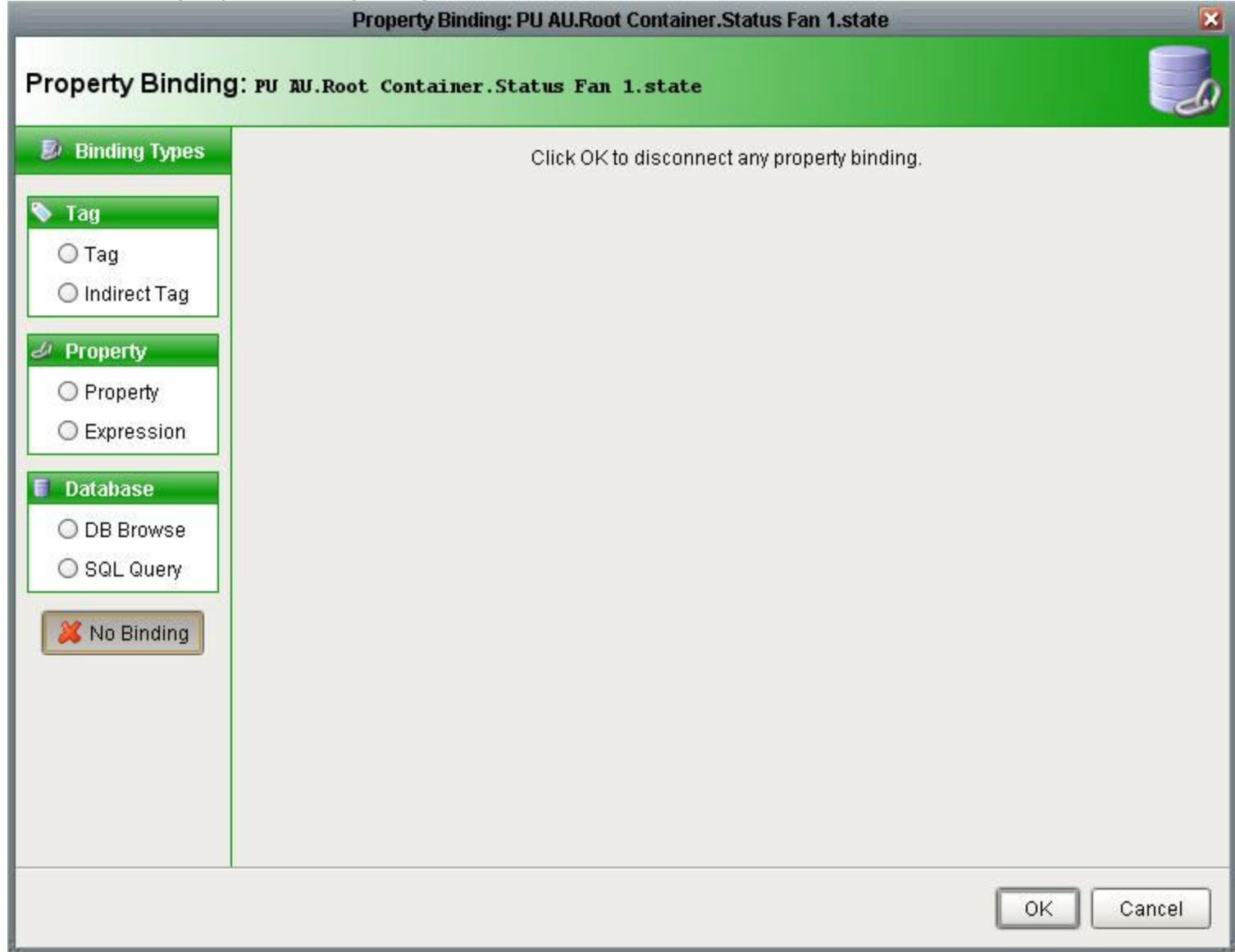
Databinding is a topic that is central to FactoryPMI. It is the mechanism that links properties to actual data, to each other, and to expressions in order to create a web of interaction that powers projects.

There are 3 primary ways to bind properties: To other properties, to SQL statements, and to expressions. Often, it will require a combination of bindings (accomplished through [Dynamic Properties](#)) to complete a task.

Did you know...

[SQLTags™](#) simplifies databinding with a [drag and drop](#) approach!

Nearly every property is bindable in FactoryPMI. Bindable properties are marked by the Bind Icon (). Clicking on this button brings up the binding dialog:



The radio boxes on the left side show the available binding options:

Data Binding Options

No Binding	No databinding. The value is whatever is entered in the designer. Choose this to remove any previous binding.
Tag	Binds the selected field to a property of a SQLTag .
Indirect Tag	Binds the selected field to an Indirect Tag . This builds up the tag path dynamically.
Property	Binds the selected field to the value of a different property . Displays a tree view of all the components in the window so you can find the property to bind to.
Expression	Binds the value of the field to the result of an expression, written in the FactoryPMI

[Expression Language](#). You can plug in all properties to your expression.

DB Browse	Presents a graphical way to browse the database for a field to bind to. Equivalent to binding to an SQL query.
SQL Value	Allows you to enter an SQL query to bind to. The SQL query designed by the DB Browser will be entered, if you used that tool. Allows you to specify other options for how the query is executed.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Data Types

FactoryPMI components have many different types of properties. As you are making dynamic properties of your own, binding properties' values to database values, etc, it is important to have a good grasp of the difference between data types.

Numbers

- **Short.** A Short is a 16 bit signed number, ranging from -32,768 to 32,767.
- **Integer.** An Integer is a 32 bit signed number, ranging from -2,147,483,648 to 2,147,483,647.
- **Long.** A Long is a 64 bit signed number, ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
- **Float.** A Float is a 32 bit floating-point number.
- **Double.** A Double is a 64 bit floating-point number.

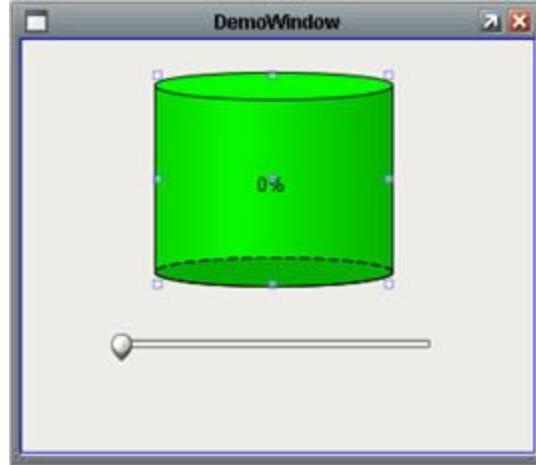
Other Types

- **Boolean.** A value that is either *true* or *false*. If a number is treated as a boolean, 0 is *false*, anything nonzero is *true*. If a boolean is treated as a number, *false* is 0, *true* is 1.
- **String.** A sequence of characters.
- **Color.** A color in the RGB (Red-Green-Blue) colorspace. Also includes an alpha (transparency) channel.
- **Date.** Represents a point in time with millisecond accuracy.
- **DataSet.** A DataSet is a two-dimensional array of values, where each column has a particular datatype and name.

Binding to a Component's Property

Binding the value of a property to that of another property is as simple as choosing which to bind, and which to bind to. You can bind to any property in the current window.

In this example, we will bind the value of a tank to the value of a slider.

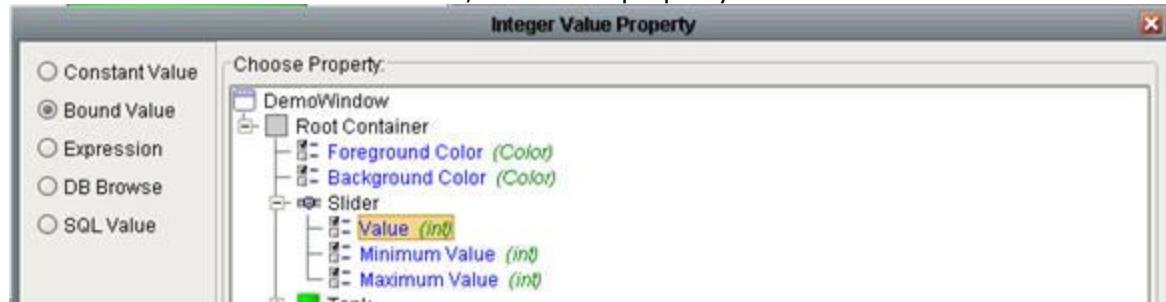


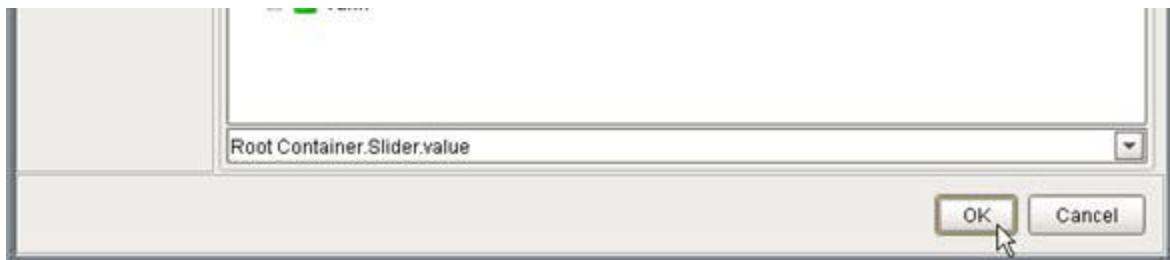
With the tank selected, click the "Bind Property" button for the value:

The screenshot shows the 'DemoWindow' application interface. On the left, a tree view shows the window structure: DemoWindow > Root Container > Slider > Tank. The 'Slider' node is expanded. On the right, the main view displays a green cylinder labeled '0%' with a horizontal slider below it. A red arrow points from the text 'Tank Selected' to the cylinder. Another red arrow points from the text '\"Bind Property\" button' to the 'Value' property's binding button in the property grid. The property grid shows the following properties for the 'Tank' component:

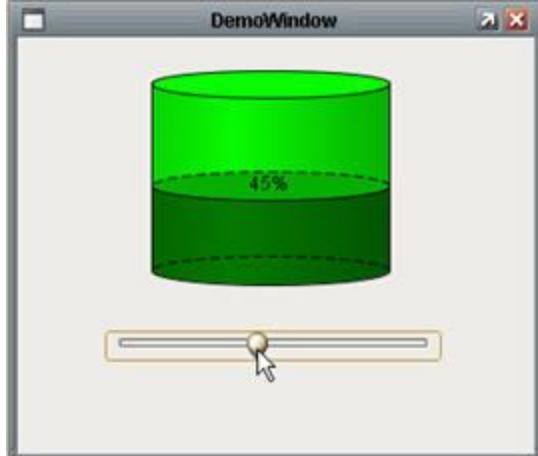
Property	Value
Name	Tank
Enabled	<input checked="" type="checkbox"/> true
Visible	<input checked="" type="checkbox"/> true
Border	[Color Chooser]
Mouseover Text	[Text Box]
Font	Dialog, 12 Dialog, Plain, 12
Foreground Co...	[Color Chooser] 0,0,0,255
Background C...	[Color Chooser] 238,236,232,255
Opaque	<input type="checkbox"/> false
Rotation	0° [Up/Down Buttons] Custom...
Value	0 [Binding Button]
Capacity	100
Units	[Text Box]

In the property binding window, choose "Bound Value" on the left side, and then browse to the value that you wish to bind to in the tree. In this case, the Value property of the slider. Click OK.

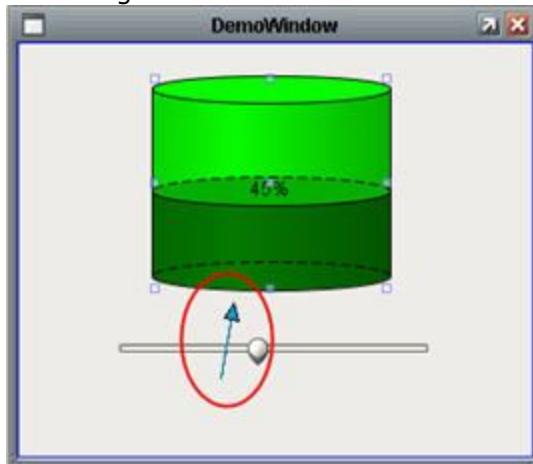




That's it! Now, if you click on the "Preview" button (▶), you can drag the slider and watch the value of the tank change instantly:



Also, now when you click on the tank (not in preview mode), you see a binding arrow pointing to it from the slider, indicating that the slider is bound to one of its properties.

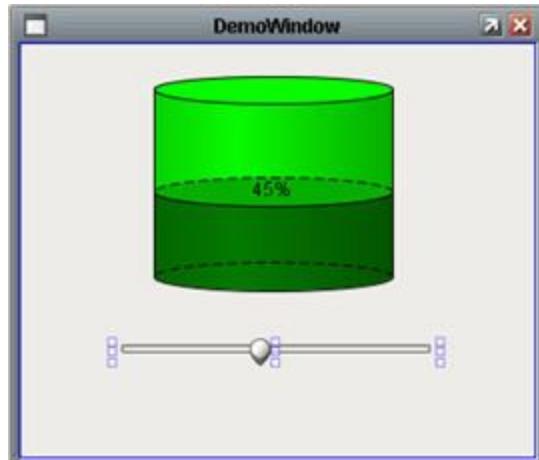




Binding to an SQL Query

Binding to an SQL query can be accomplished in 2 ways: by using the database browser, or by writing the query yourself.

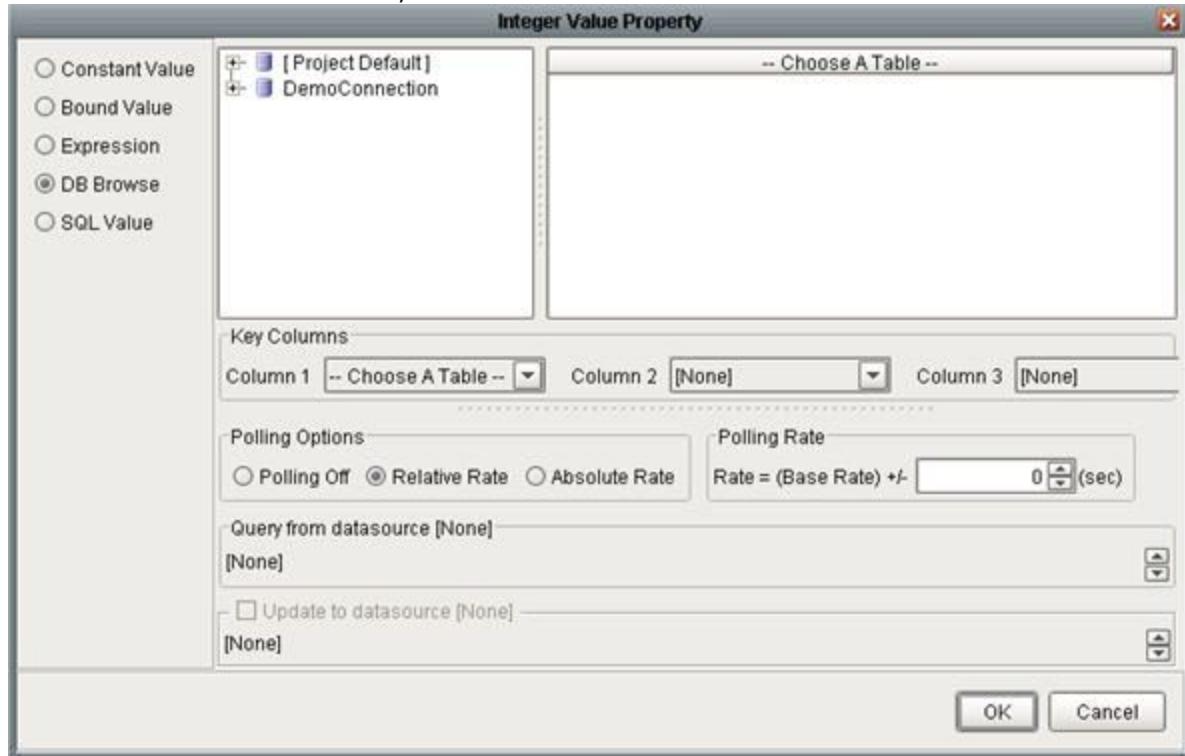
Working off of the example in [Binding to a Component Property](#), we will now bind the value of the slider to a database value:



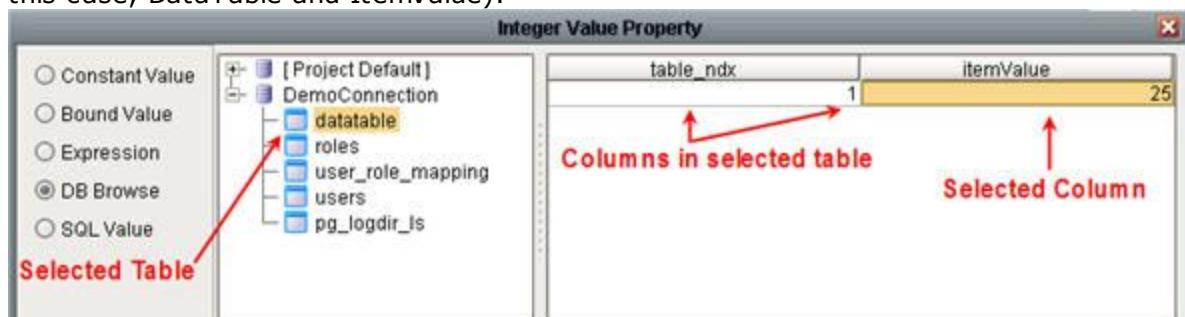
To begin, select the slider, and click the "Bind Property" button next to the Value property (☞)

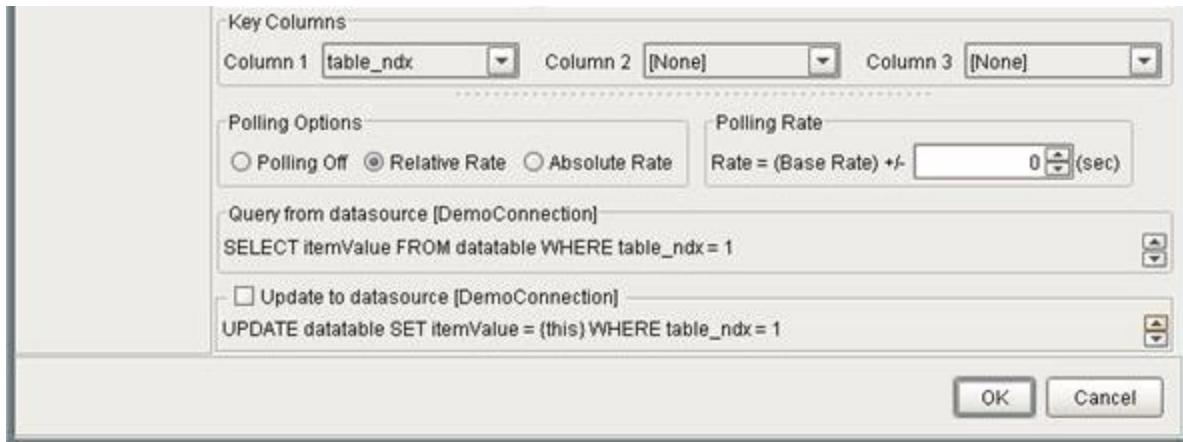
Using the Database Browser

To use the database browser, select "DB Browse" on the left side. You will see the following screen:



In the upper left pane, browse to the table you want to bind to, and click on the exact field in the right pane (in this case, DataTable and ItemValue):





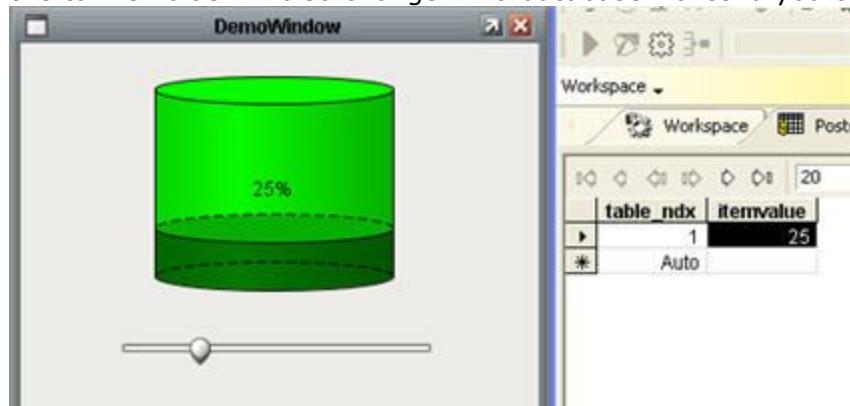
FactoryPMI will adjust the basic settings for you, detecting the primary key of the table and generating the queries. The following table outlines the options available on this screen:

Database Mapping Options	
Key Columns	FactoryPMI must be able to locate the value to bind to in the database, and thus there must be a field (or combination of fields, up to three) that is unique to that specific row. Usually database tables have one predetermined field, called the "Primary Key", that serves this purpose. In this case, FactoryPMI should automatically detect and use it.
Polling Options	Defines how often FactoryPMI will check the database for a change. The options are: <ul style="list-style-type: none"> Polling Off: The query will run when the window opens, and whenever any of the properties that are embedded in the query change (which would change the query). Relative Rate: Query will run according to the base rate, +/- the polling rate amount given. Absolute Rate: Query will run on the specified interval.
Polling Rate	If Relative Rate is selected, will specify the offset from the base rate. If Absolute Rate is selected, specifies exactly how often to run. An absolute value of 0 specifies to run as often as possible.
Query from Datasource	For informational purposes. The query that will retrieve the data from the database. To edit, choose to bind to an "SQL Value" on the left side of the screen. The current query will be copied over.
Update to Datasource	If selected, the communication is two-way, and the value of the property will be written down to the database on a change.

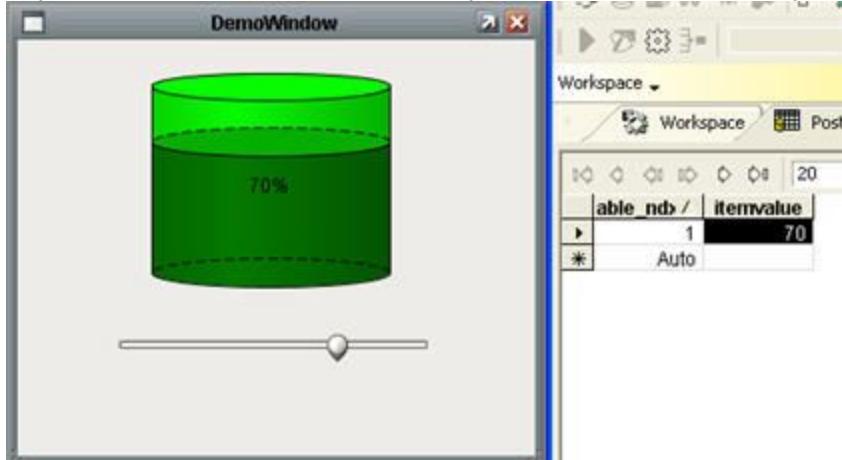
Note: The base rate is set in the property panel for the [Project Workspace](#)

For this example, leave the polling rate at "Relative", and choose "Update to Datasource"

Click "OK" to close the window. Now click "Preview" to turn the workspace into the live program. We must also enable data communication with the gateway, so click the "Gateway Read-Write" button as well (). Now your slider value will adjust to show the value in the database. If you've followed from the example in property binding, the tank's value will also change. In a database frontend you can view the values side by side:



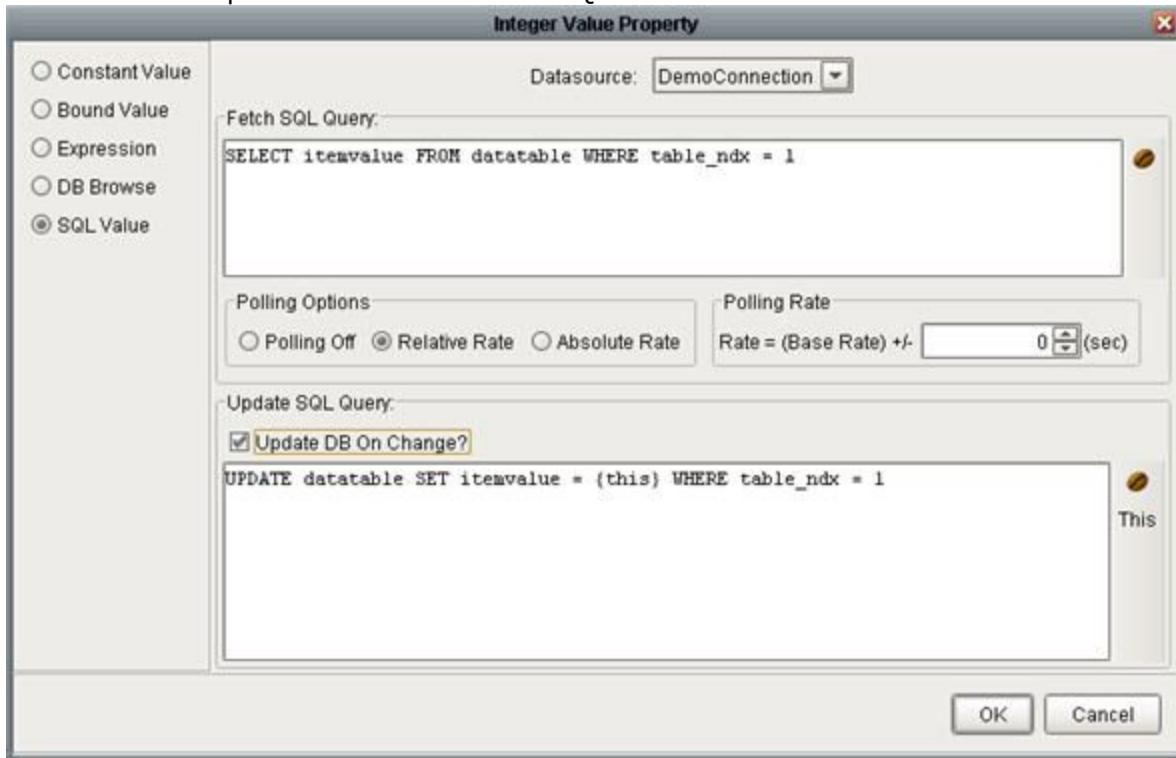
If you move the slider and refresh your database browser, you should see that the values match.



Manually Writing the Query

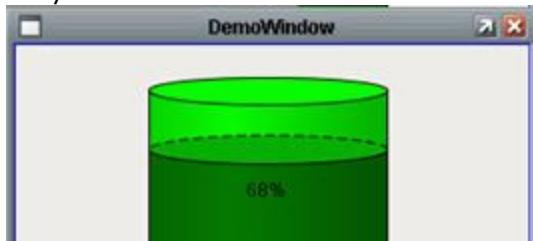
When you select "SQL value" from the databinding dialog, you can manually write the queries used to read and write to the database. The power in this, beside being able to leverage the full SQL language, is that you can also reference **other components' properties** in your queries. That is, you might select the value of a different row, depending on which value in a drop down list is selected. This opens a huge world of possibilities.

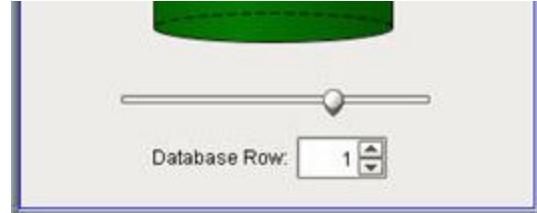
To begin with, continuing from the previous example, click "SQL Value" on the left side of the window. You will notice that the queries created from the SQL Browser are carried over:



This window is fairly straightforward, allowing you a wide range of freedom in crafting your queries.

Now, suppose we had a spinner box on the form as well, allowing the user to choose which row of the database they wanted to reference:





In the binding window, we could tie the row that the queries reference to the number selected in the spinner by using the Property Binding Button:

Property Binding: DemoWindow.Root Container.Cylindrical Tank.value

Datasource: [Project Default]

Insert Property Binding

Binding Types

Property

- Property
- Expression

Database

- DB Browse
- SQL Query

No Binding

Fetch SQL Query:

```
SELECT itemValue  
FROM datatable  
WHERE table_ndx =
```

Use Fallback Value

Polling Options

- Polling Off
- Relative Rate
-

Update SQL Query:

Update DB On Change?

```
UPDATE datatable  
SET itemvalue = {this}  
WHERE table_ndx =
```

Choose Property

Spinner Value

DemoWindow
└ Root Container
 └ Cylindrical Tank
 └ Spinner
 └ Value (int)
 └ Minimum Value (int)
 └ Maximum Value (int)

Root Container.Slider.value

OK Cancel

OK Cancel

This

After doing the same for the update, the screen looks like:

Integer Value Property

Datasource: DemoConnection

Fetch SQL Query:

```
SELECT itemvalue FROM datatable WHERE table_ndx = {Root Container.Spinner.intValue}
```

Polling Options

- Polling Off
- Relative Rate
- Absolute Rate

Polling Rate

Rate = (Base Rate) +/- 0 (sec)

Update SQL Query:

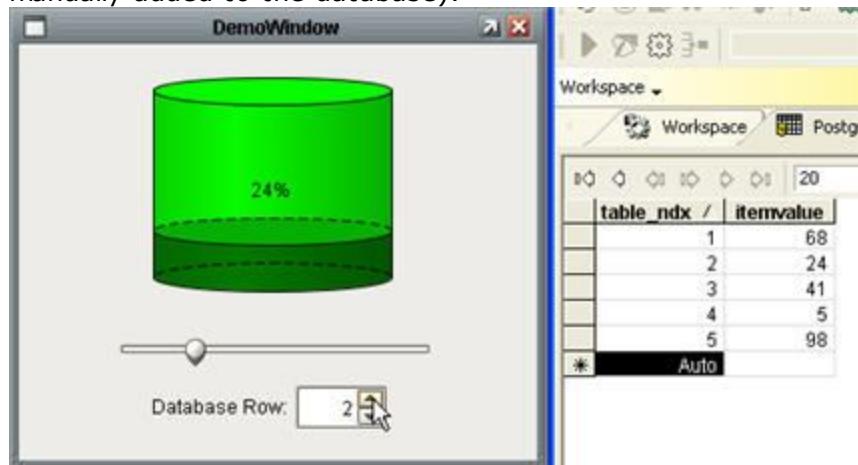
Update DB On Change?

```
UPDATE datatable SET itemvalue = {this} WHERE table_ndx = {Root Container.Spinner.intValue}
```

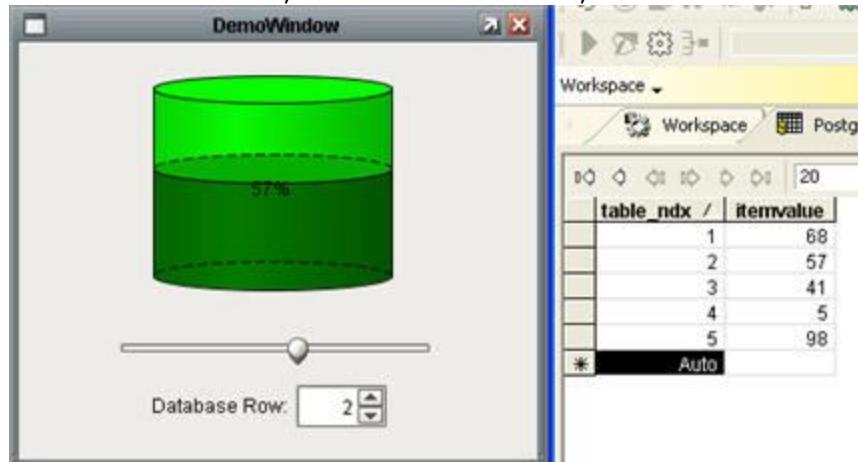
This



Now we can test the project next to our database frontend, and observe the results (note: rows have been manually added to the database):



When the value of the spinner changes, the slider and tank change to reflect the value at that row in the database. From here, if we move the slider, we see the value in the DB change:



This shows an important property: whenever a value changes, any properties that are linked to it are updated, even if they are linked indirectly. This creates a solid connection between all bound components.

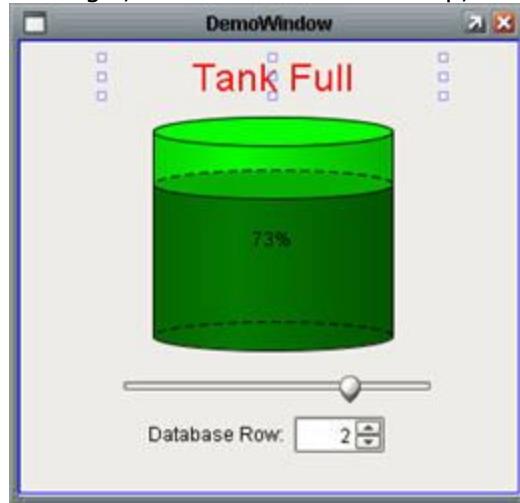
Binding to an Expression

The FactoryPMI Expression Language is a small scripting language that can be used to accomplish a variety of common tasks, such as rounding numbers, choosing between values, converting data types and more. You can reference other properties, and bind properties to the expression result, allowing you to accomplish a wide range of tasks. For a technical reference of all the functions available in the expression language, see [Technical - Expression Language](#).

To bind to an expression, simply choose "Expression" on the left side of the binding window. On the right side, there is a text box for entering the expression, and a Property Binding Button ().

We will continue off of the example in [Binding to SQL](#), with the tank and slider. Now, we will add a notification to the screen showing whether the tank is currently "Full", which we'll define as being more than 85% full. If the tanks value is greater than 85%, the notification will be shown, otherwise it will be hidden.

To begin, add the label to the top, and adjust its properties to make it a proper notification:



Next, it is a simple matter to bind the label's Visible property to an expression, evaluating whether the slider's value is greater than 85. Click the "Bind Property" button for the labels Visible property:

The property editor shows the following settings for the 'Notification' label:

Property	Value
Name	Notification
Enabled	<input checked="" type="checkbox"/> true
Visible	<input checked="" type="checkbox"/> true
Border	[Color Chooser]
Mouseover Text	
Font	Dialog, Plain, 24

Select "Expression" on the left side. The expression we want to use is simply "`SliderValue>85`". Since all statements produce a value in the expression language, this simple boolean equation will produce either a true or false. Conveniently, this is the type of value required by the Visible property.

To reference the slider's value, click the "Bind Property" button, and browse to the value:

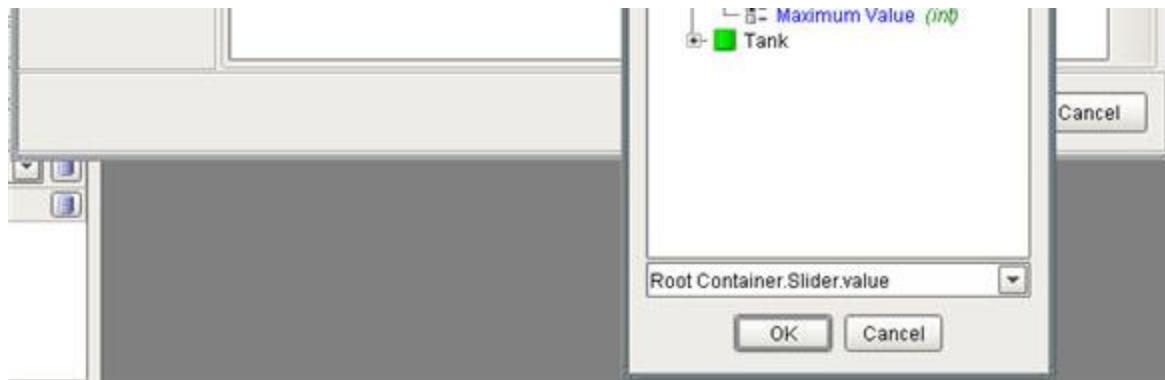
The 'Boolean Value' dialog has the following tabs:

- Constant Value
- Bound Value
- Expression** (selected)
- DB Browse
- SQL Value

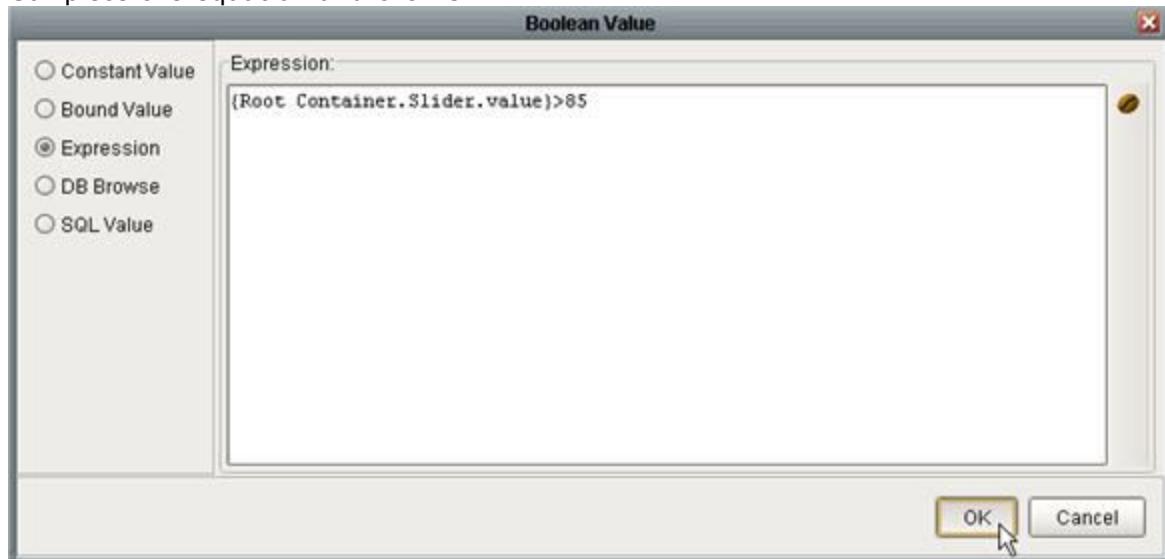
The 'Expression' field contains the expression: `SliderValue>85`.

The 'Choose Property' dialog shows a tree structure of properties for the 'Slider' component:

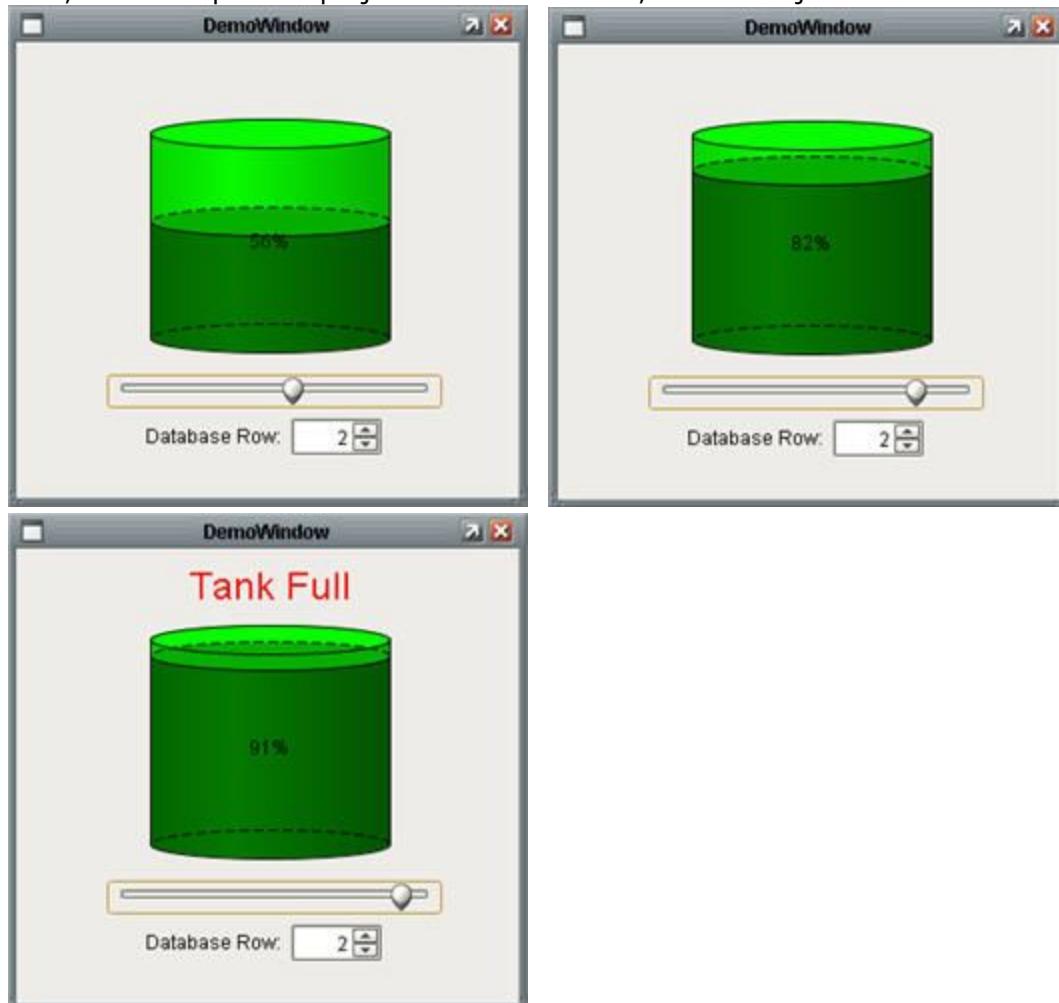
- DemoWindow
- Root Container
 - Foreground Color (Color)
 - Background Color (Color)
- Notification
- Spinner
- Label
- Slider
 - Value (int)** (highlighted)
 - Minimum (int)
 - Maximum (int)



Complete the equation and click OK:



Now, when we put the project in Preview Mode, we can adjust the slider and observe the results:



Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

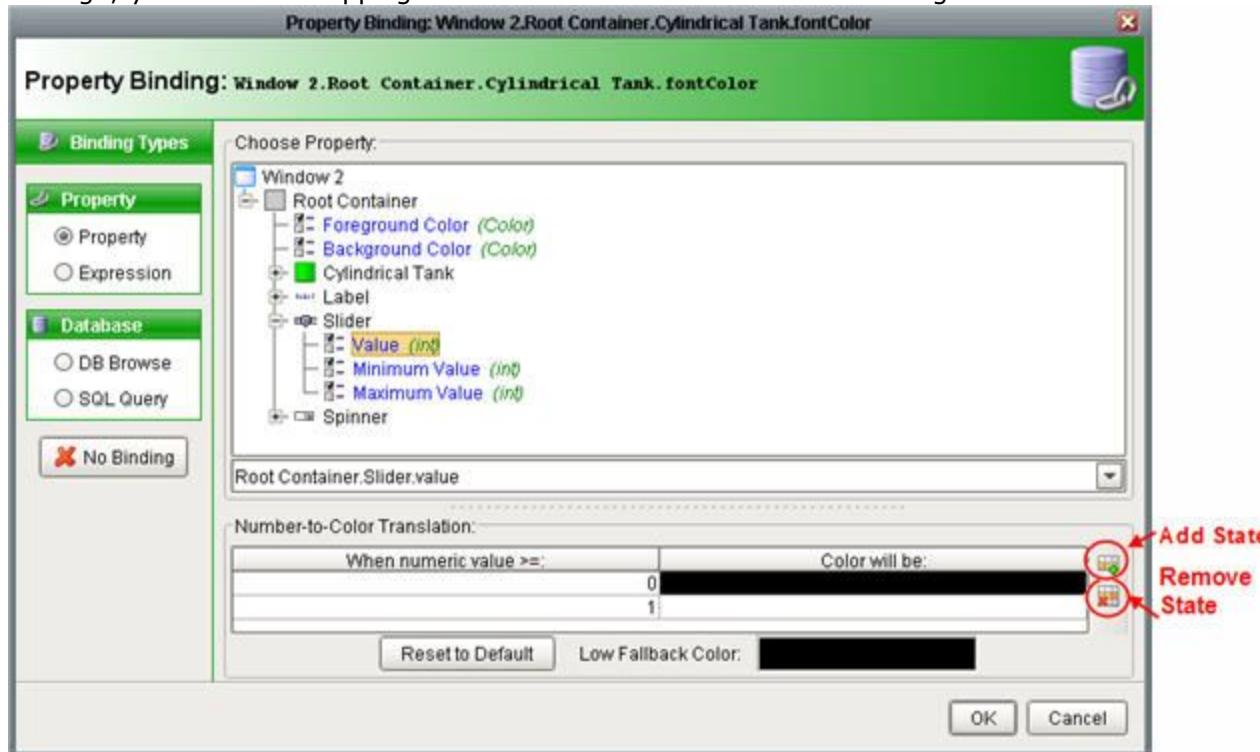


Binding Color Properties

For the most part, binding colors is not much different than any other type. However, there are a number of issues in binding colors that aren't encountered elsewhere.

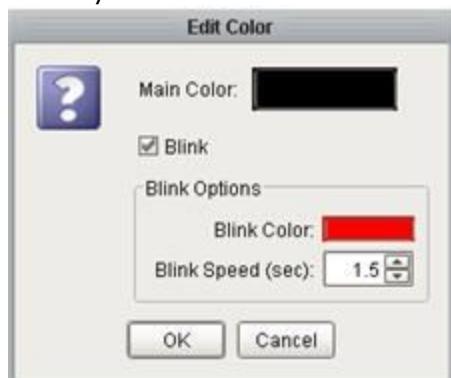
First off, it is not possible to bind a color directly to an SQL query (and accordingly, the DB mapper). FactoryPMI cannot convert the results of a query to a color automatically. However, the expression language provides functions for working with colors, so if a database-centered coloring is required, you can create a [Dynamic Property](#) bound to the SQL query, and reference that in your expression, or bind to it directly on the property binding screen (explained below).

Binding a color to another property works a little differently than normal. Instead of passing the value straight through, you define a mapping between values and colors. The binding screen looks like this:



You'll notice that there is now a value/color association table, where you can create any number of mappings between the value of the selected property, and the corresponding color. There is also a Fallback Color, which is used if there is no state for the current value.

When you double click on the color, a selection screen will pop up:

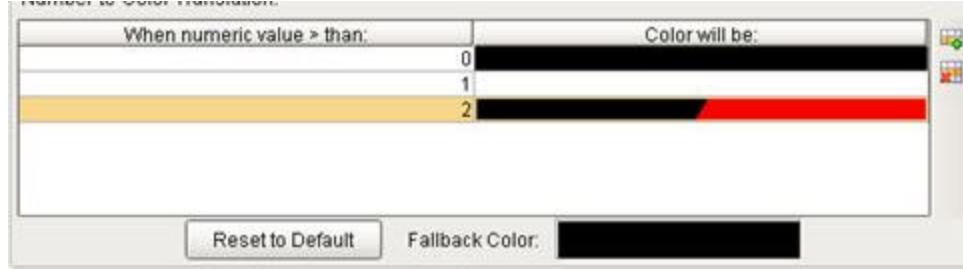


In addition to the main color, you can specify a blink color and rate. When this is set, and the value is in the specified range, the color will alternate between the two at the specified interval. This makes it very easy to create attention-grabbing notifications for important situations.

When a blink color is specified, the association table shows a split color bar:

Tip...

Color databinding is all but obsolete from [Styles](#). Learn how to use styles. It is a better, more generalized approach that can accomplish the same thing and more!



Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Events Overview

Events are an important part of FactoryPMI, providing the ability to respond to user input, or system events, in powerful ways. There are 2 primary classifications of events: component level, and global.

Component level events, as the name suggests, are set for individual components. They can be mouse clicks, key presses, or more internal events such as coming into focus, or having a property change. In order to handle these events, the developer creates "actions", where they can specify a response.

Global events are defined on the application level, and allow you to add code that will execute on startup, shutdown, or on a certain keystroke. Additionally, you can set up timers that will execute at a regular interval.

In both cases, the code that executes in an event is written in Jython. In the case of component events, there are script generators that will automatically generate Jython code to perform common tasks, but it is still a good idea to check out the [Jython Overview](#) section to become a little more familiar with the language.

This section will focus primarily on the user interface aspect of events. For information on Jython and on writing event handlers, see the [Jython Overview](#).

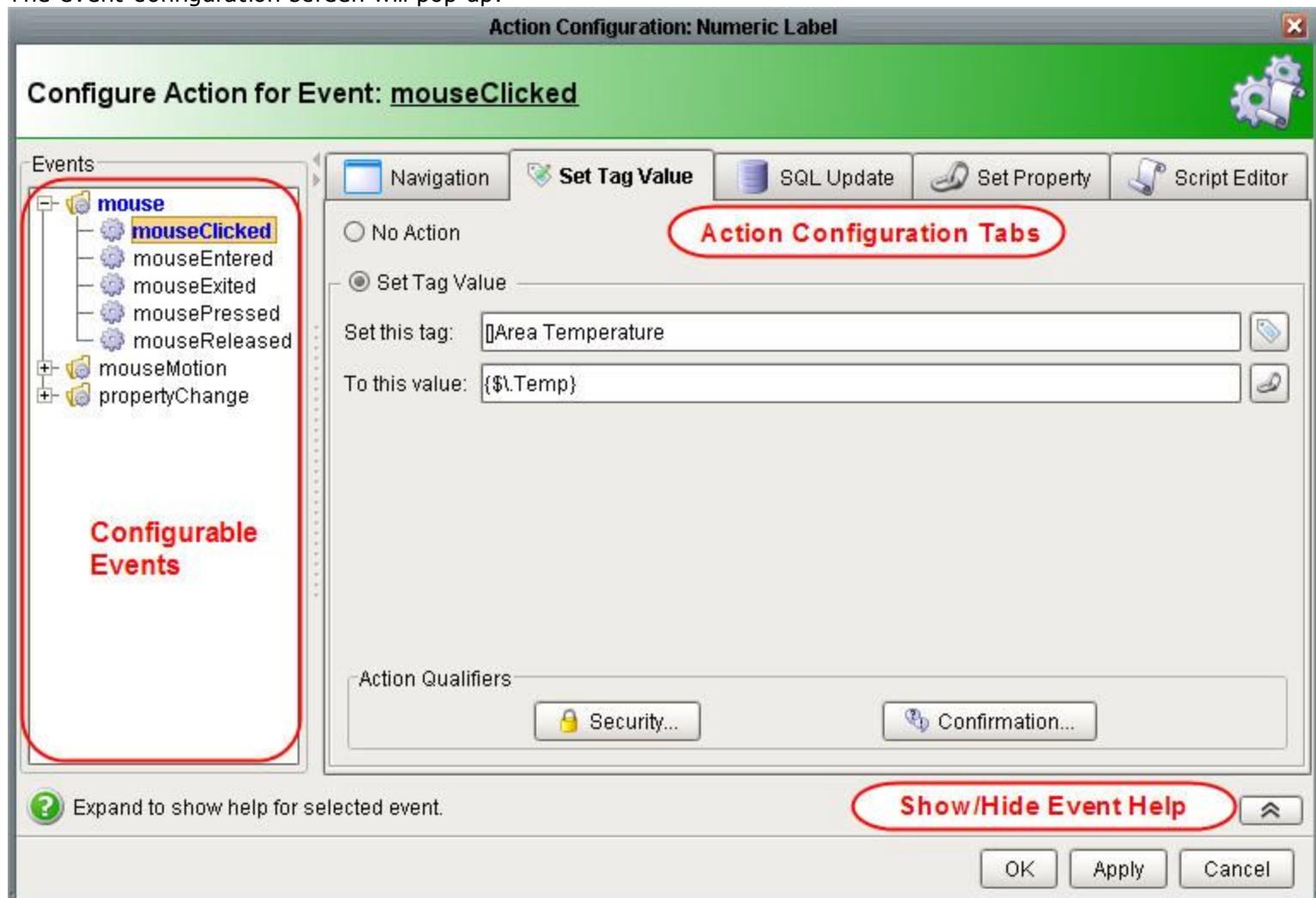
Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Component Events

Every component supports a range of basic events, such as mouse clicks, key presses, visibility changes, etc. To respond to these events, you simply need to configure an **Action**. To do so, right click on any component and select "Configure actions...":



The event configuration screen will pop up:



The left pane shows all of the events that you can configure, split up into categories. To the right are 4 tabs,

which let you configure events in different ways. The first three tabs, Navigation, SQL Update, and Set Property, allow you to quickly accomplish common tasks. In reality, they simply generate Jython code for you, which you can view or edit via the Script Editor tab.

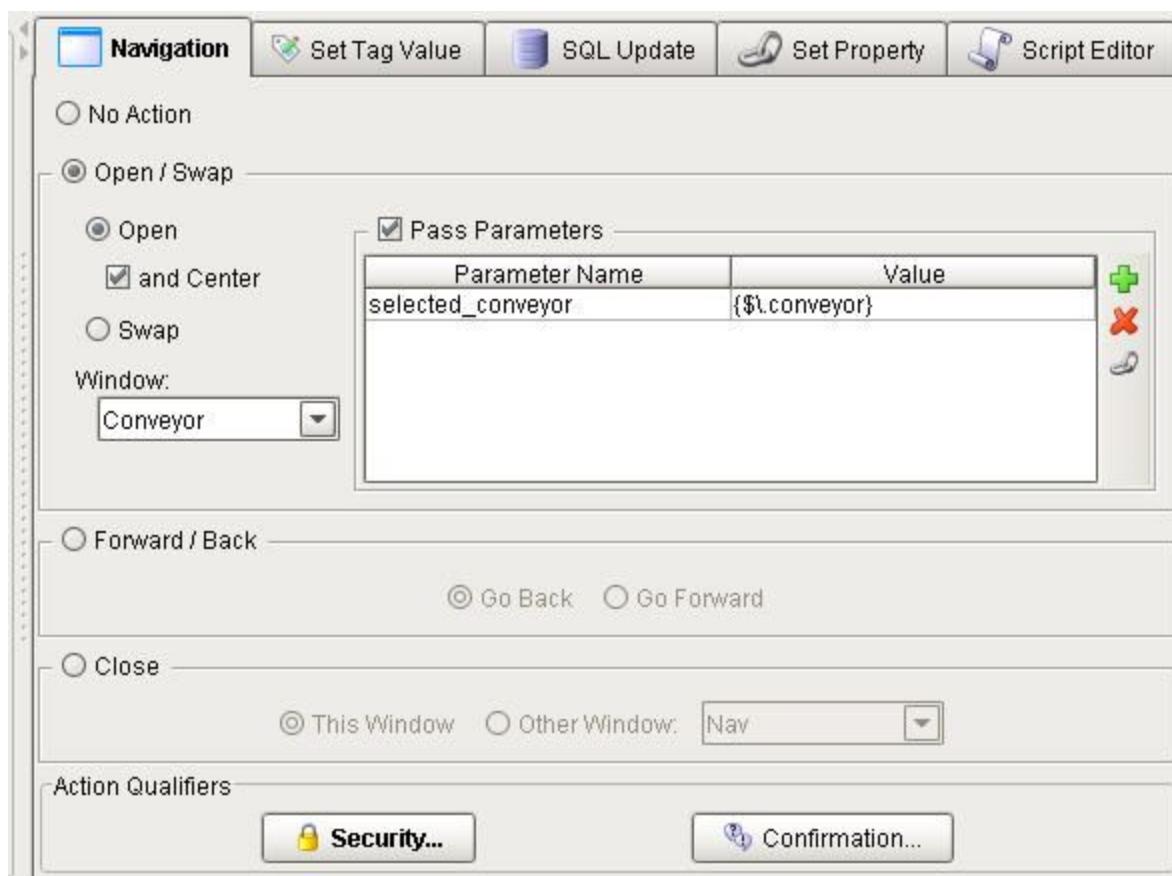
Action Qualifiers

Action qualifiers allow you to set security restrictions or confirmation dialogs on any Jython builder



Navigation Tab

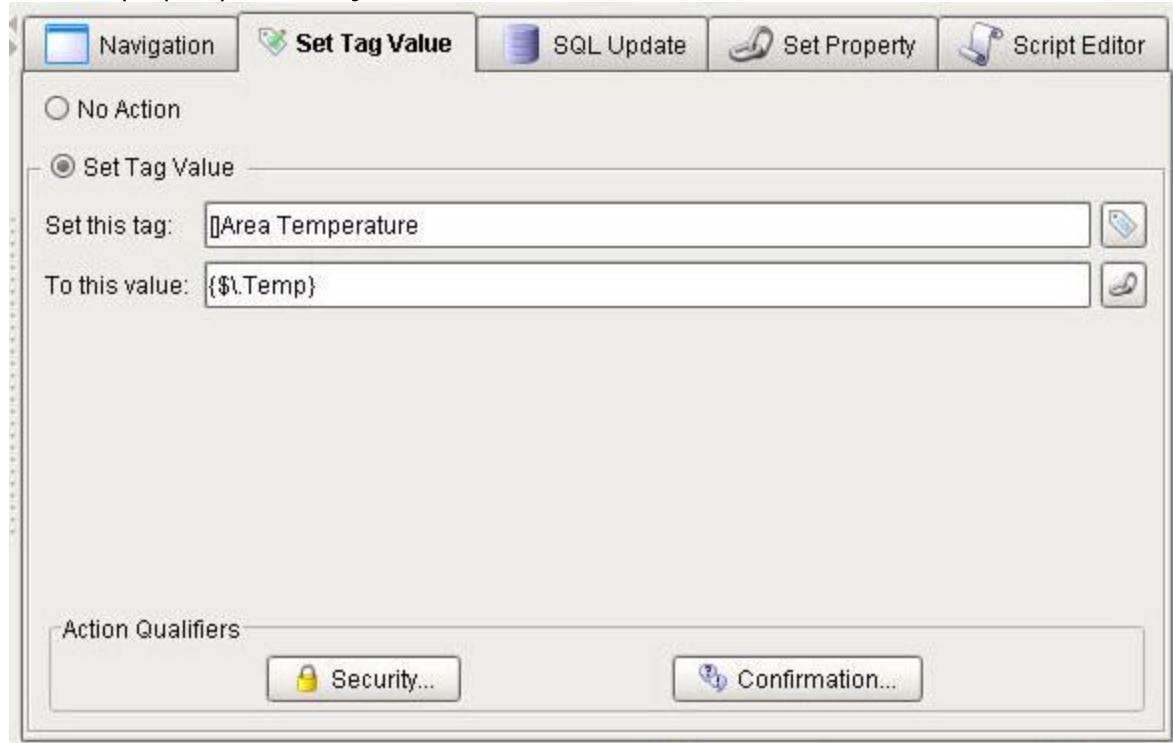
The navigation tab is a shortcut to dealing with window navigation. It builds Jython scripts from [fpmi.nav](#).



Menu Item	Function
Open	Opens the specified window as a popup. Checking " and Center " centers the window after opening it.
Swap	Swaps the "main" window (usually the maximized one) by calling fpmi.nav.swapTo .
Pass Parameters	This list sets Dynamic Properties of your target window to either static or bound values (Object properties from the calling window). When the window is opened/swapped, the dynamic properties on the Root Container of the window will be set based on the Parameter Name and Value .
Forward/Back	Takes the "main" window forward or back if the user has been Swapping the window.
Close	Closes the specified window.

Set Tag Value

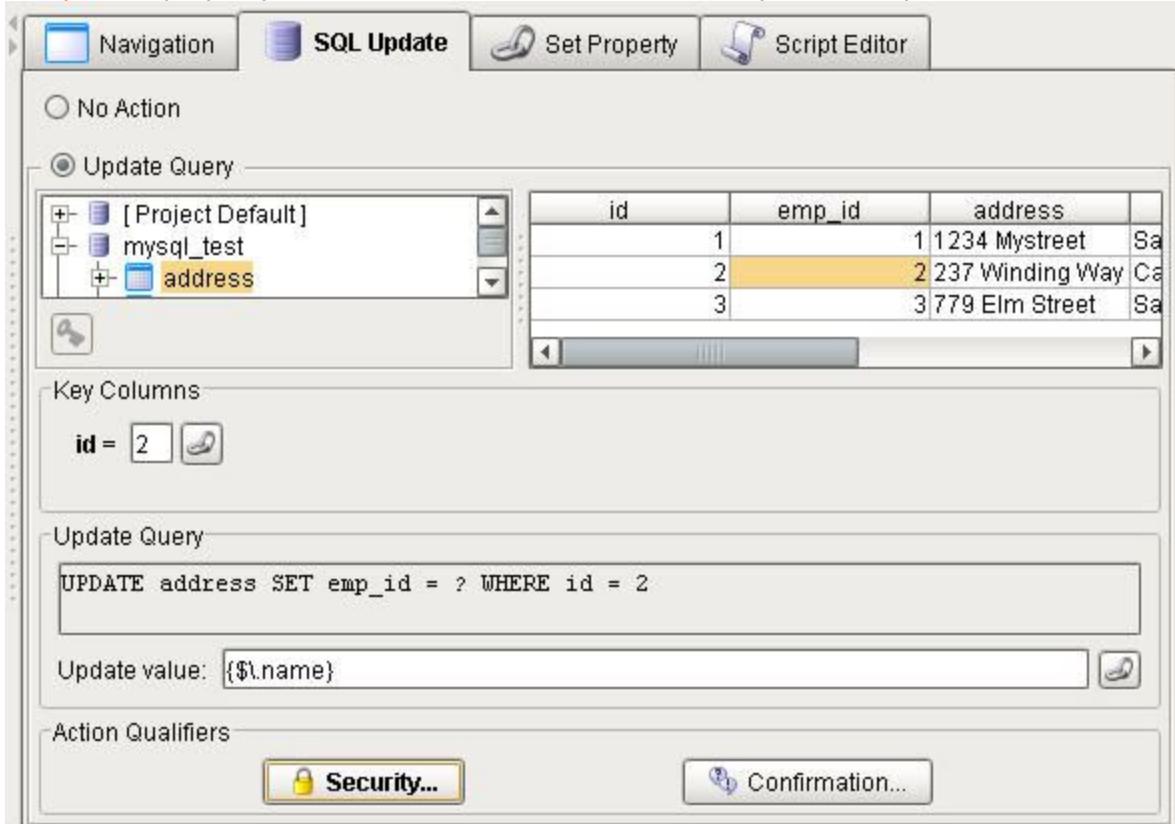
The Set Tag Value Tab allows an Event to set the value of a [SQLTag](#) property to a preset value or the value of another property of an object.



SQL Update Tab

The SQL update tab lets you build an Update query that will be executed on an event. This is useful because you will often want to write a value done instantly on an action, such as writing an Enabled bit on a button press. To do this, simply browse to the location you want to write to in the database, and set the update value. The update value can be a reference to a property value as well.

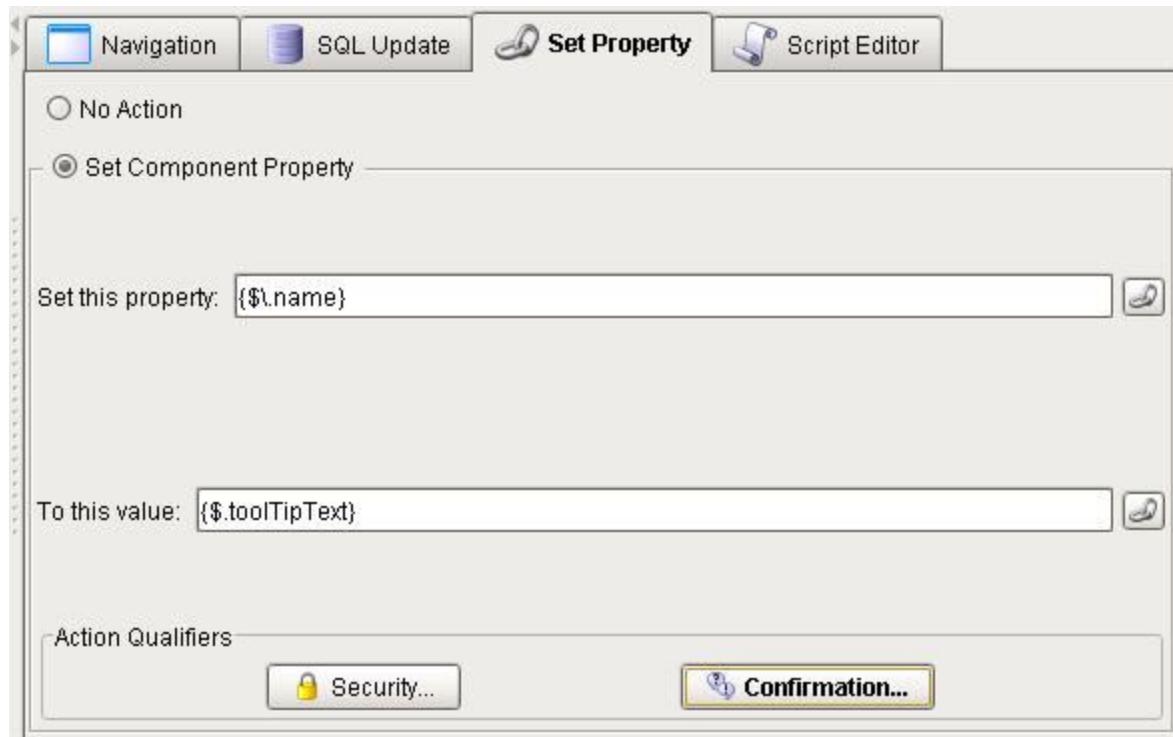
Note: it is often better practice to create a [Dynamic Property](#) that is set to update SQL Datasource on value change, then write to the dynamic property instead of the SQL database directly. This keeps more of a "Black Box"



programming paradigm.

Set Property Tab

The Set Property Tab allows an Event to set any property of an object to a preset value or the value of another property of an object.



Script Editor

The real power in the event system is that you can use the Jython scripting language to accomplish virtually any task. It is a robust, easy to use language, that has many built-in functions and features. Additionally, there is a library of FactoryPMI-specific functions. For a more thorough overview and introduction to the language, see the [Technical - Jython](#) section.

To edit the code for an event, click on the Script editor tab. Remember, both the Navigation tag and the SQL Update tab simply generate code that goes in this window, so you may start there and add on. The script editor tab for the navigation example looks like this:

This screenshot shows the 'Script Editor' tab of a software interface. At the top, there are three tabs: 'Navigation', 'SQL Update', and 'Script Editor' (which is selected). The main area contains a scrollable text editor with the following Jython code:

```
# This script was generated automatically by the navigation
# script builder. You may modify this script, but if you do,
# you will not be able to use the navigation builder to modify
# this script without overwriting your changes.

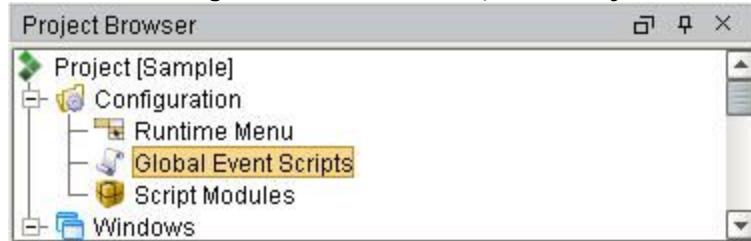
param2 = fpmi.gui.getParentWindow(event).getComponentForPath('Root Container').f

fpmi.gui.openWindow('DemoWindow', {'TankNumber' : 1, 'RealTimeUpdateEnabled' : p}
```

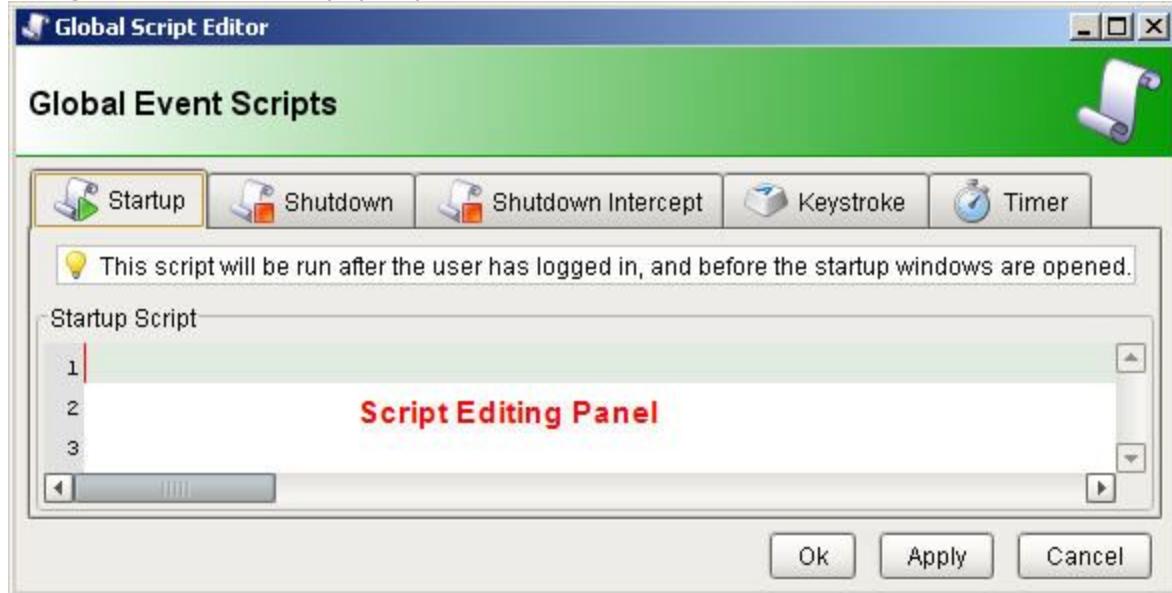
Global Events

Global events are events that apply to your entire project, and are handled in 1 central location. Like component events, you can respond to them by using [Jython](#) code.

To access the global events screen, click **Project - Global Event Scripts** or in the **Project Browser** -> Configuration



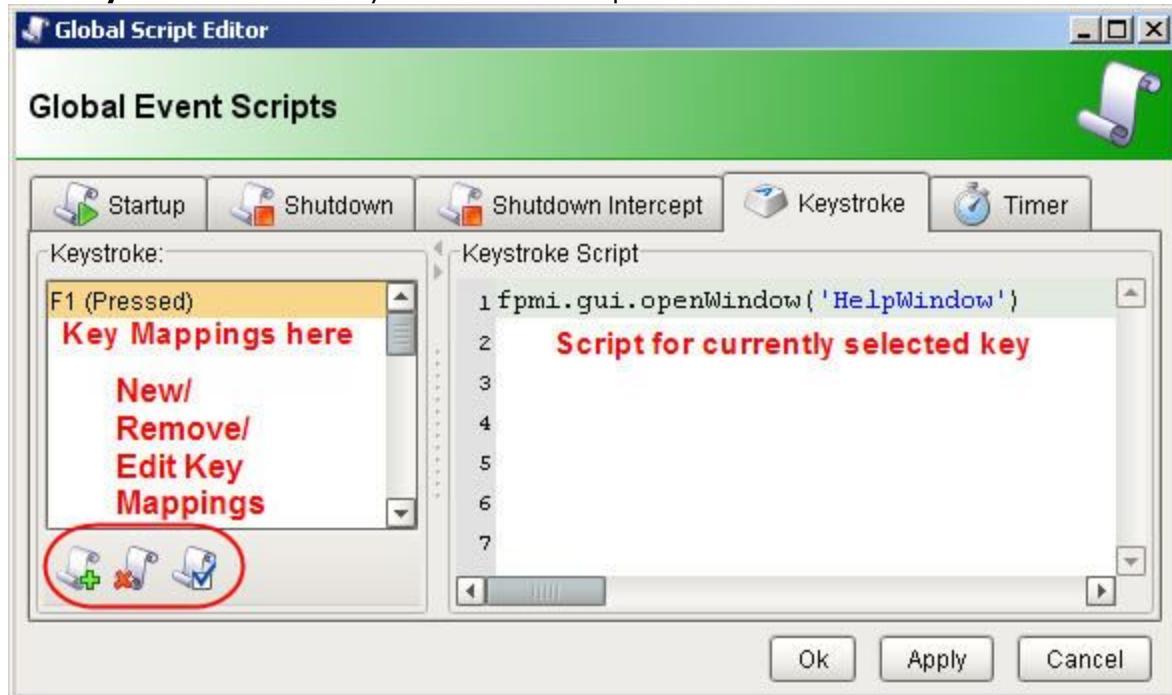
The global event editor pops up, and looks as follows:



The first two tabs, **Startup** and **Shutdown** are simple script editing windows. These scripts will be run any time a client starts up or shuts down.

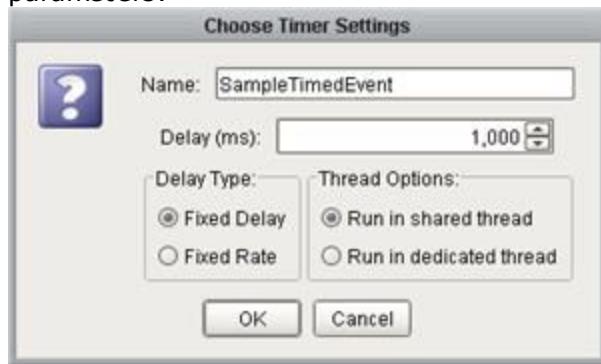
Shutdown Intercept is run when a client requests to close the runtime.

The **Keystroke** tab allows you to write a script that will execute when a certain button combo is pressed:



Keystrokes defined in this tab are global, and will be executed any time they are pressed in the run-time. Therefore, it is not a good idea to map keys such as "enter", "tab", or anything else that is commonly used.

The final tab, **Timers** allows you to set up scripts that will be run at certain intervals. Each script has its own set of parameters:



Delay - The interval of the timer. How often the script actually runs will depend on the *delay type*.

Delay Type - Specifies exactly how the timer is scheduled.

Fixed Delay - The next time is scheduled *from the end of the current event*.

Fixed Interval - The next time is scheduled *from the start of the current event*.

To illustrate, imagine you have a timer with a delay of 5000 (5 seconds), which executes a script that takes 2 seconds to run (a large SQL query, for instance). If the timer were started exactly at time 00:00, the execution times would look like this:

Fixed Delay	Fixed Rate
00:00	00:00
00:07	00:05
00:14	00:10
00:21	00:15
00:28	00:20
00:35	00:25
00:42	00:30

As you can see, fixed delay is (delay+execution time), whereas fixed rate disregards execution time.

Thread Options - Specifies which thread the timer runs in, which has the following implications:

Run in shared thread - The timer is run in a thread shared by multiple timers. Since only 1 timer can execute at a time, if there is already a script running when a second timer expires, the second script will not run immediately. Therefore, execution times are not as precise in the shared pool.

Run in a dedicated thread - The system will run the script in a separate thread. In most cases this will make execution more precisely timed.

The downside is that having many separate threads can significantly hurt system performance. Therefore, you should only use dedicated threads when you have to. In most cases timer execution will be staggered such that there should be no collisions in the shared thread.

Securing Components

It is easy to restrict access/visibility of components and windows to certain user roles. This allows you to easily alter the users capabilities dramatically, such as changing from full control to read-only.

To begin, you must first configure your user roles through the gateway. See [Gateway - Authentication](#) for more information.

Every component has a security setting. By default, everything is set to "inherit from parent", which will allow everyone with a valid log-in to see everything. To access the security settings, select a component, and click on the "Security Settings" button:



The security dialog will pop up, and will be almost entirely disabled, except for the choice to inherit settings, or override them. Choosing to override them will allow you to specify your own settings, and will enable the rest of the screen:



This screen has three main parts: *clearance mode*, *role selection*, and *response*.

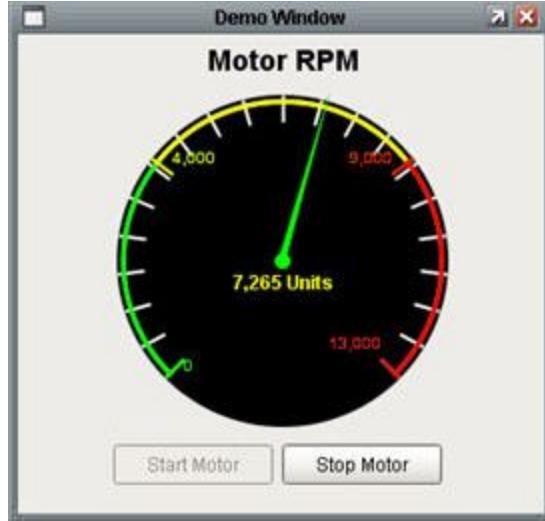
Clearance Mode - You may choose to either explicitly grant or deny access. This choice will depend on the number and types of roles you have set up, as well as what you're trying to accomplish.

Role Selection - Here you select the specific roles that will be given or denied access, depending on the clearance mode setting.

Response - How FactoryPMI will respond to denied roles. This portion is different depending on the type of object selected (component, container, or window). For instance, with components, you can either disable it, or make it invisible.

Example

In this example, we will accomplish a very common task of allowing many people to view the status of a system, but only allow some to actually change it. We start with a simple screen that shows the screen of a motor, along with buttons to start and stop it:



We want to restrict informational viewers from starting and stopping the program by hiding the buttons when they're logged in. To facilitate this, we've placed the buttons in their own container:



By selecting the container and clicking the "Security Settings" button, we can access the security options for it:

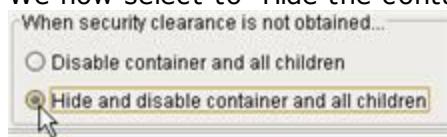


In order to specify our own settings, we select "Override parent's security settings". Now, since we have a role called "View Only", we will leave the *clearance mode* at the default (deny mode), since it is easy to deny access to 1 role than to grant access to 2 (in a real system this difference might be more dramatic).

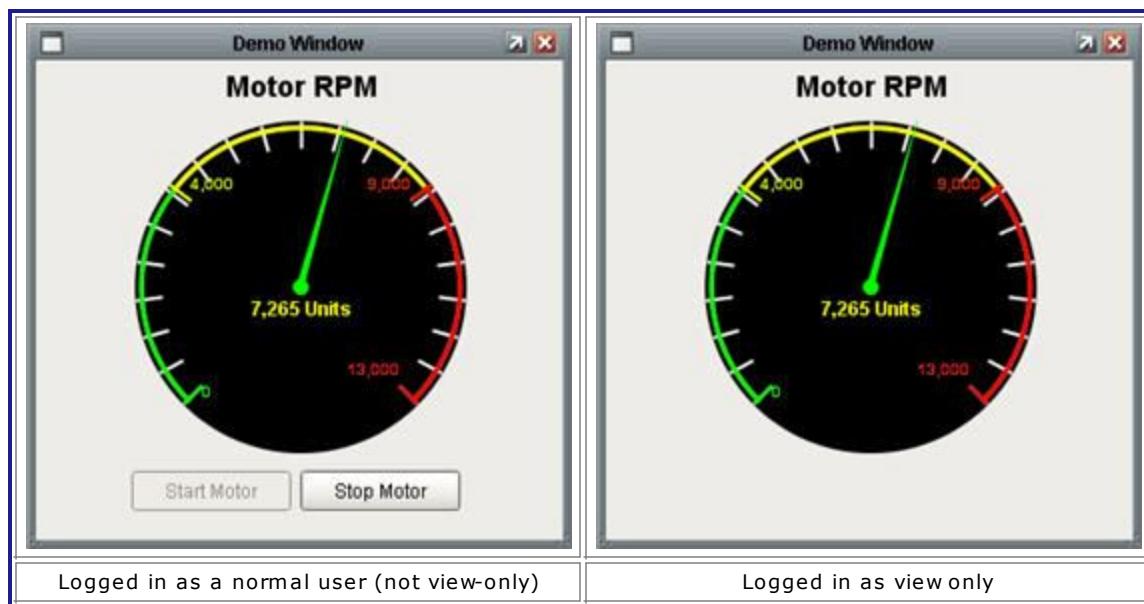
In the role selector, we choose the "ViewOnly" role, and click the large right arrow to add it to the selected list:



We now select to "Hide the container" when someone in the view only list tries to access the window:



And click "OK". We can now log into the system and view the results (or use the [Design-time Security Tool](#)).



Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

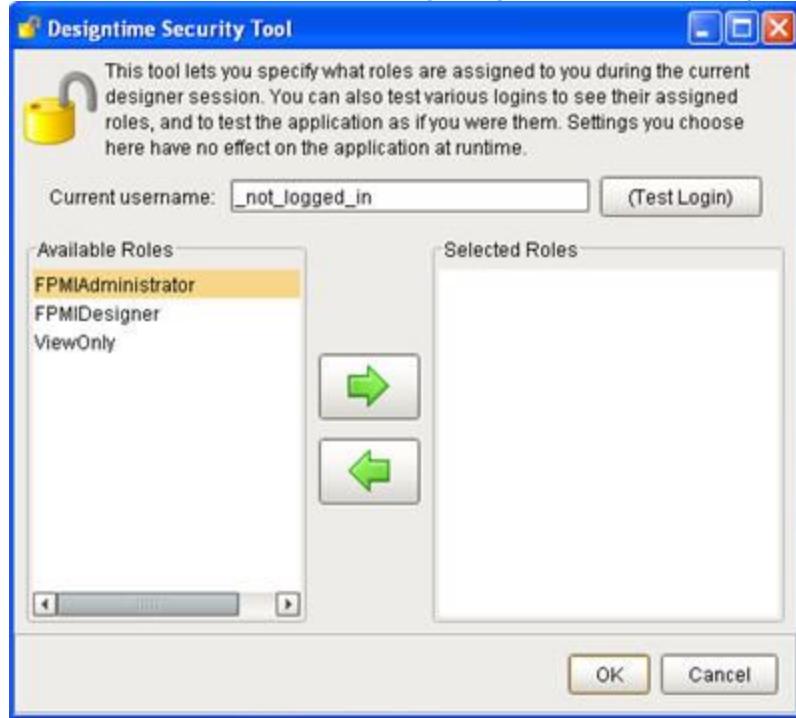
Design-time Security Tool

The design-time security tool lets you test log-ins, and test how the application works under different roles without leaving the designer.

To access the tool, click **Tools - Designtime Security Tool** from the menubar:



The tools looks like the following image. The extra role (ViewOnly) is from the [Component Security Example](#):



You may now assign yourself a specific role, or log in as a different user to select the roles that are applicable to them.

To assign an arbitrary role, select it from the list of available roles, and click the large right arrow.

To log in as a different username, click the (Test Login) button. A window will ask for the username/password. After a successful authentication, the username will be displayed under "current username", and the roles to which that user is assigned will be selected.





Logging In

After successful log-in.

When you click OK, the designer will now respond to the selected roles. Any open windows may need to be closed and re-opened to show the effects. To remove the restrictions, simply open the security tool and remove all of the roles from the selected list with the large left arrow.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Runtime Menu

The menu in the FactoryPMI Runtime is configurable. Each menu item runs a Jython script when pressed, just like a button on a window. By default, each project has a **Command** menu, with items **Logout**, **Lock Screen**, and **Exit**. Want to rename the **Command** menu to **File** for familiarities sake? No problem. Feel free to customize the menu structure as you see fit.

(**Note:** during runtime there will also be the **Windows** and **Help** menus that aren't customizable).

Using the **Menu Editor Tool** , you can customize the default menu structure and add your own menus. These menus are great places for your window navigation because they are always visible, and you can update them all in one place.

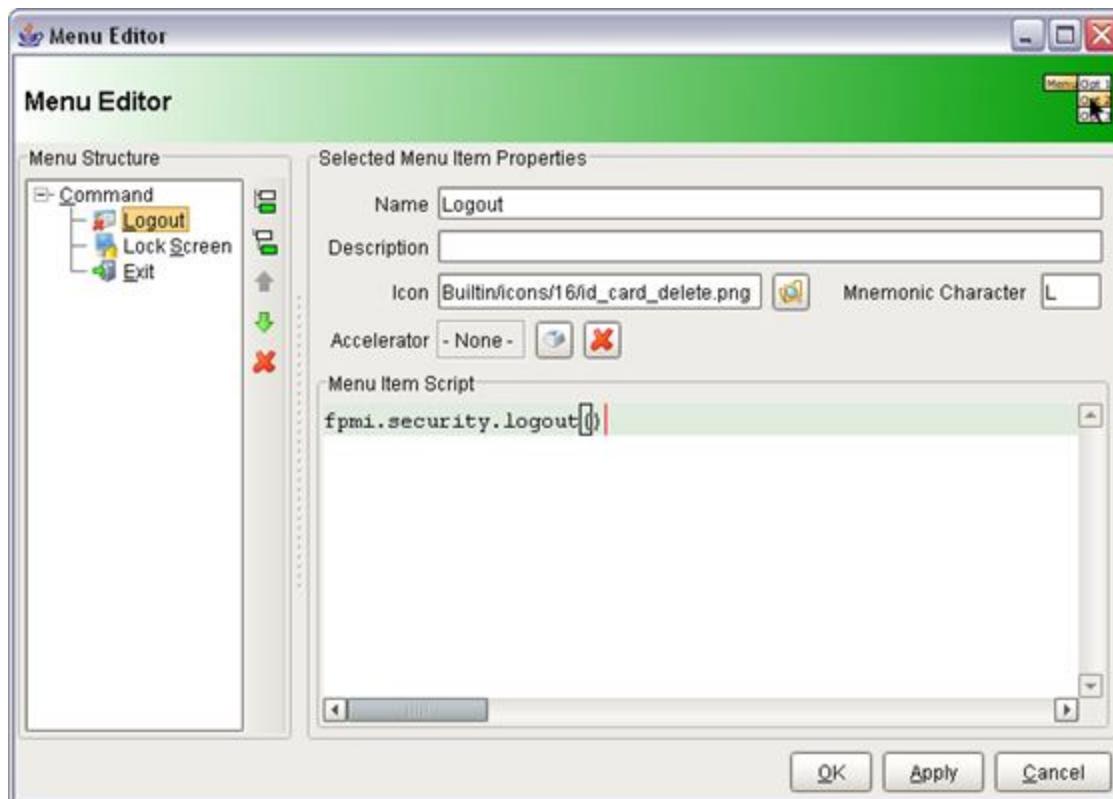


Fig 1. The Menu Editor Tool

TIP. If you are designing an application for a touchscreen, you can make the runtime menu bigger! Just increase the menu font size, which is a [property of the project](#).

By running a FactoryPMI project in [full-screen exclusive mode](#) on a touchscreen monitor, you can truly use FactoryPMI as a traditional HMI system. Industrial panel-mount touchscreen monitors paired with a standard PC can be purchased for significantly less than comparable dedicated HMI panels or Panelviews, and are much more powerful.



To enable your FactoryPMI project to be used on a touchscreen computer, check the [Touchscreen Enabled](#) property for your project in the FactoryPMI Gateway.

- The login screen of the FactoryPMI Runtime will display a touchscreen mode toggle button in the lower left-hand window.
- All input components will display a popup keyboard or numeric entry pad when they are pressed. The **Touchscreen Mode** property dictates the exact response. The default, *Single Click* opens the popup keypad on a single click. Other options include: *Double Click* and *None*.
- You can access the touchscreen mode in Jython with **fpmi.gui.isTouchscreenModeEnabled()**
- To emulate touchscreen mode in the Designer, click [View -> Emulate Touchscreen](#)



How to Use the Component Reference

In this section, you will find a detailed reference to each component available in FactoryPMI. Each component will be described in a common format. The components are broken down by palette category. The properties and scripting functions for [Windows](#) and the entire [Project](#) are also discussed here.

Description

In this section the component's various uses will be described, and perhaps a screenshot or two showing the component in different configurations.

Properties

This section will describe all of the properties unique to this component. Most components share a set of common properties, such as `name`, `background`, etc. For a description of the common properties, see the [Common Properties page](#).

Human-Readable Name

Binding / Scripting Name

Property Type

Mouseover Text

toolTipText

STRING

A short message that will be displayed for a while after the user's mouse has been hovering over this component for a few seconds. Note that HTML formatting is allowed.

Example Property

The following icons indicate special properties:

Expert: This property is not listed with a **Standard** Property Display Filter. **Expert** or **All** must be selected in the [Property Display filter](#) to view this property.

Bindable: Bindable properties are marked by the Bind Icon () in the [Property Panel](#). See [Databinding - Overview](#) for more information.

Customizer

Some components have a 'customizer' in addition to their properties. This is accessed via the customizer button



above the properties table: . Many of the common components have the Dynamic Properties Customizer, which allows you to add [your own properties](#) to the component. The more complex components, such as [charts](#) and [tables](#), have customizers that allow you to configure the component.

Events

All components fire events that can be handled with action scripts, written in Jython. These events are grouped into event sets. For a description of each event set, see the [Action Scripts \(Events\)](#) section of the Jython scripting reference.

Scripting Functions

Many components have functions that can be called in an action script. For instance, you can call the `doClick()` function on a Button component to simulate a user clicking the button. This section will describe the functions available for the component.



Common Properties

Most (not all!) components share a set of common properties. These properties are described below.

Common Properties

Name

name

STRING

The name of the component/window. This name must be unique within its parent. In other words, no sibling component may have the same name.

Enabled

enabled

BOOLEAN

Applicable mainly for input components. When disabled, the component will be visible, but not editable or selectable.

Visible

visible

BOOLEAN

Determines whether the component is shown or not. Containers that are invisible hide all of their child components.

Border

border

BORDER

The border of the component.

Cursor

cursor

CURSOR

The cursor that will be displayed when the user's mouse is hovering over the component.

Mouseover Text

toolTipText

STRING

A short message that will be displayed for a while after the user's mouse has been hovering over this component for a few seconds. Note that HTML formatting is allowed.

Font

font

FONT

The font for the component. It is wise to use one of the *logical fonts* rather than a *physical font*. Logical fonts are fonts that don't actually exist on the computer, but rather are guaranteed to be system-independent. For example, if you use the font Arial, and a user loads your application on a computer that Arial isn't installed on, things won't look right. The 5 logical fonts are:

- Serif
- SansSerif
- Monospaced
- Dialog
- DialogInput

Foreground Color

foreground

COLOR

The color of the foreground of the component. Usually this is the text color.

Background Color

background

COLOR

The color of the background of the component. Note that some components only show their background if the `Opaque` property is true.

Opaque

opaque

BOOLEAN

Determines if the component is opaque. Components such as Labels and Containers use this property to determine whether or not to fill in their background.

Maximum Size

maximumSize

DIMENSION



The maximum size of this component. Currently only used for windows.

Minimum Size

minimumSize

DIMENSION



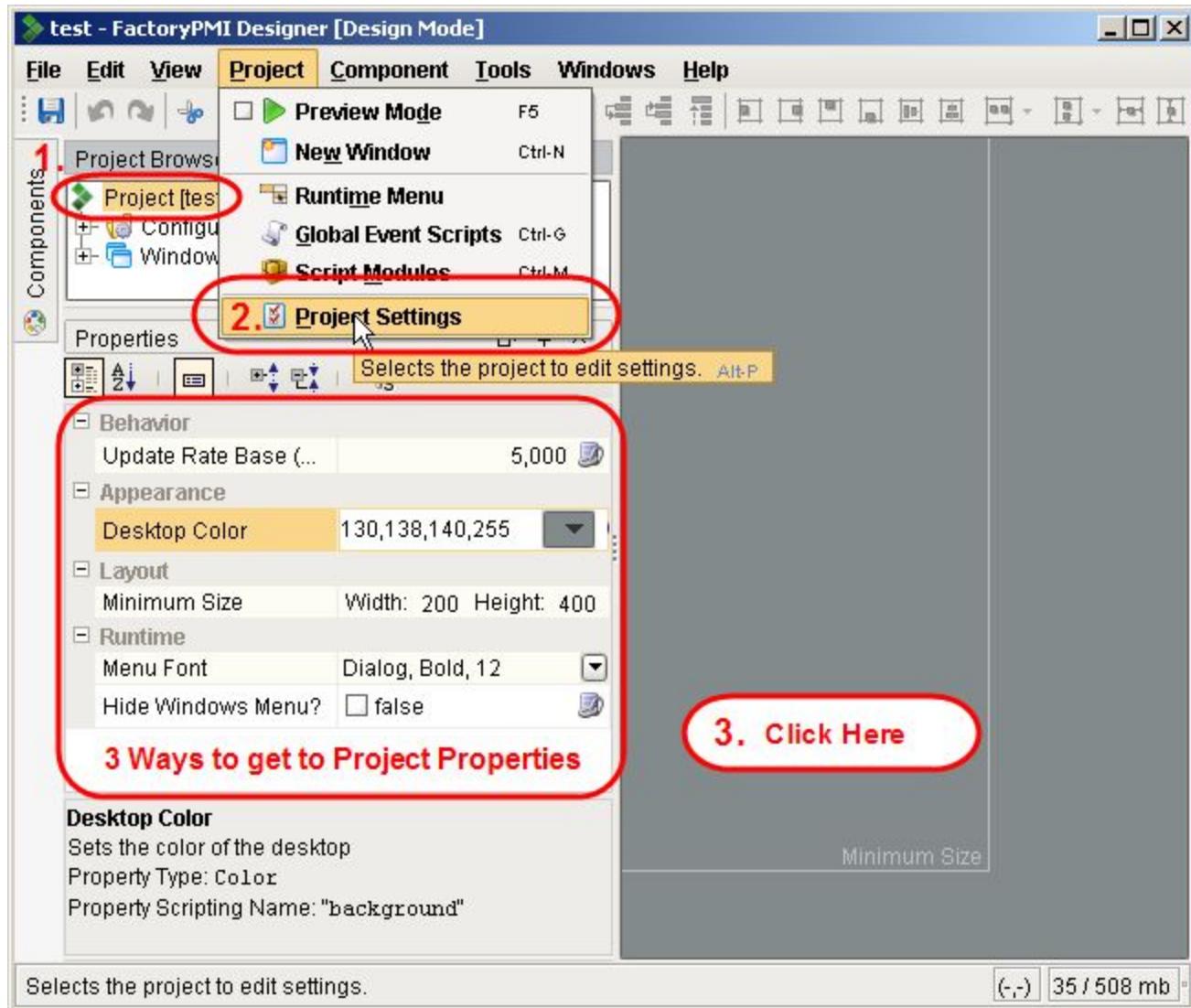
The minimum size of this component. Currently only used for windows.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Project Properties

Description

The project has a few properties that you can edit inside the designer, like component properties. Most properties are edited in the [FactoryPMI Gateway web interface](#). To see the properties that you edit inside the designer, just click on your workspace background.



Properties

Desktop Color	background	COLOR
---------------	------------	-------

The color of the desktop (the area where there are no windows showing, if any).

Menu Font	menuFont	FONT
-----------	----------	------

The font that the menu will use, in the runtime application. Making this bigger will make the menu bigger too, which can be useful for touchscreen applications.

Update Base Rate (ms)	updateBase	INT
-----------------------	------------	-----

The polling base rate for the entire project. This is the "base rate" in all database bindings.

Minimum Size

minSize

DIMENSION

Defines the smallest size that an application is allowed to be. If the enclosing window or Designer work area becomes smaller than this size, the application will scroll. This can be very useful, especially when working with projects in the Designer on a small screen, like a laptop screen.

Fetch-Thread Count

threadCount

INT



EXPERT ONLY. Controls the number of threads that are allocated for fetching SQL SELECT statements used in bindings. Only takes effect after a reload of the application.

Update-Thread Count

updateThreadCount

INT



EXPERT ONLY. Controls the number of threads that are allocated for executing SQL UPDATE statements used in bindings. Only takes effect after a reload of the application.

Customizer

None

Events

None.

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Window Properties

Icon in toolbar:

Description

The window is the top-level unit of organization in a FactoryPMI project. All components are placed inside windows. You can have multiple windows open at a time. Please see the [Windows](#) section of the Designer Concepts portion of the manual for more about windows.



Properties

Title	title	STRING
-------	-------	--------

The title is displayed in the Window's title bar.

Dock Position	dockPosition	INT
---------------	--------------	-----

Controls how the window behaves with regards to docking. Most windows are Floating, but they can also be docked to one of the sides. A window that is docked will not be obscured by other windows, and change the effective size of the application workspace, including for maximized floating windows.

Possible values are:

- **Floating (0)**. The window can be moved by the user, maximized or not, and resizable in any direction.
- **North (1)**. The window is docked to the north (top) edge of the project. Only the window's bottom edge can be resized.
- **East (2)**. The window is docked to the east (right) edge of the project. Only the window's left edge can be resized.
- **West (3)**. The window is docked to the west (left) edge of the project. Only the window's right edge can be resized.
- **South (4)**. The window is docked to the south (bottom) edge of the project. Only the window's top edge can be resized.

Closable	closable	BOOLEAN
----------	----------	---------

Determines whether or not the controls to close the window are visible. The window can always be closed via the `fpmi.gui.closeWindow()` and `fpmi.gui.closeParentWindow()` functions.

Maximizable	maximizable	BOOLEAN
-------------	-------------	---------

Determines whether or not the window can be maximized.

Resizable	resizeable	BOOLEAN
-----------	------------	---------

Controls whether or not the user can drag the edges of the window to resize it.

Start Maximized	startMaximized	BOOLEAN
-----------------	----------------	---------

If true, this window will start in the maximized state automatically. Note that windows that are opened with the various `swapWindow` functions open in the same state and size as the window they are swapped from, regardless of this property.

Border Display Policy

`borderDisplayPolicy`

INT

Controls when the window's border is shown. The default is to show the border when the window is not maximized. It is common to change this to `Never` on docked windows.

Possible values are:

- **Always (0)**. The window's border is always shown, regardless of the window's state.
- **Never (1)**. The window's border is never shown, regardless of the window's state.
- **When Not Maximized (2)**. The window's border is shown when the window is not maximized, and hidden when it is maximized.

Titlebar Display Policy

`titlebarDisplayPolicy`

INT

Controls when the window's titlebar is shown. The default is to always show the titlebar. It is common to change this to `Never` on docked windows, or to `WHEN NOT MAXIMIZED` to hide the titlebar on maximized windows.

Possible values are:

- **Always (0)**. The window's titlebar is always shown, regardless of the window's state.
- **Never (1)**. The window's titlebar is never shown, regardless of the window's state.
- **When Not Maximized (2)**. The window's titlebar is shown when the window is not maximized, and hidden when it is maximized.

Titlebar Height

`titlebarHeight`

INT

The height, in pixels, of the window's titlebar. It can be useful to bump this up on touchscreen applications so that it is easier to move/close the window with a finger. It can also look nice to make this smaller on certain windows where the titlebar takes up too much screen real estate.

Titlebar Font

`titlebarFont`

FONT

The font of the window's title as displayed in the window's titlebar.

Minimum Size

`minimumSize`

DIMENSION



The Minimum Size restricts the window to only be able to be resized down to that size, but no smaller.

Maximum Size

`maximumSize`

DIMENSION



The Maximum Size restricts the window to only be able to be resized up to that size, but no larger.

Cache Policy

`cachePolicy`

INT



Controls how FactoryPMI caches this window. When a window is closed, it can be 'cached'. Next time the window is opened, if it is still stored in the cache, the window will open nearly instantaneously, bypassing the normal de-serialization time associated with loading a window. The window will be in the same state as when it was *closed*, rather than in the same state it was left in when it was last *saved*.

Why would you want to control this? In most cases you don't, and should leave it set at `Auto`. But, imagine that you had a small popup window, used for data entry. When the user hits `OK`, the data typed in is stored to the database, and the window is closed. Next time you open that window, you don't want the text fields to have the previous data still typed in! Since this is a small window and de-serializing doesn't take much time, you would want to set this window's cache policy to '`Never`', so that its field were always empty when you opened the window.

Possible values are:

- **Auto (0)**. The default caching policy. This mode lets FactoryPMI decide how long to keep the window in the cache depending on system load.
- **Never (1)**. The window is never cached. Every time it is opened, it is de-serialized fresh.
- **Always (2)**. The window is always cached, no matter what. Be careful with this setting, as using it too liberally might cause FactoryPMI to not be able to clear resources (memory) when it needs to.

Layer

layer

INTEGER



The layer property specifies which layer the window will appear in. By default, all windows are in layer zero, which is the bottom layer. A window in layer x will always be shown above any windows in a layer that is less than x, and below any windows in a layer greater than x.

Mostly, this is useful to create floating, always-on-top windows. To do so, just set the layer property of a window that you want to be always-on-top to be greater than zero, and it will always be shown on top of other windows.

Customizer

None

Events

- **internalFrame**
 - internalFrameActivated
 - internalFrameClosed
 - internalFrameClosing
 - internalFrameDeactivated
 - internalFrameOpened
- **focus**
 - focusGained
 - focusLost
- **propertyChange**
 - propertyChange

Scripting Functions

getRootContainer()

Return a reference to the Window's Root Container. Example:

```
# This code would be in the internalFrameActivated event, to give initial focus
# To a certain component
rc = event.source.getRootContainer()
textField = rc.getComponent("My Text Field")
textField.requestFocusInWindow()
```

setMaximum(Maximum)

If Maximum is true (1), the window will be maximized, if it isn't already. If Maximum is false (0), the window will be restored, if it is maximized.

Example:

```
# This code would toggle the maximized state of a window from a button.
win = fpmi.gui.getParentWindow(event)
```

wi n. s e t M a x i m u m(n o t w i n. i s M a x i m u m())

isMaximum()

Returns true (1) if the window is maximized, false (0) otherwise.

setLocation(x, y)

Sets the location of the window. Especially useful for opening popup windows and setting their location relative to another location on the screen. Note that the point (0,0) is the upper left corner of the application.

Example:

```
fpmi.gui.openWindow("PopupWindow")
myWin = fpmi.gui.getWindow("PopupWindow")
myWin.setLocation(200,200)
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Text Field Component

Icon in toolbar: 

Description

The text field is a versatile input component useful for the input of and display of any single-line text. While this component *can* be used for numeric data entry, consider using the [Numeric Text Field](#), which is more suited for this purpose.



This field features a protected mode. When you enable the protected property, the field is not editable even when it receives input focus. The user must double click on the field or press enter in order to edit the field. When they are done (press enter again or leave the field), the field becomes non-editable again.

Properties

Text

text

STRING



The text inside this text field. See the **Defer Updates** field for a description of when this changes as the user is typing.

Editable?

editable

BOOLEAN

If false, the text in this field cannot be changed by the user. Makes this component act similar to the Label component.

Protected Mode

protectedMode

BOOLEAN

When turned on, the field is not editable until the user double clicks in it or presses enter while it has focus.

Defer Updates

deferUpdates

BOOLEAN

Controls how the **Text** property is updated as the user types text into this field. If this property is false, then the **Text** property will be updated immediately with each typed change. If this property is true, then the **Text** property will only be updated when the user hits Enter or leaves the field.

Maximum Characters

maxChars

INT

The text box will be limited to this number of characters. Use -1 for unlimited characters (this is the default).

Styles

styles

DATASET



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [key](#)

- keyPressed
- keyReleased
- keyTyped
- mouse
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- mouseMotion
 - mouseDragged
 - mouseMoved
- propertyChange
 - propertyChange
- focus
 - focusGained
 - focusLost

Scripting Functions

getSelectedText()

Retrieve the portion of the text of this text field that is currently selected.

The following functions are common to all components that allow dynamic properties:

getPropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

setPropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15
event.source.setPropertyValue("MyProperty", 15)
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

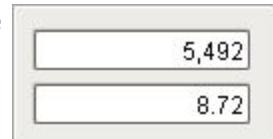


Numeric Text Field Component

Icon in toolbar:

Description

The Numeric Text Field is similar to the standard Text Field, except that it is specialized for use with numbers. Instead of a **Text** property, it has four numeric value properties. Which one you use depends on the mode of the text box.



Like the standard Text Field, this text field can operate in *protected mode*. When you enable the protected property, the field is not editable even when it receives input focus. The user must double click on the field or press enter in order to edit the field. When they are done (press enter again or leave the field), the field becomes non-editable again.

Properties

Number Type

Mode

INT

Determines what mode this numeric text field is in. The different modes determine the underlying numeric type of this field, which determines which value property to use. See the [Data Types](#) page for an explanation of types.

Possible values are:

- Integer (0)
- Double (3)
- Long (1)
- Float (2)

Protected Mode

protectedMode

BOOLEAN

When turned on, the field is not editable until the user double clicks in it or presses enter while it has focus.

Use Bounds?

useBounds

BOOLEAN

If true, the **Maximum** and **Minimum** properties will be used to enforce upper and lower bounds on the number.

Maximum

maximum

DOUBLE

If **Use Bounds** is true, this number is the maximum value of the value of this numeric text field.

Minimum

minimum

DOUBLE

If **Use Bounds** is true, this number is the minimum value of the value of this numeric text field.

Editable?

editable

BOOLEAN

If false, this numeric field becomes un-editable.

Decimal Format

decimalFormat

STRING

This formatting string is used to format the number as it is displayed in the text field. See the [numberFormat expression function](#) for a description of the syntax of this string.

Integer Value

intValue

INT



The value of this component if **Number Type** is set to Integer

Double Value	doubleValue	DOUBLE
---------------------	-------------	--------

The value of this component if **Number Type** is set to Double



Long Value	longValue	LONG
-------------------	-----------	------

The value of this component if **Number Type** is set to Long



Float Value	floatValue	FLOAT
--------------------	------------	-------

The value of this component if **Number Type** is set to Float



Reject Updates During Edit	rejectUpdatesDuringEdit	BOOLEAN
-----------------------------------	-------------------------	---------

If true, then while this numeric text field has the input focus, any updates to its value (from a database binding, for example) will be rejected. The intent of this feature is to allow a user to type in a value, and not have their typing overwritten by a polling SQL query. This is true by default.



Styles	styles	DATASET
---------------	--------	---------



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [key](#)

- keyPressed
- keyReleased
- keyTyped

- [mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased

- [mouseMotion](#)

- mouseDragged
- mouseMoved

- [propertyChange](#)

- propertyChange

- [focus](#)

- focusGained
- focusLost

Scripting Functions

The following functions are common to all components that allow dynamic properties:

getPropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

setPropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15
event.source.setPropertyValue("MyProperty", 15)
```

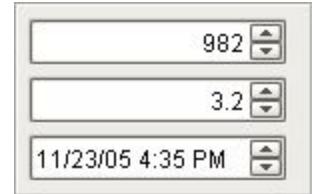


Spinner Component

Icon in toolbar:

Description

The spinner component represents a value that is part of a series of values, such as numbers and dates. It allows you to not only edit the value directly, but to 'spin' the value up or down, using the up and down buttons that are part of the component.



Properties

Editor Background

editorColor

COLOR

The background color of the editing portion of this component.

Spinner Mode

spinnerMode

INT

The mode of the spinner determines what kind of value the component represents.

Possible values are:

- Integer (0)
- Double (1)
- Date (2)

Numeric Minimum

minValue

DOUBLE

The minimum value allowed, if **Spinner Mode** is Integer or Double.

Numeric Maximum

maxValue

DOUBLE

The maximum value allowed, if **Spinner Mode** is Integer or Double.

Date Step Size

dateStepSize

INT

The step size if the **Spinner Mode** is Date. The step size is the amount the value changes up or down when the up or down buttons are pushed. Note that for dates, if the user highlights a portion of the date, then that portion will be changed, rather than the unit chosen here.

Possible values are:

- Year (1)
- Month (2)
- Week (3)
- Day (5)
- Hour (10)
- Minute (12)
- Second (13)
- Millisecond (14)

Numeric Step Size

stepSize

DOUBLE

The step size if the **Spinner Mode** is Integer or Double. The step size is the amount the value changes up or down when the up or down buttons are pushed.



Value (Integer)

intValue

INT



The current value, if **Spinner Mode** is Integer.

Value (Double)

doubleValue

DOUBLE



The current value, if **Spinner Mode** is Double.

Value (Date)

dateValue

DATE



The current value, if **Spinner Mode** is Date.

Styles

styles

DATASET



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [propertyChange](#)
 - propertyChange

Scripting Functions

The following functions are common to all components that allow dynamic properties:

get PropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters**PropName**

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

set PropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters**PropName**

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15
event.source.setPropertyValue("MyProperty", 15)
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Formatted Text Field Component

Icon in toolbar:

Description

This specialized text field allows for 2 modes of operation:

`support@inductiveautomation.com`
`(800) 266-7798`

Formatted Mask

In this mode, input is automatically formatted and restricted based on a *format mask*. For example, a format mask like: `(###) ####-####` will allow the entry of a 9-digit phone number. The formatting characters are automatically inserted if the user does not type them in. Any other characters are restricted.

Regular Expression

In this mode, input is validated against a *regular expression*. (A regular expression is a special string that defines a set of allowed strings. See http://en.wikipedia.org/wiki/Regular_expression). Any input that matches the given regular expression is allowed, and input that doesn't match, is restricted.

Properties

Text

`text`

STRING

The currently displayed text in the field. May or may not match the mask/regular expression, depending on the value of **Allows Invalid Text**. See **Committed Value**.

Validation Mode

`validationMode`

INT

The mode of this component. See the description above.

Possible values are:

- Regular Expression (1)
- Formatted Mask (2)

Formatted Mask Pattern

`formattedMaskPattern`

STRING

The format mask to use, if the **Validation Mode** is Formatted Mask. Format strings make use of the following special characters. All other characters will appear in the committed text.

Character	Description
#	Any valid number, Such as 0-9.
'	Escape character, used to escape any of the special formatting characters.
U	Any letter. All lowercase letters will be mapped to upper case automatically.
L	Any letter. All upper case letters will be mapped to lower case automatically.
A	Any letter or number.
?	Any letter, case is preserved.
*	Anything.
H	Any hex character (0-9, a-f or A-F).

Examples:

Format Mask	Description	Example
(###) ###-####	Phone numbers with area code	(800) 266-7798
HHHHHH	Six hex digits, like an HTML color.	FF0000
0xHHHH	Four hex digits, with a '0x' prepended to it	0x001A
#UUU###	A California license plate number.	4ABC123

Reg Ex Pattern

validationPattern

STRING

The regular expression to us., if the **Validation Mode** is Regular Expression. Some examples of regular expressions:

Regular Expression	Matches
\p{Upper}\p{Lower}*, \p{Upper}\p{Lower}*	A name, formatted such as Smith, John
\d{3}-\d{2}-\d{4}	A social security number, formatted such as 123-45-6789
\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}	A network IP address, like 67.82.120.116
^[a-f0-9A-F]{6}\$	A six-digit hexadecimal number.

As you can tell, regular expressions are quite ugly and non-intuitive. Luckily, good regular expressions for pretty much any kind of input you can dream of are easily found on the Internet. Try typing 'email address regular expression' into Google.

Committed Value

committedValue

STRING



The committed value is the value of the text when it was last committed successfully. A successful commit occurs when the text is 'legal' according to the format mask or regular expression. When commits occur depends on the **Commit While Typing** and **Focus Lost Behavior** properties.

Allows Invalid Text

allowsInvalid

BOOLEAN

If true, then the text box allows the user to type characters that are rejected by the format mask or regular expression. If false, then character input is strictly allowed to only legal characters.

Overwrites Text

overwriteMode

BOOLEAN

If true, then this field acts as if the computer was in 'overwrite' editing mode, rather than 'insert' mode.

Commit While Typing

commitsOnValidEdit

BOOLEAN

If this is true, the field will commit its value every time a valid edit is made. If false, commits will only occur when the user hits enter or leaves the field. See also the **Focus Lost Behavior** property.

Focus Lost Behavior

focusLostBehavior

INT

The value of this property determines what happens when the focus leaves this field. Most often, you want to leave it at 'Commit or Revert', which means that the field will commit the current edit if it is valid, otherwise it will revert the text to the committed text.

Possible values are:

- **Revert (2)**. Revert the display to match the committed value, possibly losing the current edit.

- **Commit or Revert (1)**. Similar to COMMIT, but if the value isn't legal, behave like REVERT.
- **Commit (0)**. Commits the current value. If the value being edited isn't considered a legal value by the format mask / regular expression, then the value will not change, and then edited value will persist.
- **Persist (3)**. Do nothing, leave the current edit displayed and don't commit the value.

Styles

styles

DATASET



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [key](#)
 - keyPressed
 - keyReleased
 - keyTyped
- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange
- [focus](#)
 - focusGained
 - focusLost

Scripting Functions

None.



Password Field Component

Icon in toolbar:

Description

The password field is a simple text field that is used for passwords. The difference between it and a normal text field is that it doesn't directly display what is being typed in.



Properties

Text

text

STRING



The text that this password field represents. Note that this is the real text, not the string of echo characters.

Echo Character

echoCharacter

STRING

The character to display instead of the real characters.

Styles

styles

DATASET



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

[key](#)

- keyPressed
- keyReleased
- keyTyped

[mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased

[mouseMotion](#)

- mouseDragged
- mouseMoved

[propertyChange](#)

- propertyChange

- [focus](#)

- focusGained
- focusLost

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Text Area Component

Icon in toolbar: 

Description

The text area is a multi-line text input component, suitable for large amounts of text.

This is a large text area used for multi-line text input

Properties

Text	text	STRING	
-------------	------	--------	---

The text that this text area contains.

Editable	editable	BOOLEAN	
-----------------	----------	---------	--

Controls whether or not this text area is editable.

Rows	rows	INT	
-------------	------	-----	--

The number of rows long that this text area is. This is a hint that is used for the scrollbars.

Columns	columns	INT	
----------------	---------	-----	--

The number of columns wide that this text area is. This is a hint that is used for the scrollbars.

Line Wrap	lineWrap	BOOLEAN	
------------------	----------	---------	--

If true, the text area won't scroll horizontally, rather, it will wrap lines to fit.

Styles	styles	DATASET	 
---------------	--------	---------	---

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [key](#)
 - keyPressed
 - keyReleased
 - keyTyped
- [mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange
- [focus](#)
 - focusGained
 - focusLost

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

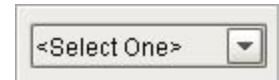


Dropdown Component

Icon in toolbar:

Description

The dropdown component displays a list of choices in a limited amount of space. The current selection is shown, and the choices are only presented when the user clicks on the dropdown button. The choices that are shown depend on the **Data** property. This is a dataset, which can be typed in manually in the FactoryPMI Designer, or it can be populated dynamically from a database.



It is often the case that you want to display choices to the user that are 'dressed up' versions of the actual choices. For instance, suppose that you are selecting choices for a downtime tracking entry. The choices might be: "Operator Error", "Machine Malfunction", and "Other". But, you really want to map these choices to some numeric code which is how the choice is stored. So, for instance, when the user chooses "Other" you really want to get the number 3. The dropdown component is perfect for such a use. The **Data** property can be set up in one of three fashions, which control how the 'selected values' properties change.

The 3 ways to set up the **Data** dataset are as follows:

Description	Example	Active Properties
Single list of values		The dropdown displays these values, and both the Selected String Value and the Selected Label properties are identical. The Selected Value property is undefined.
List of Integer, String pairs.		The dropdown displays the values in the second (label) column. The Selected Value property reflects the first column of the selection. The Selected Label property reflects the second column. The Selected String Value property is undefined.
List of String, String pairs.		The dropdown displays the values in the second (label) column. The Selected String Value property reflects the first column of the selection. The Selected Label property reflects the second column. The Selected Value property is undefined.

The dropdown component can operate in one of three *Selection Modes*. These modes affect how the dropdown's current selection (defined by the values of its `Selected Value`, `Selected String Value`, and `Selected Label` properties) behave when the selection properties are set to values not present in the choice list, or conversely, when the choice list is set to a new DataSet that doesn't contain the current selection:

- **Strict.** Selected values must always correlate to an option in the list defined by the `Data` property. If an invalid selection is set (via a binding or a script), the selection will be set to the values defined by the `No Selection` properties. If the `Data` property is set to a list that does not contain the current selection, the current selection will be reset to the `No Selection` values.
- **Lenient. (default)** Selected values are independent of the list defined by the `Data` property. This mode is useful to avoid race conditions that can cause problems in **Strict** mode when both the `Data` and the `Selected Value` properties are bound. If the current selection is not present in the `Data` list, the read-only property `Selected`

Index will be -1.

- **Editable.** The same selection rules as defined by **Lenient** mode, except that the dropdown itself becomes editable, allowing a user to type in their own arbitrary value. This value will be set as the dropdown's Selected Label.

Properties

Selection Mode

selectionMode

INT

The selection mode determines the behavior of the dropdown: whether its selected value must strictly be in the underlying set of choices, whether it is flexible, or even user-editable. See the description for more details.

Possible values are:

- Strict (0)
- Lenient (1)
- Editable (2)

Data

data

DATASET

This property determines what the dropdown component's choices are. See the description above for example data.

Selected Value

selectedValue

INTEGER



This is the *integer* value of the selected choice, if applicable. This is only used if the first column of the **Data** property is a number, and the **Data** property has more than one column.

Selected String Value

selectedStringValue

STRING



This is the *string* value of the selected choice, if applicable. This is used in all cases other than when the **Selected Value** property is used.

Selected Label

selectedLabel

STRING



This is the value that is displayed for the currently selected choice. This is always active.

Horizontal Alignment

horizontalAlignment

INT



Adjusts the horizontal alignment of the text within the box.

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Vertical Alignment

verticalAlignment

INT



Adjusts the vertical alignment of the text within the box.

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Mode

mode

INT

Changes the dropdown's display.

Possible values are:

- List (0)
- Table (1)

No Selection Value	noSelectionValue	INT	
--------------------	------------------	-----	---

The integer value to use for the **Selected Value** property when no choice is selected. Defaults to -1.

No Selection String	noSelectionString	STRING	
---------------------	-------------------	--------	---

The string value to use for the **Selected String Value** property when no choice is selected. Defaults to "" (the empty string).

No Selection Label	noSelectionLabel	STRING	
--------------------	------------------	--------	---

The string value to use as the label when nothing is selected. Defaults to '<Select One>'.

Selection Background	selectionBackground	COLOR	
----------------------	---------------------	-------	---

The background color to use for the selected choice.

Show Table Header?	showTableHeader	BOOLEAN	
--------------------	-----------------	---------	---

Selects whether or not the dropdown table header is displayed. (only used in table mode)

Hide Table Columns?	hideTableColumns	STRING	
---------------------	------------------	--------	--

A comma separated list of columns to hide from the dropdown table, eg. 0,2 (only used in table mode)

Max Table Width	maxTableWidth	INT	
-----------------	---------------	-----	---

The maximum width allowed for the dropdown table. (only used in table mode)

Max Table Height	maxTableHeight	INT	
------------------	----------------	-----	---

The maximum height allowed for the dropdown table. (only used in table mode)

Styles	styles	DATASET	 
--------	--------	---------	---

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

• [key](#)

- keyPressed
- keyReleased
- keyTyped

• [mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange
- [focus](#)
 - focusGained
 - focusLost

Scripting Functions

The following functions are common to all components that allow dynamic properties:

get PropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

set PropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15
event.source.setPropertyValue("MyProperty", 15)
```

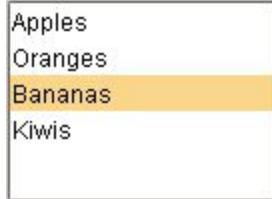


List Component

Icon in toolbar:

Description

The List component displays a list of options, allowing freeform selection of the items.



Properties

Data

data

DATASET



The list of choices to display. Items in the first column will be the ones shown.

Selection Mode

selectionMode

INT

This mode determines how selection is handled.

Possible values are:

- **Single (0)**. This means that only one option can be selected at a time.
- **Single Interval (1)**. This means that a single contiguous interval can be selected at a time using Shift-Click.
- **Multiple Interval (2)**. This means that freeform selection is allowed using a combination of Shift-Click and Control-Click.

Selected Index

selectedIndex

INT



The index of the selected row. The first row is index zero. This will be -1 if no rows are selected. This is mainly useful for the Single selection mode. [See the functions](#) for more ways to handle the selected rows for Single Interval and Multiple Interval selection modes.

Selected Foreground

selectedForeground

COLOR

The color of the foreground for the selected cell(s).

Selected Background

selectedBackground

COLOR

The color of the background for the selected cell(s).

Selected Focus Border

selectedFocusBorder

BORDER

The border for the focused selected cell.

Customizer

None.

Events

- [key](#)

- keyPressed
- keyReleased
- keyTyped
- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange
- [focus](#)
 - focusGained
 - focusLost

Scripting Functions

addSelectionInterval(start, end)

Adds the options at indexes start through end (inclusive) to the selected options.

Example:

```
# Include index 5 with the currently selected options
list = event.source.parent.getComponent("MyList")
list.addSelectionInterval(5, 5)

# Include indexes 7 through 10 with the currently selected options
list = event.source.parent.getComponent("MyList")
list.addSelectionInterval(7, 10)
```

getSelectedIndices()

Returns a list of the selected indices in increasing order. Returns an empty list if nothing is selected.

Example:

```
# Loop through the selected values, doing something with them
list = event.source.parent.getComponent("MyList")
selected = list.getSelectedIndices()
for index in selected:
    print list.data.getValueAt(index, 0)
```

getSelectedValue()

Returns the currently selected value, or None if there is none.

Example:

```
# Print the selected value, if applicable
list = event.source.parent.getComponent("MyList")
value = list.getSelectedValue()
if value != None:
    print value
else:
    print 'Nothing Selected'
```

getSelectedValues()

Returns a list of the selected values. Returns an empty list if nothing is selected.

Example:

```
# Loop through the selected values, doing something with them
list = event.source.parent.getComponent("MyList")
selected = list.getSelectedValues()
for value in selected:
    print value
```

isselectedIndex(index)

Returns true if the value at `index` is selected, false otherwise.

Example:

```
# Check if option 5 is selected
list = event.source.parent.getComponent("MyList")
if list.isSelectedIndex(5):
    print "Option 5 is selected"
```

isSelectionEmpty()

Returns true if nothing is selected.

setSelectedValue(value)

Tries to find a value equal to the parameter `value`, and if it finds one, sets it to be selected. If it doesn't find an option equal to `value`, the selection is cleared.

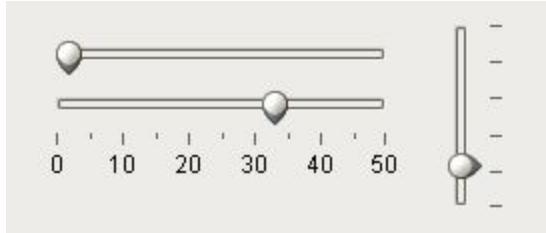


Slider Component

Icon in toolbar:

Description

The slider component lets the user drag an indicator along a scale to choose a value in a range.



Properties

Value	value	INT	
The currently selected value of the slider.			
Defer Updates	deferred	BOOLEAN	
If true, the Value property doesn't change as the user is sliding, only when they choose a resting point. This is good for the case where you have the Value property bi-directionally bound to the database, to avoid inundating the database with UPDATE queries.			
Minimum Value	minimum	INT	
The minimum value of the selectable range.			
Maximum Value	maximum	INT	
The maximum value of the selectable range.			
Horizontal Slider	horizontal	BOOLEAN	
If false, this slider displays itself vertically instead of horizontally.			
Snap To Ticks?	snapToTicks	BOOLEAN	
If true, the slider's indicator snaps to the ticks. This prevents the user from selecting values in between the ticks.			
Major Tick Spacing	majorTickSpacing	INT	
The distance between the 'major' tick marks. Should be a multiple of the minor tick spanning, and the size of the range should be divisible by this number for optimal tick placing.			
Minor Tick Spacing	minorTickSpacing	INT	
The distance between the 'minor' tick marks. The size of the range should be divisible by this number for optimal tick placing.			
Paint Track?	paintTrack	BOOLEAN	
Determines whether the 'track' that the indicator slides in is shown.			
Paint Labels?	paintLabels	BOOLEAN	
If true, numeric labels are shown on the major ticks.			

Paint Ticks?

paintTicks

BOOLEAN

If true, tick marks are painted at the major and minor tick spacings.

Inverted?

inverted

BOOLEAN

If true, the slider displays its values in descending order.

Styles

styles

DATASET



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [key](#)

- keyPressed
- keyReleased
- keyTyped

- [mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased

- [mouseMotion](#)

- mouseDragged
- mouseMoved

- [propertyChange](#)

- propertyChange

- [focus](#)

- focusGained
- focusLost

Scripting Functions

None.



Calendar Component

Icon in toolbar:



Description

The calendar component displays a calendar right on your window.



Properties

Date (immediate)

date

DATE



This date is the currently selected date, as the user selects it.

Date (latched)

latchedDate

DATE



This date is the value of the selected date the last time 'OK' was pushed.

Format String

format

STRING



The date formatting pattern used to format the string versions of the dates.

Formatted Date

formattedDate

STRING



The Date property, as formatted by the Format String.

Formatted Latched Date

formattedLatchedDate

STRING



The Latched Date property, as formatted by the Format String.

Time Style

timeStyle

INT



Determines how this calendar treats the time portion of the date.

Possible values are:

- **User Selectable (0).** A time spinner component will be shown, and the user will be able to select the desired time (default).
- **Start of Day (1).** No time component will be shown. The time portion of the selected date will always be 00:00:00.
- **End of Day (2).** No time component will be shown. The time portion of the selected date will always be 23:59:59.

Show OK Button

showOkButton

BOOLEAN

Turn this off if you don't want to show the OK button. The latched date and the immediate date will be equivalent.

Today Foreground	todayForeground	COLOR
-------------------------	-----------------	-------

Foreground color for the today indicator.

Today Background	todayBackground	COLOR
-------------------------	-----------------	-------

Background color for the today indicator.

Weekend Foreground	weekendForeground	COLOR
---------------------------	-------------------	-------

Foreground color for the weekend indicators.

Weekend Background	weekendBackground	COLOR
---------------------------	-------------------	-------

Background color for the weekend indicators.

Selected Border	selectedBorder	BORDER
------------------------	----------------	--------

The border to use for the currently selected date.

Title Background	titleBackground	COLOR
-------------------------	-----------------	-------

The background color for the title bar.

Customizer

None.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.



Popup Calendar Component

Icon in toolbar:



Description

The popup calendar component allows the selection of a date/time from a popup calendar.

11/23/2005 05:02 PM

Properties

Date

date

DATE



The date that is selected.

Format String

format

STRING

The format string that controls how the date is shown in the non-popup portion of the control. See the expression language function [dateFormat](#) for a description of this format string.

Text

text

STRING



The selected date, formatted as it is displayed in the non-popup portion of the component.

Time Style

timeStyle

INT

Determines how this calendar treats the time portion of the date.

Possible values are:

- **User Selectable (0)**. A time spinner component will be shown, and the user will be able to select the desired time (default).
- **Start of Day (1)**. No time component will be shown. The time portion of the selected date will always be 00:00:00.
- **End of Day (2)**. No time component will be shown. The time portion of the selected date will always be 23:59:59.

Calendar Background

calendarBackground

COLOR

The background color for the calendar popup.

Today Foreground

todayForeground

COLOR

Foreground color for the today indicator.

Today Background

todayBackground

COLOR

Background color for the today indicator.

Weekend Foreground

weekendForeground

COLOR

Foreground color for the weekend indicators.

Weekend Background

weekendBackground

COLOR

Background color for the weekend indicators.

Selected Border

selectedBorder

BORDER

The border to use for the currently selected date.

Title Background

titleBackground

COLOR

The background color for the title bar.

Styles

styles

DATASET



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased

- [mouseMotion](#)

- mouseDragged
- mouseMoved

- [propertyChange](#)

- propertyChange

Scripting Functions

The following functions are common to all components that allow dynamic properties:

get PropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

set PropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15  
event.source.setPropertyValue("MyProperty", 15)
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Date Range Component

Icon in toolbar:

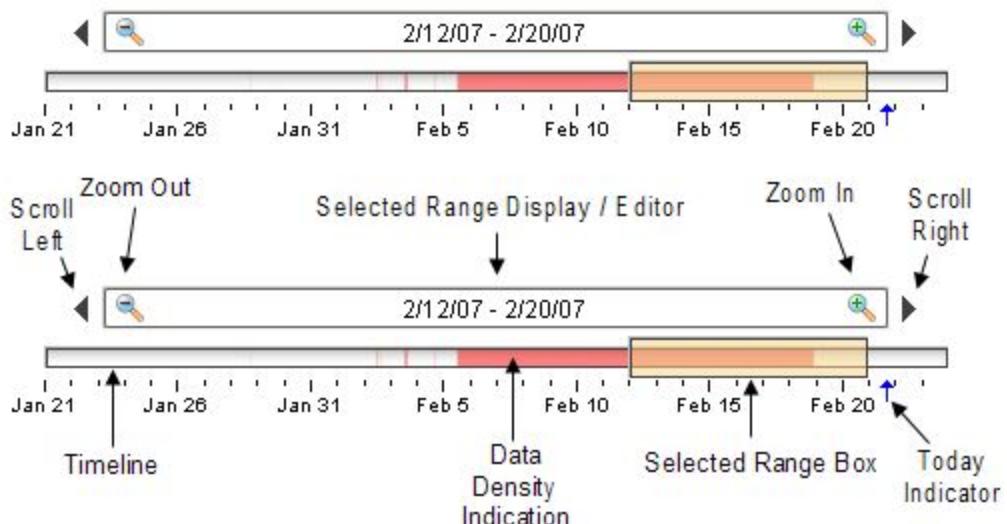


Description

The date range component provides an intuitive, drag-and-drop way to select a contiguous range of time. The user is shown a timeline and can drag or stretch the selection box around on the timeline. The selected range is always a whole number of units, where the unit is determined by the current zoom level. For instance, in the screenshot the selected range is Feb 12, 2007 through Feb 20, 2007. This means from the beginning of the 12th through the end of the 20th.

Using this component is as simple as using the **Start Date** and **End Date** properties that the component provides. Typically, you'll bind these dates into a SQL query that drives whatever you're using the date range for, such as a table. For instance, your query binding might look like this:

```
SELECT Column1, Column2, Column3  
FROM MyTable WHERE  
t_stamp >= {Root Container.Date Range.startDate} AND  
t_stamp <= {Root Container.Date Range.endDate}
```



As an advanced optional feature, the date range can display a data density histogram inside the timeline. This is useful for historical data with large gaps in it, so that the end user isn't hunting for data. (**Tip:** this is also great for demos, to make it easy to find historical data in a database that isn't being actively updated).

Properties

Startup Mode

startupMode

INT

Controls whether or not this date range automatically assigns itself a starting range based on the current time whenever it is displayed on a window that is opened. Automatic mode is very convenient in the majority of cases when you want the initial range to be some amount of time from the present into the past.

Possible values are:

- None (0)
- Automatic (1)

Startup Range

startupRange

STRING

If startup mode is Automatic, this will be the starting range of time available for selection (the extend of the timeline). This is a string of the form "# units", such as "5 days" or "2 hours" or "3 weeks".

Startup Selection

startupSelection

STRING

If startup mode is Automatic, this will be the starting selected range (the extend of the selection

box). This is a string of the form "# units", such as "5 days" or "2 hours" or "3 weeks".

Max Selection	maxSelectionSize	STRING
----------------------	------------------	--------

The maximum size of the selected date range. This is a string of the form "# units", such as "5 days" or "2 hours" or "3 weeks".

Start Date	startDate	DATE
-------------------	-----------	------

The starting date of the currently selected range.

End Date	endDate	DATE
-----------------	---------	------

The ending date of the currently selected range.

Outer Range Start	outerRangeStartDate	DATE
--------------------------	---------------------	------

The starting date of the available timeline.

Outer Range End	outerRangeEndDate	DATE
------------------------	-------------------	------

The ending date of the available timeline.

Today Color	todayIndicatorColor	COLOR
--------------------	---------------------	-------

The color of the "Today Arrow" indicator

Editor Background	editorBackground	COLOR
--------------------------	------------------	-------

The background color of the textual date range editor portion of this component.

Box Fill	boxFill	COLOR
-----------------	---------	-------

The fill color for the selection box.

Selection Highlight	selectionHighlight	COLOR
----------------------------	--------------------	-------

The focus highlight color for the selection box

Track Margin	trackMargin	INT
---------------------	-------------	-----

The amount of room on either side of the slider track. May need adjusting if default font is changed.

Tick Density	tickDensity	FLOAT
---------------------	-------------	-------

This is multiplied by the width of the timeline to determine the current ideal tick unit.

Data Density	densityData	DATASET
---------------------	-------------	---------

A dataset that is used to calculate a histogram of data density. The dataset should have one column which should be a date column. Fill this in with a date for each sample of data that you have. For example, the query that drives this DataSet might be like:

```
SELECT t_stamp
FROM MyTable WHERE
t_stamp >= {Root Container.Date Range.outerRangeStartDate} AND
t_stamp <= {Root Container.Date Range.outerRangeEndDate}
```

High Density Color	highDensityColor	COLOR
---------------------------	------------------	-------

The color used to indicate high data density, if you fill in the **Data Density** DataSet.

Date Style	dateStyle	INT
-------------------	-----------	-----

The style to display dates in. For international support.

Possible values are:

- Auto (0)
- MDY (1)
- DMY (2)
- YMD (3)

Time Style

timeStyle

INT



The style to display times of day. For international support.

Possible values are:

- Auto (15)
- 12 HR (16)
- 24 HR (17)

Customizer

This component's customizer is the Dynamic Properties Customizer. This mean that you can add [your own properties](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.



Table Component

Icon in toolbar:

Description

The Table component in FactoryPMI is very powerful and easy to configure. Its unprecedented flexibility allows you to easily display your tabular data in a variety of ways. Important features include:

- **Column Sorting.** Your users can easily sort the data by clicking on the column headers. The sorting is a 3-mode sort: Ascending, Descending, and "Natural", which uses the default order of the data.
- **Mapped Row Coloring.** Map the background color of each row to a particular column. This allows you to give powerful visual indication of different types of rows in your tables, such as differentiating between alarm states.
- **Column Translation.** Allow the table component to handle all code mapping, such as mapping 0 to "Off" and 1 to "On". No fancy SQL knowledge required.
- **Images.** Map values to images, allowing intuitive visual cues.
- **Progress Bar Indication.** Display numeric data as progress bars inside cells, providing fast visual reference for bounded amounts.
- **Number and Date formatting.** Format numbers and dates to your exact specification.
- **Column Hiding.** Hide columns from view that contain identifying data used by the row coloring or by other components.
- **Printing.** Print tables directly to multi-paged printouts.
- **Editing.** Columns can be made editable. Changes will be reflected in the underlying dataset, at which point they can be mapped back to a database.

Col 1	Col 2	Col 3
17	Test Row 4	2.34 °F
50	Test Row 2	11.92 °F
83	Test Row 14	16.06 °F
66	Test Row 8	22.72 °F
58	Test Row 16	49.76 °F
95	Test Row 12	54.00 °F
57	Test Row 19	58.94 °F
22	Test Row 4	60.75 °F
94	Test Row 1	67.26 °F
39	Test Row 0	72.22 °F
81	Test Row 4	80.94 °F
89	Test Row 17	87.88 °F
76	Test Row 16	88.38 °F
20	Test Row 1	94.08 °F
86	Test Row 16	97.33 °F

Unit	Amps	Running?
Compressor 1	13.32	
Compressor 2	0.00	
Pump 1	24.30	
Pump 2	10.20	

Tips and Tricks

Binding into a Table.

Often you will want to bind components such as labels and text boxes values to the currently selected value in the table. The tricky part is making sure the binding doesn't fail when nothing is selected. The following expression does this quite nicely:

```
if ({Root Contai ner. MyTable. selectedRow} = -1,
    "----", //this is the fail case
    {Root Contai ner. MyTable. data}[{Root Contai ner. MyTable. selectedRow}, "Col 1"])
```

If you're binding this to an integer property, like the `intValue` property of a numeric text field, you will need to cast the value to an integer, like this:

```
if ({Root Contai ner. MyTable. selectedRow} = -1,
    0, //this is the fail case
    toInt({Root Contai ner. MyTable. data}[{Root Contai ner. MyTable. selectedRow}, "Col 1"]))
```

Changing the Column Widths.

To change a table's column's widths, simply switch into preview mode and use your mouse to resize the columns, and then switch back to design mode. Simple!

Exporting to HTML.

You can export the table to an HTML file that retains the table's formatting. To do this, use a script like this: (more about the table's **exportHTML** function is [here.](#))

```
table = event.source.parent.getComponent("Table") # Get a reference to the table  
table.exportHTML("MyTable.html", "My Table Header", 500) # Prompt user to save the exported file
```

Exporting to CSV.

You can export the table's raw data to a CSV file. To do this, use a script like this: (more about the **fpmi.db.exportCSV** function is [here.](#))

```
table = event.source.parent.getComponent("Table") # Get a reference to the table  
fpmi.db.exportCSV("mydata.csv", 1, table.data)
```

Printing.

Printing a table is a snap! Simply use the table's built in print function like this: (more about the **print** function is [here.](#))

```
table = event.source.parent.getComponent("Table") # Get a reference to the table  
table.print()
```

Properties

Data

data

DATASET



This is the most important property of the table, as it contains the actual data that the table displays. This can be entered directly in the designer, but more commonly it is bound to some SQL query.

Column Attributes Data

columnAttributesData

DATASET

This is the dataset that backs the current column attributes configured via the customizer.

Selection Mode

selectionMode

INT

Possible values are:

- Single (0)** Only one thing (row, column, or cell) can be selected at a time.
- Single Interval (1)** One contiguous interval can be selected at a time.
- Multiple Interval (2)** Selection is free-form, number of noncontiguous selection intervals can be made.

Row Selection Allowed

rowSelectionAllowed

BOOLEAN

If true, row selection is allowed. If Column Selection is disallowed, this means that entire rows will be selected. If Column Selection is also allowed, this means that individual cells can be selected.

Column Selection Allowed

columnSelectionAllowed

BOOLEAN

If true, column selection is allowed. If Row Selection is disallowed, this means that entire columns will be selected. If Row Selection is also allowed, this means that individual cells can be selected.

Row Height

rowHeight

INT

The height of each row, in pixels. Default is 16.

Background Mode

backgroundColorMode

INT

Controls the background of each row.

Possible values are:

- Constant (1).** All cells have the same background, which is the background color of the table.
- Alternating (2).** Rows alternate between the background color of the table, and the **Odd Row**

Background color. Good for making large tables pleasing to the eye, and easier to read.

- **Mapped (3)**. Background color for each row is controlled by the background color map and the mapping column. See the [Customizer](#) section.

Odd Row Background	oddBackground	COLOR
---------------------------	---------------	-------

The background color of every odd row if the **Background Mode** is set to Alternating.

Auto-Resize Mode	autoResizeMode	INT
-------------------------	----------------	-----

In all modes other than Off, the table fits all columns into the table so that there is no horizontal scrolling. When you resize one column, this mode controls how the other columns are resized to fit.

Possible values are:

- **All Columns (4)**. All other columns are proportionally resized.
- **Last Column (3)**. The table adjusts the last column to make everything fit. If it is the last column that is changing, the second to last is adjusted.
- **Next Column (1)** The column after the column that is changing is adjusted.
- **Off (0)**. Columns aren't fit to the table width in the first place. All columns have some initial width, and resizing them just initiates scrolling.
- **Subsequent Columns (2)**. Like **All**, but only the columns after the one that is changing are adjusted proportionately.

TestData	test	BOOLEAN
-----------------	------	---------

This quasi-property is for experimenting with tables. Every time it is turned on, the **Data** property is filled with random data.

Initially Selected Row	initialRowSelection	INT
-------------------------------	---------------------	-----

If non-negative, the table will auto-select the row at this index whenever the table's data transitions from data with zero rows to data with a positive number of rows.

Selected Column	selectedColumn	INT
------------------------	----------------	-----

The currently selected column, or -1 if no columns are selected.

Selected Row	selectedRow	INT
---------------------	-------------	-----

The currently selected row, or -1 if no columns are selected.

Properties Loading	propertiesLoading	INT
---------------------------	-------------------	-----

True if the any properties are currently being loaded from a SQL query. Useful if the table is bound to a very large query that you want to give some kind of "Loading" indication for.

Selection Background	selectionBackground	COLOR
-----------------------------	---------------------	-------

This property controls the background color of selected cells. Note that you can make this color semi-transparent to merely tint the selected cells, which preserves any mapped or alternating background color.

Selection Foreground	selectionForeground	COLOR
-----------------------------	---------------------	-------

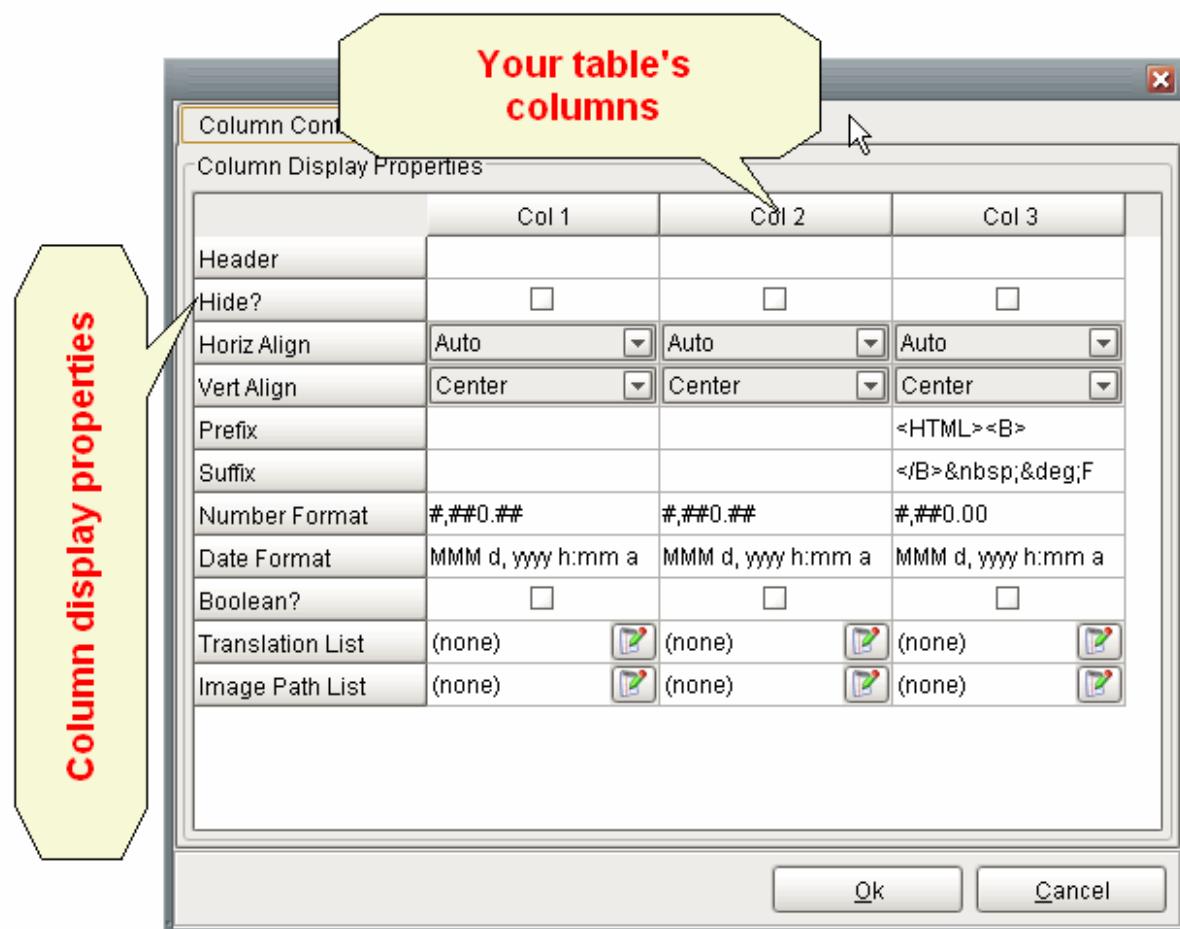
This property controls the foreground (text) color of selected cells.

Show Horizontal Grid Lines?	showHorizontalLines	BOOLEAN
------------------------------------	---------------------	---------

Show Vertical Grid Lines?	showVerticalLines	BOOLEAN
----------------------------------	-------------------	---------

Grid Line Color	gridColor	COLOR
------------------------	-----------	-------

The color used to draw grid lines.



The Table Customizer is where you configure your columns' display properties, as well as the row mapping configuration.

Column Display Properties

When you open the Table Customizer, you will see a table that has all of your data's columns across the top, and all of the column display properties across the left. You can configure each column to have its own display properties.

- **Header** By default, each column's header will be the column name as reported by the **Data** property, which is usually the column name in the database. Sometimes you want to dress this up a bit and make it more readable. If you enter a header here, it will be displayed instead of the column name.
- **Hide?** If you check this, the column will not be displayed. This is useful for data that is used by other things, like row color mapping or other logic on your window, but isn't very human-readable.
- **Horiz Align** Controls the horizontal alignment of the contents of each cell. Auto's rules are as follows: Numbers: Right, Images: Center, Everything else: Left.
- **Vert Align** Controls the vertical alignment of the contents of each cell.
- **Prefix** A string that will be prepended in front of the data in the cell. Useful for HTML formatting within the cell.
- **Suffix** A string that will be appended to the end of the data. Useful for HTML formatting and displaying units.
- **Number Format** A number format string that will be used to format the data *if it is numeric data*. See the [numberFormat expression](#) documentation for the syntax of this string.
- **Date Format** A date format string that will be used to format the data *if the column's data are dates*. See the [dateFormat expression](#) documentation for the syntax of this string.
- **Boolean?** If true, the data in this column will be rendered as a checkbox. The checkbox will be checked if the data evaluates to true. Numbers that are non-zero are true. Strings like "yes", "true", "on" are also true.
- **Progress Bar?** If true, the column, which should be numeric, will display a horizontal progress bar as the background of the cell.
- **Progress Bar Range** Defines the range (far left and far right) of the progress bar. If the cell's value falls outside of this range, the progress bar will simply display as empty or full.
- **Hide Text Over P-Bar?** If true, the textual representation of the cell's numeric value will not be shown
-

- **P-Bar Color** The color of the filled part of the progress bar.
- **P-Bar Background** The color of the background of the progress bar.
- **Translation List** If you define mappings in this list, if any data is found in this column that matches the **key** of a mapping, that mapping's **value** will be displayed instead. For instance, suppose you were selecting a Motor HOA status field. You could map 0 to "Off", 1 to "Auto", and 2 to "Hand", and your data would be human-readable. Note that *only the display* is translated; the data is still numeric in the **Data** property.
- **Image Path List** Similar to the **Translation List** property, but the values are treated as image paths, and that image is shown in the cell.
- **Background Color List** Similar to the **Translation List** property, but the values are treated as colors, and that color is shown as the background in the cell.
- **Foreground Color List** Similar to the **Translation List** property, but the values are treated as colors, and that color is shown as the foreground in the cell.
- **Editable** If true, the cell will become editable when double-clicked. Valid changes will be reflected back in a change to the table's **data** property. The table will also fire a `cellEdited` event that contains the row, column, previous value, and new value for the cell. Remember, if the table's data is bound to a polling query binding, the edited dataset will be overwritten with whatever is in the database. You can use the `cellEdited` event to issue a SQL UPDATE query that will make the edit in the database as well.

Background Color Mapping

If you have set the table's **Background Mode** to Mapped, you use this section of the Table Customizer to define your mapping. Choose a numeric column to govern the mapping, and then map the value to colors.

Events

- [key](#)
 - keyPressed
 - keyReleased
 - keyTyped
- [mouse](#)
 - mouseClicked
 - mousePressed
 - mouseReleased
- [propertyChange](#)
 - propertyChange
- [cell](#)
 - cellEdited

Scripting Functions

`addRow(row)`

Adds the given row to the dataset. The length of the list must be equal to the number of columns in the table's dataset, and the types of the values in the list must match the types in the dataset.

Parameters

`row`

A list representing the row to be added to the dataset.

Example:

```
# This code adds a row to a table's dataset, if that dataset had three columns, and integer, a string, and a float.
table = event.source.parent.getComponent("Table")
table.addRow([5, "hello", 8.9])
```

deleteRow(row)

Returns the given row number from the dataset.

Parameters

row

An integer representing the index of the row in the dataset to delete.

getRowsInViewOrder()

Returns a list of row indexes, in the order that they are currently sorted. So, for example, if you have 3 rows and the user hasn't sorted the table in any special way, you'll get [0, 1, 2]. If they sort the table in reverse order, you'll get [2, 1, 0]. This is useful if you need to do something with the rows of the table in the order that the user is currently viewing them.

Example:

```
# Loop through the rows, updating each task with a priority number in the
# order that the user has sorted them.
table = event.source.parent.getComponent("MyTable")
order = table.getRowsInViewOrder()
priority=0
for row in order:
    taskId = table.data.getValueAt(row, 0)
    query = "UPDATE mytable SET task_pri = %d WHERE id= %d" % (priority, taskId)
    fpmi.db.runUpdateQuery(query)
    priority = priority + 1
```

getSelectedColumnCount()

Returns the number of columns that are currently selected.

getSelectedColumns()

Returns a list of the selected column indices. Returns an empty list if nothing is selected.

Example:

```
# Loop through the selected columns, printing out the data in the first row of each column
table = event.source.parent.getComponent("MyTable")
columns = table.getSelectedColumns()
for col in columns:
    print table.data.getValueAt(0,col)
```

getSelectedRowCount()

Returns the number of rows that are currently selected.

getSelectedRows()

Returns a list of the selected row indices. Returns an empty list if nothing is selected.

Example:

```
# Loop through the selected rows, printing the entire row
table = event.source.parent.getComponent("MyTable")
rows = table.getSelectedRows()
for row in rows:
    for col in range(table.data.columnCount):
        # Note the trailing comma here - to suppress newline
        print table.data.getValueAt(row,col),
    print "" #for newline
```

isCellSelected(row, column)

Returns true if the cell at the given coordinates is selected.

isRowSelected(row)

Returns true if the given row is selected.

exportHTML(filename, title, width)

Creates an HTML file that looks very similar to the current contents of the table. Visual properties like colors, number and date formatting, alignment, etc are all preserved. (as opposed to the `fpmi.db.exportHTML` function). This function will return the path to the file that was created, or None if the action was canceled by the user.

Parameters

filename

A suggested filename for the user.

title

The title of the web page that will be created.

width

The width (in pixels) of the resulting HTML table.

Example 1:

```
# This code, which would go in a button that is in the same container as  
# the table, would export the table's data as a web page.
```

```
table = event.source.parent.getComponent("Table")  
table.exportHTML("table_data.html", "My Table Data", 500)
```

Example 2:

```
# Tip: you can open the file afterwards by using the fpmi.net.openURL() function
```

```
table = event.source.parent.getComponent("Table")  
filename = table.exportHTML("table_data.html", "My Table Data", 500)  
if filename != None:  
    fpmi.net.openURL("file://" + filename)
```

getDataAsHTML(title, width)

Returns the same HTML that is generated via `exportHTML`, but as a string in memory, so that you can do with it is you please. (For instance, send it as an email via [fpmi.net.sendEmail\(\)](#))

Parameters

title

The title of the web page that will be created.

width

The width (in pixels) of the resulting HTML table.

print([fitWidth], [HeaderText], [FooterText])

Prints the data in the table. Note that this is a 'smarter' print than the printing function found in the `fpmi.print` builtin package. That print function simply prints the table as it appears on the screen - even if the table has a scrollbar. This print function prints all of the data in the table on multiple pages if necessary.

Parameters

[fitWidth] (optional)

If true, the table is stretched to fit the width of the page. If false, a wide table's columns width may

spill onto multiple pages. (Default: true)

[HeaderText] (optional)

Some text to display on the top of each page. If "{0}" appears in the text, it will be replaced with the current page number. (Default: None)

[FooterText] (optional)

Some text to display on the bottom of each page. If "{0}" appears in the text, it will be replaced with the current page number. (Default: "Page {0}")

Example:

```
# This code would go in the actionPerformed event of a button placed next to a table to print out the table
table = event.source.parent.getComponent("Table")
table.print()
```

setColumnLabel(columnIndex, label)

Programmatically sets the column header label at index columnIndex to the given string label.

Parameters

columnIndex

The index of the column to set the label for. Note that this is the visible index, not the index in the DataSet, so hidden columns don't apply.

label

The desired header label for the column.

Example:

```
# This code would be placed in the propertyChange event for the table component to
# rename the columns every time the data is loaded.
if event.propertyName == "data":
    table = event.source
    table.setColumnLabel(0, "Machine")
    table.setColumnLabel(1, "Operator")
    table.setColumnLabel(2, "Status")
```

setColumnWidth(columnIndex, width)

Programmatically sets the column at index columnIndex to the given width. Useful for tables whose columns are dynamically named, so the normal column widths don't stick.

Parameters

columnIndex

The index of the column to set the width for. Note that this is the visible index, not the index in the DataSet, so hidden columns don't apply.

width

The desired width of the column. Note that unless **auto-resize mode** is **Off**, this is a relative width.

Example:

```
# This code would set the widths of the column, after setting the table data to some dynamic data
table = event.source.parent.getComponent("Table")
newData = fpmi.db.runQuery(something)
table.data = fpmi.db.toDataSet(newData)
table.setColumnWidth(0, 100)
table.setColumnWidth(1, 200)
```

setValue(row, column, value)

Sets the value in the table's dataset at the given location.

Parameters

row

The index of the row to update.

column

The index or name of the column to update

value

The value to set at the location.

Example:

```
# This code would set the value of column "ColX"=10 and on row number 5.
```

```
table = event.source.parent.getComponent("Table")
```

```
table.setValue(5, "ColX", 10)
```

updateRow(row, changes)

Updates a row in the dataset.

Parameters

row

An integer representing the index of the row in the dataset to delete.

changes

A dictionary of changes to make to the row. Keys in this dictionary represent column names or column indexes, and the values represent the values to update.

Example:

```
# This code would set the value of column "ColX"=5 and "ColY"=8.9 on row number 5.
```

```
table = event.source.parent.getComponent("Table")
```

```
table.updateRow(5, {"ColX":5, "ColY":8.9})
```



Button Component

Icon in toolbar:

Description

The Button component is a versatile component used for things like opening/closing windows and setting values in the database (turning machines on, etc). It can be used for showing status, as well. For example, if you have three buttons, Hand, Off, and Auto, not only can they set those modes, but their background color can display the current mode.

To get buttons to do things, you add an Action to the `actionPerformed` event. Many new users to FactoryPMI will configure an Action for the `mouseClicked` event instead. While this will work, it is better to use the `actionPerformed` event. Why? Buttons can also be activated by tabbing over to them and hitting **Enter** or **Space**, or they could be activated by pressing **Alt** and the button's *mnemonic* character. So, to make sure that your button works in all of these cases, configure your Action on the `actionPerformed` event, *not* the `mouseClicked` event. For more information on Actions, see the [Events](#) section of the Designer manual.



Properties

Background Color

buttonBG

COLOR

The background color of the button. Note that this button's scripting name is `buttonBG`, not `background`, like most components.

Text

text

STRING



The text that the button displays.

Mnemonic

mnemonicChar

STRING

The mnemonic characters should be a character found in the **Text** property. This character will be underlined, and the button will be activated when the user presses the Alt key plus this character.

Image Path

path

STRING



The path to the image that this button shows.

Icon-Text Spacing

iconTextGap

INT

The space between the image and the text.

Horizontal Alignment

horizontalAlignment

INT

The horizontal alignment of the button contents (image and text).

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

INT

Vertical Alignment

verticalAlignment

INT

The vertical alignment of the button contents (image and text).

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Background 3D?

background3D

BOOLEAN



If you turn this off, the background of the button will be drawn flat. You may need to change the background color to see the effect.

Fill Area?

contentAreaFilled

BOOLEAN



If you turn this off, the button will not fill in its background. (it will be transparent). Make sure Opaque is off as well.

Border Painted?

borderPainted

BOOLEAN



If you turn this off, the button won't draw its border.

Rollover?

rolloverEnabled

BOOLEAN



If you turn this off, the mouseover effect of the button won't occur.

Default Button

defaultBtn

BOOLEAN



If you turn this on, this button will be the 'default' button for the window. This makes it activate when enter is pressed anywhere on the window, unless an input component like a text box has input focus.

Disabled Image Path

disabledPath

STRING



Normally when a button is disabled, its image is merely grayed out. If you set this property, this image will be used instead.

Margin

margin

INSETS



This determines the minimum amount of padding between the button contents and the button border.

Horizontal Text Position

horizontalTextPosition

INT



This is the position of the button's text *relative* to its image.

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Vertical Text Position

verticalTextPosition

INT



This is the position of the button's text *relative* to its image.

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [action](#)
 - actionPerformed (Remember, use this to respond to the button being pushed.)
- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

doClick()

Programmatically simulates a user clicking the button. Your action script will be run.

Example:

```
# Click some button on the window
button = event.source.parent.getComponent("MyButton")
button.doClick()
```

The following functions are common to all components that allow dynamic properties:

getPropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

setPropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15
event.source.setPropertyValue("MyProperty", 15)
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



One-Shot Button Component

Icon in toolbar:

Description

The one-shot button is great for telling a PLC to do something. It simply writes a value, and waits for it to be reset. To use the one-shot button, bind an OPC tag bidirectionally to the one-shot button's **Value** property. When clicked, the button will write the value in its **Set Value** property to the **Value** property. Typically, **Set Value** is 1, and **Value** is 0 in a ready state, although the logic could be reversed or change simply by altering **Set Value**. The button can disable itself when it is writing, and will display different text. Note that the button considers itself to be writing whenever **Value** equals **Set Value** - you must make sure that the PLC resets this value, otherwise the button will remain in a writing state.



Properties

Value

value

INT



The current value. Should be bound bi-directionally to a tag

Set Value

setValue

INT



The value to set the control value to when the button is pushed.

Idle Text

normalText

STRING



The text of the button while its value is not being written

Writing Text

writePendingText

STRING



The text of the button while its value is being written

Disable While Writing

disableWhileWriting

BOOLEAN

If true, the button will be disabled while it is writing.

Confirm?

confirm

BOOLEAN

If true, a confirmation box will be shown.

Confirm Text

confirmText

STRING

The message to ask the user if confirmation is turned on.

Background Color

buttonBG

COLOR

The background color of the button.

Background 3D?

background3D

BOOLEAN



Should this button have a 3d type background, or a flat color one?

Fill Area?

contentAreaFilled

BOOLEAN



Controls whether or not this button's internal area is filled

Border Painted?	borderPainted	BOOLEAN	
------------------------	---------------	---------	---

Should the border of this button be displayed?

Rollover?	rolloverEnabled	BOOLEAN	
------------------	-----------------	---------	---

If true, the button may indicate that the mouse is hovering over it.

Image Path	path	STRING	
-------------------	------	--------	---

The relative path of the image.

Disabled Image Path	disabledPath	STRING	
----------------------------	--------------	--------	---

The relative path of the image to be displayed when this component is not enabled.

Icon-Text Spacing	iconTextGap	INT	
--------------------------	-------------	-----	--

The space (in pixels) between the icon (if any) and the text (if any)

Margin	margin	INSETS	
---------------	--------	--------	---

The space between a button's text and its borders.

Horizontal Alignment	horizontalAlignment	INT	
-----------------------------	---------------------	-----	--

The horizontal alignment of the button's contents (text and/or image)

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Horizontal Text Position	horizontalTextPosition	INT	
---------------------------------	------------------------	-----	---

The horizontal position of the button's text relative to its image

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Vertical Alignment	verticalAlignment	INT	
---------------------------	-------------------	-----	--

The vertical alignment of the button's contents (text and/or image)

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Vertical Text Position	verticalTextPosition	INT	
-------------------------------	----------------------	-----	---

The vertical position of the button's text relative to its image

Possible values are:

- Top (1)

- Center (0)
- Bottom (3)

Styles

Contains the component's styles

DATASET



Data Quality

dataQuality

INT



The data quality code for any tag bindings on this component.

Mnemonic

mnemonicChar

STRING



A single letter that will activate the button using 'ALT-*mnemonic*'.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [action](#)
 - actionPerformed
- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

doClick()

Programmatically simulates a user clicking the button.

Example:

```
# Click some button on the window
button = event.source.parent.getComponent("MyButton")
button.doClick()
```

The following functions are common to all components that allow dynamic properties:

getPropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window  
val = event.source.getPropertyValue("MyProperty")  
fpmi.gui.messageBox(val)
```

setPropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15  
event.source.setPropertyValue("MyProperty", 15)
```



2 State Toggle Component

Icon in toolbar:

Description

This button is similar to the basic [Toggle Button](#), but more finely tuned to work in realistic controls environments. Use this button any time you want to toggle a value between two states, such as On/Off, Stop/Run, etc. If you have more than two states (for example, Hand/Off/Auto, use the [Multi-State Button](#)).



How do I use it? It is very simple. If you have a tag whose value you want to toggle between 2 values (like zero and one), you simply drag and drop the tag onto the button. This will bind both the **Control Value** and **Indicator Value** to that tag. Now set the **State 1 Value** and **State 2 Value** to your two states (they default to zero and one, respectively). Lastly, use the [Styles Customizer](#) to define the styles for your two states.

How does it work? This button has four integer values that you use to set it up: the **Control Value**, the **Indicator Value**, and values that define the 2 different states (**State 1 Value** and **State 2 Value**). Every time you press the button, one of the state values is written to the control value. The **Indicator Value** is used to determine which state you're in. For example, suppose that **State 1 Value** was zero and **State 2 Value** is one. If **Indicator Value** is zero and you press the button, it'll write a one to the **Control Value**. The **Style** of the component is typically driven by the read-only property **Current State**. Current State equals zero when **Indicator Value=State 1 Value** and one otherwise.

Properties

Control Value

controlValue

INT



Bind this to the tag that controls the state. (Typically, this is bound to the same location as *Indicator Value*)

Indicator Value

indicatorValue

INT



Bind this to the tag that indicates the current state. (Typically, this is bound to the same location as *Control Value*)

State 1 Value

state1Value

INT



The value that will be written to **controlValue** when the button is pushed in state 2.

State 2 Value

state2Value

INT



The value that will be written to **controlValue** when the button is pushed in state 1.

Confirm?

confirm

BOOLEAN

If true, a confirmation box will be shown.

Confirm Text

confirmText

STRING

Background Color

buttonBG

COLOR

The background color of the button.

Background 3D?

background3D

BOOLEAN



Should this button have a 3d type background, or a flat color one?

Fill Area?	contentAreaFilled	BOOLEAN	
Controls whether or not this button's internal area is filled			
Border Painted?	borderPainted	BOOLEAN	
Should the border of this button be displayed?			
Rollover?	rolloverEnabled	BOOLEAN	
If true, the button may indicate that the mouse is hovering over it.			
Text	text	STRING	
Text of this component			
Image Path	path	STRING	
The relative path of the image.			
Disabled Image Path	disabledPath	STRING	
The relative path of the image to be displayed when this component is not enabled.			
Icon-Text Spacing	iconTextGap	INT	
The space (in pixels) between the icon (if any) and the text (if any)			
Margin	margin	INSETS	
The space between a button's text and its borders.			
Horizontal Alignment	horizontalAlignment	INT	
The horizontal alignment of the button's contents (text and/or image)			
Possible values are:			
<ul style="list-style-type: none">• Left (2)• Center (0)• Right (4)• Leading (10)• Trailing (11)			
Horizontal Text Position	horizontalTextPosition	INT	
The horizontal position of the button's text relative to its image			
Possible values are:			
<ul style="list-style-type: none">• Left (2)• Center (0)• Right (4)• Leading (10)• Trailing (11)			
Vertical Alignment	verticalAlignment	INT	
The vertical alignment of the button's contents (text and/or image)			
Possible values are:			
<ul style="list-style-type: none">• Top (1)			

- Center (0)
- Bottom (3)

Vertical Text Position

verticalTextPosition

INT



The vertical position of the button's text relative to its image

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Styles

styles

DATASET



Contains the component's styles

Data Quality

dataQuality

INT



The data quality code for any tag bindings on this component.

Current State

state

INT



Read-only property that shows what state (0 or 1) this button is currently in.

The message to ask the user if confirmation is turned on.

Mnemonic

mnemonicChar

STRING



A single letter that will activate the button using 'ALT-mnemonic'.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [action](#)
 - actionPerformed
- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

doClick()

Programmatically simulates a user clicking the button.

Example:

```
# Click some button on the window  
button = event.source.parent.getComponent("MyButton")  
button.doClick()
```

The following functions are common to all components that allow dynamic properties:

getPropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window  
val = event.source.getPropertyValue("MyProperty")  
fpmi.gui.messageBox(val)
```

setPropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15  
event.source.setPropertyValue("MyProperty", 15)
```



Multi-State Button Component

Icon in toolbar:

Description

This button is really a series of two or more buttons. Each button represents an integer-valued state. Each state defines two styles for a button: the selected style, and the unselected style. Each button is automatically displayed with the correct style based on the current state (the value of **Indicator Value**). When a button is pressed, its state value is written to the **Control Value**.



How do I use it? Simply drag a tag that represents your state onto the Multi-State Button. This will bind both the **Control Value** and **Indicator Value** to that tag. Now open up the Multi-State Button customizer, and define your states: their order, values and styles. Lastly choose if you want the buttons to be a column, row, or grid by setting the **Display Style** property.

Properties

Control Value

controlValue

INT



Bind this to the tag that controls the state. (Typically, this is bound to the same location as *Indicator Value*)

Indicator Value

indicatorValue

INT



Bind this to the tag that indicates the current state. (Typically, this is bound to the same location as *Control Value*)

Display Style

displayStyle

INT

The display style (rows or columns) for this N-state button.

Possible values are:

- Column (0)
- Row (1)
- Grid (2)

Horizontal Gap

hGap

INT

The horizontal spacing between buttons

Vertical Gap

vGap

INT

The vertical spacing between buttons

Grid Rows

gridRows

INT

The number of rows if the Display Style is set to "Grid" mode.

Grid Cols

gridCols

INT

The number of columns if the Display Style is set to "Grid" mode.

Confirm?	confirm	BOOLEAN
-----------------	---------	---------

If true, a confirmation box will be shown.

Confirm Text	confirmText	STRING
---------------------	-------------	--------

The message to ask the user if confirmation is turned on.

States	states	DATASET
---------------	--------	---------



A DataSet that stores the information for the different states.

Customizers

- Multi-State Button Customizer. This customizer lets you define the states for this component. Each state will correspond to a button, in order. Set up both the selected style and unselected style for each state. Each style lets you define the button's: Text, Foreground, Background, Border, and Image Path.
- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

The following functions are common to all components that allow dynamic properties:

get PropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

set PropertyValue(PropName, Value)

Set the value of the dynamic property named `PropName` to the value `Value`.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

Change the value to 15

```
event.source.setPropertyValue("MyProperty", 15)
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Toggle Button Component

Icon in toolbar: ON

Description

The toggle button represents a bit: on (selected) or off (not selected). Visually the button looks down or depressed when it is selected, and up when it is not selected. This component is very similar to the Check Box component.



Properties

Background Color

buttonBG

COLOR

The background color of the button. Note that this button's scripting name is `buttonBG`, not `background`, like most components.

Label

text

STRING



The text that this button displays.

Selected

selected

BOOLEAN



This will be `true` when the Toggle Button is selected (down), and `false` otherwise.

Image Path

path

STRING



If set, the Toggle Button will show this image.

Selected Image Path

selectedPath

STRING

If set, this Toggle Button will show this image *when selected*.

Background 3D?

background3D

BOOLEAN



If you turn this off, the background of the button will be drawn flat. You may need to change the background color to see the effect.

Fill Area?

contentAreaFilled

BOOLEAN



If you turn this off, the button will not fill in its background. (it will be transparent). Make sure Opaque is off as well.

Border Painted?

borderPainted

BOOLEAN



If you turn this off, the button won't draw its border.

Rollover?

rolloverEnabled

BOOLEAN



If you turn this off, the mouseover effect of the button won't occur.

Margin

margin

INSETS



This determines the minimum amount of padding between the button contents and the button border.

Styles

styles

DATASET





Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [item](#)
 - itemStateChanged
- [key](#)
 - keyPressed
 - keyReleased
 - keyTyped
- [action](#)
 - actionPerformed
- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange
- [focus](#)
 - focusGained
 - focusLost

Scripting Functions

None.

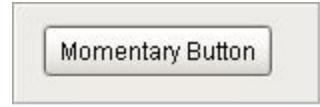


Momentary Button Component

Icon in toolbar: 

Description

The Momentary button is a specialized button that has four special properties: **Value**, **On Value**, **Off Value**, and **On Time**. By default, **On Value** is 1, and **Off Value** is 0. When the button is pushed, it sets **Value** to **On Value** (1), and then after **On Time** milliseconds, it sets it back to **Off Value** (0). This **Value** property can then be easily bound bi-directionally to a database field.



The purpose of this field is to mimic the behavior of a physical momentary push button. It allows easy interfacing to a PLC memory location that needs to be set for a small amount of time to turn a machine on or off, but shouldn't stay set.

Warning. This button merely sets a database field for a certain amount of time. It is up to you to make sure that FactorySQL is polling that same field at a quick enough rate that ensures that the value will be noticed. For instance, if you set your Momentary Button to hold the value on for 4000 milliseconds, FactorySQL should be polling at around 1500 milliseconds.

Advanced Use. Sometimes you want to have some more control over your button's action, such as displaying a confirmation box before the value is set. To do this, set the Momentary Button's **Automatic** property to false. It is now up to you to add an action script to the `actionPerformed` event to start the timing process by calling the `start()` function. Such a script might look like this:

```
# Activate this momentary button if the user is really sure
if fpmi.gui.confirm("Are you sure you want to shut down the machine?"):
    event.source.start()
```

Properties

this value becomes the value of **On Value** for **On Time** milliseconds. Bind this value bi-directionally to some database field for easy configuration.

Background Color

buttonBG

COLOR

The background color of the button. Note that this button's scripting name is `buttonBG`, not `background`, like most components.

Text

text

STRING



The text that the button displays.

Mnemonic

mnemonicChar

STRING

The mnemonic characters should be a character found in the **Text** property. This character will be underlined, and the button will be activated when the user presses the Alt key plus this character.

Value

value

INT



This is the value of the button. Normally, this has the same value as **Off Value**. When the button is pushed (or the `start()` function is called

On Value

onValue

INT

The value that **Value** will be set to when the button is pushed. Default is 1.

Off Value

offValue

INT

The value that **Value** is most of the time. Default is 0.

On Time	onTime	INT
---------	--------	-----

The time in milliseconds that **Value** will be held at **On Value**.



Automatic	auto	BOOLEAN
-----------	------	---------

If you turn this off, you will need to call the `start()` function manually in a script.



Image Path	path	STRING
------------	------	--------

The path to the image that this button shows.

Icon-Text Spacing	iconTextGap	INT
-------------------	-------------	-----

The space between the image and the text.

Horizontal Alignment	horizontalAlignment	INT
----------------------	---------------------	-----

The horizontal alignment of the button contents (image and text).

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Vertical Alignment	verticalAlignment	INT
--------------------	-------------------	-----

The vertical alignment of the button contents (image and text).

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Background 3D?	background3D	BOOLEAN
----------------	--------------	---------

If you turn this off, the background of the button will be drawn flat. You may need to change the background color to see the effect.



Fill Area?	contentAreaFilled	BOOLEAN
------------	-------------------	---------

If you turn this off, the button will not fill in its background. (it will be transparent). Make sure Opaque is off as well.



Border Painted?	borderPainted	BOOLEAN
-----------------	---------------	---------

If you turn this off, the button won't draw its border.



Rollover?	rolloverEnabled	BOOLEAN
-----------	-----------------	---------

If you turn this off, the mouseover effect of the button won't occur.



Disabled Image Path	disabledPath	STRING
---------------------	--------------	--------



Normally when a button is disabled, its image is merely grayed out. If you set this property, this image will be used instead.

Margin	margin	INSETS
--------	--------	--------



This determines the minimum amount of padding between the button contents and the button border.

Horizontal Text Position

horizontalTextPosition

INT



This is the position of the button's text *relative* to its image.

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Vertical Text Position

verticalTextPosition

INT



This is the position of the button's text *relative* to its image.

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Styles

styles

DATASET



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

[key](#)

- keyPressed
- keyReleased
- keyTyped

[action](#)

- actionPerformed

[mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased

[mouseMotion](#)

- mouseDragged
- mouseMoved

[propertyChange](#)

- propertyChange

- focus

- focusGained
- focusLost

Scripting Functions

start()

Starts the momentary button as if it were pushed in 'automatic' mode. Technically, this sets **Value** to **On Value**, waits for **On Time** milliseconds, and then sets **Value** to **Off Value**. See the component description above for an example.

The following functions are common to all components that allow dynamic properties:

getPropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window  
val = event.source.getPropertyValue("MyProperty")  
fpmi.gui.messageBox(val)
```

setPropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15  
event.source.setPropertyValue("MyProperty", 15)
```



CheckBox Component

Icon in toolbar:

Description

A CheckBox is a familiar component that represents a bit - it is either on (selected) or off (not selected). It is functionally equivalent to the Toggle Button component.

Check Box (Off)

Check Box (On)

Properties

Text	text	STRING	
-------------	------	--------	--

The text that this checkbox displays.

Selected	selected	BOOLEAN	
-----------------	----------	---------	--

This will be true when the Checkbox is checked, and false otherwise.

Styles	styles	DATASET	
---------------	--------	---------	------

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [item](#)
 - itemStateChanged
- [key](#)
 - keyPressed
 - keyReleased
 - keyTyped
- [action](#)
 - actionPerformed
- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)

- mouseDragged
- mouseMoved
- [propertyChange](#)
 - propertyChange
- [focus](#)
 - focusGained
 - focusLost

Scripting Functions

The following functions are common to all components that allow dynamic properties:

get PropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

set PropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15
event.source.setPropertyValue("MyProperty", 15)
```



Radio Button Component

Icon in toolbar:

Description

The radio button is similar to the CheckBox component, except for one special property. All radio buttons in the same [container](#) (including the *Root Container*) will automatically be mutually exclusive. This means that only one radio button can be selected at a time. Radio buttons are a good way to let the user choose just one of a number of options. ([Dropdown Lists](#) are another good way to do this)



Properties

Text	text	STRING	
-------------	------	--------	--

The text that is displayed in this radio button.

Selected	selected	BOOLEAN	
-----------------	----------	---------	--

This property will be `true` if the radio button is selected, and `false` otherwise.

Styles	styles	DATASET	
---------------	--------	---------	--

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [item](#)
 - itemStateChanged
- [key](#)
 - keyPressed
 - keyReleased
 - keyTyped
- [action](#)
 - actionPerformed
- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased

- [mouseMotion](#)

- mouseDragged
- mouseMoved

- [propertyChange](#)

- propertyChange

- [focus](#)

- focusGained
- focusLost

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



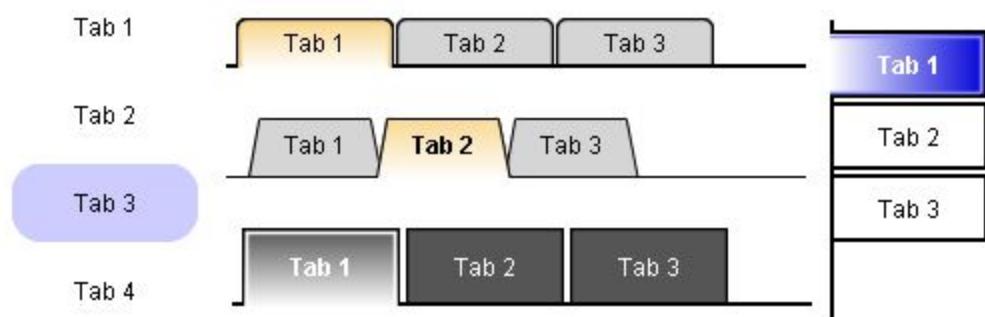
Tab Strip Component

Icon in toolbar:

Description

The Tab Strip component lets you easily add a tab strip to your project. A tab strip can be used anywhere that a user needs to be able to select between multiple windows or screens. It is most commonly used in a docked window to provide window navigation. To support this typical use-case, the tab strip has two **navigation modes**:

- **Swap to Window** (default): The tab strip will automatically call `fpmi.nav.swapTo()` with the name of the selected tab. This facilitates very easy navigation for most common FactoryPMI projects.
- **Disabled**: The tab strip doesn't do anything when the tab selection changes. Users can implement their own via data bindings or by responding to the `propertyChange` scripting event.



The tab strips visual style is highly customizable. There are different rendering styles, and things such as fonts, colors, line thicknesses, hover colors, and gradients are customizable within each rendering style. Use the Tab Strip's customizer to come up with a style that suits your project, as well as to manage the tabs that are present. The tabs and their styles are all stored in a DataSet property (called `Tab Data`), so they can be modified at runtime as well.

Properties

Selected Tab

selectedTab

STRING



Name of the selected tab. This is also the name of the window that, if it exists, will be swapped to when this tab is pressed if the Navigation Mode = "Swap To Window"

Tab Data

tabData

DATASET

A DataSet that holds the configuration that determines what tabs exist, and what they look like. The columns that the TabStrip understand are:

- NAME
- DISPLAY_NAME
- HOVER_COLOR
- SELECTED_FOREGROUND_COLOR
- SELECTED_BACKGROUND_COLOR
- SELECTED_FONT
- SELECTED_GRADIENT_START_COLOR
- SELECTED_GRADIENT_END_COLOR
- UNSELECTED_FOREGROUND_COLOR
- UNSELECTED_BACKGROUND_COLOR

- UNSELECTED_FONT
- UNSELECTED_GRADIENT_START_COLOR
- UNSELECTED_GRADIENT_END_COLOR
- USE_SELECTED_GRADIENT
- USE_UNSELECTED_GRADIENT
- MOUSEOVER_TEXT

Orientation	orientation	INT
--------------------	-------------	-----

Orientation of the tab strip.

Possible values are:

- Top (0)
- Left (1)
- Bottom (2)
- Right (3)

Navigation Mode	navigationMode	INT
------------------------	----------------	-----

Navigation mode. Disabled does nothing when a tab is pressed, other than fire a propertyChangeEvent. Swap to window swaps to the window whose name corresponds to the name of the selected tab, provided that window exists.

Possible values are:

- Disabled (0)
- Swap to window (1)

Renderer	renderer	INT
-----------------	----------	-----

The renderer to use when rendering tabs.

Possible values are:

- Simple (0)
- Fancy (1)
- Folder (2)

Size Mode	sizeMode	INT
------------------	----------	-----

The sizing mode tabs use when deciding their size. Automatic means every tab is the same fixed size. Individual lets each tab decide its own size based on the size of its text.

Possible values are:

- Automatic (0)
- Individual (1)

Intertab Space	interTabSpace	INT
-----------------------	---------------	-----

The amount of space between each tab.

Text Padding	textPadding	INT
---------------------	-------------	-----

Padding on each side of the text inside a tab.

Rounding Radius	roundingRadius	INT
------------------------	----------------	-----

Rounding radius for the tab corners.

Separator Thickness	separatorThickness	FLOAT
----------------------------	--------------------	-------

Thickness of the line drawn across the bottom and around each tab.

Separator Color	separatorColor	COLOR
------------------------	----------------	-------

Color of the line drawn across the bottom and around each tab.

Customizers

- Tab Strip Customizer. Allows you to add and remove tabs to the tab strip, as well as modify and preview the style.
- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



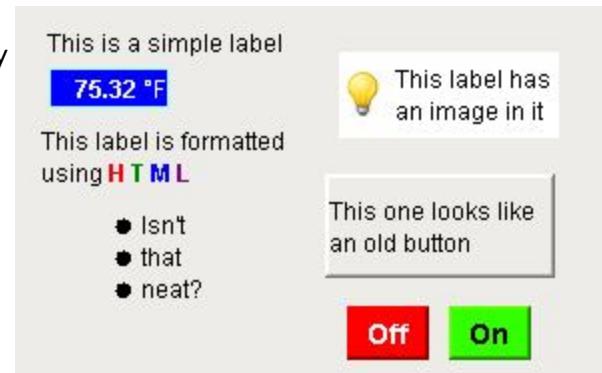
Label Component

Icon in toolbar: Label

Description

The Label is one of the most versatile components in FactoryPMI. It will probably be the component that you use most often. It can display text, images, or both. Its text can be HTML formatted (like most components). It can even be made to respond to user interaction through its events.

Labels are one of the most common Components that you will want to add [Custom Properties](#) to. For instance, put an integer custom property "state" on a label, and then bind the text to be "On" when the state=1 and "Off" otherwise. Bind the background color to be red when the state is 0, and green when the state is 1. Now you have a re-usable binary state indicator. You can see how the flexibility of bindings and custom properties make the Label extremely versatile.



Properties

Text

text

STRING



The main property of a label, controls what the label displays.

Image Path

path

STRING



Labels can contain images, as well. Use this to specify an image to display.

Disabled Image Path

disabledPath

STRING



Just like Buttons, Labels can display a different image when they are disabled, instead of a gray-ed out version of their standard image. Use this property to specify that image.

Icon-Text Spacing

iconTextGap

INT

Controls the space between the image and the text of the label.

Horizontal Alignment

horizontalAlignment

INT

Controls the horizontal alignment of the Label's contents (Image and Text).

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Horizontal Text Position

horizontalTextPosition

INT

Controls the horizontal alignment of the text *relative to the image*.

Possible values are:

- Left (2)

- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Vertical Alignment	verticalAlignment	INT
---------------------------	-------------------	-----

Controls the vertical alignment of the Label's contents (Image and Text).

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Vertical Text Position	verticalTextPosition	INT
-------------------------------	----------------------	-----

Controls the vertical alignment of the text *relative to the image*.

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Rotation	rotation	INT
-----------------	----------	-----

Controls the rotation of the label. Make sure to size the label large enough to contain its rotated version.

Antialias	antialias	BOOLEAN	
------------------	-----------	---------	---

Turn this on for extra-high quality text rendering. Looks good with very large fonts, for things like headers. Default: false.

Styles	styles	DATASET	 
---------------	--------	---------	---

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)

- `propertyChange`

Scripting Functions

The following functions are common to all components that allow dynamic properties:

`getPropertyValue(PropName)`

Retrieve the current value of the dynamic property named `PropName`.

Parameters

`PropName`

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

`setPropertyValue(PropName, Value)`

Set the value of the dynamic property named `PropName` to the value `Value`.

Parameters

`PropName`

The name (a string) of the dynamic property to change.

`Value`

The value to change the property to.

Example:

```
# Change the value to 15
event.source.setPropertyValue("MyProperty", 15)
```



Numeric Label Component

Icon in toolbar:

Description

This component is a specialized label designed to display a number. It can include units, and has an integrated number format string. By default the number is displayed bold and the units are not. This can be customized, see the **Prefix** and **Suffix** expert properties. This label's text is constructed as follows:

87.6 **87.63°F**

88ppm **8.76pH**

\$87.63 **87.632**

Prefix + numberFormat(Value, Pattern) + Suffix + Units

It is important to note that you could customize the standard Label component using custom properties and bindings to mimic this component *exactly*. If this component doesn't do something that you need, you can make your own numeric label and use it everywhere in your project.

Properties

Value

value

DOUBLE



The numeric value that this label displays.

Number Format Pattern

pattern

STRING

The number format pattern used to format the **Value** for display. The default is "#,##0.##", which will display up to two digits past the decimal point if applicable. If you want to display *exactly* two places past the decimal point, use "#,##0.00". See the [numberFormat](#) expression for the complete syntax of a number format pattern string.

Units

units

STRING

A string to display after the value, intended for engineering units, like °F, pH, gal/sec, etc.

Prefix

prefix

STRING



This string comes before the text. This can be useful if your number needs something before it, like a dollar sign. Be default, this is "<HTML>", which makes the number appear **bold**.

Suffix

suffix

STRING



This string comes after the value, and before the units. By default this is "", which ends the bold started by the default suffix.

Text

text

STRING



This read-only property is the text that the label displays. The text is constructed as follows:

Prefix + numberFormat(Value, Pattern) + Suffix + Units

Image Path

path

STRING



Just like the normal Label component, the numeric label can have an image.

Disabled Image Path

disabledPath

STRING



Just like the normal Label component, the numeric label can have a disabled image.

Icon-Text Spacing	iconTextGap	INT
--------------------------	-------------	-----

The spacing between the text and the label.

Horizontal Alignment	horizontalAlignment	INT
-----------------------------	---------------------	-----

The horizontal alignment of this label's contents.

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Horizontal Text Position	horizontalTextPosition	INT
---------------------------------	------------------------	-----

The horizontal position of the text relative to the image.

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Vertical Alignment	verticalAlignment	INT
---------------------------	-------------------	-----

The vertical alignment of the contents of the label.

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Vertical Text Position	verticalTextPosition	INT
-------------------------------	----------------------	-----

The vertical position of the text relative to the image.

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Rotation	rotation	INT
-----------------	----------	-----

The amount in degrees to rotate this label.

Antialias	antialias	BOOLEAN
------------------	-----------	---------

If true, large text will appear smoother.

Styles	styles	DATASET
---------------	--------	---------

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased

- [mouseMotion](#)

- mouseDragged
- mouseMoved

- [propertyChange](#)

- propertyChange

Scripting Functions

The following functions are common to all components that allow dynamic properties:

get PropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

set PropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15
event.source.setPropertyValue("MyProperty", 15)
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Multi-State Indicator Component

Icon in toolbar: Auto

Description

The Multi-State Indicator is just a [Label](#) component with a preset style. This component is taking advantage of the [styles feature](#).



This component has a State property that drives the style, eg. whether the background is green or red.

Properties

State	state	INT	
--------------	-------	-----	--

The current state of the component. This property drives the style of the component.

Text	text	STRING	
-------------	------	--------	--

The main property of a label, controls what the label displays.

Image Path	path	STRING	
-------------------	------	--------	--

Labels can contain images, as well. Use this to specify an image to display.

Disabled Image Path	disabledPath	STRING	
----------------------------	--------------	--------	--

Just like Buttons, Labels can display a different image when they are disabled, instead of a gray-ed out version of their standard image. Use this property to specify that image.

Icon-Text Spacing	iconTextGap	INT	
--------------------------	-------------	-----	--

Controls the space between the image and the text of the label.

Horizontal Alignment	horizontalAlignment	INT	
-----------------------------	---------------------	-----	--

Controls the horizontal alignment of the Label's contents (Image and Text).

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Horizontal Text Position	horizontalTextPosition	INT	
---------------------------------	------------------------	-----	--

Controls the horizontal alignment of the text *relative to the image*.

Possible values are:

- Left (2)
- Center (0)
- Right (4)
- Leading (10)
- Trailing (11)

Vertical Alignment

verticalAlignment

INT

Controls the vertical alignment of the Label's contents (Image and Text).

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Vertical Text Position

verticalTextPosition

INT

Controls the vertical alignment of the text *relative to the image*.

Possible values are:

- Top (1)
- Center (0)
- Bottom (3)

Rotation

rotation

INT

Controls the rotation of the label. Make sure to size the label large enough to contain its rotated version.

Antialias

antialias

BOOLEAN



Turn this on for extra-high quality text rendering. Looks good with very large fonts, for things like headers. Default: false.

Styles

styles

DATASET



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased

- [mouseMotion](#)

- mouseDragged
- mouseMoved

- [propertyChange](#)

- propertyChange

Scripting Functions

The following functions are common to all components that allow dynamic properties:

get PropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window  
val = event.source.getPropertyValue("MyProperty")  
fpmi.gui.messageBox(val)
```

set PropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15  
event.source.setPropertyValue("MyProperty", 15)
```



LED Display Component

Icon in toolbar:

Description

The LED display is a display component like a Label that looks like varying styles of physical LED displays. The three styles are:

- **7-Segment.** This style uses seven segments to display numerals 0-9, and most characters. Some characters like 'w' and 'v' do not display very well, simply due to lack of options with 7 segments.
- **14-Segment.** Like the 7-segment display, but adds more segments for higher quality alphanumeric characters. All letters A-Z are supported, but only in upper-case.
- **5x7 Matrix.** This style uses a matrix of dots to represent each character. All letters are supported in both upper and lower cases.



This component functions like both the standard [Label](#) component and the [Numeric Label](#) component, depending on this component's **mode** property, which can be set to **Numeric** or **Alphanumeric**.

Properties

Background Color

background

COLOR

The color of the background of the component.

LED Lit

glyphForeground

COLOR

The color of lit LED segments

LED Unlit

glyphBackground

COLOR

The color of unlit LED segments

Style

style

INT

The visual style of the display.

Possible values are:

- 7 Segment (7)
- 14 Segment (14)
- 5x7 Matrix (34)

Mode

mode

INT

The mode of the display.

Possible values are:

- Numeric (0)
- Alphanumeric (1)

Value

value

DOUBLE



The numeric value of the display, used when **Mode** is **Numeric**

Number Format Pattern

numberFormat

STRING

The number formatting string used to format the value.

Text

text

STRING



The text value of the display, used when **Mode** is **Alphanumeric**

Horizontal Alignment

horizontalAlignment

INT

Determines the alignment of the display's contents along the X axis

Possible values are:

- Left (2)
- Center (0)
- Right (4)

Letter Gap

gap

FLOAT



The percentage of the height to be used as an inter-character spacing

Margin

margin

INSETS



The margin for the interior of the display

Styles

styles

DATASET



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

[mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased

[mouseMotion](#)

- mouseDragged
- mouseMoved

[propertyChange](#)

- propertyChange

Scripting Functions

None.



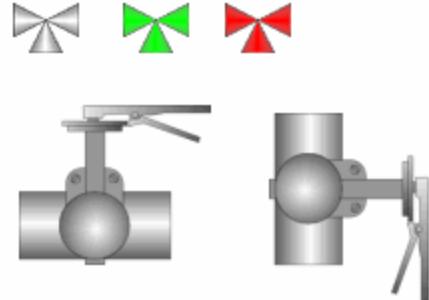
Image Component

Icon in toolbar:

Description

The Image component is a powerful tool, allowing you to quickly make visually intuitive HMI screens without spending time in an image editing program. Like the Label component, the Image component displays an image. However, the Image component lets you alter the look of that image dynamically. Some features include:

- **Scaling.** Scale your images up or down just by resizing the image component.
- **Color Tinting.** Tint entire images to display status. Tint your motor red or green depending on its status.
- **Color Swapping.** Swap one color for another. Have region of an image that you want a certain color? No problem.
- **Rotation.** Rotate images any which way. Hook the rotation up to a Timer, and get rotating animation.



NOTE. Images in FactoryPMI are stored *on the gateway server*, not in the window. This allows you to replace an image globally in your projects. To learn how to manage your images, see the [Image Tool](#) section of the Designer Reference.

Properties

Rotation

rotation

INT

The amount the image is rotated, in degrees. Note that if you need to rotate and stretch your image, stretch mode `Bounds` isn't the best choice, because the corners of the image will get clipped. Use stretch modes `% Bounds` or `Parameters` instead.

Image Path

path

STRING



The path of the image to display.

Disabled Image Path

disabledPath

STRING



The path if the image to display if the component is disabled. If you don't set this, a grayed out version of the normal image will be used.

Stretch Mode

stretchMode

INT

Determines if this image stretches, and if so, to what size.

Possible values are:

- **No Stretch (0).**
- **Bounds (1).** Stretch the image to the size of the component.
- **% Bounds (3).** Stretch the image to a percentage of the size of the component. Useful for rotation.
- **Parameters (2).** Stretch the image to the width and height specified (see the next two properties).

Stretch Width	stretchWidth	INT	
----------------------	--------------	-----	---

The width to stretch the image to in **Stretch Mode** Parameters.

Stretch Height	stretchHeight	INT	
-----------------------	---------------	-----	---

The height to stretch the image to in **Stretch Mode** Parameters.

Color Swap Filter	useColorSwap	BOOLEAN	
--------------------------	--------------	---------	---

If true, any pixels in the image that are the same color as the **Swap From** color will be changed to the **Swap To** color.

Swap From	swapFromColor	COLOR	
------------------	---------------	-------	---

See **Color Swap Filter**.

Swap To	swapToColor	COLOR	
----------------	-------------	-------	---

See **Color Swap Filter**.

Tint Filter	useTint	BOOLEAN	
--------------------	---------	---------	---

If true, the entire image will be tinted the color of **Tint Color**.

Tint Color	tintColor	COLOR	
-------------------	-----------	-------	---

See **Tint Filter**.

Flip Horizontal	flipHorizontal	BOOLEAN	
------------------------	----------------	---------	--

Flips the image horizontally.

Flip Vertical	flipVertical	BOOLEAN	
----------------------	--------------	---------	--

Flips the image vertically.

Styles	styles	DATASET	 
---------------	--------	---------	---

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved

- [propertyChange](#)

- propertyChange

Scripting Functions

The following functions are common to all components that allow dynamic properties:

get PropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window  
val = event.source.getPropertyValue("MyProperty")  
fpmi.gui.messageBox(val)
```

set PropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15  
event.source.setPropertyValue("MyProperty", 15)
```

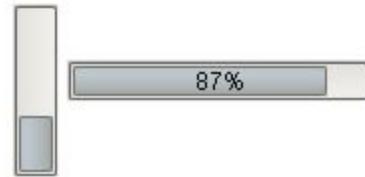


Progress Bar Component

Icon in toolbar:

Description

Visually indicates the progress of some task. Can be used to display any value that has an upper and lower bound.



Properties

Value	value	INT	
--------------	-------	-----	--

The current value of the progress bar. Needs to be \geq to **Minimum** and \leq to **Maximum**.

Maximum	maximum	INT	
----------------	---------	-----	--

The maximum value of the **Value** property.

Minimum	minimum	INT	
----------------	---------	-----	--

The minimum value of the **Value** property.

Horizontal?	horizontal	BOOLEAN	
--------------------	------------	---------	--

If true, will be drawn horizontally. If false, will be drawn vertically.

Show Percentage?	stringPainted	BOOLEAN	
-------------------------	---------------	---------	--

If true, the percentage will be shown as a number.

Indeterminate?	indeterminate	BOOLEAN	
-----------------------	---------------	---------	--

If true, the **Value** will be ignored, and the progress bar will display an animation showing that something is happening, but it is uncertain how far along it is.

Direction	Direction	INT	
------------------	-----------	-----	--

Controls the direction of positive growth for the progress bar. Only works for horizontal progress bars..

Possible values are:

- Left to Right (0)
- Right to Left (2)

Styles	styles	DATASET	
---------------	--------	---------	--

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



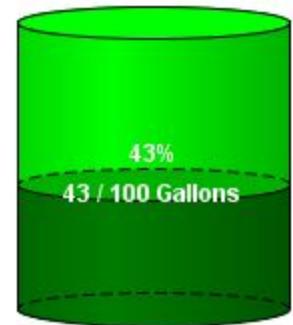
Cylindrical Tank Component

Icon in toolbar:



Description

A component that looks like a 3D cylindrical tank, with some liquid inside. The liquid rises and falls as the **Value** property changes.



Properties

Rotation

rotation

INT

Controls the rotation (in degrees) of this tank.

Anti Alias

antiAlias

BOOLEAN



Controls whether or not this component is drawn with Anti-Alias. Default: true.

Value

value

INT



The value of the tank represents the amount of liquid in the tank. Should be less than the capacity.

Capacity

capacity

INT



The total capacity of the tank.

Units

units

STRING

The name of the units (e.g. "Gallons") of the contents of the tank. Used in conjunction with the **Show Value** property.

Show Value

showValue

BOOLEAN

If true, the value, capacity, and units are shown (e.g. 34/200 Gallons).

Show Percentage

showPercent

BOOLEAN

If true, the percentage full is shown.

Font Color

fontColor

COLOR

The color of the font.

Tank Color

tankColor

COLOR



The color of the tank.

Liquid Color

liquidColor

COLOR



The color of the liquid in the tank.

Styles

styles

DATASET

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events• [mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased

• [mouseMotion](#)

- mouseDragged
- mouseMoved

• [propertyChange](#)

- propertyChange

Scripting Functions

None.



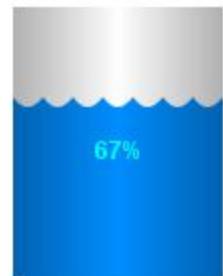
Level Indicator Component

Icon in toolbar:



Description

A component that displays the level of fullness of some container. This is basically a visually simplified version of the [Cylindrical Tank](#) component. By turning on and off the **Gradient** and **Liquid Waves** properties, you can control how fancy this component looks. The main use for this is to put behind images of tanks with transparent cutaways, to show how full the tank is.



Properties

Value	value	INT	
The value of the level indicator controls how 'full' it looks. Should be less than or equal to the Capacity .			
Capacity	capacity	INT	
The maximum Value of the level indicator.			
Units	units	STRING	
The units of the contents of this vessel. Used in conjunction with Show Value .			
Show Value	showValue	BOOLEAN	
If true, the value, capacity, and units will be shown.			
Show Percentage	showPercent	BOOLEAN	
If true, the percentage full will be shown.			
Gradient	gradient	BOOLEAN	
If true, the level indicator will be drawn with a gradient to make it look 3D.			
Liquid Waves	waves	BOOLEAN	
If true, the top of the fill line will look like 'waves' to indicate that it is a liquid.			
Wave Length	waveLength	INT	
The length of each wave, in pixels.			
Wave Height	waveHeight	INT	
The height of each wave, in pixels.			
Font Color	fontColor	COLOR	
The color of the font for Show Value and Show Percentage .			
Anti Alias	antiAlias	BOOLEAN	
Controls if this component is drawn with Anti-Alias on or off.			

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.



Barcode Component

Icon in toolbar:

Description

The barcode component displays some text as a barcode. The supported formats are:

- Code 128
- Code 39
- Extended Code 39
- Codabar
- Interleaved Code 25
- MSI
- EAN-13
- EAN-8



34-2382-298

Properties

Barcode Background

barcodeBackground

COLOR

The color behind the barcode proper (not the displayed text, if any)

Code

code

STRING



The code (text) do display as a barcode.

Barcode Format

barcodeType

INT

The format of the barcode.

Possible values are:

- Code 39 (0)
- Code 39 (narrow) (1)
- Extended Code 39 (2)
- Extended Code 39 (narrow) (3)
- Code 128 (4)
- Codabar (5)
- Codabar (narrow) (6)
- Interleaved Code 25 (7)
- Interleaved Code 25 (narrow) (8)
- MSI (9)
- EAN-13 (10)
- EAN-8 (11)

Show Text?

showText

BOOLEAN

If true, the human-readable version of the **Code** will be displayed beneath the barcode.

Check Digit

checkDigit

BOOLEAN

If true, a check digit will be encoded into the barcode. Some barcode formats force this to be true, and some don't support it.

Barcode Height

barcodeHeight

INT

The height (in pixels) of the barcode.

Narrowest Bar Width

narrowestBarWidth

INT

The narrowest width (in pixels) of a barcode bar.

Rotation

angleDegrees

DOUBLE



The amount (in degrees) to rotate the barcode

Customizer

None.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.



Meter Component

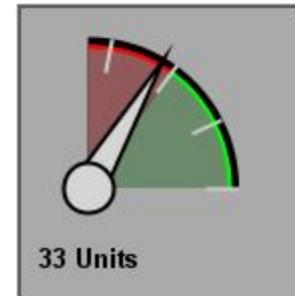
Icon in toolbar:



Description

A meter display shows a value on a needle-gauge. The gauge's range can be broken up into five intervals. The intervals can have their own edge and background colors.

How the meter looks is affected by its appearance properties. You can modify colors, thicknesses, start and extend angles, needle size, etc to get the meter that you want. For example, the meter on the far right of the example has a Meter Angle Extent of 90°, a Meter Angle of 45°, a reversed range, and 2 intervals with background colors.



Properties

Value

value

DOUBLE



The current value of the meter. The needle in the meter will rotate to display this value, and the value label will show the value as a number.

Units

units

STRING

The units that this meter measures. Used in the value label.

Dial Background

dialBackground

COLOR

The background of the dial face of the meter.

Needle Color

needleColor

COLOR

The color of the needle.

Needle Size

needleSize

FLOAT

The size of the needle.

Needle Stroke Color

needleStrokeColor

COLOR

The color of the needle's outline stroke.

Needle Stroke Size

needleStrokeSize

FLOAT

The size of the needle's outline stroke.

Value Color

valueColor

COLOR

The color of the value and unit label.

Tick Label Color

tickLabelColor

COLOR

Color of the numeric labels near the tick marks.

Tick Color

tickColor

COLOR

The color of the tick marks.

Tick Size	tickSize	DOUBLE
------------------	----------	--------

The distance between tick marks.

Show Tick Labels?	ticks	BOOLEAN
--------------------------	-------	---------

If false, tick marks labels won't be shown.

Value Label Font	valueFont	FONT
-------------------------	-----------	------

The font of the value display.

Tick Label Font	labelFont	FONT
------------------------	-----------	------

The font of the tick mark labels.

Value Format	valueLabelFormat	STRING
---------------------	------------------	--------

A number format pattern to use for the meter's value.

Tick Format	tickLabelFormat	STRING
--------------------	-----------------	--------

A number format pattern to use for the meter's tick marks.

Arc Width	arcWidth	FLOAT
------------------	----------	-------

The thickness of the colored arcs that represent intervals.

Meter Angle Extent	meterAngleExtent	INT
---------------------------	------------------	-----

The angle (in degrees) as a portion of a circle that will be used for the meter.

Meter Angle	meterAngle	INT
--------------------	------------	-----

The angle (in degrees) that the center of the meter extend will point to. Straight up is 90°, increasing counter-clockwise. This lets you rotate the meter.

Dial Shape	dialType	INT
-------------------	----------	-----

Controls the shape of the dial. Determines how the area of the circle that isn't used (See **Meter Angle Extent** is drawn.

Possible values are:

- Chord (1)
- Circle (0)
- Pie (2)

Overall Low Bound	overallLow	DOUBLE
--------------------------	------------	--------

The overall low bound of the range that the value can be in.

Overall High Bound	overallHigh	DOUBLE
---------------------------	-------------	--------

The overall high bound of the range that the value can be in.

Reverse Range?	reverseRange	BOOLEAN
-----------------------	--------------	---------

If this is true, the value range will be shown reverse. This means that clockwise meter movement will be a decrease in value.

Interval 1 Low	interval1Low	DOUBLE
-----------------------	--------------	--------

The lower bound of this interval.

Interval 1 High	interval1High	DOUBLE
------------------------	---------------	--------

The upper bound of this interval.

Interval 1 Outline	interval1Outline	COLOR
---------------------------	------------------	-------

The color to draw the outline of this interval's arc.

Interval 1 Background	interval1Background	COLOR
------------------------------	---------------------	-------

The color to fill the wedge of this interval.

Interval 2 Low	interval2Low	DOUBLE
-----------------------	--------------	--------

The lower bound of this interval.

Interval 2 High	interval2High	DOUBLE
------------------------	---------------	--------

The upper bound of this interval.

Interval 2 Outline	interval2Outline	COLOR
---------------------------	------------------	-------

The color to draw the outline of this interval's arc.

Interval 2 Background	interval2Background	COLOR
------------------------------	---------------------	-------

The color to fill the wedge of this interval.

Interval 3 Low	interval3Low	DOUBLE
-----------------------	--------------	--------

The lower bound of this interval.

Interval 3 High	interval3High	DOUBLE
------------------------	---------------	--------

The upper bound of this interval.

Interval 3 Outline	interval3Outline	COLOR
---------------------------	------------------	-------

The color to draw the outline of this interval's arc.

Interval 3 Background	interval3Background	COLOR
------------------------------	---------------------	-------

The color to fill the wedge of this interval.

Interval 4 Low	interval4Low	DOUBLE
-----------------------	--------------	--------

The lower bound of this interval.

Interval 4 High	interval4High	DOUBLE
------------------------	---------------	--------

The upper bound of this interval.

Interval 4 Outline	interval4Outline	COLOR
---------------------------	------------------	-------

The color to draw the outline of this interval's arc.

Interval 4 Background	interval4Background	COLOR
------------------------------	---------------------	-------

The color to fill the wedge of this interval.

Interval 5 Low	interval5Low	DOUBLE
-----------------------	--------------	--------

The lower bound of this interval.

Interval 5 High	interval5High	DOUBLE
------------------------	---------------	--------

The upper bound of this interval.

Interval 5 Outline	interval5Outline	COLOR
---------------------------	------------------	-------

The color to draw the outline of this interval's arc.

Interval 5 Background	interval5Background	COLOR
------------------------------	---------------------	-------

The color to fill the wedge of this interval.

Customizer

None.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



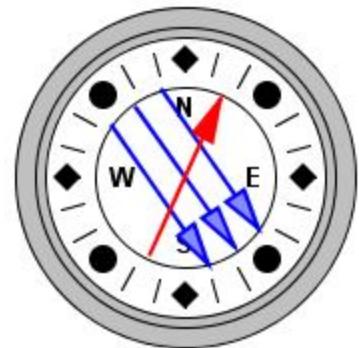
Compass Component

Icon in toolbar:

Description

The compass is a component that displays up to three needles at once on a cardinal direction compass. This can be useful for plotting anything that has a cardinal direction, such as the wind direction.

Each needle can be one of 9 different styles. Use the "Disabled" style to turn off any needle.



Properties

Value 1	value1	DOUBLE	
----------------	--------	--------	--

Value 1 for the compass.

Value 1 Needle	value1Needle	INT	
-----------------------	--------------	-----	--

The needle type for this value.

Possible values are:

- Disabled (-1)
- Arrow (0)
- Line (1)
- Long (2)
- Pin (3)
- Plum (4)
- Pointer (5)
- Ship (6)
- Wind (7)
- Middle Pin (9)

Value 1 Color	value1Color	COLOR	
----------------------	-------------	-------	--

The main color for Value 1's needle

Value 1 Outline	value1OutlineColor	COLOR	
------------------------	--------------------	-------	--

The outline color for value 1's needle

Value 2	value2	DOUBLE	
----------------	--------	--------	--

Value 2 for the compass.

Value 2 Needle	value2Needle	INT	
-----------------------	--------------	-----	--

The needle type for this value.

Possible values are:

- Disabled (-1)
- Arrow (0)
- Line (1)
- Long (2)
- Pin (3)
- Plum (4)
- Pointer (5)
- Ship (6)
- Wind (7)
- Middle Pin (9)

Value 2 Color	value2Color	COLOR
----------------------	-------------	-------

The main color for Value 2's needle

Value 2 Outline	value2OutlineColor	COLOR
------------------------	--------------------	-------

The outline color for value 2s needle

Value 3	value3	DOUBLE
----------------	--------	--------

Value 3 for the compass.



Value 3 Needle	value3Needle	INT
-----------------------	--------------	-----

The needle type for this vaule.

Possible values are:

- Disabled (-1)
- Arrow (0)
- Line (1)
- Long (2)
- Pin (3)
- Plum (4)
- Pointer (5)
- Ship (6)
- Wind (7)
- Middle Pin (9)

Value 3 Color	value3Color	COLOR
----------------------	-------------	-------

The main color for Value 3's needle

Value 3 Outline	value3OutlineColor	COLOR
------------------------	--------------------	-------

The outline color for value 3s needle

Label Font	labelFont	FONT
-------------------	-----------	------

The font to use for the compass's labels.

Rose Color	roseColor	COLOR
-------------------	-----------	-------

The background color of the rose.

Rose Highlight	roseHighlightColor	COLOR
-----------------------	--------------------	-------

The highlight color of the rose.

Center Color

centerColor

COLOR

The center color of the compass

Styles

styles

DATASET



Contains the component's styles

Data Quality

dataQuality

INT



The data quality code for any tag bindings on this component.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased

- [mouseMotion](#)

- mouseDragged
- mouseMoved

- [propertyChange](#)

- propertyChange

Scripting Functions

The following functions are common to all components that allow dynamic properties:

get PropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

set PropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15
event.source.setPropertyValue("MyProperty", 15)
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



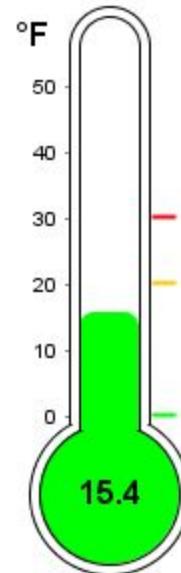
Thermometer Component

Icon in toolbar:



Description

This component displays a temperature value depicted as a level in a mercury thermometer. Three temperature intervals can optionally be defined with their own colors. The mercury will change color based on the range that it is in.



Properties

Value	value	DOUBLE	
--------------	-------	--------	--

The current temperature of the thermometer.

Overall Low Bound	overallLow	DOUBLE	
--------------------------	------------	--------	--

The lowest value that the temperatuer can reach.

Overall High Bound	overallHigh	DOUBLE	
---------------------------	-------------	--------	--

The highest value that the temperature can reach.

Units	units	INT	
--------------	-------	-----	--

The type of this temperature.

Possible values are:

- None (0)
- Fahrenheit (1)
- Celcius (2)
- Kelvin (3)

Thermometer Color	thermometerColor	COLOR	
--------------------------	------------------	-------	--

The color of the outline of the thermometer.

Mercury Color	mercuryColor	COLOR	
----------------------	--------------	-------	--

The color of the mercury. Note that if Use Range Color (see below) is true (true is the default), then the mercury will adopt the interval color that it is in, if any.

Value Color

valueColor

COLOR

The color to paint the numeric display of the temperature.

Value Label Font

valueFont

FONT

The font to use for the numeric display of the temperature.

Interval 1 Low

interval1Low

DOUBLE

The low bound of interval number one.

Interval 1 High

interval1High

DOUBLE

The high bound of interval number one.

Interval 1 Color

interval1Color

COLOR

The color to use for interval number one.

Interval 2 Low

interval2Low

DOUBLE

The low bound of interval number two.

Interval 2 High

interval2High

DOUBLE

The high bound of interval number high.

Interval 2 Color

interval2Color

COLOR

The color to use for interval number two.

Interval 3 Low

interval3Low

DOUBLE

The low bound of interval number three.

Interval 3 High

interval3High

DOUBLE

The high bound of interval number high.

Interval 3 Color

interval3Color

COLOR

The color to use for interval number three.

Thermometer Width

strokeWidth

INT



The width to use for the outline of the thermometer.

Use Range Color

useSubrangePaint

BOOLEAN



If true, then the mercury color will be the color of the range (interval) that it is currently in, if any.
(Default: true)

Follow data in ranges

followDataInSubranges

BOOLEAN



If true, then the temperature axis will zoom into the range that it is currently in. (Default: false)

Customizer

None.

Events• [mouse](#)

- mouseClicked
- mouseEntered

- mouseExited
- mousePressed
- mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Document Viewer Component

Icon in toolbar:



Description

The document viewer is capable of loading and displaying a document that is available over the network at a URL. It is capable of displaying simple HTML pages and RTF pages. Although HTML links will be followed, it is *not* a fully functional interactive web browser. Its HTML support is rudimentary at best, and there is no JavaScript support. See the [openURL\(\)](#) Jython function for a more robust solution for launching webpages, PDFs, etc.

This component is useful for viewing machine manuals or operator protocol in HTML or RTF format. Note that in addition to HTML URLs (like "http://www.google.com"), you can load files as well using the URL format for files. Some examples:

```
"file:///localhost/C:/myfolder/file.txt"  
"file:///MyFileServer/resources/manuals/instructions.r
```



Properties

Page URL

page

STRING



The URL of the page to view.

Content Type

contentType

STRING



The MIME-type of the contents. Used when specifying the contents explicitly using the `text` property. Examples: `text/plain`, `text/html`

Text

text

STRING



The text of the document viewer. Can be set explicitly. Make sure to set the `Content-Type` to match the kind of text that is entered here.

Launch Links Externally

launchLinksExternally

BOOLEAN



If checked, hyperlinks will be launched in an external browser instead of the document viewer itself.

Customizer

None.

Events

- [key](#)

- `keyPressed`
- `keyReleased`
- `keyTyped`

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange
- [focus](#)
 - focusGained
 - focusLost

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



IP Camera Viewer Component

Icon in toolbar:



Description

The IP camera viewing component displays a video stream from a network camera directly in one of your FactoryPMI windows. This can be a very powerful tool for allowing operators to view remote or inaccessible locations. Cameras can provide positive feedback about the state and position of machinery, weather, and other factors.

This component is capable of displaying two types of video:

- **MJPEG** (a.k.a. Motion JPEG) is a streaming video protocol that compresses video frames using standard JPEG compression. Compression rates are quite good, requiring low network bandwidth utilization. Framerates depend greatly on the dimensions of the video, but typically range from 1-20 frames per second.
- **JPEG stills** is not a true video protocol, but is rather the practice of continually refreshing an image that a camera is constantly overwriting. Its simplicity means that many cameras support it (usually along with another protocol). Frame rates are typically lower than MJPEG.



Most network cameras on the market support one, if not both of these protocols. Even better, if you have an existing CCTV camera system, **video server** devices are available that CCTV camera inputs and provide MJPEG streams to the network.

Finding the URL for your network camera's video stream is usually the only challenge in connecting this component. Most, if not all, network cameras have an internal web server, allowing viewers to use web browsers to view their video stream. If you go to that webpage, and look at the HTML source of the page, you should be able to find the URL of the MJPEG or JPEG still stream.

Some examples:

- **Axis 2100** (MJPEG): <http://ip.address.here/axis-cgi/mjpg/video.cgi?resolution=640x480>
- **Panasonic BL-C10A** (MJPEG): <http://ip.address.here/nphMotionJpeg?Resolution=640x480&Quality=Standard>
- **StarDot Netcam** (JPEG stills): <http://ip.address.here/netcam.jpg>

Properties

Video Mode

mode

INT

Possible values are:

- **MJPEG (0)**. Displays a MJPEG video stream.
- **JPEG (1)**. Displays a still JPEG image, refreshing it every `Refresh Rate` milliseconds.

Refresh Rate

refreshRate

INT

The amount of time, in milliseconds, to wait between each image refresh if the mode is **JPEG**.

Use Authentication?

useAuthentication

BOOLEAN

If true, the Username and Password will be used to connect to the URL

Username	username	STRING
The username to authenticate the URL connection, if Use Authentication is true.		
Password	password	STRING
The password to authenticate the URL connection, if Use Authentication is true.		
URL	url	STRING
The HTTP URL locating the video stream.		
Connection Retries	connectRetries	INT
The number of times to try to connect to the URL before giving up.		
Retry Delay	retryDelay	INT
The time, in milliseconds, to wait between connection attempts.		
Show Stats	showStats	BOOLEAN
If true, the Frames per Second and Kilobytes per Second will be shown in the lower left corner of the component.		

Customizer

None.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.

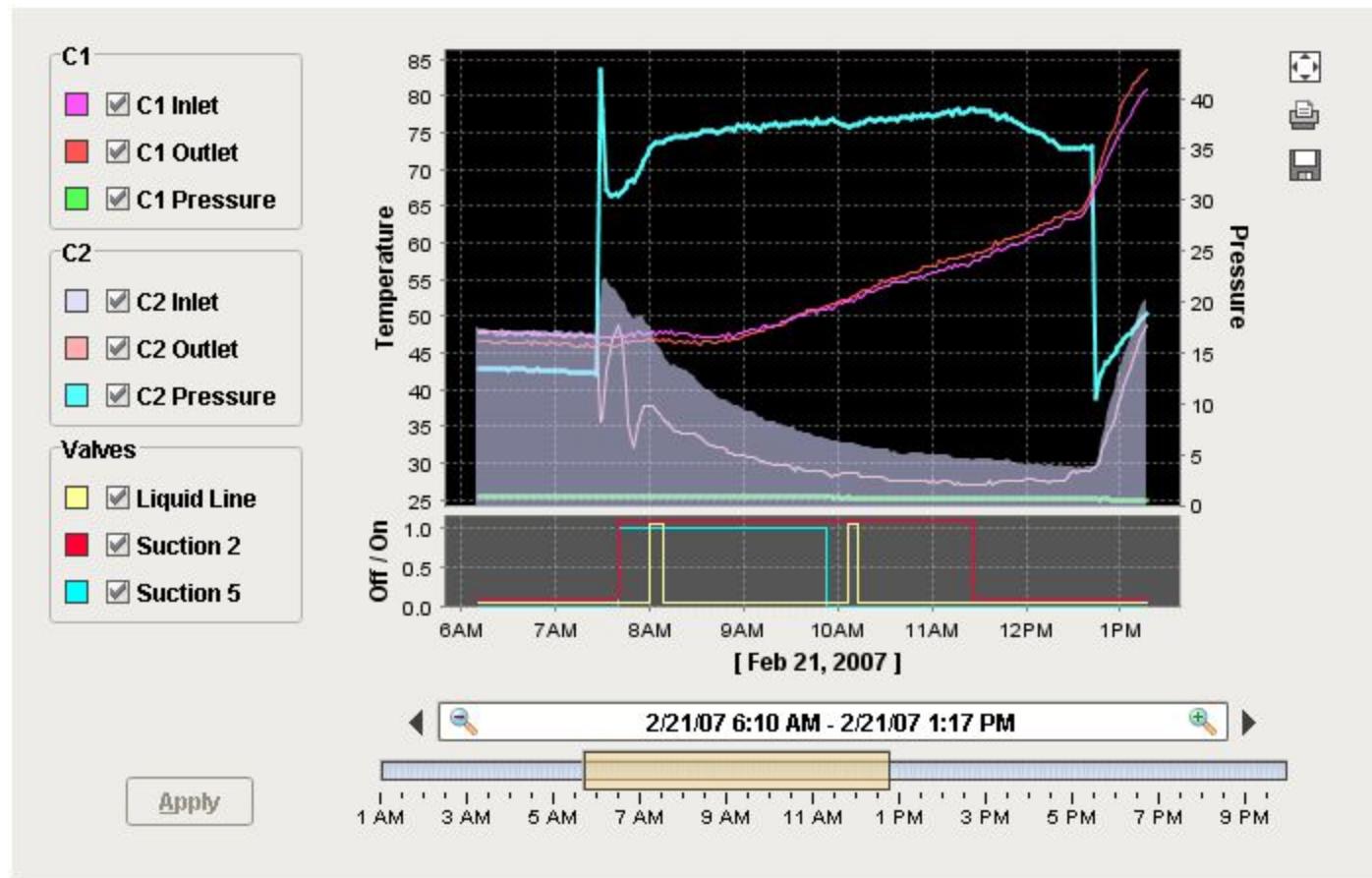


Easy Chart Component

Icon in toolbar:



Description



Introduction & Features

The FactoryPMI Easy Chart (introduced in version 2.0.0) is a very easy to configure timeseries chart component. The main differences between this component and our normal [Chart](#) component are:

- Easy configuration
- Built-in pen selection
- Built-in date range selection.

This eliminates what was often a large integration effort required for the standard chart.

The Easy Chart is not only far easier than the old chart, it is also more powerful. Features include:

- **Everything you loved** about the old (standard) chart, such as:
 - Zoom, Pan, Mark, and X-Trace modes
 - Any number of Y axes
 - Subplots
 - Large numbers of datapoints supported
- **Three modes** of operation:
 - Historical Mode. Integrates the [Date Range](#) component for easy, convenient date range selection.

Did you know...

The Easy Chart component is the result of three years of Inductive Automation support staff helping customers set up chart windows. While we know that this chart component is much easier and more powerful than the old one, we're still open to ideas! If you have a feature

- Realtime Mode. User chooses the window of time, and data scrolls by.
- Manual Mode. You configure the start time and end time for the data however you want

you'd love to see in our charts, let us know!

- **User-selectable pens.** Users can turn on and off pens at will.

- **User-selectable pen colors.**

- **Utility buttons**

- Maximize. Lets the chart take the entire component space, hiding date range and pen selection elements.
- Print. Lets the user print the chart.
- Save for Excel. Saves the underlying chart data as an Excel spreadsheet

- **More Pen Styles.** Ten different pen styles, each with their own options. Line, 3D Line, Area, Digital, Shape, Dot, Bar, etc.

- **Really Easy Configuration.** No SQL. No bindings. No scripts. Everything is configured by the Pen (using the Customizer). No more confusion about DataSets and Series. You browse the database for your pens, and that's it. Pen style, Y-Axis, and Subplots are all options on the pen.

- **Data Density Histogram.** The date range component's data density histogram shows you where there is data, and where there isn't.

Basic Setup

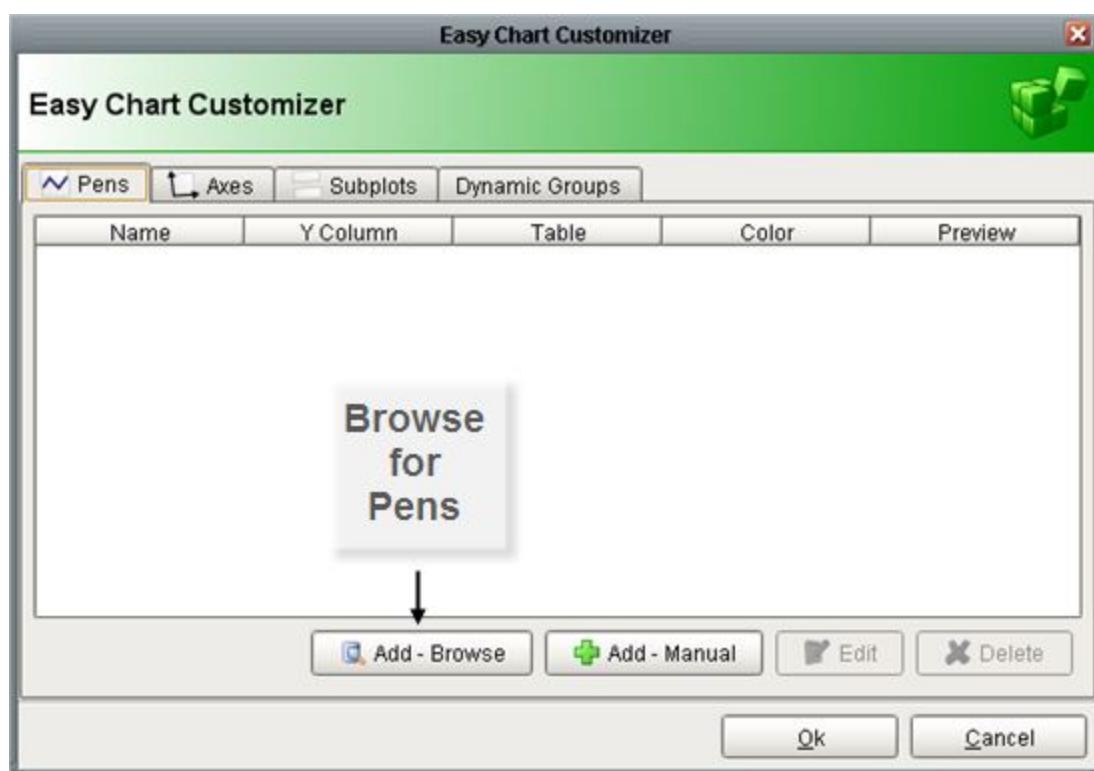
Adding Pens

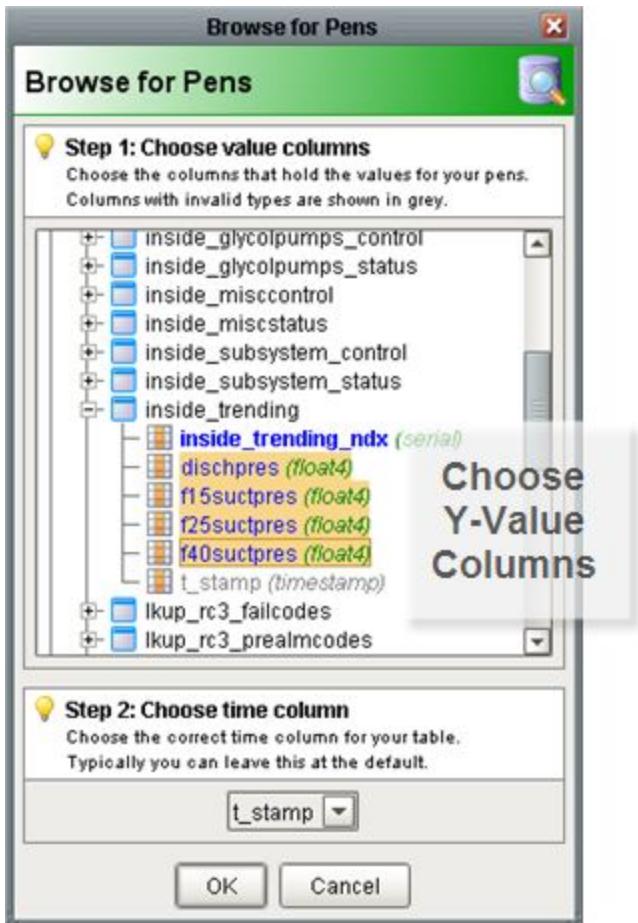
Setting up the Easy Chart is, well, easy. Simply enter the customizer, and start browsing for pens.

What is a Pen?

A pen is a series of datapoints, with the X value being time, and the Y value being a numeric value. The Easy Chart stores lots of configuration options with each pen, like the database table, Y-value column, and X-value column where the data can be found.

Additionally it stores options like the color, style, Y-axis, subplot, etc.





When you are browsing for pens you're looking for the **Y-value columns** in your chart history tables. All columns whose types aren't right for being a Y-value (anything non-numeric) will be shown in gray. When you find your Y-value columns, you can choose your X-value column for the selected pens. Typically, this will be the column named *t_stamp* that FactorySQL created.



Now you will see your pens shown in the customizer, along with a preview of how the pen will look. Select the pen and choose *Edit Pen* to see all of the pen options. Experiment with the visual styles to see what is available. Pen options are mostly self-explanatory, hover your mouse over an option to get details.



At this point you can test your chart out. Click **OK** in the customizer, turn your database traffic on, and go into preview mode. You should see your data when you choose an appropriate date range. You can also turn on and off your pens.

Y-Axes

The easy chart supports any number of Y-axes. To add an axis, go to the **Axes** tab of the chart customizer. When adding an axis, you get a number of options such as the type (numeric or logarithmic), label, color, autorange vs fixed range, and auto-ticks vs fixed ticks. You can also modify the position of the axis, but note that by default the Chart's **Auto Axis Positioning** property is enabled, which means that the chart will balance the axes automatically between left and right depending on demand. As pens are turned on and off by the user, only the axes that are used by visible pens are shown.

After you add your axes, you edit any pens that you want to use your new axes. Simply choose the new axis in the axis dropdown of the pen editing window.

Subplots

The Subplots feature lets you break up the chart's plot area into multiple distinct subplots that share the X axis, but have their own Y axes. This is often useful for digital data, as shown in the screenshot above. By default the chart has 1 subplot (the main plot). To add a new subplot, simply hit the add button in the **Subplots** tab of the chart customizer.

Subplots have relatively few options. The **Weight** option determines how much room the subplot gets relative to the other subplots. For example, in the screenshot above subplot #1's weight is 5, and subplot #2's weight is 1, leading to a 5-to-1 distribution of space. Just like axes, once you add your subplots you should go back to your pens and modify your pens' subplot property for any pens you want to appear on the subplot.

Pen Groups

You can put your pens in groups to break up the pens into some logical separation. For instance, in the screenshot above there are three pen groups: **C1**, **C2**, and **Values**. The group name is used as the titled border for the pens' grouping container. Groups also have another purpose, but it is more advanced and most people won't have to worry about it. For more, read the **Dynamic Pens** section below.

Those are the basics of the chart configuration. Note that there are lots of display options on the chart itself (in the property table, not in the customizer). All of the options are explained below, but feel free to experiment to see what they do.

Chart Modes

The easy chart has three modes of operation that affect how it creates the underlying SQL queries to pull your chart data from the database. These modes affect how the chart determines the date range of the data that will be displayed.

Historical Mode. This is the default mode. In this mode, the user is given a Date Range component to affect the data shown. The data does not poll periodically in this mode, rather, it is queried when the date range changes.

Realtime Mode. In this mode, the user is given the option to choose a span of time, such as "5 minutes", and the last 5 minutes of data will be shown on the chart. In this mode, the **Poll Rate** property is used to determine how often to refresh the data.

Manual Mode. In this mode, you can set the chart's **Start Date** and **End Date** properties however you like in order to affect the date range shown. If you'd rather use two date dropdowns instead of the Date Range component, this is how you'd do it. Note that in this mode the **Poll Rate** property is used, so if you don't want the data to poll periodically, set the **Poll Rate** to 0.

Dynamic Pens

It is often the case that you'll want to make one chart window that services many similar pieces of equipment. For instance, if you have 30 tanks and they all have the same datapoints, you want to be able to use one window for all 30 of them and simply pass the tank number into the chart window as a parameter. There are actually a number of ways to accomplish this, each method suitable for different scenarios. Let's start with the simplest.

Option 1. All data stored in one table

Suppose you have all 30 tanks' data stored to one history table, where there is a column designating the **TankNum**. In this case, simply use the chart's **Where Clause** property. This property is a snippet of SQL WHERE clause that will be applied to all pens. So, put a dynamic property named **TankNum** on your window's Root Container, and bind the chart's **Where Clause** property to an expression like:

```
"TankNum = " + {Root Container.TankNum}
```

Now simply pass the **TankNum** parameter into your window, and your pens will all reflect the data for the correct tank.

Option 2. Multiple tables / Multiple conditions

Suppose your multiplexed chart is representing something more complex, like a bottling line. The historical data involved will likely be stored in multiple tables, and one WHERE clause might not apply to all pens. Or, suppose you added 2 sets of similar pens so that the user can compare and contrast between 2 similar things. Certainly, those WHERE clauses won't be the same. This is where **dynamic pen groups** come into play. To achieve this put pens who require the same WHERE clause snippet into the same group, and then add that group name into the list of *dynamic groups* in the customizer. Your chart will now have a new dynamic String property called **group_MyGroupName**. You can put your snippet of WHERE clause into this string (and of course it can be bound to an expression).

Option 3. Click-to-Chart™. Graph any pen against any other pens across the entire system.

This third option, what we call Click-to-Chart™, is a powerful method of setting up charting across an entire SCADA system. This idea takes advantage of the fact that the Easy Chart stores its pen information in an expert property DataSet. This means that your pens could be defined in the database, rather than inside your window. The idea of Click-to-Chart is that you define all pens across your entire system in one table. There could easily be thousands of pens in this table. Then, across the entire system, on every valve, every tank, every temperature indicator, etc, you put a right-click menu that says "Add this to the chart". Then, when they go to the chart window, they see the pens they have selected charted.

If you're curious about more details of how to set this up, please call us. The new Easy Chart component enables this feature far more simply than with the old chart. It's a bit of work up front (defining all your pens and enabling their selection), but it offers an unprecedented amount of charting flexibility.

Properties

Plot Background	plotBackground	COLOR
------------------------	----------------	-------

The background color for all subplots, unless they override it.

Chart Mode	chartMode	INT
-------------------	-----------	-----

Affects the mode that the chart operates in.

- **Manual Mode:** the data selected is determined by the values of the **Start Date** and **End Date** properties, which must be set manually.
- **Historical Mode:** a date range component will be displayed by the chart, allowing the user to select the time period they are interested in
- **Realtime Mode:** the user will be given the chance to choose a span of time, like 15 minutes, and that span will be updated at the poll rate as the data scrolls across

Possible values are:

- Manual (0)
- Historical (1)
- Realtime (2)

Gridline Color	gridlineColor	COLOR
-----------------------	---------------	-------

The color of the chart's gridlines

Gridline Width	gridlineWidth	FLOAT
-----------------------	---------------	-------

The thickness of the gridlines (default is .5)

Gridline Dash Pattern	gridlineDashPattern	STRING
------------------------------	---------------------	--------

The dash pattern for the gridlines, or empty string for no dashing. Example: "2,2" means 2 pixels on, 2 pixels off.

Pen Manipulation?	allowPenManipulation	BOOLEAN
--------------------------	----------------------	---------

Controls whether or not end-users can turn on and off pens.

Auto Apply	autoApply	BOOLEAN
-------------------	-----------	---------

If true, user changes to pen visibility will occur immediately.

Date Range	dateRangeLocation	INT
-------------------	-------------------	-----

Affects the position of the date range control.

Possible values are:

- Top (1)
- Bottom (2)

Legend	legend	INT
---------------	--------	-----

Where the legend should appear, if any.

Possible values are:

- None (0)

- Top (1)
- Bottom (2)
- Left (3)
- Right (4)

Chart Border	chartBorder	BORDER
---------------------	-------------	--------

The border for the chart itself

Pen Control Border	penBorder	BORDER
---------------------------	-----------	--------

The border for the pen control panel, if visible

Date Range Border	dateRangeBorder	BORDER
--------------------------	-----------------	--------

The border for the date range control, if visible

Poll Rate	pollRate	INT
------------------	----------	-----

The rate (in milliseconds) at which this chart's queries poll. Historical charts don't use this property.

Chart Title	title	STRING
--------------------	-------	--------

Sets an optional title to be displayed above the chart

X Axis Label	xAxisLabel	STRING
---------------------	------------	--------

The label shown on the X Axis (time axis)

Axis Font	axisLabelFont	FONT
------------------	---------------	------

The font for axis labels

Tick Font	axisTickLabelFont	FONT
------------------	-------------------	------

The font for tick labels

Empty Group Name	emptyGroupName	STRING
-------------------------	----------------	--------

The grouping title to use for pens that are not in a pen group.

Group Pens	penGrouping	BOOLEAN
-------------------	-------------	---------

If true, pens will be grouped by their group name

Auto Axis Positioning	autoPositionAxes	BOOLEAN
------------------------------	------------------	---------

If true, axes alternate automatically between left and right, rather than being placed explicitly.

Auto Pen Coloring	autoColorPens	BOOLEAN
--------------------------	---------------	---------

If true, pens are assigned different colors automatically.

Auto Color List	autoColorList	COLOR;
------------------------	---------------	--------

The list of colors to use if auto pen coloring is enabled

Show Loading	showLoading	BOOLEAN
---------------------	-------------	---------

If true, an animated indicator will be shown when data is loading

Show Warnings	showWarnings	BOOLEAN
----------------------	--------------	---------

If true, warnings generated during chart configuration will be printed to the console.

Horiz Gap	hGap	INT
------------------	------	-----



The horizontal spacing to use for the pen checkboxes

Vert Gap	vGap	INT	
-----------------	------	-----	--

The vertical spacing to use for the pen checkboxes

Alphabetize Pens	alphabetizePens	BOOLEAN	
-------------------------	-----------------	---------	---

If true, pens visibility checkboxes will be alphabetized.

Show Tooltips?	tooltips	BOOLEAN	
-----------------------	----------	---------	--

If true, tooltips showing point values will be displayed on the chart.

Show Maximize Button?	showMaximize	BOOLEAN	
------------------------------	--------------	---------	--

If true, a small maximize button will be displayed next to the chart.

Show Print Button?	showPrint	BOOLEAN	
---------------------------	-----------	---------	--

If true, a small print button will be displayed next to the chart.

Show Save Button?	showSave	BOOLEAN	
--------------------------	----------	---------	--

If true, a small save button will be displayed next to the chart.

Button Size	utilityButtonSize	INT	
--------------------	-------------------	-----	---

The size of the utility button icons.

Date Format	dateFormat	STRING	
--------------------	------------	--------	---

The format string to use when inserting dates into queries

Date Quote Start	dateQuoteStart	STRING	
-------------------------	----------------	--------	---

The starting quote for quoting a date

Date Quote End	dateQuoteEnd	STRING	
-----------------------	--------------	--------	---

The ending quote for quoting a date

Bar Margin	barMargin	DOUBLE	
-------------------	-----------	--------	---

The margin to use for the 'Bar' pen style

Gap Threshold	gapThreshold	DOUBLE	
----------------------	--------------	--------	---

The relative threshold to use for determining continuity breaks for the 'Discontinuous Line' pen style

3D X Offset	xOffset3D	INT	
--------------------	-----------	-----	---

The offset to use in the x direction for the '3D Line' pen style

3D Y Offset	yOffset3D	INT	
--------------------	-----------	-----	---

The offset to use in the y direction for the '3D Line' pen style

Digital Gap	digitalGap	DOUBLE	
--------------------	------------	--------	---

The size of the gap to use between digital pens

Startup Range	startupRange	STRING	
----------------------	--------------	--------	--

For historical-mode date range:

If startup mode is Automatic, this will be the starting range of time available for selection.

Startup Selection	startupSelection	STRING
--------------------------	------------------	--------

For historical-mode date range:
If startup mode is Automatic, this will be the starting selected range.

Max Selection	maxSelectionSize	STRING
----------------------	------------------	--------

For historical-mode date range:
The maximum size of the selected date range

Date Style	dateStyle	INT	
-------------------	-----------	-----	---

The style to display dates in. For international support.

Possible values are:

- Auto (0)
- MDY (1)
- DMY (2)
- YMD (3)

Time Style	timeStyle	INT	
-------------------	-----------	-----	---

The style to display times of day. For international support.

Possible values are:

- Auto (15)
- 12 HR (16)
- 24 HR (17)

Show Density	showHistogram	BOOLEAN
---------------------	---------------	---------

For historical-mode date range:

If true, a data density histogram will be shown in the date range.

Today Color	todayIndicatorColor	COLOR	
--------------------	---------------------	-------	---

For historical-mode date range:

The color of the "Today Arrow" indicator

Box Fill	boxFill	COLOR	
-----------------	---------	-------	---

For historical-mode date range:

The fill color for the selection box.

Selection Highlight	selectionHighlight	COLOR	
----------------------------	--------------------	-------	---

For historical-mode date range:

The focus highlight color for the selection box

Track Margin	trackMargin	INT	
---------------------	-------------	-----	---

For historical-mode date range:

The amount of room on either side of the slider track. May need adjusting if default font is changed.

Tick Density	tickDensity	FLOAT	
---------------------	-------------	-------	---

For historical-mode date range:

This is multiplied by the width to determine the current ideal tick unit.

High Density Color	highDensityColor	COLOR	
---------------------------	------------------	-------	---

For historical-mode date range:

The color used to indicate high data density.

Unit Count

For realtime-mode date range:

The number of units back to display

unitCount

INT

Unit

unit

INT

For realtime-mode date range:

The selected unit of the realtime date control

Possible values are:

- Seconds (1)
- Minutes (60)
- Hours (3600)
- Days (86400)

Realtime Text

rtLabel

STRING

For realtime-mode date range:

The text to display on the realtime date control.

Where Clause

globalWhereClause

STRING

A snippet of where clause that will be applied to all pens, like **TankNum = 2**

Start Date

startDate

DATE



For manual-mode:

The start date to use for selecting pen data

End Date

endDate

DATE



For manual-mode:

The end date to use for selecting pen data

Pens

pens

DATASET



This DataSet defines all of the pens that this chart will graph.

Axes

axes

DATASET



This DataSet defines all axis that can be used by the pens.

Subplots

subplots

DATASET



This DataSet defines all subplots' relative size and color.

Properties Loading

propertiesLoading

INT



The number of properties currently being loaded

Total Datapoints

datapoints

INT



The number of datapoints being displayed by the graph.

Customizer

The customizer is used to define Pens, Axes, and Subplots for the chart.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

exportExcel([filename])

Exports the chart as an excel spreadsheet. Each dataset is given its own worksheet. You can optionally suggest a filename to the user, if you don't the chart will suggest a filename of "Chart.xls".

Example:

```
chart = event.source.parent.getComponent("My Chart")
chart.exportExcel()
```

print()

Prompts the user to print the chart.

Example:

```
chart = event.source.parent.getComponent("My Chart")
chart.print()
```

setMode(mode)

Sets the current mode for the chart. The mode is an integer, options are:

- 0 : Zoom mode. This is the default mode, where the user can draw a zoom rectangle with the mouse.
- 1 : Pan mode. This mode lets the user use the mouse to pan the data left and right.
- 2 : Mark mode. This mode lets the user click near a datapoint to annotate the point with its X and Y value.
- 3 : X-Trace mode. This mode lets the user click on the chart and see all values that fall along that X value.

Example:

```
# This code would set the chart to X-Trace mode
chart = event.source.parent.getComponent("My Chart")
chart.setMode(2)
```

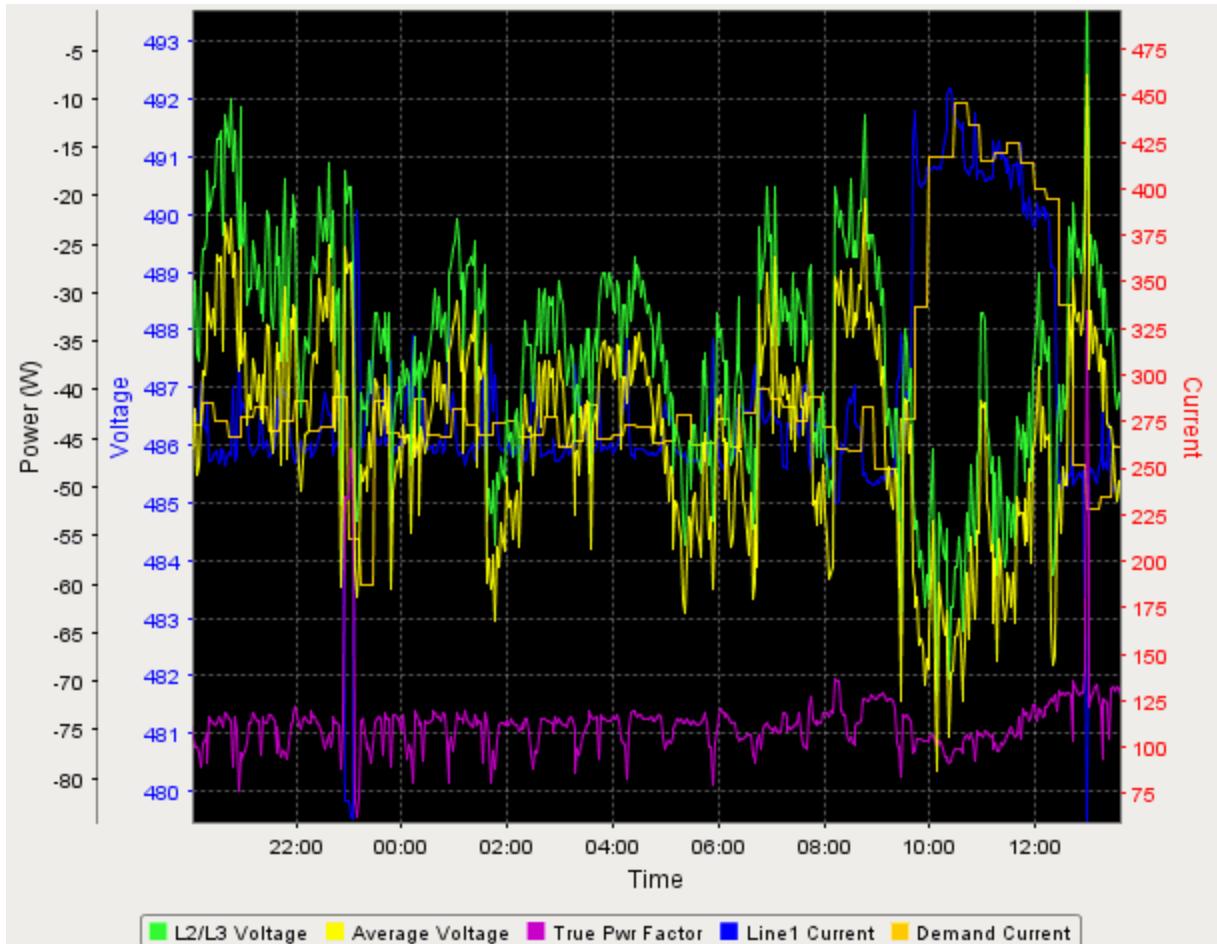


Chart Component

Icon in toolbar:



Description



The FactoryPMI Chart control is a very powerful graphing component. Some of the more important features are:

- **Interactive Zooming.** The user can zoom into a portion of the chart control by simply drawing the rectangle that they want to view.
- **Arbitrary Axes.** Any number of X and Y axes.
- **Support for Large Datasets.** Graph over 400,000 datapoints.
- **User Controllable.** Set your chart up any way you want. Have the user turn on and off individual 'pens', control date range, etc.
- **Realtime or Historical.** The same Chart can be used to display a realtime scrolling chart or historical chart.

Before we delve into how to use the Chart component, we'd like to point your attention to the FactoryPMI Goodies page at <http://www.inductiveautomation.com/products/factorypmi/goodies/>. There we have a chart template window for realtime and historical graphs. While you should still read this page to understand how to use chart controls, we just want you to know that you don't have to build your own chart windows from scratch.

Terminology

Before we delve into how to set up a Chart, we need to get some terminology straight.

A list of datapoints, which appears as a line on the chart is called a **Series**. Each datapoint has an X and a Y value. Usually, the X value is a date, and the Y value is a number. The X value could also be a number.

On graphs with multiple **Series**, it will often be the case that they all share common X values. For instance, it will usually be the case that you have a table in the database that is being logged to from FactorySQL. It will have a `t_stamp` (date) column, and then multiple numeric columns, each representing an address in a PLC. Each row in the table contains the values of the PLC addresses at a specific point in time. The `t_stamp` column is the shared X value for all of the other columns. Multiple **Series** with a common X value are grouped together into **Datasets**.

For example, suppose you had the following table:

<code>t_stamp</code> (DateTime)	Fan1Speed (Integer)	Fan1Amps (Double)	Fan2Speed (Integer)	Fan2Amps (Double)
2005-10-03 3:20pm	30	2.34	78	8.92
2005-10-03 3:30pm	24	1.23	81	9.67
2005-10-03 3:40pm	21	1.11	79	9.12

If you selected this entire table, it would be *one Dataset* with *four Series*: Fan1Speed, Fan1Amps, Fan2Speed, and Fan2Amps. This **Dataset** would have **12 Datapoints**: four datapoints per row, and 3 rows.

Basic Setup

There are 3 main steps to setting up a basic graph:

- **Add Datasets.** Add one or more datasets, each of which must have at least one series. Bind these datasets to a datasource.
- **Configure Axes.** Configure your Y-Axes. You usually have an axis per distinct unit of measure found in your series.
- **Configure Datasets.** Map each dataset to an axis. Configure the color of the series in each dataset.

Y Axes. Your basic unit of configuration of the Chart is the **Dataset**, not the **Series**. You cannot map an individual series to a Y axis, you can only map Datasets to Y axes. Of course, there is no limit to the number of datasets that you can have, so you could have each series in its own dataset. In general, you will try to group all of your series that have the same units into the same dataset, if you can (if they are in the same table). You *can* map multiple datasets to the same axis. There are four different types of y axes: number, date, category and logarithmic.

Example. Continuing with our example above, we can see that our table contains two distinct units of measure: Speed (measured in percentage) and Amps. So, we will add two datasets to the Chart, named appropriately (See Fig 1). This configuration is done in the Chart Customizer.

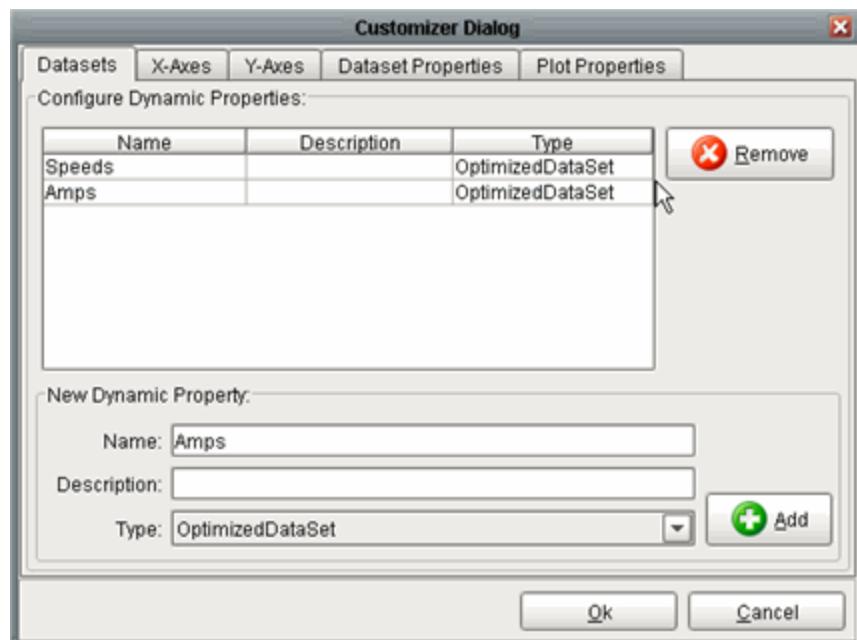


Fig 1. Adding Datasets to a Chart

Now we need to bind our datasets to some data. Lets say that our example table is called "MyTable". (NOTE: binding Chart Dataset data is one of the most varied parts of chart configuration, and where some real customizability presents itself. See the [Databinding](#) section for some ideas) For simplicity's sake, lets just bind the datasets to all of the rows in MyTable. So, the Speeds dataset would be bound to the following SQL query:

```
SELECT t_stamp, Fan1Speed, Fan2Speed FROM MyTable ORDER BY t_stamp
```

... and the Amps dataset would be bound to:

```
SELECT t_stamp, Fan1Amps, Fan2Amps FROM MyTable ORDER BY t_stamp
```

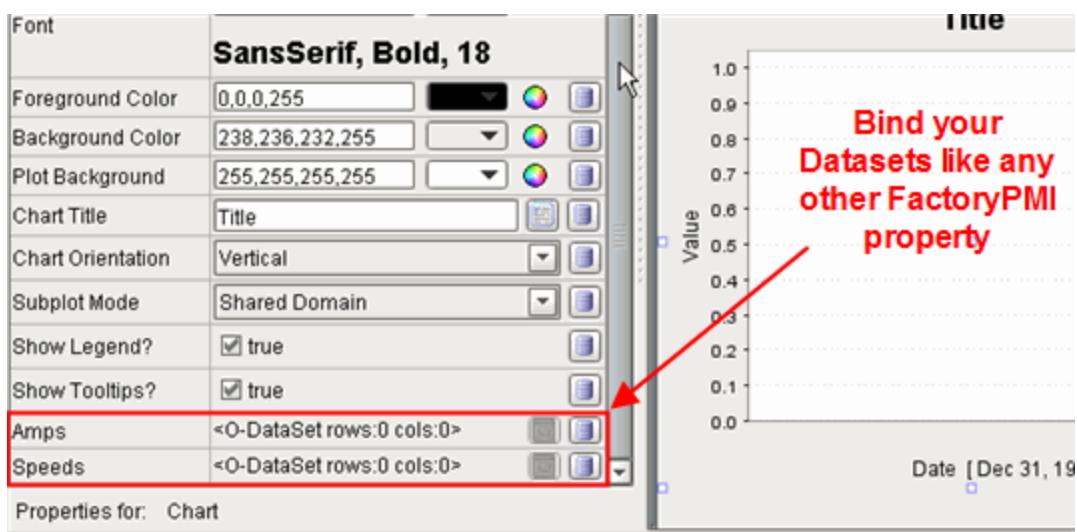


Fig 2. Binding Datasets

NOTE. You should always use the **ORDER BY t_stamp** clause in your chart queries, because otherwise you don't know what order the database will return your rows to you. The Chart component graphs points in the order they are received, so use ORDER BY to prevent your series from becoming a jumbled mess.

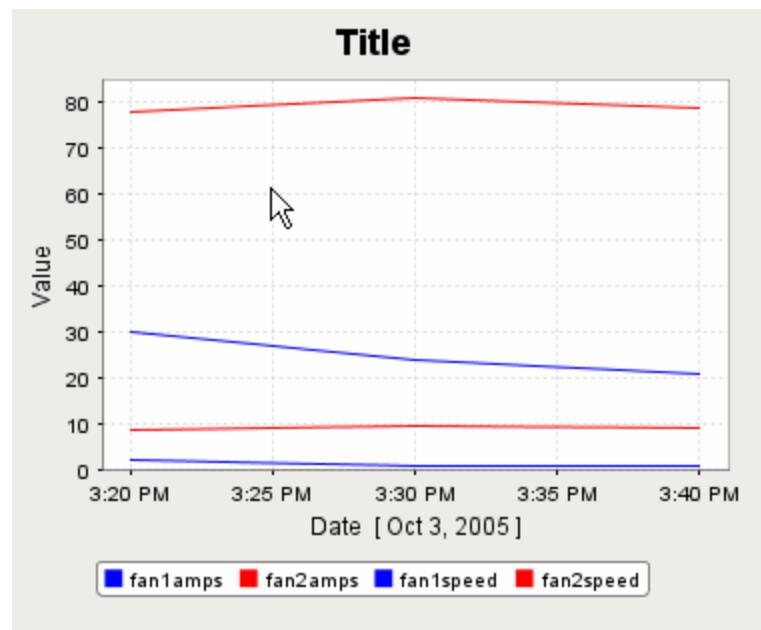


Fig 3. Initial Chart

The chart will now look like it does in Figure 3. You can see that it has some problems. The two datasets have assigned the same colors to their first two series. Each dataset has a list of colors that is assigned to its series in order. We will change the colors soon. Also, both datasets are graphed against the same, generic Y axis. Lets fix these problems.

Configuring Axes. First lets add our second axis. We'll use the default Y axis for the Speeds. Back in the customizer, go to the Y-Axes tab. Add a new Numeric Axis named "AmpsAxis" (See Fig. 4). Set the "Axis Label" for your two Y axes appropriately. Also set one of their "Axis Positions" to "Bottom/Right".

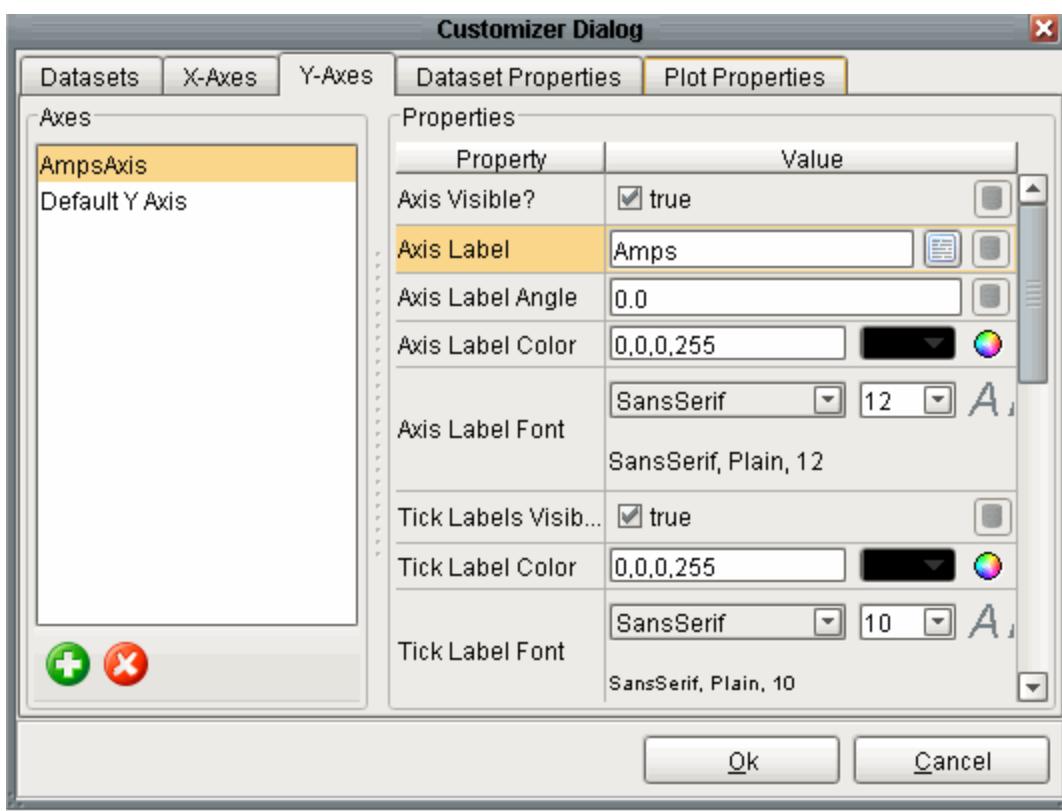


Fig 4. The Y-Axes Configuration Tab

Configuring Datasets. Now we need to tell the "Amps" dataset to use the "AmpsAxis" axis. Use the "Dataset Properties" tab to assign axes to datasets. (See Fig. 5)

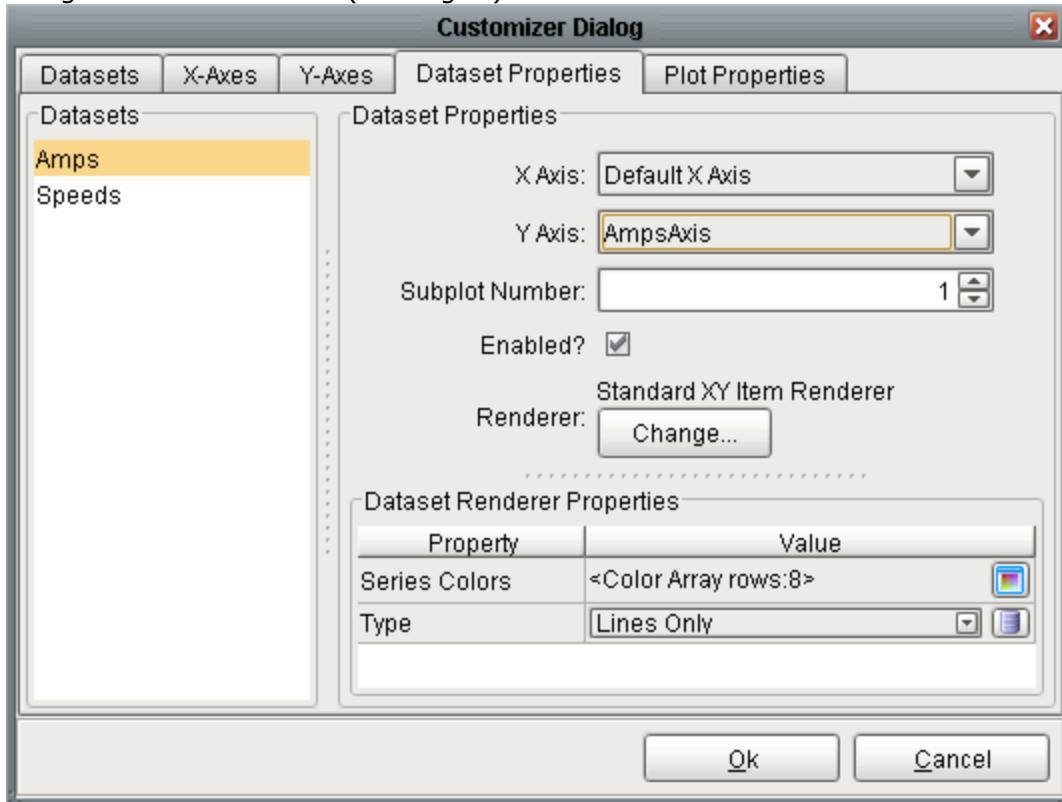


Fig 5. Setting Dataset Axes

Lastly, we will change the colors for the datasets. In Fig. 6, you can see that we have chosen greens for Fan 1, and blues for Fan 2, to logically group those series.



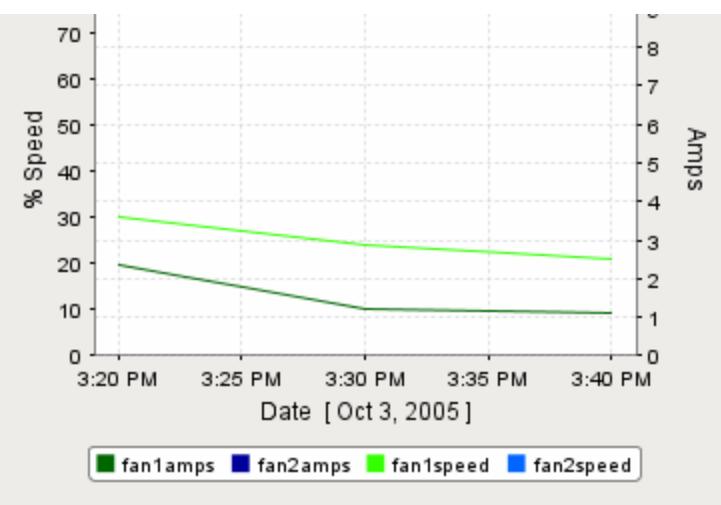


Fig 6. The Finished Product

Dataset Binding Techniques

Now that you understand the basic setup and terminology of the Chart component, we can discuss various dataset binding techniques used in conjunction with the Chart. As you may have noticed, the Chart control has no notion of date ranges, pen (series) selection, realtime vs. historical, etc. The chart component simply graphs the datapoints given to it. Using various databinding techniques, we alter the datapoints given to the chart to achieve various effects.

Realtime Charts. A realtime chart scrolls by, showing the most recent values for some fixed window of time. For instance, a rolling view of the last 15 minutes. There are a variety of ways to achieve this.

- Have FactorySQL only log 15 minutes worth of data (using the "Delete records older than..." option). The upside if this is simplicity. The downside is that you don't keep data for a historical view, and the user can't change the length of the sliding window of time. In this technique, you bind datasets to a polling SQL query. The poll rate determines the refresh rate of the chart.
- Use a polling SQL query that simply selects the last however many minutes of data. For instance, in PostgreSQL the query would look like this:

```
SELECT t_stamp, Fan1Speed, Fan2Speed FROM MyTable
WHERE t_stamp >= CURRENT_TIMESTAMP - interval '15 minutes'
ORDER BY t_stamp
```

It is easy to allow the user to select the time interval with this method. Simply bind the value of a spinner or numeric text field in place of the '15' in the query. The query polling rate determines the refresh rate of the chart in this technique.

Historical Charts. A historical chart lets the user select some time period. This can be done by selecting a starting date and an ending date with two date dropdowns. It can also be done by letting the user pick an end date and a span, where the span is a number of hours or minutes, chosen from a dropdown list. The latter is a bit safer, as you don't want some hapless user trying to chart three years worth of data that was stored every 10 seconds; the resulting data deluge would probably cause the client to run out of memory. To implement this second method, you'd bind your datasets to a query such as:

```
SELECT t_stamp, Fan1Speed, Fan2Speed FROM MyTable
WHERE t_stamp >= '{Root Container.EndDate.text}' - interval '{Root
Container.Span.selectedValue} hours'
AND t_stamp <= '{Root Container.EndDate.text}'
ORDER BY t_stamp
```

Where 'Root Container.EndDate' is a date dropdown and 'Root Container.Span' is a dropdown whose values are hours. Note that there are *many* variations on this theme, but this should give you the basic idea.

User Selectable Series. Sometimes you want to make a chart with 30 series, and let the user turn them on and off one at a time, so that they can graph a variety of things against each other. There are two main ways of doing this: Let your queries bring in all of the data, and use the Chart component's Jython functions to turn series on and off, responding to checkbox events. Or, let the other components in your window (checkboxes, buttons) generate the list of series to request from the database, and have your Chart's datasets' queries bound to these lists. The first method is good if the user turns on and off pens very frequently, since all of the data is already there. The second method is better if the user doesn't change pens as frequently, but changes date ranges frequently. In this

case, you will get a performance boost because you are only requesting the pens you need from the database.

To see an example of the second method, download the Chart Template from the FactoryPMI goodies. Dissecting it will show you how it works. Even better, you can use the template as a starting point, and make your own super-chart window with it!

Don't Forget!

There are a few things to keep in mind when designing chart windows:

- Always use the **ORDER BY t_stamp** clause at the end of your queries!
- Always make sure that your graph tables *have an index on the t_stamp column!* In PostgreSQL, the command to add an index on the t_stamp column is: `CREATE INDEX your_index_name ON YourTable (t_stamp);` Don't forget this! It can make the difference between data taking 90 seconds to load, and data taking 3 seconds to load.
- The sky is the limit when designing FactoryPMI chart windows! Let your imagination run wild. Here are some of the innovative ideas we have seen people come up with:
 - Chart windows with user-selectable-series, whose parameters could be saved to a database table as a "Favorite Graph". For instance, they could save a date range and a list of columns to create a saved "January Compressor 1 Pressure vs Temperature" chart.
 - A project where every data point displayed on every window had a right-click menu that said "Add to Chart", so they had one chart window for the entire project, that could chart anything vs anything else.

Properties

Plot Background	plotBackground	COLOR
------------------------	----------------	-------

The background color of the plot (the inner part of the chart background).

Chart Title	title	STRING
--------------------	-------	--------

A title for the chart.

Chart Type	chartType	INT
-------------------	-----------	-----

The type of chart to display. This determines how the X values of the chart are interpreted. The default is **XY Plot**, which has numeric or datetime X values. The other type is **Category Chart**, where the X values are String categories. Category charts are useful for bar charts, pareto charts, etc. Note that if you switch the chart type to **Category Chart**, you'll need to configure a categorical X axis, and you'll need to choose category renderers for your datasets.

Possible values are:

- XY Plot (2)
- Category Chart (0)

Extract Order	extractOrder	INT
----------------------	--------------	-----

If the Chart Type is "Category Chart", this determines how the data is interpreted. More specifically, do columns in the DataSet define series and rows define categories, or vice versa.

Possible values are:

- By Col (0)
- By Row (1)

Chart Orientation	orientation	INT
--------------------------	-------------	-----

Allows you to flip the chart on its side.

Possible values are:

- Horizontal (0)
- Vertical (1)

Subplot Mode

subplotMode

INT

Controls the subplot mode. In shared Domain, all subplots share the same X axis, and in shared range, all subplots share the Y axis.

Possible values are:

- Shared Domain (0)
- Shared Range (1)

Show Legend?

legend

BOOLEAN

If false, the legend will be hidden.

Show Tooltips?

tooltips

BOOLEAN

If false, tooltips won't be shown.

Selection Enabled?

selectionEnabled

BOOLEAN

If true, mouse clicks will select datapoints/bars on the chart. You can access what datapoint is currently selected (if any) via the **Selected Datapoint** property. Note that the **Selected Datapoint** property is a read only property, so you will only see it in binding windows, not in the property editor.

Selected Datapoint

selectedData

STRING

A String that represents the currently selected datapoint or bar. If there is nothing selected, this will be an empty string. Note that this property is *read-only*, which means that it only appears in the binding windows, not in the property editor. You can also access it via scripting. The format of this string varies based on the type of chart:

XY Plot: "DatasetName;SeriesName;x-val;y-val"

Category Chart: "DatasetName;Category;Series;y-val"

Selection Highlight Color

selectionHighlightColor

COLOR



The color that datapoints / bars will be outlined in if selected.

Selection Highlight Width

selectionHighlightWidth

FLOAT



The width of the outline for selected datapoints / bars.

Properties Loading

propertiesLoading

INT



This variable reflects the number of properties that are currently loading data from the database. In the Chart's case, this means datasets that are loading. Use this variable to detect when the chart's datasets are changing. This can be used to set the visibility of a "Loading..." label. If you have lots of datasets, you can even use this to show a progress bar filling in: set the progress bar's maximum to your total number of datasets, and set its value to that maximum - this property.

Customizer

The customizer for this class lets you add, remove, and edit properties for the Chart's datasets, X-axes, Y-axes, and subplots.

Events**• mouse**

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
-

- mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

clearSelection()

Programmatically clears any selected data. Example:

```
chart = event.source.parent.getComponent("My Chart")
chart.clearSelection()
```

setDatasetEnabled(DSName, value)

Programmatically sets the dataset named "DSName" to be enabled (visible) or not, depending on the boolean value of value.

Example:

```
# This code would work in the itemStateChanged event of a CheckBox
# It would set the dataset named "MyDataset" to be enabled if the
# checkbox was checked, and vice-versa.
chart = event.source.parent.getComponent("My Chart")
chart.setDatasetEnabled("MyDataset", event.stateChange == event.SELECTED)
```

setSeriesEnabled(DSName, SeriesName, value)

Programmatically sets the series named "SeriesName" in the dataset named "DSName" to be enabled (visible) or not, depending on the boolean value of value.

Example:

```
# This code would work in the itemStateChanged event of a CheckBox
# It would set the series "Temperature" in the dataset named "MyDataset"
# to be enabled if the checkbox was checked, and vice-versa.
chart = event.source.parent.getComponent("My Chart")
chart.setSeriesEnabled("MyDataset", "Temperature", event.stateChange == event.SELECTED)
```

setDatasetYAxis(DSName, AxisName)

Switches the dataset named "DSName" from whatever Y-axis it is currently using to the Y-axis named "AxisName". Will cause an error if either the dataset or axis doesn't exist. Example:

```
# This code would make the dataset "temps" to use the "OtherAxis" Y-axis.
chart = event.source.parent.getComponent("My Chart")
chart.setDatasetYAxis("temps", "OtherAxis")
```

setDatasetXAxis(DSName, AxisName)

Switches the dataset named "DSName" from whatever X-axis it is currently using to the X-axis named "AxisName". Will cause an error if either the dataset or axis doesn't exist. Example:

```
# This code would make the dataset "temps" to use the "OtherAxis" X-axis.
chart = event.source.parent.getComponent("My Chart")
chart.setDatasetXAxis("temps", "OtherAxis")
```

setDatasetPlotNumber(DSName, PlotNumber)

Switches the dataset named "DSName" from subplot it is currently on to the subplot numbered "PlotNumber". Will cause an error if either the dataset or subplot doesn't exist. Example:

```
# This code would make the dataset "temps" to use the "Plot 2" subplot.
```

```
chart = event.source.parent.getComponent("My Chart")
chart.setDatasetXAxis("temps", 2)
```

setMode(mode)

Sets the current mode for the chart. The mode is an integer, options are:

- 0 : Zoom mode. This is the default mode, where the user can draw a zoom rectangle with the mouse.
- 1 : Pan mode. This mode lets the user use the mouse to pan the data left and right.
- 3 : Mark mode. This mode lets the user click near a datapoint to annotate the point with its X and Y value.
- 4 : X-Trace mode. This mode lets the user click on the chart and see all values that fall along that X value.

Example:

```
# This code would set the chart to X-Trace mode
chart = event.source.parent.getComponent("My Chart")
chart.setMode(2)
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Bar Chart Component

Icon in toolbar:



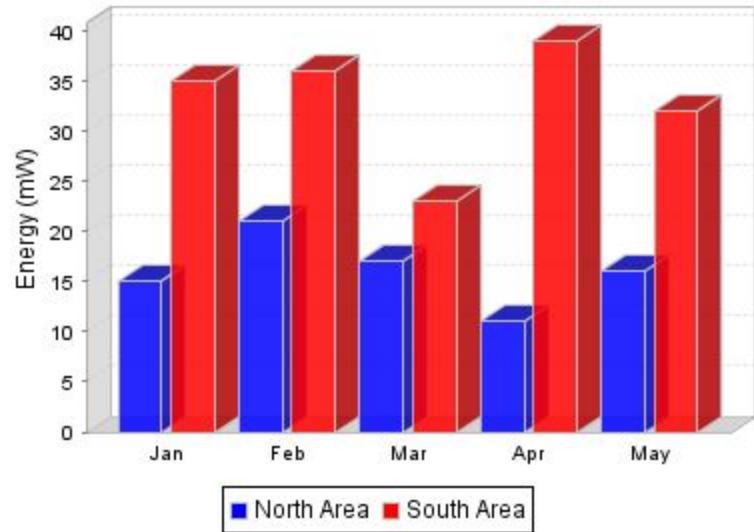
Description

The FactoryPMI Bar Chart is a very easy-to-use chart that provides familiar bar charts. It also can be configured to display other kinds of *category* charts. A category chart is a chart whose X-values are categories (strings) rather than numeric values (numbers, dates). See the [Chart Type](#) property for a description of the different category charts supported by the Bar Chart component.

Like most chart components, the `Data` property drives the chart. The first column in the `Data` `DataSet` defines the names of the categories. The rest of the columns define the values for each of the series (if there is more than one series per category), and thus should be numeric. Note - if your data is 'turned on its side', meaning that the columns define the categories and rows define the series, then set the [Extract Order](#) to "By Column".

For example, the dataset for the chart in the screenshot would have looked like this:

Label (String)	North Area (Integer)	South Area (Integer)
Jan	15	35
Feb	21	36
Mar	17	23
Apr	11	39
May	16	32



Properties

Data	data	DATASET
The data driving the chart. See the description above for formatting rules.		
Chart Title	title	STRING
The title of the chart. Optional.		
Value Axis Label	valueLabel	STRING
The label for the value axis		
Category Axis Label	categoryLabel	STRING
The label for the category axis		
Chart Type	rendererType	INT

Controls how the bar chart is displayed.

Possible values are:

- Bar (0)
- 3D Bars (1)
- Stacked Bars (2)
- 3D Stacked Bars (3)
- Layered (4)
- Area (5)

The following diagram shows examples of all types:

Chart Type: Bar

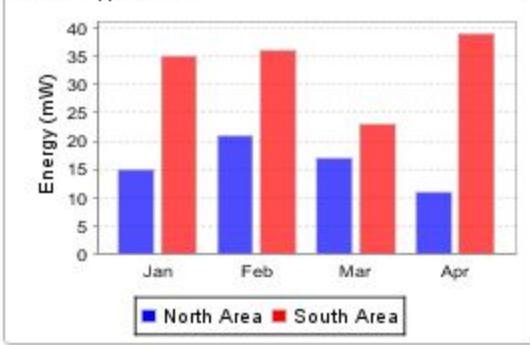


Chart Type: 3D Bars

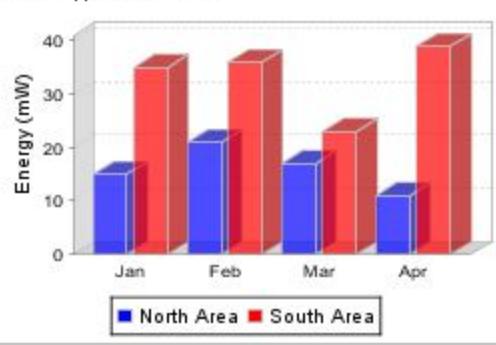


Chart Type: Stacked Bars

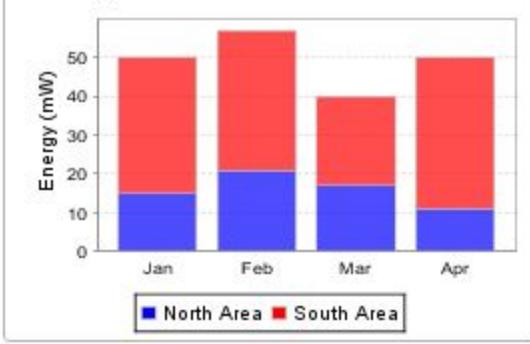


Chart Type: 3D Stacked Bars

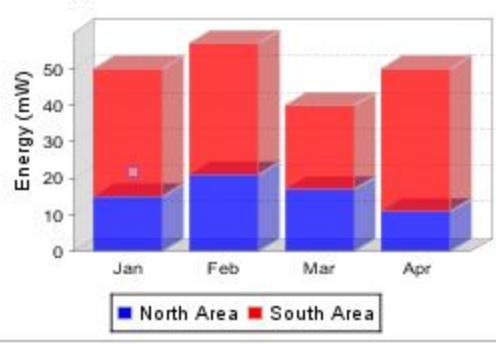


Chart Type: Layered

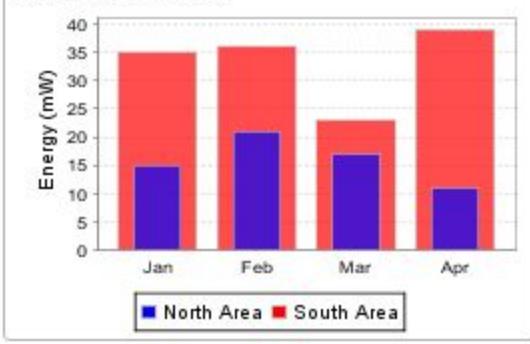
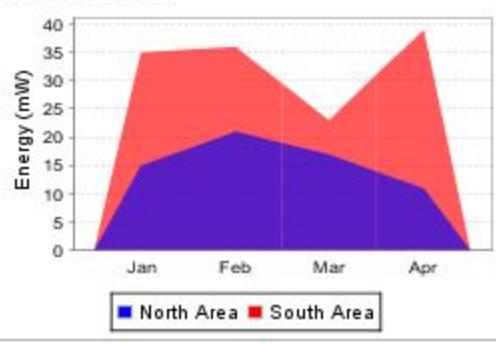


Chart Type: Area



Plot Background

plotBackground

COLOR

The background color for the plot.

Series Colors

seriesColors

COLOR ARRAY

The sequence of colors used for series in the bar chart.

Legend?

legend

BOOLEAN

Determines whether the legend is shown or not.

Tooltips?

tooltips

BOOLEAN

Determines whether or not tooltips are displayed when the mouse is paused over a bar.

Labels?

labels

BOOLEAN

Determines whether or not the bars should have numeric labels on top of them.

Foreground Transparency

foregroundAlpha

FLOAT

The transparency of the bar (useful for 3D bars). Should range from 0.0 to 1.0

Vertical

vertical

BOOLEAN

Sets the orientation of the chart to vertical (true) or horizontal (false)

Category Margin

categoryMargin

DOUBLE

The margin between categories as a fraction of the total space. Set this to 0 for an Area chart.

Item Margin

itemMargin

DOUBLE

The margin between bars in a category as a fraction

Extract Order

extractOrder

INT

Controls whether the first row defines the categories or the series.

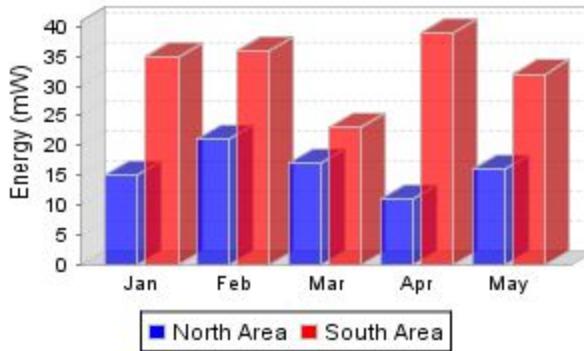
Possible values are:

- By Column (0)
- By Row (1)

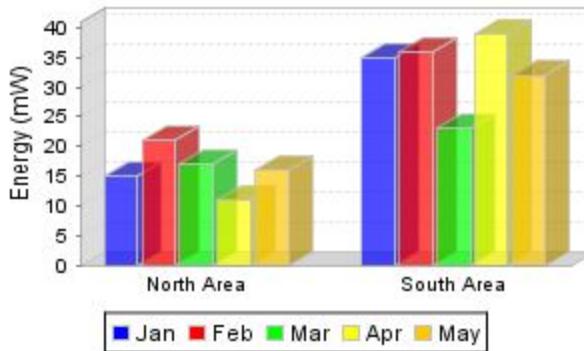
An example should make this clear. The following diagram shows the different interpretations of this DataSet:

Label (String)	North Area (Integer)	South Area (Integer)
Jan	15	35
Feb	21	36
Mar	17	23
Apr	11	39
May	16	32

Extract Order: By Row



Extract Order: By Column



Customizer

None.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

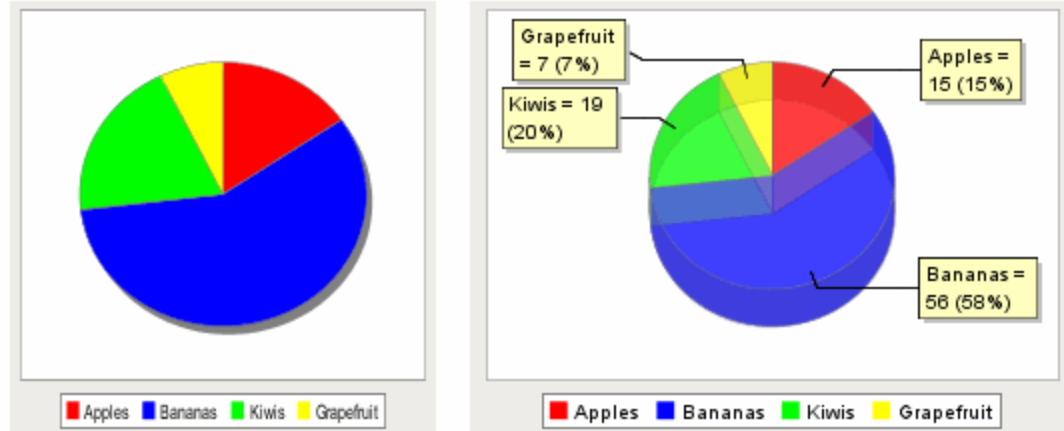


Pie Chart Component

Icon in toolbar:

Description

The Pie Chart component displays a familiar-looking pie chart. A Pie Chart displays a list of named items, each of which has a value that is part of a total. The total is the sum of the value of each item. The key to the Pie Chart component is the **Data** property. Each row of the dataset will be an item in the pie chart. The dataset should have two columns: a String column for the item name and a numeric column for the items value. For example, the dataset that makes up the screenshot would be:



Label (String)	Value (Integer)
Grapefruit	7
Apples	15
Bananas	56
Kiwis	19

NOTE. Is your data stored horizontally? That is, each of your columns is an item, and there is one row, which contains the amounts? See the [Extract Order](#) property.

Properties

f

Data

data

DATASET

The list of items that make up the pie chart. See description above for the format of this dataset.

Chart Title

title

STRING

A title to display above the chart.

Plot Background

plotBackground

COLOR

The background color of the chart's plot.

Section Colors

sectionColors

COLOR ARRAY

The list of colors to use for the pie sections. If there are more sections than colors in this array, the colors will be repeated.

Outline Colors

outlineColors

COLOR ARRAY

The list of colors to use for the pie section outlines. If there are more sections than colors in this array, the colors will be repeated.

Outline Stroke	outlineStroke	FLOAT
-----------------------	---------------	-------

The width of the pie section's outline stroke.

Legend?	legend	BOOLEAN
----------------	--------	---------

If true, a legend mapping colors to item labels will be shown.

Tooltips?	tooltips	BOOLEAN
------------------	----------	---------

If true, an item-specific tooltip will be shown when the mouse is hovering over that item's wedge. (See **Tooltip Format**).

Selection Enabled?	selectionEnabled	BOOLEAN
---------------------------	------------------	---------

If true, mouse clicks will select wedges on the chart. You can access what wedge is currently selected (if any) via the **Selected Wedge** property. Note that the **Selected Wedge** property is a read only property, so you will only see it in binding windows, not in the property editor.

Selected Wedge	selectedData	STRING
-----------------------	--------------	--------

A String that represents the currently selected wedge. If there is nothing selected, this will be an empty string. Note that this property is *read-only*, which means that it only appears in the binding windows, not in the property editor. You can also access it via scripting. The format of this string is: "SeriesName;Value"

Selection Highlight Color	selectionHighlightColor	COLOR
----------------------------------	-------------------------	-------



The color that wedges will be outlined in if selected.

Selection Highlight Width	selectionHighlightWidth	FLOAT
----------------------------------	-------------------------	-------



The width of the outline for selected wedge.

Labels?	labels	BOOLEAN
----------------	--------	---------

If true, labels for each wedge will be shown, showing information like labels, values, and percentages. (See **Label Format**).

Label Format	labelFormat	STRING
---------------------	-------------	--------

A format string that determines what is shown in the labels. The strings in this table are placeholders that will be replaced with their respective values for each item.

Placeholder	Will become...
{0}	Item Label
{1}	Item Value
{2}	Item Percentage

Example: The string: "{0}: {1}lbs ({2})" would become something like: "Kiwis: 19lbs (34%)"

Tooltip Format	tooltipFormat	STRING
-----------------------	---------------	--------

A format string for the tooltips. See the table for **Label Format** for a description.

Legend Font	legendFont	FONT
--------------------	------------	------

The font to use for legend items.

Label Font	labelFont	FONT
-------------------	-----------	------

The font to use for labels.

Starting Angle	startAngle	INT
-----------------------	------------	-----

The angle to start the initial item at. Has the effect of rotating the pie chart.

Rotation	rotation	INT
-----------------	----------	-----

Changes the direction that the items are drawn, in the order they are found in the dataset.

Possible values are:

- Clockwise (0)
- Counter-Clockwise (1)

Enforce Circularity?	circular	BOOLEAN
-----------------------------	----------	---------

Makes the pie chart enforce that the pie chart is a circle, not just an oval.

Style	style	INT
--------------	-------	-----

The display style of the pie chart.

Possible values are:

- Pie (0)
- 3D Pie (1)
- Ring (2)

Foreground Transparency	foregroundAlpha	DOUBLE
--------------------------------	-----------------	--------

The amount of transparency (0.0-1.0) of the foreground color.

3D Depth Factor	depthFactor	DOUBLE
------------------------	-------------	--------

The depth factor (viewing angle) (0.0-1.0) of the 3-dimensionality.

Extract Order	extractOrder	INT
----------------------	--------------	-----

Controls how the Pie Chart reads the **Data** dataset. As described, it is **By Column**. If you change this to **By Row**, the Pie Chart will expect a one-row dataset of the following format:

Grapefruit	Apples	Bananas	Kiwis
7	15	56	19

Possible values are:

- By Column (0)
- By Row (1)

Customizer

None.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved

- [propertyChange](#)

- propertyChange

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

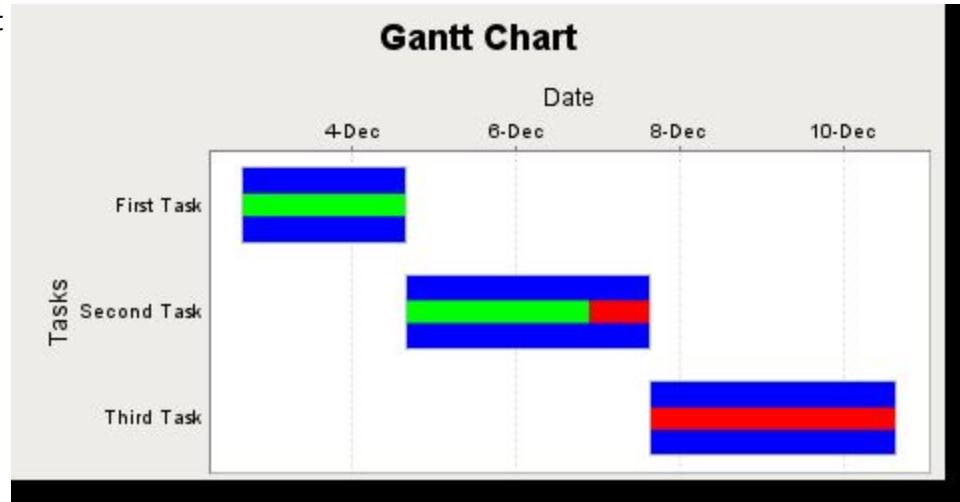


Gantt Chart Component

Icon in toolbar: 

Description

A Gantt chart is used for task scheduling. It shows a list of named tasks, each of which have a start date, an end date, and a percentage complete. This allows an easy way to visualize tasks, and easily see things like what tasks are holding others up.



Properties

Data

data

DATASET

The dataset for a Gantt chart contains a list of tasks. Each task consists of a name, which is shown on the vertical axes, a start and end date, and a percentage complete (0-100). Here is a (contrived) example:

Label (String)	Start Date (Date)	End Date (Date)	Percent Done (Integer)
Get Dressed	3-12-2005 6:00 am	3-12-2005 7:00 am	100
Commute	3-12-2005 7:00 am	3-19-2005 8:30 am	100
Work	3-12-2005 9:00 am	3-19-2005 5:30 pm	76

Chart Title

title

STRING

A title to display at the top of the chart

Task Axis Title

taskAxisTitle

STRING

The axis title for the vertical axis.

Date Axis Title

dateAxisTitle

STRING

The axis title for the horizontal axis.

Task Color

taskColor

COLOR

The color of a task.

Complete Color

completeColor

COLOR

The color of the 'complete' section of a task's progress bar.

Incomplete Color	incompleteColor	COLOR
The color of the 'incomplete' section of a task's progress bar.		
Plot Background	plotBackground	COLOR
The background color of the chart's plot.		
Tooltips?	tooltips	BOOLEAN
Show tooltips?		

Customizer

None.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.



Box and Whisker Chart Component

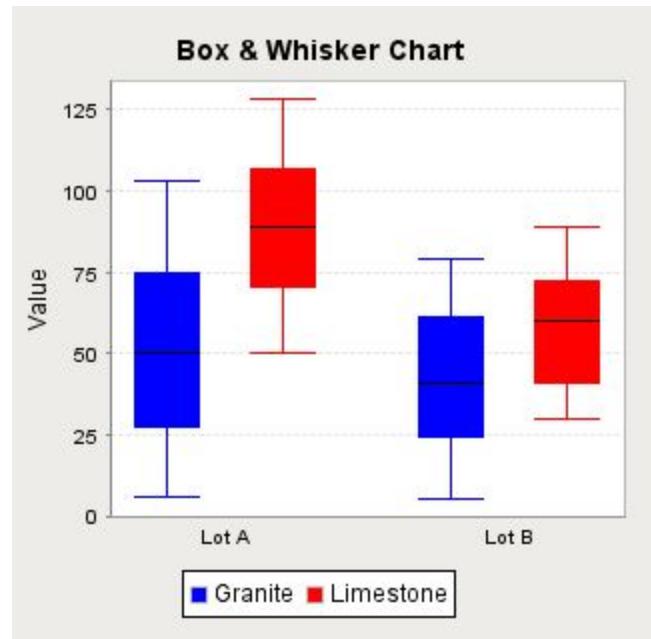
Icon in toolbar:



Description

A Box and Whisker chart displays pertinent statistical information about sets of data. Each box represents a set of numbers. The upper and lower bounds of the box represent the 1st and 3rd quartiles. The line in the box represents the *median*. The *whiskers* represent the max and min outliers. For a more detailed description, see <http://mathworld.wolfram.com/Box-and-WhiskerPlot.html>.

The main trick to using a box and whisker chart, like most charts, is populating the `data` property. See the `data` property description below for more details.



Properties

Data

data

DATASET

The dataset for a box and whisker chart contains sets of numbers. The numbers can be broken down in two ways. One is by column: each column of numbers will be a separate box. The second, which is **optional**, is by a key column. If this is used, then the first column must be a string 'key', breaking up the data into categories. For example, the data that generated the plot in the screenshot would have looked like this:

Key (String)	Granite (Integer)	Limestone (Integer)
Lot A	23	39
Lot A	24	23
Lot A	93	54
Lot A	76	72
Lot B	21	83
Lot B	4	21
Lot B	76	98
Lot B	89	102

Chart Title

title

STRING

A title to display at the top of the chart

Value Axis Title

valueAxisTitle

STRING

The axis title for the vertical axis.

Category Axis Title	categoryAxisTitle	STRING
The axis title for the horizontal axis.		
Series Colors	Series Colors	COLOR ARRAY
A list of colors with which to color the first breakdown (by column) of the data.		
Plot Background	plotBackground	COLOR
The background color of the plot.		
Tooltips?	tooltips	BOOLEAN
If true, a tooltip will be displayed over each box showing the statistical values.		
Fill Boxes?	fillBoxes	BOOLEAN
Determines whether or not the boxes are filled with their color.		
Legend?	legend	BOOLEAN
If true, a legend will be shown.		

Customizer

None.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.



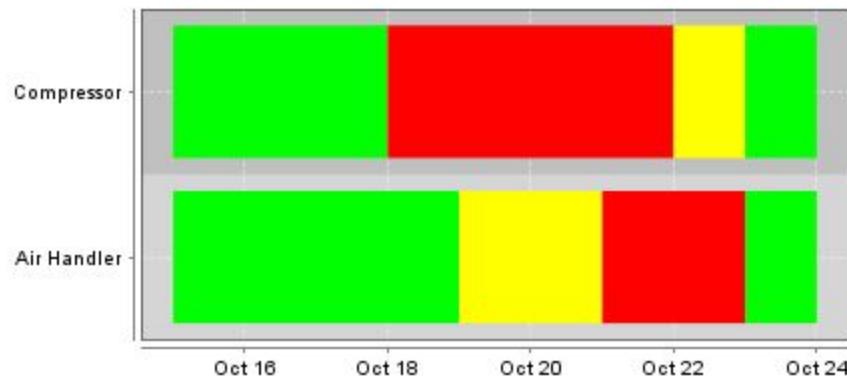
Status Chart Component

Icon in toolbar:



Description

The status chart component allows you to visualize the status of one or more discrete datapoints over a time range.



Properties

Data Format

dataFormat

INT

Format of the incoming dataset. In "wide" format, there are 2 or more columns: the first column of the dataset needs to be a timestamp, and every subsequent column represents one series in the chart. In "tall" format, there are always three columns: the first column is a timestamp, the second column is a series name, and the third is a value.

Possible values are:

- Wide (0)
- Tall (1)

Series Data

data

DATASET



The actual raw timeseries data that describes the status of each series. Data can be in either "wide" or "tall" format. Typically bound to a SQL query.

Series Properties Data

properties

DATASET



A dataset that contains the number-to color mappings for each series

Chart Title

chartTitle

STRING

Title of this chart.

Title Font

titleFont

FONT

Font of the chart title.

Title Color

titleColor

COLOR

Color of the chart title.

Series Spacing

seriesSpacing

DOUBLE

Affects the amount of spacing between series. Can be between 0.0 and 1.0. The series present on this chart are given equal space to display themselves. Series spacing is the percentage of that space that they use to do so.

Domain Axis Label

domainAxisLabel

STRING

Label on the domain axis.

Domain Axis Font	domainAxisFont	FONT
-------------------------	----------------	------

Font used on the domain axis.

Domain Axis Color	domainAxisColor	COLOR
--------------------------	-----------------	-------

Color used on the domain axis.

Domain Axis Location	domainAxisLocation	INT
-----------------------------	--------------------	-----

Location of the domain axis.

Possible values are:

- Left (2)
- Right (3)

Range Axis Label	rangeAxisLabel	STRING
-------------------------	----------------	--------

Label on the range axis.

Range Axis Font	rangeAxisFont	FONT
------------------------	---------------	------

Font used on the range axis.

Range Axis Color	rangeAxisColor	COLOR
-------------------------	----------------	-------

Color used on the range axis.

Range Axis Location	rangeAxisLocation	INT
----------------------------	-------------------	-----

Location of the range axis.

Possible values are:

- Top (0)
- Bottom (1)

Date Style	dateStyle	INT
-------------------	-----------	-----



The style to display dates in. For international support.

Possible values are:

- Auto (0)
- MDY (1)
- DMY (2)
- YMD (3)

Time Style	timeStyle	INT
-------------------	-----------	-----



The style to display times of day. For international support.

Possible values are:

- Auto (15)
- 12 HR (16)
- 24 HR (17)

Properties Loading	propertiesLoading	INT
---------------------------	-------------------	-----



The number of properties currently being loaded

Customizers

- Series Properties Customizer. Allows you to set a value and a corresponding color for each series in the chart.
- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



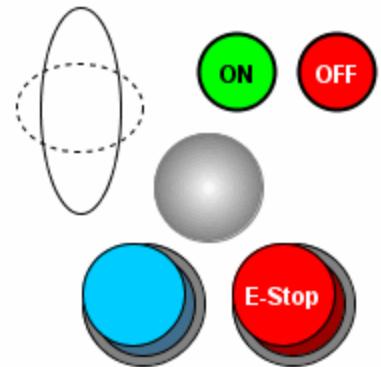
Circle Component

Icon in toolbar:

Description

The circle component displays an oval. The oval has a variable width outline and can optionally be filled with a color, or a circular gradient. In addition, you can enter a string which will be displayed in the center of the circle. To make the oval a perfect circle, remember to hold down Shift while creating/resizing a component to constrain proportions.

You can add custom properties to circles, so they make great status indicators. The buttons in the screenshot are made up of three circles overlapping. You can download the 3D circular button component as part of Component Pack #1 on the FactoryPMI goodies page at <http://www.inductiveautomation.com/products/factorypmi/goodies/>.



Properties

Fill Circle?

fillCircle

BOOLEAN

If true, the center of the circle will be filled with the **Fill Color**. If false, the center of the circle will be transparent.

Fill Color

fillColor

COLOR

The color to fill the center of the circle with, if **Fill Circle?** is true. Note that if **Gradient Fill?** is true, this color becomes the inside color of the gradient fill.

Outline Color

outlineColor

COLOR

The color to draw the outline of the circle in.

Line Width

lineWidth

INT



The width of the outside of the circle.

Line Style

lineStyle

INT



The style in which to draw the outline of the circle.

Possible values are:

- Plain (0)
- Dashed (1)

Dash Pattern

strokePattern

STRING

The pattern to draw the dashed outline, if **Line Style** is Dashed. This is a comma-separated list of integer values, which *must have an even number of values*. This list defines a pattern, # pixels on, # pixels off, # pixels on, # pixels off, etc. For example:

"3,3":

"3,10":

"10,3":

Rotation

rotation

INT

The amount (in degrees) to rotate this oval.

Label	label	STRING
--------------	-------	--------

A string to draw in the middle of the circle.

Label Color	textColor	COLOR
--------------------	-----------	-------

The color to draw the label, if present.

Anti Alias	antiAlias	BOOLEAN
-------------------	-----------	---------

If true the circle is drawn with anti-alias on.

Gradient Fill?	fillGradient	BOOLEAN
-----------------------	--------------	---------

If true, the center is filled with a circular gradient.

Gradient Background	gradientColor	COLOR
----------------------------	---------------	-------

If gradient fill is on, the outside color of the gradient.

Gradient Radius	gradientRadius	INT
------------------------	----------------	-----

If gradient fill is on, the radius of the radial gradient. If this is zero, the radius of the circle will be used.

Styles	styles	DATASET
---------------	--------	---------

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

The following functions are common to all components that allow dynamic properties:

getPropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

setPropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15
event.source.setPropertyValue("MyProperty", 15)
```

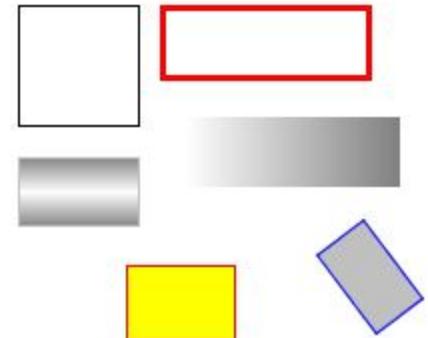


Rectangle Component

Icon in toolbar:

Description

The rectangle component displays a rectangle. Remember, if you want a square, hold down Shift while creating/resizing the rectangle. The rectangle component has control over what sides are shown, how thick they are, if the center is filled, gradient fill, etc.



Properties

Rotation

rotation

INT

The amount (in degrees) to rotate the rectangle. You will probably want to override the size of the rectangle in order to prevent the corners from being clipped.

Anti Alias

antiAlias

BOOLEAN



If true, the rectangle will be drawn with anti-alias on.

Draw Fill

drawFill

BOOLEAN

If true, the rectangle will be filled with the background color.

North Edge

drawNorth

BOOLEAN

If true, the north edge will be drawn.

East Edge

drawEast

BOOLEAN

If true, the east edge will be drawn.

South Edge

drawSouth

BOOLEAN

If true, the south edge will be drawn.

West Edge

drawWest

BOOLEAN

If true, the west edge will be drawn.

North Edge Line Width

northLineWidth

INT

The thickness of the north edge.

East Edge Line Width

eastLineWidth

INT

The thickness of the east edge.

South Edge Line Width

southLineWidth

INT

The thickness of the south edge.

West Edge Line Width

westLineWidth

INT

The thickness of the west edge.

Rounding Radius

roundingRadius

FLOAT

If non-zero, and if all edges have the same width, this rectangle will be rounded with the given radius.

Gradient Type

gradientType

INT

The type of gradient fill, if any.

Possible values are:

- Off (0)
- North-South (1)
- East-West (2)

Gradient Style

gradientStyle

INT

The style of the gradient, if applicable.

Possible values are:

- Round (1)
- Slope (0)

Gradient Color

gradientColor

COLOR



The secondary color for the gradient. The primary color is the background color.

Override Size?

overrideSize

BOOLEAN

If true, the rectangle's size won't be dependent on the component's bounds, rather, it will be the width and height specified.

Overridden Width

overrideWidth

INT

The width of the rectangle if override size is true.

Overridden Height

overrideHeight

INT

The height of the rectangle if override height is true.

Styles

styles

DATASET



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

• [mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased

- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

The following functions are common to all components that allow dynamic properties:

get PropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window
val = event.source.getPropertyValue("MyProperty")
fpmi.gui.messageBox(val)
```

set PropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15
event.source.setPropertyValue("MyProperty", 15)
```



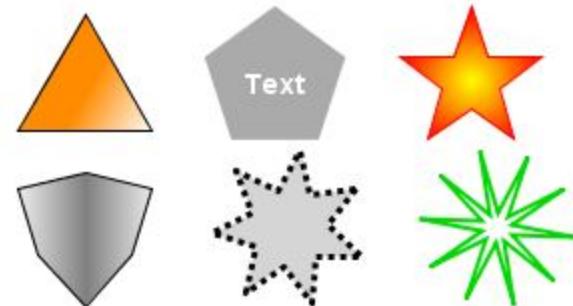
Polygon Component

Icon in toolbar:



Description

The polygon is a versatile shape displaying a regular polygon of 3 or more vertices at any degree of rotation. As an added mode, the polygon can become a star, which means an additional vertex is added between each normal vertex, but at a different distance from the center of the shape. Combine this with line styles, widths, colors, gradient fills, and the ability to display text, and you get a simple but powerful shape component.



Properties

Outline Color

foreground

COLOR



The color to use to draw the outline of the polygon

Fill Color

background

COLOR



The color to use to fill the polygon.

Gradient Color

gradientColor

COLOR



The color to mix with the fill color to make the gradient.

Label Color

textColor

COLOR

The color of the text in the polygon/star, if any.

Vertices

vertices

INT



The number of vertices (corners) for the polygon/star.

Star?

star

BOOLEAN



If true, the shape will be a star. Otherwise, the shape will be a polygon.

Spoke Ratio

spokeRatio

DOUBLE

The ratio of the star's inner spoke vertices' radii compared to the outer ones.

Fill?

fill

BOOLEAN

If true, the polygon/star will be filled in.

Outline?

outline

BOOLEAN

If true, an outline will be drawn around the polygon/star.

Label

label

STRING

Use this to display some text in the middle of your polygon/star.

Line Width	lineWidth	FLOAT	
-------------------	-----------	-------	--

Set the width of the outline in pixels.

Line Style	lineStyle	INT	
-------------------	-----------	-----	---

The line style determines how the shape of the line looks

Possible values are:

- Plain (0)
- Dashed (1)

Dash Pattern	strokePattern	STRING	
---------------------	---------------	--------	--

Enter a string of comma-delimited numbers which indicate the stroke pattern for a dashed outline.
For instance, "3,5" means three pixels on, five pixels off.

Gradient Type	gradientType	INT	
----------------------	--------------	-----	--

The type gradient of gradient to use to fill the polygon/star

Possible values are:

- None (0)
- Linear (1)
- Spherical (2)

Gradient Length	gradientLength	DOUBLE	
------------------------	----------------	--------	---

This is multiplied by the radius to use to determine gradient length.

Gradient Angle	gradientAngle	INT	
-----------------------	---------------	-----	---

The starting angle for linear gradients

Gradient Cyclic?	gradientRepeat	BOOLEAN	
-------------------------	----------------	---------	---

If true, the linear gradient will repeat

Rotation	rotation	INT	
-----------------	----------	-----	--

The angle of rotation in degrees.

Antialias	antiAlias	BOOLEAN	
------------------	-----------	---------	---

Draw with antialias on? Makes shape smoother

Styles	styles	DATASET	 
---------------	--------	---------	---

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed
- mouseReleased
- [**mouseMotion**](#)
 - mouseDragged
 - mouseMoved
- [**propertyChange**](#)
 - propertyChange

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

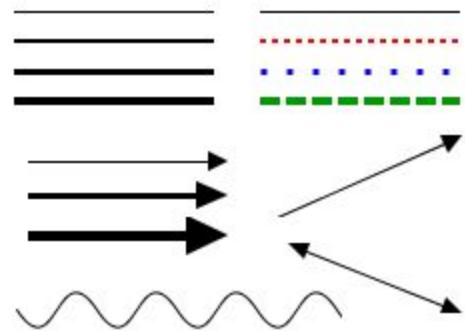


Line Component

Icon in toolbar: →

Description

The line component displays a straight line. It can run north-south, east-west, or diagonally. You can add arrows to either side. The line can be dashed using any pattern you want. You can even draw the line like a sinusoidal wave!



Properties

Line Width

lineWidth

INT



The width of the line, in pixels.

Line Mode

lineMode

INT



The line mode. By default, the line runs horizontally or vertically, depending on the overall shape of the component. If you change this to uphill or downhill, the line runs from one corner of the component to the opposite corner.

Possible values are:

- Horizontal/Vertical (0)
- Uphill (Left-Right) (1)
- Downhill (Left-Right) (2)

Line Style

lineStyle

INT



Possible values are:

- Plain (0)
- Dashed (1)
- Sinusoidal (2)
- Sinusoidal-Dashed (3)

Dash Pattern

strokePattern

STRING

The dash pattern to draw the line in, if **Line Style** is Dashed or Sinusoidal-Dashed. This is a comma-separated list of integer values, which *must have an even number of values*. This list defines a pattern, # pixels on, # pixels off, # pixels on, # pixels off, etc. For example:

"3,3":

"3,10":

"10,3":

Sine Length

sineLength

INT

The length of each sine period.

Sine Height	sineHeight	INT
-------------	------------	-----

The height (amplitude) of the sine wave.

Left Arrow	leftArrow	BOOLEAN
------------	-----------	---------

If true, an arrow will be drawn on the left side of the line.

Right Arrow	rightArrow	BOOLEAN
-------------	------------	---------

If true, an arrow will be drawn on the right side of the line.

Left Arrow Size	leftArrowSize	INT
-----------------	---------------	-----

The size of the left arrow.

Right Arrow Size	rightArrowSize	INT
------------------	----------------	-----

The size of the right arrow.

Anti Alias	antiAlias	BOOLEAN	
------------	-----------	---------	---

If true, the line will be drawn with anti-alias on. This looks good for diagonal and sinusoidal lines.

Styles	styles	DATASET	 
--------	--------	---------	---

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

The following functions are common to all components that allow dynamic properties:

getPropertyValue(PropName)

Retrieve the current value of the dynamic property named PropName.

Parameters

PropName

The name (a string) of the dynamic property to retrieve.

Example:

```
# Display the current value in a popup window  
val = event.source.getPropertyValue("MyProperty")  
fpmi.gui.messageBox(val)
```

setPropertyValue(PropName, Value)

Set the value of the dynamic property named PropName to the value Value.

Parameters

PropName

The name (a string) of the dynamic property to change.

Value

The value to change the property to.

Example:

```
# Change the value to 15  
event.source.setPropertyValue("MyProperty", 15)
```



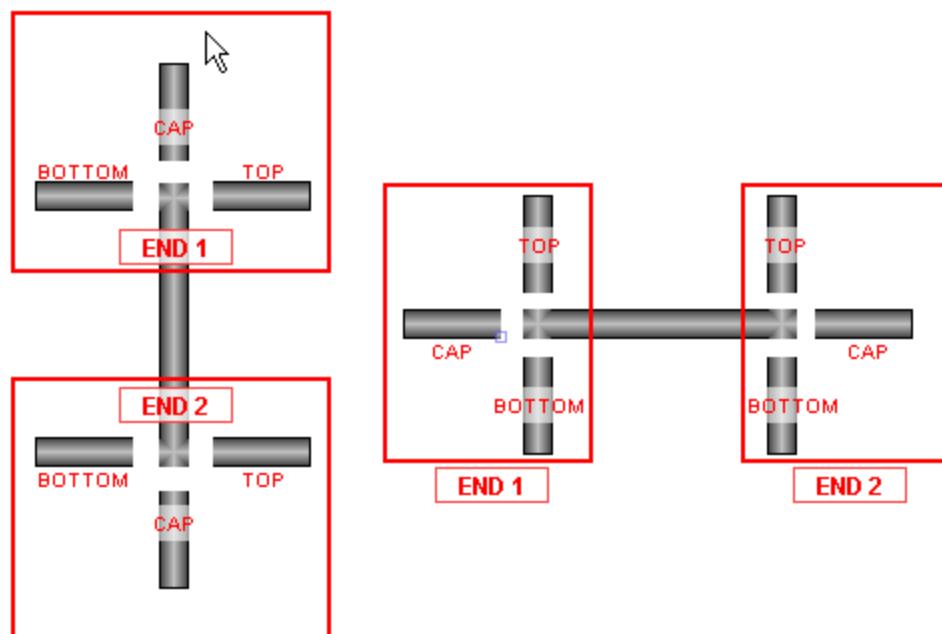
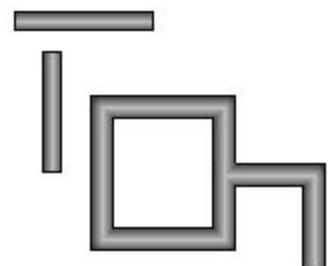
Pipe Segment Component

Icon in toolbar:

Description

The pipe segment component displays a 3-D pipe. In its basic form it looks very much like a rectangle with a round gradient. The difference comes in its advanced rendering of its edges. You can configure each pipe segment's end to mate perfectly with another pipe segment butted up against it perpendicularly. The result looks like a pipe welded together in a 90° corner.

The control of the pipe's ends can be a bit confusing to a new user. It is done via 6 booleans - three per 'end'. **End 1** is the top/left end, and **End 2** is the bottom/right end. You turn off each boolean if there will be another pipe butted up against that side. The following diagram should make the naming conventions more clear:



Properties

Center Fill

mainColor

COLOR



The center fill color of the pipe segment's gradient fill.

Edge Fill

secondaryColor

COLOR



The edge fill color of the pipe segment's gradient fill.

Outline Color

outlineColor

COLOR



The color of the 1px outline.

End 1 Top?

end1Top

BOOLEAN

If this is turned off, the pipe will be drawn to accept another pipe butted up against this location (see diagram in description).

End 1 Cap?

end1Cap

BOOLEAN

If this is turned off, the pipe will be drawn to accept another pipe butted up against this location (see diagram in description).

End 1 Bottom?

end1Bottom

BOOLEAN

If this is turned off, the pipe will be drawn to accept another pipe butted up against this location (see diagram in description).

End 2 Top?

end2Top

BOOLEAN

If this is turned off, the pipe will be drawn to accept another pipe butted up against this location (see diagram in description).

End 2 Cap?

end2Cap

BOOLEAN

If this is turned off, the pipe will be drawn to accept another pipe butted up against this location (see diagram in description).

End 2 Bottom?

end2Bottom

BOOLEAN

If this is turned off, the pipe will be drawn to accept another pipe butted up against this location (see diagram in description).

Styles

styles

DATASET



Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.

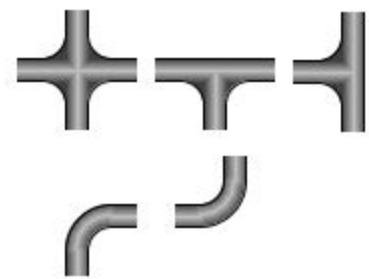


Pipe Joint Component

Icon in toolbar:

Description

The pipe joint displays a fancy joint component two join two pipe segments together. By turning off the cardinal directions, this will display a 2,3, or 4-pipe union. This component is rarely used, as pipes can butt up against each other and look joined.



Properties

Center Fill	mainColor	COLOR	
The color of the center of the gradient fill.			
Edge Fill	secondaryColor	COLOR	
The color of the edge of the gradient fill.			
Outline Color	outlineColor	COLOR	
The outline color.			
Top?	top	BOOLEAN	
If true, the pipe joint will accept a pipe coming in from the top.			
Right?	right	BOOLEAN	
If true, the pipe joint will accept a pipe coming in from the right.			
Bottom?	bottom	BOOLEAN	
If true, the pipe joint will accept a pipe coming in from the bottom.			
Left?	left	BOOLEAN	
If true, the pipe joint will accept a pipe coming in from the left.			
Styles	styles	DATASET	
Contains the component's styles.			

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Container Component

Icon in toolbar:

Description

The container is a very important component in FactoryPMI. All components are always inside of a container, except for the special "**Root Container**" of each window. A container is different than other components in that it can contain other components, including other containers. Uses for containers include:

- **Organization.** Containers can be used to [group](#) components together. These components can then easily be moved, copied, or deleted as a group. Furthermore, they will all be listed inside of their parent container in the navigation tree, which makes them easier to find.
- **Re-usability.** Containers allow a unique opportunity to create a complex component that is made up of multiple other components. The Container's ability to have custom properties aids this greatly. For instance, if you wanted to make an HOA control, you can put three buttons inside of a container and configure them to all use a 'status' property that you add to their parent Container. Now you have built an HOA control that can be re-used and treated like its own component. The possibilities here are endless. Create a date range control that generates an SQL WHERE clause that can be used to control Charts and Tables. Create a label/button control that can be used to display datapoints, and pop up a parameterized window that displays meta-data (engineering units, physical location, notes, etc) about that datapoint. Creating re-usable controls with Containers containing multiple components is the key to rapid development in FactoryPMI.
- **Layout.** Containers are a great way to improve window aesthetics through borders and [layout](#) options.



Properties

Styles	styles	DATASET	
--------	--------	---------	--

Contains the component's styles.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)

- mouseDragged
- mouseMoved
- [propertyChange](#)
- propertyChange

Scripting Functions

getComponent(CompName)

Return a reference to the component named "CompName" that is contained within this container. If no such component exists within this container, `None` is returned.

Parameters

CompName

The name (a string) of the component to retrieve.

Example:

This code would be inside a button, to display the text in a sibling Text Field component

```
MyParent = event.source.parent
# event.source.parent returns a reference to the Container component that the event-firing
# component is in.
TextField = MyParent.getComponent("Text Field")
fpmi.gui.messageBox(TextField.text)
```

getComponents()

Retrieve a list (sequence) of all components in a container. Example:

```
# Print out the names of all of the sibling components
siblings = event.source.parent.getComponents()
for sib in siblings:
    print sib.name
```



Paintable Canvas Component

Icon in toolbar:



Description

The Paintable Canvas component is a component that can be custom "painted" using Jython scripting. By responding to the component's *repaint* event, a designer can draw using Java2D anything within the component's bounds. Whenever any dynamic properties on the component change, the component is re-painted automatically, making it possible to create dynamic, vector-drawn components that can represent anything.



This component is an advanced component for those who are very comfortable using scripting. It is extraordinarily powerful, however, as your imagination is the only limit with what this component can be.

When you first drop a Paintable Canvas onto a window, you'll notice that it looks like a placeholder. If you switch the Designer into preview mode, you'll see an icon of a pump displayed. The pump is an example that comes pre-loaded into the Paintable Canvas. By editing the component's event scripts, you can dissect how the pump was drawn. You will notice that the script uses Java2D. You can read more about Java2D here <http://java.sun.com/docs/books/tutorial/2d/index.html>. You will notice that as you resize the pump, it scales beautifully in preview mode. Java2D is a *vector* drawing library, enabling you to create components that scale very gracefully.

Tips:

- Don't forget that you can add Dynamic Properties to this component, and use the Styles feature. Use the values of dynamic properties in your `repaint` code to create a dynamic component.
- You can create an interactive component by responding to mouse and keyboard events
- You can store your custom components on a custom palette and use them like standard components.

Properties

Focusable

focusable

BOOLEAN



If the component is focusable, it will receive keyboard input and can detect if it is the focus owner.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.
- Style Customizer. This means that you can add [your own styles](#) to this component.

Events

[mouse](#)

- mouseClicked
- mouseEntered
- mouseExited
- mousePressed

- mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [focus](#)
 - focusGained
 - focusLost
- [paint](#)
 - repaint This is a special event for this component only. The event object in this code has special properties: `graphics`, `width`, and `height`.
- [propertyChange](#)
 - propertyChange
- [key](#)
 - keyPressed
 - keyReleased
 - keyTyped

Scripting Functions

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Sound Player Component

Icon in toolbar:



Description

The Sound Player component is an invisible component that facilitates audio playback in a FactoryPMI client. Each Sound Player component has one sound clip associated with it, and will play that clip on demand. There is a built in triggering system, as well as facilities to loop the sound while the trigger is set.

Properties

Play Mode

playMode

INT

The Play Mode determines whether the sound is played automatically on trigger or manually. "Manually" means that you call the component's [play\(\)](#) function from a script.

Possible values are:

- Manual (0)
- On Trigger (1)

Loop Mode

loopMode

INT

The Loop Mode determines how many times the sound is played when triggered.

Possible values are:

- Play Once (0)
- Loop Forever (1)
- Loop N Times (2)

Loop Count

loopCount

INT

If Loop Mode is "Loop N Times", this is the "N".

Volume

volume

DOUBLE

The volume to use for playback (from 0.0 to 1.0).

Mute

mute

BOOLEAN

If true, the clip will be muted during playback.

Trigger

trigger

BOOLEAN



The clip will be played when the trigger is true, if Play Mode is "On Trigger"

Sound Data

soundData

BINARY DATA (BYTE[])

The clip that this component will play. Load a *.wav file or other compatible audio clip into this property. The file will be saved with the component; clients do not need access to a shared file.

Data Quality

dataQuality

INT



The data quality code for any tag bindings on this component.

Customizers

- Dynamic Properties Customizer. This means that you can add [your own properties](#) to this component.

Events

- [mouse](#)
 - mouseClicked
 - mouseEntered
 - mouseExited
 - mousePressed
 - mouseReleased
- [mouseMotion](#)
 - mouseDragged
 - mouseMoved
- [propertyChange](#)
 - propertyChange

Scripting Functions

play()

Tells the sound player to play the clip. The clip may be played once, some number of times, or continuously depending on the Loop Mode.

Example:

```
player = event.source.parent.getComponent("Sound Player")
button.play()
```

stop()

Tells the sound player to stop playing the clip. Useful when loop mode is set to "Loop Forever"

Example:

```
player = event.source.parent.getComponent("Sound Player")
button.stop()
```



Timer Component

Icon in toolbar:



Description

The timer control is usually used for animation. When running, its **Value** is incremented by the **step by** value, until the value hits the **Bound**, at which point the count starts over. Other objects can then bind to the **Value** for various purposes.



For instance, if you set the timer's **Bound** property to 360, and bind an object's rotation to the **Value** property, the object will spin in a circle when the timer is running.

Or, suppose that you have images that make up frames of animation. Name your images: "Frame0.png", "Frame1.png", "Frame2.png". Set the timer's **Bound** to be 3, Then bind the image path of an image to the following expression:

```
"Frame" + {Root Container.Timer.value} + ".png"
```

How fast the timer counts is up to the **Delay** property, which is the time between counts in milliseconds.

Want to run a Jython script every time the timer counts? First, make sure you don't actually want to write a [Global Timer Script](#), which will run on some interval whenever the application is running. In contrast, a script that works via a Timer component will only run while the window that contains the Timer is open, and the Timer is running. The way to do this is to attach an event script on the **propertyChange** event, making sure that `event.propertyName == "value"`.

Properties

Delay (ms)

delay

INT



The time between successive counts (after the first count) in milliseconds.

Initial Delay (ms)

initialDelay

INT



The time between turning the timer on and the first count, in milliseconds.

Running?

running

BOOLEAN



Controls whether this timer is currently counting, or not.

Value

value

INT



The current value of the timer. Will increment between 0 and **Bound**-1 while the timer is running.

Step by

step

INT

The amount to increment the **Value** by every **Delay** milliseconds.

Bound

max

INT

The upper bound on the **Value** property.

Customizer

None.

Events

- [action](#)
 - actionPerformed
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Signal Generator Component

Icon in toolbar:

Description

The signal generator is similar to the Timer component, but its value isn't simply a counter. Instead, you can choose from a variety of 'signals' familiar to anyone with basic knowledge of electric circuits. The value will update every **Period/Values per Period** milliseconds. In other words, the period is the number of milliseconds it will take for the signal to complete one period of its function. The Values per Period is the resolution, or number of times the **Value** will be updated every period.



Properties

Signal Type

signalType

INT

The kind of signal to emulate.

Possible values are:

- Sine (0)
- Triangular (2)
- Ramp (1)
- Square (3)
- Random (4)

Running?

running

BOOLEAN



Turns the signal on or off. The value only updates when this is true.

Value

value

DOUBLE



The current signal value.

Upper Bound

upper

DOUBLE

The upper bound of the signal's magnitude.

Lower Bound

lower

DOUBLE

The lower bound of the signal's magnitude.

Period

period

INT

The amount of time in which the **Value** will complete one period of the signal.

Values/Period

valuesPerPeriod

INT

The number of times the **Value** will be updated in one period.

Customizer

None.

Events

- [action](#)
 - actionPerformed
- [propertyChange](#)
 - propertyChange

Scripting Functions

None.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Expression Language

As anyone with a little bit of experience using FactoryPMI knows, its real power comes from the abundant ways that one can bind component properties. While direct property binding and database/SQL binding are easy to figure out, they do not provide one important binding capability - the ability to bind a property to a calculation based on one or more other properties. This is where the Expression Language comes into play. The Expression Language is a very simple language similar in concept to LISP. The following example should shed some light on the purpose behind the Expression Language.

Suppose that you have a variable in your root container called `TempC` that you want to display in a Label. You could bind the text property of the Label to `TempC` and be done with it. But suppose that you want to display " $^{\circ}\text{C}$ " next to the value. You could bind the text of the Label to the expression:

```
{Root Container.TempC} + " ^\circ C"
```

Now lets suppose that you want to show the temperature in Fahrenheit instead of Celsius. You could bind the text of the Label to the expression:

```
(1.8 * {Root Container.TempC} + 32) + " ^\circ F"
```

Now the Fahrenheit equivalent to the temperature will be dynamically calculated and shown instead of the Celsius value.

Expression Language Syntax

As its name suggests, everything in the Expression Language is an "Expression". This means that *everything returns a value*. An **expression** is:

- A Number (integer, floating point, or hexadecimal)
- A Boolean (true or false)
- A String (characters enclosed in double or single quotes)
- An equation of expressions using **operators** (see below)
- An expression in parenthesis
- A **bound SQLTag** (see below)
- A **bound property** (see below)
- A **function call** (see below)
- A **DataSet access** (see below)

The expression language is loosely typed, meaning that there *is* a type distinction (numbers, booleans, strings), but that most types can automatically be converted to another.

Operators

The following operators are available, and can be used to link together any other expressions:

Operator	Name	Description
-	Unary Minus	Returns the opposite (negative) of a number.
!	Not	Returns the logical opposite of a boolean.
[^]	Power	Raises a number to the power of another number, returns a number. a^b is a^b
%	Modulus	Performs a mod operation on two numbers, returns a number. $A\%B$ will return the remainder of A/B .
*	Multiplication	Multiplies two numbers, returns a number.
/	Division	Divides two numbers, returns a number.
+	Addition or Concatenation	If both operands are numbers, then adds the two numbers, returns a number. Otherwise, returns a string which is the first operand concatenated with the second number.
-	Subtraction	Subtracts two numbers, returns a number.

&	Bitwise AND	Performs a bitwise AND of two numbers.
	Bitwise OR	Performs a bitwise OR of two numbers.
xor	Bitwise XOR	Performs a bitwise exclusive OR of two numbers.
<<	Left Shift	Performs a signed bitwise left shift operation
>>	Right Shift	Performs a signed bitwise right shift operation
>	Greater	Performs a greater-than test, returns a boolean. The operands must be numbers.
<	Less	Performs a less-than test, returns a boolean. The operands must be numbers.
>=	Greater or Equal	Performs a greater-than or equal test, returns a boolean. The operands must be numbers.
<=	Less or Equal	Performs a less-than or equal test, returns a boolean. The operands must be numbers.
=	Equal	Tests for equality between two expressions, returns a boolean. The operands must be either booleans or numbers. Anything non-zero is considered true.
!=	Not Equal	Tests for non-equality between two expressions, returns a boolean. The operands must be either booleans or numbers. Anything non-zero is considered true.
&&	Logical AND	Performs a logical AND operation between two expressions, returns a boolean. The operands must be either booleans or numbers. Anything non-zero is considered true.
	Logical OR	Performs a logical OR operation between two expressions, returns a boolean. The operands must be either booleans or numbers. Anything non-zero is considered true.

Bound SQLTag

A **Bound SQLTag** is the use of a SQLTag in an expression. You can reference the SQLTags **value**, or any of its other properties. The syntax for a tag's value is:

{ [TagSource]Path/To/Tag}, or to reference another property:

{ [TagSource]Path/To/Tag.property}. Learn more about SQLTags paths [here](#).

(Tip: From within the expression binding window, click on the "Insert Tag" button to browse for tags. The tag you choose will be inserted at the cursor's current position in your expression)

Bound Properties

A **Bound Property** is the use of another component's property within an expression. The syntax for a bound property expression is

{component_path.property_name}

For example, the text of a button named Button in the Root Container would be accessible from within the Expression Language using the Expression: {Root Container.Button.text}

(Tip: From within the expression binding window, click on the "Insert Property Value" button to browse for properties. The property you choose will be inserted at the cursor's current position in your expression)

Functions

The expression language has a number of functions that you can call. All functions follow the format

FunctionName(Expression1, Expression2,...)

Data types are inferred automatically when possible. See the rest of this section for information on available functions.

DataSet Access

Most bound properties are values that you can use directly, such as Strings, Integers, etc. However, sometimes

you need to access a value inside a DataSet. To do this, you use the expression language's dataset accessor operators. These are the square brackets ("[", "]"). Inside the square brackets you can use one or two expressions.

Syntax	Description	Example
<code>DataSet[String]</code>	[Column Name] Mode. Returns the value in the first row, at the column with the same name as the enclosed <i>String</i>	{Root Container.Table.data}["PartName"]
<code>DataSet[Integer]</code>	[Column Index] Mode. Returns the value in the first row, at the column at the position of the value of the enclosed <i>Integer</i> .	{Root Container.Table.data}[3]
<code>DataSet[Integer, String]</code>	[Row Index, Column Name] Mode. The first argument, the <i>Integer</i> is the index of the row to look at. The second argument, the <i>String</i> is the name of the column.	{Root Container.Table.data}[23, "PartName"]
<code>DataSet[Integer, Integer]</code>	[Row Index, Column Index] Mode. The first argument is the index of the row to look at. The second argument, is the index of the column.	{Root Container.Table.data}[23, 3]

Note that the expression system doesn't know what data type a given location in a DataSet is going to contain at runtime. For this reason, you often have to enclose the DataSet Access expression in a type casting function. For example, to bind an integer property to a DataSet Access expression, you'd have to do it like this:

```
toInt({Root Container.Table.data}[23, 3])
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



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
```

coalesce(val1, val2, val3...)

This function, which accepts any number of arguments, evaluates each in order, and returns the first non-null argument. Typically, you would call this with two arguments - the first being something dynamic, the second being a static value to use as a guard in case the dynamic value is null. The function itself is given the type of the last argument.

Examples:

```
coalesce(null, "abc")
...would return "abc"
coalesce("xyz", "abc")
...would return "xyz"
coalesce({Root Container.MyDataSet}["ColumnName"], 0)
...would return the value in the dataset if it isn't null, but 0 if it is null.
```

getBit(value, position)

This function returns the bit (an integer, 0 or 1) in the number **value** at position **position**. The least significant bit in a number is position 0.

Examples:

```
getBit(0, 0)
...would return 0
getBit(1, 0)
...would return 1
getBit(8, 3)
...would return 1
getBit(8, 2)
...would return 0
```

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 (usually for binding a color to).

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)
```

lookup(DataSet, LookupValue, NoMatchValue, [LookupColumn], [ResultColumn])

This looks for LookupValue in the LookupColumn of DataSet. If it finds a match, it will return the value from the ResultColumn on the same row as the match. If no match is found, NoMatchValue is returned. Note: The type of

the value returned will always be coerced to be the same type as the `NoMatchValue`.

Parameters

DataSet

The `DataSet` to look through

LookupValue

The value to search for in the `DataSet`

NoMatchValue

The value to return if no match is found in the `DataSet`

[LookupColumn]

Optional (Default 0). The column to look for the `LookupValue` in. Can be a column index (starting at 0), or a column name (case insensitive).

[ResultColumn]

Optional (Default 1). The column to find the return value in. Can be a column index (starting at 0), or a column name (case insensitive).

For example, suppose `{RootContainer.MyDataSet}` is:

PRODUCT	PRICE	CATEGORY
"Apples"	1.99	"Fruit"
"Carrots"	3.50	"Vegetable"
"Walnuts"	6.25	"Nut"

```
lookup({Root Container.MyDataSet}, "Carrots", -1.0) //Returns 3.50
lookup({Root Container.MyDataSet}, "Grapefruit", -1) //Returns -1
lookup({Root Container.MyDataSet}, "Walnuts", "Unknown", 0, "Category") //Returns "Nut"
lookup({Root Container.MyDataSet}, "Pecans", "Unknown", 0, 2) //Returns "Unknown"
```



Mathematical Functions

The math functions are all very similar, except for round. 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 Euler's number 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*])

If *decimals* is omitted, rounds *value* to the nearest integer. Otherwise, rounds to the number of decimal places specified.

Parameters	
Value	A value to round.
Decimals	If present, this value will be the number of digits past the decimal point that the number is rounded to.



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 required parameter, and 1 optional parameter: the `value`, and the `failover`. The `value` is the actual value to convert, and `failover` is an optional value that will be returned if the conversion fails, or the value was `null`.

`toBoolean(value, [failover])`

This function tries to convert `value` to a boolean, according to the following rules:

- If `value` is a number, 0 is `false` and all other values are `true`.
- If `value` is a string, the strings (case insensitive) "on", "true", "t", "yes", "y" are considered `true`. The strings (case insensitive) "off", "false", "f", "no", "n" are considered `false`. If the string represents a number, the first rule applies. All other strings fail type casting.
- All other types of `value` fail type casting

`toColor(value, [failover])`

This function tries to convert `value` to a color. It assumes that `value` is a string. If you have integers representing Red, Green, and Blue values see the [color](#) expression. The string `value` is converted to a color according to these rules:

- If `value` is a name of a color as defined in the table below, the corresponding color will be returned. Note that color names are case insensitive.
- If `value` is a hex color string (with or without a leading "#"), the color equivalent of that hex string will be used. Examples: "#FF0000", "556B2F"
- If `value` is a list of 3 or 4 integers, a color will be created that uses the first three integers as red, green, and blue values, and the optional fourth integer as an alpha channel value. All values should be between 0 and 255. The list is free-form, any non-digit characters may be used as delimiters between the digits. Examples: "(0,0,0)", "23-99-203", "[255,255,33,127]"

`toDataSet(value, [failover])`

Tries to coerce `value` into a `DataSet`. Not many things can be coerced into `DataSets`. Namely, only `DataSets` and `PyDataSets` can be coerced into `DataSets`. This is useful for the `runScript()` expression, to convince the expression compiler to let you assign the return value of a scripting function to a `DataSet` property.

`toDate(value, [failover])`

Tries to coerce `value` into a `Date`. If `value` is a number or a string that represents a number, the number is treated as the number of milliseconds since the epoch, January 1, 1970, 00:00:00 GMT. If `value` is a string, it is parsed to see if it represents a date in one of these two formats: "yyyyMMdd.HHmssSSSZ" or "yyyy-MM-dd HH:mm:ss". If not, type casting fails.

The `failover` value must be a number or string with the same restrictions.

`toDouble(value, [failover])`

Tries to coerce `value` into a double. If `value` is a number, the conversion is direct. If `value` is a string, it is parsed to see if it represents a double. If not, type casting fails.

`toFloat(value, [failover])`

Tries to coerce `value` into a float. If `value` is a number, the conversion is direct (with possible loss of precision). If `value` is a string, it is parsed to see if it represents a float. If not, type casting fails.

`toInt or toInteger(value, [failover])`

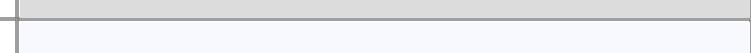
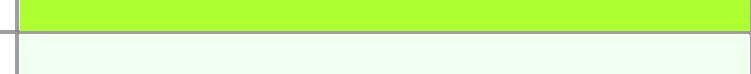
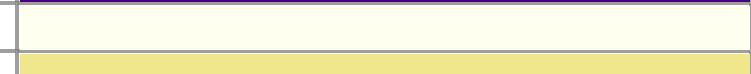
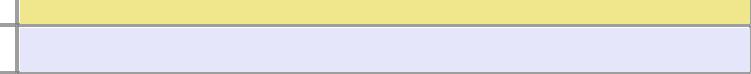
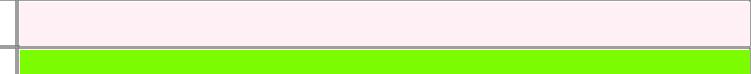
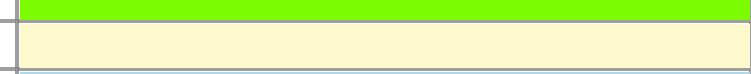
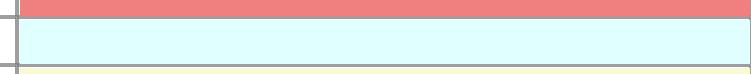
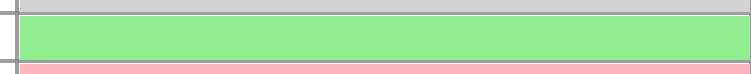
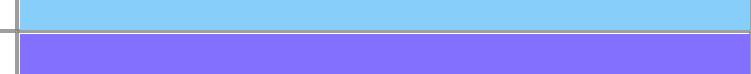
Tries to coerce `value` into an integer. If `value` is a number, the conversion is direct (with possible loss of precision). If `value` is a string, it is parsed to see if it represents an integer. If not, type casting fails.

toLong(value, [failover])

Tries to coerce value into a long. If value is a number, the conversion is direct (with possible loss of precision). If value is a string, it is parsed to see if it represents a long. If not, type casting fails.

Named Colors Table

Color Name	Color HEX	Color
AliceBlue	#F0F8FF	
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFAD	
Azure	#F0FFFF	
Beige	#F5F5DC	
Bisque	#FFE4C4	
Black	#000000	
BlanchedAlmond	#FFEBCD	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	
Chartreuse	#7FFF00	
Chocolate	#D2691E	
Clear	(transparent)	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	
Crimson	#DC143C	
Cyan	#00FFFF	
DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGray	#A9A9A9	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	
DarkOliveGreen	#556B2F	
Darkorange	#FF8C00	
DarkOrchid	#9932CC	
DarkRed	#8B0000	
DarkSalmon	#E9967A	
DarkSeaGreen	#8FBBC8	
DarkSlateBlue	#483D8B	
DarkSlateGray	#2F4F4F	
DarkTurquoise	#00CED1	

DarkViolet	#9400D3	
DeepPink	#FF1493	
DeepSkyBlue	#00BFFF	
DimGray	#696969	
DodgerBlue	#1E90FF	
Feldspar	#D19275	
FireBrick	#B22222	
FloralWhite	#FFFFA0	
ForestGreen	#228B22	
Fuchsia	#FF00FF	
Gainsboro	#DCDCDC	
GhostWhite	#F8F8FF	
Gold	#FFD700	
GoldenRod	#DAA520	
Gray	#808080	
Green	#008000	
GreenYellow	#ADFF2F	
HoneyDew	#F0FFF0	
HotPink	#FF69B4	
IndianRed	#CD5C5C	
Indigo	#4B0082	
Ivory	#FFFFFF	
Khaki	#F0E68C	
Lavender	#E6E6FA	
LavenderBlush	#FFF0F5	
LawnGreen	#7CFC00	
LemonChiffon	#FFFACD	
LightBlue	#ADD8E6	
LightCoral	#F08080	
LightCyan	#E0FFFF	
LightGoldenRodYellow	#FAFAD2	
LightGrey	#D3D3D3	
LightGreen	#90EE90	
LightPink	#FFB6C1	
LightSalmon	#FFA07A	
LightSeaGreen	#20B2AA	
LightSkyBlue	#87CEFA	
LightSlateBlue	#8470FF	
LightSlateGray	#778899	
LightSteelBlue	#B0C4DE	
LightYellow	#FFFFE0	
Lime	#00FF00	
LimeGreen	#32CD32	

Linen	#FAF0E6	
Magenta	#FF00FF	
Maroon	#800000	
MediumAquaMarine	#66CDAA	
MediumBlue	#0000CD	
MediumOrchid	#BA55D3	
MediumPurple	#9370D8	
MediumSeaGreen	#3CB371	
MediumSlateBlue	#7B68EE	
MediumSpringGreen	#00FA9A	
MediumTurquoise	#48D1CC	
MediumVioletRed	#C71585	
MidnightBlue	#191970	
MintCream	#F5FFFA	
MistyRose	#FFE4E1	
Moccasin	#FFE4B5	
NavajoWhite	#FFDEAD	
Navy	#000080	
OldLace	#FDF5E6	
Olive	#808000	
OliveDrab	#6B8E23	
Orange	#FFA500	
OrangeRed	#FF4500	
Orchid	#DA70D6	
PaleGoldenRod	#EEE8AA	
PaleGreen	#98FB98	
PaleTurquoise	#AFEEEE	
PaleVioletRed	#D87093	
PapayaWhip	#FFEFD5	
PeachPuff	#FFDAB9	
Peru	#CD853F	
Pink	#FFC0CB	
Plum	#DDA0DD	
PowderBlue	#B0E0E6	
Purple	#800080	
Red	#FF0000	
RosyBrown	#BC8F8F	
RoyalBlue	#4169E1	
SaddleBrown	#8B4513	
Salmon	#FA8072	
SandyBrown	#F4A460	
SeaGreen	#2E8B57	
SeaShell	#FFF5EE	

Sienna	#A0522D	
Silver	#C0C0C0	
SkyBlue	#87CEEB	
SlateBlue	#6A5ACD	
SlateGray	#708090	
Snow	#FFFFFA	
SpringGreen	#00FF7F	
SteelBlue	#4682B4	
Tan	#D2B48C	
Teal	#008080	
Thistle	#D8bfd8	
Tomato	#FF6347	
Transparent	(transparent)	
Turquoise	#40E0D0	
Violet	#EE82EE	
VioletRed	#D02090	
Wheat	#F5DEB3	
White	#FFFFFF	
WhiteSmoke	#F5F5F5	
Yellow	#FFFF00	
YellowGreen	#9ACD32	

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



String Functions

concat(*value1, value2, ..., valueN*)

Concatenates all of the parameters into one string. The plus (+) operator does the same.

indexOf(*String, Part*)

Searches for the first occurrence of the string *Part* inside of the string *String*. Returns the index of *Part* within *String*, or -1 if *Part* wasn't found.

Examples:

```
indexOf("Hamburger", "urge")
...returns 4
indexOf("Test", "")
...returns 0
indexOf("Disfunctional", "fun")
...returns 3
indexOf("Disfunctional", "marble")
...returns -1
indexOf("banana", "n")
...returns 2
```

lastIndexOf(*String, Part*)

Searches for the last occurrence of the string *Part* inside of the string *String*. Returns the index of this last occurrence of *Part* within *String*, or -1 if *Part* wasn't found.

Examples:

```
lastIndexOf("Hamburger", "urge")
...returns 4
lastIndexOf("Test", "")
...returns 4
lastIndexOf("Disfunctional", "fun")
...returns 3
lastIndexOf("Disfunctional", "marble")
...returns -1
lastIndexOf("banana", "n")
...returns 4
```

len(*value*)

Returns the number of characters in the String *value*. If *value* is a DataSet, returns the number of rows in the DataSet.

lower(*String*)

This expression will take a string and convert it to lower case.

Examples:

```
lower("STOP")
...returns "stop"
```

numberFormat(*number, pattern*)

This function formats the given *number* according to the *pattern*.

The pattern consists of several special characters:

Pattern Characters

0	A required digit.
#	Optional digit - 0 is not shown.
,	Grouping separator.
-	Minus sign
E	Scientific notation operator.

;
Separates positive and negative patterns (see below)

%
Multiply by 100 and show as a percent.

The *subpattern separator* (";") allows you to define positive and negative patterns. However, this only applies to outside characters, such as - and (), not grouping. The positive pattern grouping will always be used. To clarify this, see the examples below. If no negative pattern is specified, the local negation indicator will be added to the positive pattern ("-" in most cases).

Pattern Examples

Number	Pattern	Result
5	0	5
5	0.0	5.0
5	00.0	05.0
123	#,##0	123
1234	#,##0	1,234
87.329	#,##0	87
1234	#,##0.#	1,234
1234.5	#,##0.#	1,234.5
1234.5	#,##0.00	1,234.50
87.329	#,##0.##	87.33
87.329	#,##0.0000	87.3290
-1234	#,##0	-1,234
-1234	#,##0;(#)	(1,234)
1234	0.###E0	1.234E3

repeat(string, count)

Returns the given string, repeated some number of times.

```
repeat("hello", 2)  
...returns "hellohello"  
repeat("hello", 0)  
...returns ""
```

split(string, regex, [limit])

This function takes the string `string` and splits it into a bunch of substrings. The substrings are returned as a DataSet with one column called "parts". The split occurs wherever the regular expression `regex` occurs. Don't be intimidated by the regular expression, this is normally just another string, like "," for comma separated lists.

The optional `limit` argument, if greater than zero, limits the number of times the regex pattern is applied to `limit-1`. Put another way, it limits the length of the resulting dataset to length `limit`. If `limit` is non-positive then the `regex` pattern will be applied as many times as possible and the returned dataset can have any length. If `limit` is zero (the default) then the pattern will be applied as many times as possible, the returned dataset can have any length, and trailing empty strings will be discarded.

Examples:

```
split("hello,world","","")  
... returns
```

parts
"hello"
"world"

```
split("boo:and:foo", ":")  
... returns
```

parts
"boo"
"and"
"foo"

```
split("boo:and:foo", ":", 2)  
... returns
```

parts
"boo"
"and:foo"

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"
```

trim(*String*)

This expression will take a string and *trim* off any leading and/or trailing whitespace.

Examples:

```
trim("Hello Bob ")  
...returns "Hello Bob"  
trim(" Exit")  
...returns "Exit"
```

upper(*String*)

This expression will take a string and convert it to upper case.

Examples:

```
upper("Hello")  
...returns "HELLO"
```

Color Functions

brighter(color)

Returns a color one shade brighter than the argument color.

```
brighter(color(200,0,0))
```

...returns this color: 

color(red, green, blue, [alpha])

Creates a color using the given red, green, and blue amounts, which are integers between 0-255. The optional alpha channel to the color controls transparency. See the documentation for the [toColor](#) function for more flexible color creation.

Parameters

red	The amount of red in the color (between 0-255).
green	The amount of green in the color (between 0-255).
blue	The amount of blue in the color (between 0-255).
[alpha]	How transparent the color is (between 0-255; 0=totally transparent, 255=opaque).

```
color(255,0,0)
```

...returns this color: 

darker(color)

Returns a color one shade darker than the argument color.

```
darker(color(255,0,0))
```

...returns this color: 

gradient(Value, Low, High, LowColor, HighColor)

Based on the position of parameter value between parameters *Low* and *High*, returns a color which is the appropriate gradient mix of *LowColor* and *HighColor*.

Parameters

Value	The numeric value to determine the color mix.
Low	The numeric lower bound. If Value is equal to or less than Low, LowColor will be returned.
High	The numeric upper bound. If Value is equal to or greater than High, HighColor will be returned.
LowColor	The low color of the gradient.
HighColor	The high color of the gradient.

```
gradient(0, 0, 100, color(255,0,0), color(0,0,255))
```

...returns this color: 

```
gradient(100, 0, 100, color(255,0,0), color(0,0,255))
```

...returns this color: 

```
gradient(60, 0, 100, color(255,0,0), color(0,0,255))
```

...returns this color: 

```
gradient({Root Container.Tank.value}, 0, 100, color(255,0,0), color(0,0,255))
```

...will return a gradient from red to blue based on the level of a tank.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Date Functions

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({Root Contai ner. DatePi cker. date}, 5, "hour")
```

...returns a new date, five hours after the date in the DatePicker component.

```
dateArithmet ic({Root Contai ner. DatePi cker. date}, -8, "days")
```

...returns a new date, eight days before the date in the DatePicker component.

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:

```
dateDiff(toDate("2008-2-24 8:00:00"), toDate("2008-2-24 8:15:30"), "minute")
```

```

...returns a 15.5
dateDiff(toDate("2008-2-24 8:00:00"), toDate("2008-3-12 9:28:00"), "month")
...returns a 1.0
dateDiff(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. **Note:** months are returned zero-indexed. That is, January is month 0, February is month 1, and so on.

Examples:

```
dateExtract({Root Container.DatePicker.date}, "year")
```

...returns the year in the date in the DatePicker component.

dateFormat(*Date*, *Pattern*)

Returns the given Date as a string formatted according to Pattern.

For the following table, assume the Date is 7/8/2005 3:05:00 PM (July 8th, 2005).

Pattern Components		
Character	Function	Example
M	Month	7
MM	Month, forced 2 digits	07
MMM	Name of month, abbreviated.	Jul
MMMM	Name of month, full	July
d	Day of the month.	8
dd	Day of the month,forced 2 digits.	08
E	Day of the week, abbreviated.	Sun
EEEE	Day of the week, full.	Sunday
yy	Year - abbreviated.	05
yyyy	Year - Full	2005
H	Hour of the day (0-23)	15
h	Hour of the day (1-12)	3
m	Minute	5
mm	Minute, forced 2 digits.	05
s	Seconds	00

a	AM/PM marker	PM
z	Time zone, abbreviated.	PST
zzzz	Time zone, full	Pacific Standard Time

Pattern Examples

Pattern	Result
MM/dd/yyyy	07/08/2005
MMMM d, yyyy	July 8, 2005
M/d/yy h:mm a	7/8/05 3:05 PM
yyyy-MM-dd H:mm:ss	1999-01-08 14:05:06 Database format

now([pollRate])

Returns the current time. Will update every 1second by default, or at the rate specified by the `pollRate` parameter if present. Use a poll rate of zero to turn off updates.

Parameters

[pollRate]

An integer representing how long (in milliseconds) between updates. Whenever this function updates, the expression bind that it is a part of will execute. The default is 1000. Use 0 to disable updates.

Example:

To simply bind to the current date:

```
now()  
...would return a Date
```

To set up a clock, you can format the date like this:

```
dateFormat(now(), "MMM d, h:mm a")  
...would return a string like "Feb 12, 9:54 AM"
```



Aggregate Functions

groupConcat(DataSet, Column, Separator)

Concatenates all of the values in the given `Column` of the given `DataSet` into a string, with each value separated by the string `Separator`. Any null values in the column are ignored.

Parameters

DataSet

A `DataSet` that contains the column of data that this aggregate function will operate on.

Column

A Column index (starting at zero) or column name (case insensitive) of the column in which to operate

Separator

A string that will be placed in between each value

Example:

For this dataset:

Label
"bar"
"foo"
"baz"

```
groupConcat({Path.To.MyDataSet}, 0, ", ")  
...would return the string "bar, foo, baz"  
groupConcat({Path.To.MyDataSet}, "Label", "$")  
...would return the string "bar$foo$baz"
```

max(DataSet, Column)

Finds and returns the maximum value in the given `Column` of the given `DataSet`. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Parameters

DataSet

A `DataSet` that contains the column of data that this aggregate function will operate on.

Column

A Column index (starting at zero) or column name (case insensitive) of the column in which to operate

Example:

For this dataset:

Value
5
6
8
9

```
max({Path.To.MyDataSet}, 0)  
...would return 9
```

maxDate(DataSet, Column)

Finds and returns the maximum date in the given `Column` of the given `Dateset`. Any null values in the column are ignored. If there are no rows in the dataset, null is returned.

Parameters

DataSet

A `DataSet` that contains the column of data that this aggregate function will operate on.

Column

A Column index (starting at zero) or column name (case insensitive) of the column in which to operate

Example:

For this dataset:

Date
2006-09-17 8:00:00
2006-09-17 8:30:00
2006-09-17 9:00:00
2006-09-17 9:30:00

```
maxDate({Path. To. MyDataSet}, 0)
```

...would return 2006-09-17 9:30:00

mean(DataSet, Column)

Calculates the mean (a.k.a average) for the numbers in the given `Column` of the given `Dateset`. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Parameters

DataSet

A `DataSet` that contains the column of data that this aggregate function will operate on.

Column

A Column index (starting at zero) or column name (case insensitive) of the column in which to operate

Example:

For this dataset:

Value
5
6
8
9

```
mean({Path. To. MyDataSet}, 0)
```

...would return 7

median(DataSet, Column)

Calculates the median for the numbers in the given `Column` of the given `Dateset`. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Parameters

DataSet

A DataSet that contains the column of data that this aggregate function will operate on.

Column

A Column index (starting at zero) or column name (case insensitive) of the column in which to operate

Example:

For this dataset:

Value
5
6
8
9

```
median({Path. To. MyDataSet}, 'Value')
```

...would return 7

min(DataSet, Column)

Finds and returns the minimum value in the given Column of the given Dateset. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Parameters

DataSet

A DataSet that contains the column of data that this aggregate function will operate on.

Column

A Column index (starting at zero) or column name (case insensitive) of the column in which to operate

Example:

For this dataset:

Value
5
6
8
9

```
min({Path. To. MyDataSet}, 0)
```

...would return 5

minDate(DataSet, Column)

Finds and returns the minimum date in the given Column of the given Dateset. Any null values in the column are ignored. If there are no rows in the dataset, null is returned.

Parameters

DataSet

A DataSet that contains the column of data that this aggregate function will operate on.

Column

A Column index (starting at zero) or column name (case insensitive) of the column in which to operate

Example:

For this dataset:

Date
2006-09-17 8:00:00
2006-09-17 8:30:00
2006-09-17 9:00:00
2006-09-17 9:30:00

```
minDate({Path To. MyDataSet}, 0)  
...would return 2006-09-17 8:00:00
```

stdDev(DataSet, Column)

Calculates the standard deviation for the numbers in the given `Column` of the given `Dateset`. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Parameters

DataSet

A `DataSet` that contains the column of data that this aggregate function will operate on.

Column

A Column index (starting at zero) or column name (case insensitive) of the column in which to operate

Example:

For this dataset:

Value
5
6
8
9

```
stdDev({Path To. MyDataSet}, 'Value')  
...would return 1.5811
```

sum(DataSet, Column)

Calculates the sum of the numbers in the given `Column` of the given `Dateset`. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Parameters

DataSet

A `DataSet` that contains the column of data that this aggregate function will operate on.

Column

A Column index (starting at zero) or column name (case insensitive) of the column in which to operate

Example:

For this dataset:

Value
5
6
8
9

```
sum({Path.To.MyDataSet}, 'value')
```

...would return 28

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Advanced Functions

forceQuality(*value, [quality]*)

Returns the given value, but overwrites the quality of that value. If the quality argument is omitted, the quality will be GOOD. This is a way to have expressions opt-out of the quality overlay system. You can also force a specific quality code here by including the quality argument.

`forceQuality([]Tanks/Tank15)`

...returns value of the tag, but always with a good quality regardless of the quality of the tag.

`forceQuality([]Tanks/Tank15), 410)`

...returns value of the tag, but always with a TAG_DISABLED(410) quality.

runScript(*scriptFunction, [pollRate]*)

Runs a single line of Python code as an expression. If a poll rate is specified, the function will be polled. This is a very powerful way for you to add extensions to the Expression Language. For instance, one could write a script library function called **app.weather.getTempAt(zip)** that queried a web service for the current temperature at a given zipcode, and then bind the value of a label to the return value of that function.

`runScript("app.weather.getTempAt('95818')", 15000)`

...returns temperature in beautiful Sacramento, CA, USA. Polls for a new temperature every 15 seconds. Note: app.weather.getTempAt is a hypothetical scripting function that you could write. Just to show that such a function is not only possible, but simple, here is an implementation for the hypothetical app.weather module:

```
# This function would query Yahoo Weather for the temperature at
# the given zipcode and find the temperature using a regular expression
def getTempAt(zipCode):
    import fpmi
    import re #Regular Expression library

    response = fpmi.net.httpGet("http://xml.weather.yahoo.com/forecastrss?p=" + str(zipCode))

    # NOTE - if you've never seen regular expressions before, don't worry, they look
    # confusing even to people who use them frequently.
    pattern = re.compile('.*?<yweather:condition (.*)/>', re.DOTALL)
    match = pattern.match(response)
    if match:
        subText = match.group(1)
        temp = re.compile('.*?temp="(.*?)"').match(subText).group(1)
        return int(temp)
    else:
        fpmi.gui.errorBox("Yahoo weather service changed")
        return -1
```

tag(*path*)

Returns the value for the given tag path. The path can point to a non-value property of a tag.

`tag("[]Tanks/Tank15")`

...returns value of the tag.

`tag("[]Tanks/Tank15.Tooltip")`

...returns tooltip meta property of the tag.

`tag("[]Tanks/Tank" + {Root Container.TankNum} + ".Tooltip")`

...returns tooltip meta property of the tank referenced by the dynamic property. While this is possible, we recommend using indirect tag binding instead.



The Jython Language

FactoryPMI makes extensive use of the *Jython* scripting language. Jython is the Java version of the popular scripting language Python. The language is identical to Python, the main difference is that for advanced users, the entire Java standard library is available to use. FactoryPMI uses Jython because it gracefully combines ease-of-use, readability, and power. Jython is an *extremely* powerful programming language.

Many users will be able to use FactoryPMI without ever using Jython (directly). However, if you are working on an advanced project with some very custom requirements, you will probably want to spend some time learning Jython. While we provide a simplified introduction to the Jython language [here](#), covering every aspect of it would require an entire book! Fortunately, other more qualified authors have already written these books. We recommend anyone using FactoryPMI in an advanced manner pick up a copy of *Jython Essentials* by Samuele Pedroni and Noel Rappin, published by O'Reilly & Associates. (ISBN 0-596-00247-5).

Using Jython in FactoryPMI

There are 3 different areas where one will encounter Jython within a FactoryPMI project.

1. **Action Scripts.** Action scripts are the most commonly encountered Jython scripts in FactoryPMI. These scripts, which can also be thought of as *event handlers*, run every time an *event* is fired from a component. These are often in direct response to the user, such as mouse clicks, mouse movement, key presses, component focus, etc. Events can also be fired for other reasons, such as a property of a component changing. See [Action Scripts](#) for more detail.
2. **Global Event Scripts.** Global scripts are scripts that run in response to global events - they are not tied to a particular window or component. Types of global event scripts are:
 - **Project Startup.** Use these to run initializing code if needed.
 - **Project Shutdown.** Use this to run clean-up code before an application is closed, if needed.
 - **Project Shutdown Interception.** This can be used to intercept the shutdown event and cancel it. This is useful if you want to restrict the ability to shut down the FactoryPMI client.
 - **Keystroke.** Use these scripts to respond to specific keystrokes (like the F-keys) to implement mouse-free navigation, or to access commonly used functions
 - **Timer.** Use these to run scripts repeatedly on a timer.
3. **Global Script Modules.** Never cut-and-paste Jython code into multiple action scripts! Instead, write commonly used functions once and put them in the `app` module, a user-defined global module. This makes project maintenance much nicer, as well as allowing the opportunity to write complicated, industry-specific modules and copy them to multiple projects.

Built-in Jython Modules

Much of the Jython you write will involve calling functions from the FactoryPMI built-in modules. These functions facilitate interaction with your datasources, window navigation, etc. A detailed reference to the FactoryPMI built-in modules can be found [here](#).



Jython is a very easy to use and human-readable programming language. Those *not* familiar with procedural languages (Ladder Logic is not a procedural language. C, FORTRAN, Java, Pascal, and Delphi are examples of procedural languages), it should be quite easy to pick up and use. Those already familiar with procedural languages will likely find Jython not only easy to use, but refreshingly concise and elegant.

As explained below, Jython and Python are identical languages. Because of the popularity of the Python language, there is a veritable plethora of tutorial resources available, both in print and electronically. Also, we have sprinkled small example Jython scripts that should be very readable and useful in real-life situations. As you go through some of these tutorials, we'd like to point your attention to the FactoryPMI "Script Playground", which lets run Jython scripts in a console-like environment. This is found in the Designer under Tools > Advanced.

- [Non-Programmers Tutorial For Python](#). (by Josh Cogliati) This is a great resource for people who aren't very familiar with procedural languages. It starts at the beginning - introducing statements and variables, and covers the various Python variable types.
- [Learn to Program using Python](#). (by Richard Baldwin) This page works well as a reference and as a tutorial.
- [Python Tutorial](#). This is the official python tutorial. It covers pretty much everything, and is well organized. You can skip chapter 2, as you will be using FactoryPMI as your interpreter.

Advanced users: don't be fooled by Python's elegant, readable and forgiving syntax: it is *not* just a beginner's language. Python supports a variety of programming styles, from simple scripts to a fully polymorphic Object-Oriented application.

Nuts and Bolts

Jython is the same language as the popular language Python. Jython is simply Python running in Java (as opposed to Python running in C, which is what most people are familiar with). So, if you know Python, you know Jython. The major difference is that in Jython, the entire Java standard library is available to use, for the more advanced user. In FactoryPMI, there are some more differences. In order to keep the Jython scripting engine smaller, only parts of the Python standard library are included.



Action Scripts

Many of the scripts that you write will be **Action Scripts**. These are scripts that are linked to a specific component action. For instance, you could pop up a message when an image component is clicked, or you could send an SQL query to the database whenever the letter 'p' is typed into a text box.

Each action script has three things in its namespace before the script starts running:

- **fpmi** The fpmi module is automatically imported into action scripts. This saves you from having to type 'import fpmi' at the top of all of your action scripts. More on the fpmi module can be found in the [Built-in Modules](#) section.
- **app** The app module is automatically imported as well. See the [Global Script Module](#) page for more information.
- **event** The event object is the heart of the action script. This object provides specific information about the event you are handling, as well as providing a handle into the window.

The Event Object

The event object is what sets action scripts apart from other scripts that you may write (like global modules or global events). You can think of the event object as your link into the context of the event you are responding to. This object will contain information about that event. So each type of event will have different information in its event object. For instance, to find out how many times the mouse was clicked on a mouseClicked event, you can use event.clickCount. To find out the name of a property that changed on a propertyChange event, you can use event.propertyName. But you can't use event.propertyName on a mouseClicked event, because the mouse event doesn't support that property.

The event object for *all types* of events share one property: the source property. This property gives you a reference to the component that caused the event you are responding to. For instance, if you are responding to the mouseClicked event on a Label component, event.source in that script will be the Label. This allows you to do things like:

```
event.source.text = "I was clicked"
```

This would change the text of the label to "I was clicked" when the user clicks the label.

Event Sets

The various events that occur in FactoryPMI are organized into *event sets*. An event set is a set of related events, such as mouse events (clicking, pressing, hover, etc) or key events. All of the events in an event set share the same kind of event object.

The rest of this page serves as a reference to the types of events that you will come across while writing action scripts in FactoryPMI.

- [action](#)
- [cell](#)
- [focus](#)
- [item](#)
- [key](#)
- [mouse](#)
- [mouseMotion](#)
- [propertyChange](#)

action

Events:

- [actionPerformed](#)

This event is fired when the 'action' of the component occurs. What this action is depends on the type of the component. Most commonly, this is used with buttons, where the action is that the button was pushed, via a mouse click or a key press. See the component reference for details on what the action means for other components.

event object properties:

- **source**

The source component for this event.

cell

Events:

- **cellEdited**

This event is fired when a cell in a table component is edited.

event object properties:

- **source**

The source table component for this event.

- **oldValue**

The old value in the cell.

- **newValue**

The value the cell has changed to.

- **row**

The row in the dataset this edit occurred at.

- **column**

The column in the dataset this edit occurred at.

focus

Events:

- **focusGained**

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

- **focusLost**

This event occurs when a component that had the input focus lost it to another component.

event object properties:

- **source**

The source component for this event.

- **oppositeComponent**

The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vice versa.

item

Events:

- **itemStateChanged**

This event is found on components that can be 'on' or 'off', such as Checkboxes, Radio Buttons, and Toggle Buttons. This event is the best way to respond to the state of that component changing.

event object properties:

- **source**

The source component for this event.

- **stateChange**

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.

event object constants:

- **SELECTED**

The code for a selected state.

- **DESELECTED**

The code for a deselected state

Example:

```
if event.stateChange == event.SELECTED:  
    print 'This component was selected'  
else:  
    print 'This component was deselected'
```

key

Events:

- **keyPressed**

Fires when a key is pressed and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

- **keyReleased**

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

- **keyTyped**

Fires when a key is pressed and then released when source component has the input focus. Only works for characters that can be printed on the screen.

event object properties:

- **source**

The source component for this event.

- **keyCode**

The key code for this event. Used with the keyPressed and keyReleased events. See below for the key code constants.

- **keyChar**

The character that was typed. Used with the keyTyped event.

- **keyLocation**

Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants below. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.

- **altDown**

True (1) if the Alt key was held down during this event, false (0) otherwise.

- **controlDown**

True (1) if the Control key was held down during this event, false (0) otherwise.

- **shiftDown**

True (1) if the Shift key was held down during this event, false (0) otherwise.

event object constants:

- **VK_0** through **VK_9**

The key codes for the 10 digits.

- **VK_A** through **VK_Z**

The key codes for the 26 letters.

- **VK_F1** through **VK_F24**
The key codes for the F-keys.
- **VK_ALT**
The key code for the Alt key.
- **VK_CONTROL**
The key code for the Control (CTRL) key.
- **VK_DOWN**
The key code for the Down arrow key.
- **VK_END**
The key code for the End key.
- **VK_ENTER**
The key code for the Enter key.
- **VK_HOME**
The key code for the Home key.
- **VK_INSERT**
The key code for the Insert key.
- **VK_LEFT**
The key code for the Left arrow key.
- **VK_PAGE_DOWN**
The key code for the Page Down key.
- **VK_PAGE_UP**
The key code for the Page Up key.
- **VK_RIGHT**
The key code for the Right arrow key.
- **VK_SHIFT**
The key code for the Shift key.
- **VK_SPACE**
The key code for the Space key.
- **VK_TAB**
The key code for the Tab key.
- **VK_UP**
The key code for the Up arrow key.
- **KEY_LOCATION_LEFT**
The location code for the left.
- **KEY_LOCATION_NUMPAD**
The location code for the numeric keypad.
- **KEY_LOCATION_RIGHT**
The location code for the right.
- **KEY_LOCATION_STANDARD**
The location code for standard keys.
- **KEY_LOCATION_UNKNOWN**
The location code for the an unknown location.

Examples:

```
# This would be for a keyPressed or keyReleased event
# on a textfield
if event.keyCode == event.VK_ENTER and event.controlDown:
    # The user pressed CTRL-Enter, insert the text in the database
    fpmi.db.runUpdateQuery("INSERT INTO MyTable (text_column) VALUES ('%s')" %
event.source.text)
```

```
# This would be for a keyTyped event
# on any input component
print 'You typed ' + event.keyChar
```

mouse

Events:

- mouseClicked

This event signifies a mouse click on the source component. A mouse click is the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

- mouseEntered

This event fires when the mouse enters the space over the source component.

- mouseExited

This event fires when the mouse leaves the space over the source component.

- mousePressed

This event fires when a mouse button is pressed down on the source component.

- mouseReleased

This event fires when a mouse button is released, if that mouse button's press happened over this component.

event object properties:

- **source**

The source component for this event.

- **button**

The code for the button that caused this event to fire.

- **clickCount**

The number of mouse clicks associated with this event.

- **x**

The x-coordinate (with respect to the source component) of this mouse event.

- **y**

The y-coordinate (with respect to the source component) of this mouse event.

- **popupTrigger**

Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.

- **altDown**

True (1) if the Alt key was held down during this event, false (0) otherwise.

- **controlDown**

True (1) if the Control key was held down during this event, false (0) otherwise.

- **shiftDown**

True (1) if the Shift key was held down during this event, false (0) otherwise.

event object constants:

- **NOBUTTON**

Indicates no mouse buttons were involved in this event.

- **BUTTON1**

Indicates mouse button #1.

- **BUTTON2**

Indicates mouse button #2.

- **BUTTON3**

Indicates mouse button #3.

Example:

```
# This script would run on double-click of the main mouse button
if event.clickCount == 2 and event.button == event.BUTTON1:
    fpmi.gui.messageBox("You double clicked this component.")
```

mouseMotion

Events:

- **mouseDragged**
Fires when the mouse moves over a component after a button has been pushed.
- **mouseMoved**
Fires when the mouse moves over a component, but no buttons are pushed.

event object properties:

- **source**
The source component for this event.
- **button**
The code for the button that caused this event to fire.
- **clickCount**
The number of mouse clicks associated with this event.
- **x**
The x-coordinate (with respect to the source component) of this mouse event.
- **y**
The y-coordinate (with respect to the source component) of this mouse event.
- **popupTrigger**
Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
- **altDown**
True (1) if the Alt key was held down during this event, false (0) otherwise.
- **controlDown**
True (1) if the Control key was held down during this event, false (0) otherwise.
- **shiftDown**
True (1) if the Shift key was held down during this event, false (0) otherwise.

event object constants:

- **NOBUTTON**
Indicates no mouse buttons were involved in this event.
- **BUTTON1**
Indicates mouse button #1.
- **BUTTON2**
Indicates mouse button #2.
- **BUTTON3**
Indicates mouse button #3.

propertyChange

Events:

- **propertyChange**
Fires whenever a *bindable* property of the source component changes. This works for standard and custom (dynamic) properties.

event object properties:

- **source**
The source component for this event.
- **newValue**
The new value that this property changed to.
- **oldValue**
The value that this property was before it changed. Note that not all components include

an accurate oldValue in their events.

- **propertyName**

The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Example:

```
# This script would run on the propertyChange event of a numeric text
# field, so that processing could be done whenever the number changed.
if event.propertyName == 'int Value':
    fpmi.gui.messageBox("The value changed to: %d" % event.newValue)
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Global Python Module - the app package.

Often when writing your action scripts, you will find that you may write the same code over and over. Or maybe you write the code once, and then copy that component multiple times. These are both situations where you should write a global function.

For example, suppose you want to make a button that sets a bit in the database, but only if the user has the appropriate security clearance. Lets say that the bit is a control bit that turns a motor on. Your button's action script would look like this:

```
if "Supervisor" in fpmi.security.getRoles():
    fpmi.db.runUpdateQuery("UPDATE Motors SET Running = 1 WHERE MotorNum=2")
else:
    fpmi.gui.errorBox("You don't have access to turn the motor on.")
```

Now, you copy that button around on your screens, modifying the query to read "WHERE MotorNum=3", "WHERE MotorNum=4", etc. What's wrong with this? One day, the operators are going to approach you saying, *"Hey, how come only supervisors can turn motors on. We should be able to also"*. Then, you would have to edit each button to allow users with the "Operator" role to turn the motor on. This is an error-prone headache that you want to avoid.

Instead, what you should do is add a global module for motor control. To access your application's custom modules, click on the **Project > Script Modules** menu. You will see that you can add new *packages* and *modules*. A *package* is simply a way to organize multiple modules. All of your modules and packages will be in the `app` package. A *module* is a python script that usually defines python functions.

Create a new module under the `app` package, called `motorcontrol`. In the new `app.motorcontrol` module, we want to create a function that can turn any motor on. It would look like this:

```
def startMotor(motorNum):
    import fpmi #NOTE - functions don't get the fpmi module imported automatically
    if "Supervisor" in fpmi.security.getRoles() or "Operator" in fpmi.security.getRoles():
        fpmi.db.runUpdateQuery("UPDATE Motors SET Running=1 WHERE MotorNum=%d" % motorNum)
    else:
        fpmi.gui.errorBox("You don't have access to turn the motor on.")
```

Now, all of our buttons should be modified to run the following script:

```
app.motorcontrol.startMotor(2)
```

Now, your buttons are insulated against changes! If later they decide that everyone should be able to turn on a motor, or if an "Are you sure?" confirmation box should be added, you only have to change one script.

Tips & Tricks - Global Variables

It is often useful to use a global variable. Of course, in Jython, all running scripts have two namespaces, their local namespace, and the global namespace. So, you can simply declare that a name should be found in the global namespace, and use it. However, you usually want to be able to control the variable a bit more, including what it gets initialized to. For this reason, we like to wrap access to global variables in *accessor methods*. For example, the following two accessor methods allow getting and setting the value of the global variable "MyVar", which would have an initial value of "Initial Value". This would be a in a global module, like `app.variables`

```
def getMyVar():
    global MyVar
    try:
        return MyVar
    except NameError:
        MyVar='Initial Value'
    return MyVar

def setMyVar(newValue):
    global MyVar
    MyVar = newValue
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Modules - Overview

Much of what you will do in Jython will be calling functions in the FactoryPMI internal jython modules. These modules reside in the fpmi module, which is automatically imported into component event scripts. These modules contain useful functions to do common tasks such as communicating with the database and switching between windows. There are five modules contained in the fpmi module. They are:

[**fpmi.gui**](#)

Contains many functions to manipulate the FactoryPMI GUI. These include opening, closing, and swapping between windows, and displaying error and message dialogs.

[**fpmi.db**](#)

Contains functions to communicate with datasources

[**fpmi.security**](#)

Contains functions to deal with security issues, such as username and roles

[**fpmi.net**](#)

Contains network related functions.

[**fpmi.print**](#)

Contains functions to facilitate printing from FactoryPMI

In general, to use a function simply call its fully qualified name, with the required parameters. For instance, to close a window you might call: *fpmi.gui.closeWindow("MyWindow")*. Remember, function names are case sensitive.

Modules - gui

The `fpmi.gui` module provides a number of useful functions dealing with the user interface:

chooseColor(Color, [Title])

Prompts the user to pick a color using the default color-chooser dialog box.

Parameters

Color

The color that the chooser dialog will use as a starting point.

Title

The title of the dialog box. (optional - default is "Choose Color")

Example:

```
# This code would be placed in the actionPerformed event of a button,  
# and would change the background color of the container the button was placed in.
```

```
parent = event.source.parent  
newColor = fpmi.gui.chooseColor(parent.background)  
parent.background = newColor
```

color(Red, Green, Blue, [Alpha])

Returns a color object.

Parameters

Red

The amount of red (0-255) in the color.

Green

The amount of green (0-255) in the color.

Blue

The amount of blue (0-255) in the color.

[Alpha]

(Optional) The transparency of the color, with 0 being totally transparent, and 255 being totally opaque.

Example:

```
myComponent = event.source  
myComponent.background = fpmi.gui.color(255, 0, 0) # turn the component red
```

color(Name)

Returns a color based on a name.

Parameters

Name

The name of a color. Possibilities are:

- black
- blue
- cyan
- darkGray
- gray
- green
- lightGray
- magenta
- orange
- pink
- red
- white
- yellow

Example:

```
myComponent = event.source  
myComponent.background = fpmi.gui.color('red') # turn the component red
```

confirm(Message, Title)

Displays a confirmation dialog box with "Yes" and "No" options. Returns true (non-zero) if "Yes" is chosen, zero otherwise.

Parameters

Message

The message (question) to show.

Title

The title of the dialog box.

Example:

```
# By using the confirm function in an if statement, we can let the user
# confirm an action. In this case, we shut down the plant if the user
# confirms it, otherwise, we don't do anything
if fpmi.gui.confirm("Are you sure you want to shutdown the plant?", "Really Shutdown?"):
    fpmi.db.runUpdateQuery("UPDATE ControlTable SET Shutdown=1")
```

createPopupMenu(FunctionNames, Functions)

Creates a popup menu object, which can then be used in mouse click events to display the popup menu over a component.

Parameters

FunctionNames

A Jython sequence of function names.

Functions

A Jython sequence of function objects OR a sequence of (FunctionNames, Functions) sequences for submenu entries..

To use this function, first create a Jython sequence whose entries are strings, and another sequence whose entries are Jython functions. The strings will be the items that are displayed in your popup menu, and when an item is clicked, its corresponding function will be run. Your functions must accept an event object as an argument.

To show the popup menu, just store the menu object that this function returns, and then call that object's `show(eventObject)` function, where `eventObject` is the event object for a `mousePressed` or `mouseReleased` event on the component you wish the popup menu to be shown on.

Best Practices. It is best to have the menu object created only once via an application specific library function. Then, call the `show(eventObject)` function on *both* the `mousePressed` and `mouseReleased` events on your component. The reason for this is that different operating systems (Windows, Linux, MacOS) differ in when they like to show the popup menu. The `show(eventObject)` function detects when the right time is to show itself, either on mouse press or release.

The following example shows a popup menu being used to acknowledge alarms in an alarm table by right-clicking on the table, and choosing either to acknowledge the selected alarm or all alarms.

This would be put in a project library module called `app.util`:

```
def getAlarmPopup():
    # Declare that alarmPopup should be found in the global namespace
    global alarmPopup
    try:
        # If this function has been run before, this will work
        return alarmPopup
    except NameError:
        # If this function hasn't been run before, this will run
        import fpmi, app

        # This function will be the "Acknowledge" entry in the popup menu
        def ack(event):
            import fpmi, app
            table = event.source
            selRow = table.selectedRow
            if selRow == -1:
                fpmi.gui.warningBox("No alarm selected")
            elif table.model.getValueAt(selRow, 0) == 0:
                # In my table, the first column is the alarm's unacknowledged bit.
                fpmi.gui.warningBox("Alarm already acknowledged")
            else:
                desc = table.model.getValueAt(selRow, 1)
                path = table.model.getValueAt(selRow, 2)
                if fpmi.gui.confirm("<HTML>Are you sure you want to acknowledge<BR>%s?" % desc, "Confirm"):
                    app.auth.ackAlarm(desc, path)
                    table.setSelectedRow(-1)

        # This function will be the "Acknowledge All" entry in the popup menu
        def ackAll(event):
            import fpmi, app
```

```

if fpmi.gui.confirm("Are you sure you want to acknowledge all alarms?", "Confirm"):
    app.auth.ackAllAlarms(event)

# Finally, create the actual popup menu and assign it to global variable alarmPopup
alarmPopup = fpmi.gui.createPopupMenu(["Acknowledge Alarm", "Acknowledge All"], [ack, ackAll])
return alarmPopup

```

This is the code in both the `mousePressed` and `mouseReleased` events of the Table component:

```

menu = app.util.getAlarmPopup()
menu.show(event)

```

errorBox(Message, [Title])

Displays an error box to the user.

Parameters

Message

The error message to show.

[Title] (optional)

The title of the dialog box, or "Error" if omitted.

Example:

```
# Turn on compressor #12, but only if the user has the right credentials
```

```

if 'Supervisor' in fpmi.security.getRoles():
    fpmi.db.runUpdateQuery("UPDATE CompressorControl SET running=1 WHERE compNum = 12")
else:
    fpmi.gui.errorBox("Unable to turn on Compressor 12. You don't have proper security privileges.")

```

getOpenedWindows()

Returns a tuple of the currently opened windows.

Example:

```

windows = fpmi.gui.getOpenedWindows()
print 'There are %d windows open' % len(windows)
for window in windows:
    print window.name

```

getParentWindow(eventObject)

Returns a reference to the window containing `eventObject.source`.

Parameters

eventObject

The event object generated for any event.

Example:

```
# Use this in an event script to change the window's title
window = fpmi.gui.getParentWindow(event)
window.title='This is a new title'
```

getSibling(eventObject, Name)

Returns a reference to the sibling component of `eventObject.source` named `name` or `None` if the sibling doesn't exist. This is a short-cut for: `eventObject.parent.getComponent("MySiblingsName")`

Parameters

eventObject

The event object generated for any event.

Name

The name of the sibling you want to retrieve.

Example:

```
# Get my sibling text box's text, and use it
textField = fpmi.gui.getSibling(event, 'TextField (1)')
if textField is None:
    fpmi.gui.errorBox("There is no text field!")
else:
    fpmi.gui.messageBox("You typed: %s" % textField.text)
```

getWindow(Name)

Returns a reference to the window named `name`. Throws a `ValueError` if the window is not currently open.

Parameters

Name

The name of the window you wish to retrieve.

Example:

```
# Get the window named 'Overview' and then close it
try:
    window = fpmi.gui.getWindow('Overview')
    fpmi.gui.closeWindow(window)
except ValueError:
    fpmi.gui.warningBox("The Overview window isn't open")
```

getWindowNames()

Returns a list of all of the current project's windows names sorted in alphabetical order.

inputBox(Prompt, initialValue)

Opens up a popup input dialog box. This dialog box will show the message `Prompt`, and allow the user to type in a string. The text box in the dialog will initially have the value of `InitialValue`. When the user is done, they can press OK or Cancel. If OK is pressed, this function will return with the value that they typed in. If Cancel is pressed, this function will return the value `None`.

Parameters

Prompt

The message that will be displayed in the input dialog box.

InitialText

The initial text value in the text box.

Example:

```
# This could go in the mouseClicked event of a label
# to allow the user to change the label's text
txt = fpmi.gui.inputBox("Enter text:", event.source.text)
if txt != None:
    event.source.text = txt
```

isTouchscreenModeEnabled()

Note - As of FactoryPMI 3.0, all input components are already touchscreen-aware. You don't need to use this anymore.

Returns true if the FactoryPMI runtime was opened in touchscreen mode. Used in applications that may be opened on touchscreen monitors, to detect if there is a need to pop up a touchscreen keyboard when a text box is clicked.

Example (to be used in the `mouseClicked` event of a text input component):

```
# The mouse was clicked in this component - check to see if we need a touchscreen keyboard
if fpmi.gui.isTouchscreenModeEnabled():
    # Set the text to the return value of the keyboard
    event.source.text = fpmi.gui.showTouchscreenKeyboard(event.source.text)
```

Example (to be used in the `mouseClicked` event of a numeric input component):

```
# The mouse was clicked in this component - check to see if we need a touchscreen keypad
if fpmi.gui.isTouchscreenModeEnabled():
    # Set our value to the return value of the keypad
    event.source.intValue = fpmi.gui.showNumericKeypad(event.source.intValue)
```

messageBox(Message, [Title])

Displays an informational message box to the user.

Parameters

Message

The message to show.

[Title] (optional)

The title of the dialog box, or "Information" if omitted.

Example:

```
# Show how many hours a motor has been running when it is clicked
# First, get the motor number, which is a custom property of the motor component
```

```

motorNumber = event.source.getPropertyValue('MotorNumber')

# Now retrieve the hours running from the database
hours = fpmi.db.runScalarQuery("SELECT HoursRunning FROM MotorStatus WHERE motor=%d" % motorNumber)

fpmi.gui.messageBox("The motor has been running for %d hours" % motorNumber)

```

moveComponent(component, x, y)

Sets a component's position. Note that if the component is set to relative layout mode, the position is relative to the size of its parent container as it was when the window was saved.

Parameters

component

A reference to the component to reshape.

x

An integer representing the new X position for the component.

y

An integer representing the new Y position for the component.

Example:

```

# This code would go in a Timer's propertyChange script for animation
if event.propertyName == "value":
    newX = event.newValue;
    rect = event.source.parent.getComponent("Rectangle")
    fpmi.gui.moveComponent(rect, newX, 250)

```

passwordBox(Message, [Title], [EchoChar])

Pops up an input box that is made for passwords (typed text is masked). Returns the password that the user typed in, or `None` if the user hit **Cancel**.

Parameters

Message

A string to display to the user. Something like: "Please enter the password".

Title (optional)

A string used as the title of the password input box's window. Default is "Password".

EchoChar (optional)

The character used to echo back to the user. Default is an asterisk.

Example:

```

password = fpmi.gui.passwordBox("Please enter the password.")
if password == "abra-ca-dabra":
    fpmi.gui.messageBox("Access granted!")

```

reshapeComponent(component, x, y, width, height)

Sets a component's position and size. Note that if the component is set to relative layout mode, the position and size are relative to the size of its parent container as it was when the window was saved.

Parameters

component

A reference to the component to reshape.

x

An integer representing the new X position for the component.

y

An integer representing the new Y position for the component.

width

An integer representing the new width for the component.

height

An integer representing the new height for the component.

Example:

```
# This code would go in a Timer's propertyChange script for animation
if event.propertyName == "value":
    newX = event.newValue;
    newWidth = int(event.newValue*1.5)
    rect = event.source.parent.getComponent("Rectangle")
    fpmi.gui.reshapeComponent(rect, newX, 150, newWidth, 80)
```

resizeComponent(component, width, height)

Sets a component's size. Note that if the component is set to relative layout mode, the size is relative to the size of its parent container as it was when the window was saved.

Parameters

component

A reference to the component to reshape.

width

An integer representing the new width for the component.

height

An integer representing the new height for the component.

Example:

```
# This code would go in a Timer's propertyChange script for animation
if event.propertyName == "value":
    newWidth = event.newValue;
    rect = event.source.parent.getComponent("Rectangle")
    fpmi.gui.resizeComponent(newWidth, 80)
```

setTouchscreenModeEnabled(value)

Sets (0 or 1) whether touchscreen mode is on or off. Useful for touchscreen applications that are auto-login, so that you can enable touchscreen mode in the startup script.

Parameters

value

A boolean (0 or 1) to set the touchscreen mode to.

Example:

```
fpmi.gui.setTouchscreenModeEnabled(1)
```

showNumericKeypad(Value, [FontSize])

Note: As of FactoryPMI 3.0, this will be done for you on all input components

Opens a numeric input keypad with an initial value specified. The given FontSize will be used, or font size 20 if omitted. This is useful for touchscreens, and works particularly well in mouseClicked actions for text boxes. This function returns the number chosen. The keypad will be 30% of the size of the window, and anchored to the bottom.

Parameters

Value

The initial value to display.

[FontSize] (optional)

The font size to use. Default is 20.

Example:

```
# This would go in the MouseClicked or MousePressed action of a Text Field or Numeric Text Field.
```

```
# For Integer Numeric Text Field:
```

```
if fpmi.gui.isTouchscreenModeEnabled():
    event.source.intValue = fpmi.gui.showNumericKeypad(event.source.intValue)
```

```
# For Double Numeric Text Field:
```

```
if fpmi.gui.isTouchscreenModeEnabled():
    event.source.doubleValue = fpmi.gui.showNumericKeypad(event.source.doubleValue)
```

```
# For Text Field:
```

```
# notice the str() and int() functions used to convert the text to a number and vice versa
# str() and int() are built-in Jython functions
if fpmi.gui.isTouchscreenModeEnabled():
    event.source.text = str(fpmi.gui.showNumericKeypad(int(event.source.text)))
```

showTouchscreenKeyboard(*Text*, [*FontSize*], [*Password?*])

Note: As of FactoryPMI 3.0, this will be done for you on all input components

Opens a keyboard on screen for use with touchscreens. The keyboard starts up with the Text given as a parameter, and returns the edited text when the keyboard closes.

Parameters

Text

The initial text to edit.

FontSize (optional, default=20)

The size of the font of the keyboard

Password? (optional, default=false)

If true, the popup keyboard won't echo back the characters that are typed.

Example:

```
# This would go in the MouseClicked or MousePressed action of a Text Field or similar component.
```

```
if fpmi.gui.isTouchscreenModeEnabled():
    event.source.text = fpmi.gui.showTouchscreenKeyboard(event.source.text)
```

warningBox(*Message*, [*Title*])

Displays a warning message box to the user.

Parameters

Message

The warning to show.

[*Title*] (optional)

The title of the dialog box, or "Warning" if omitted.

Example:

```
# Start the motor, or, warn the user if in wrong mode
runMode = event.source.parent.getPropertyValue('RunMode')
if runMode == 1: Cannot start the motor in mode #1
    fpmi.gui.warningBox("Cannot start the motor, current mode is <B>VIEW MODE</B>")
else:
    fpmi.db.runUpdateQuery("UPDATE MotorControl SET MotorRun=1")
```



Modules - nav

The fpmi.nav module provides functions for navigating between windows. Note, this library is new as of FactoryPMI 3.0, and implements functions that used to be in fpmi.gui. The new functions, **swapTo**, **goBack**, **goForward**, and **goHome** are aimed towards projects that have one central, maximized window that swaps out with other windows to create the effect of 'screens', docked windows, and popup windows.

centerwindow(*WindowName*)

Centers the window on the screen

Parameters

WindowName

The name of the window to center.

Example 1:

```
# Center the window named 'Overview'.
fpmi.nav.centerWindow('Overview')
```

closeWindow(*Window*)

Closes the specified window.

Parameters

Window

The name of the window to close, or the actual window reference itself (as returned by functions such as [getWindow\(*WindowName*\)](#)).

Example 1:

```
# Get the window named 'Overview' and then close it.
# If the window isn't open, show a warning
try:
    window = fpmi.gui.getWindow('Overview')
    fpmi.nav.closeWindow(window)
except ValueError:
    fpmi.gui.warningBox("The Overview window isn't open")
```

Example 2:

```
# Close the window named 'Overview' in one step.
# If the window isn't open, the call to closeWindow will have no effect
fpmi.nav.closeWindow('Overview')
```

closeParentWindow(*EventObject*)

Closes the window that is the parent of the source of *EventObject*

Parameters

EventObject

The object that is generated with any event (action).

Example:

```
# This code would be placed in the actionPerformed event of a button,
# and would close the window that contained the button.

fpmi.nav.closeParentWindow(event)
```

openWindow(*Name*, [Parameters])

Opens the window named *Name*. If the window is already open, brings it to the front. The optional Dictionary parameter *Parameters* contains key, value pairs which will be used to set the target window's root container's dynamic variables.

For instance, if the window that you are opening is named "TankDisplay" has a dynamic variable in its root container named "TankNumber", then calling `fpmi.nav.openWindow("TankDisplay", {"TankNumber" : 4})` will open the "TankDisplay" window and set Root Container.TankNumber to four. This is useful for making *parameterized windows*, that is, windows that are re-used to display information about like pieces of equipment.

Parameters

Name

The name of a window to open.

Parameters (optional)

A Jython Dictionary listing parameter names and values that will be set in the new window. A windows "parameters" are the variables defined in the root container.

Example:

```
# This is the simplest form of openWindow
fpmi.nav.openWindow("SomeWindowName")

# A more complex example - a setpoint screen for multiple valves
titleText = "Third Valve Setpoints"
tankNo = fpmi.nav.openWindow("ValveSetPts", {"valveNum":3, "titleText":titleText})
```

swapTo(*Name*, [Parameters])

This function *swaps* from the current window to the window specified by *Name*. Swapping windows means that the new window is opened and set to the same size as the current window, and then the current window is closed. The current window is defined as a window that is currently opened, not docked (a.k.a. Floating), and maximized. This gives the impression that the current window has simply turned into the new window. The optional parameters dictionary has the same meaning as in [openWindow](#).

Parameters

Name

The name of the window to swap to.

Parameters (optional)

A Jython Dictionary listing parameter names and values that will be set in the new window. A window's "parameters" are the variables defined in the root container.

Example 1:

```
# This code would go in a button's ActionPerformed event to swap out of the
# current window and into a window named MyWindow

fpmi.nav.swapTo("MyWindow")
```

Example 2:

```
# This code would go in a button's ActionPerformed event to swap out of the
# current window and into a window named MyWindow
# It also looks at the selected value in a dropdown menu and passes that value into the new window.
# MyWindow's Root Container must have a dynamic property named "paramValue"
```

```
dropdown = event.source.parent.getComponent("Dropdown")
fpmi.nav.swapTo("MyWindow", {"paramValue":dropdown.selectedValue})
```

swapWindow(eventObject, Name, [Parameters])

This window swaps from the current window (the parent of eventObject.source) to the window specified by *Name*. Swapping windows means that the new window is opened and set to the same size as the current window, and then the current window is closed. This gives the impression that the current window has simply turned into the new window. The optional parameters dictionary has the same meaning as in [openWindow](#).

Parameters

eventObject

The event object in the window that will be closed.

Name

The name of the window to swap to.

Parameters (optional)

A Jython Dictionary listing parameter names and values that will be set in the new window. A window's "parameters" are the variables defined in the root container.

Example 1:

```
# This code would go in a button's ActionPerformed event to swap out of the
# window containing the button and into a window named MyWindow

fpmi.nav.swapWindow(event, "MyWindow")
```

Example 2:

```
# This code would go in a button's ActionPerformed event to swap out of the
# window containing the button and into a window named MyWindow
# It also looks at the selected value in a dropdown menu and passes that value into the new window.
# MyWindow's Root Container must have a dynamic property named "paramValue"
```

```
dropdown = event.source.parent.getComponent("Dropdown")
fpmi.nav.swapWindow(event, "MyWindow", {"paramValue":dropdown.selectedValue})
```

swapWindow(SwapFrom, SwapTo, [Parameters])

This function is the same as the other swapWindow, but uses a name for the window to be closed rather than finding it from an eventObject.

Parameters

SwapFrom

The name of the window to close.

SwapTo

The name of the window to open.

Parameters (optional)

A Jython Dictionary listing parameter names and values that will be set in the new window.

Example 1:

```
# This code would swap from window named WindowA to a window named WindowB

fpmi.nav.swapWindow("WindowA", "WindowB")
```

Example 2:

```
# This code would swap from window named WindowA to a window named WindowB.
```

It also looks at the two calendar popup controls and passes the two selected dates to
WindowB. WindowB's Root Container must have dynamic properties named "startDate" and
"endDate"

```
date1 = event.source.parent.getComponent("Start Date").date
date2 = event.source.parent.getComponent("End Date").date
fpmi.nav.swapWindow("WindowA", "WindowB", {"startDate":date1, "endDate":date2})
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.

Modules - db

The fpmi.db module provides functions that allow you to easily execute queries against a datasource, and deal with data.

dateFormat(Date, FormatString)

This function formats the given Date using the FormatString. This function works the same as the [dateFormat expression](#).

Parameters

Date

The date object to format.

FormatString

The format string to use. See the [dateFormat expression](#) for the syntax of this string.

refresh(Component, PropertyName)

Forces the property named PropertyName of the given component to refresh its data. This only works if the property is bound to an SQL query. This is most useful for bound SQL queries that are set to POLLING OFF.

Parameters

Component

A reference to a component.

PropertyName

The name of a property on that component that has an SQL binding.

Example:

```
# This code would go in some component that altered the database,  
# and then wanted to refresh a table to reflect those changes  
  
table = event.source.parent.getComponent("Table (1)")  
fpmi.db.refresh(table, "data")
```

runPrepStmt(Query, [Datasource], Parameters[])

Runs a prepared statement. Prepared statements are very useful for INSERTS and UPDATES that involve user-provided text, because they eliminate the danger that the user's text will break the syntax of the query (for instance, by using a single quote). They are also necessary if you are uploading to BLOB columns.

Parameters

Query

The query to run. Use questionmarks for the placement of your parameters.

[Datasource]

Optional. The name of the datasource to run against. If omitted, the project's default datasource will be used.

Parameters[]

A list of parameters, one for each question mark in the query.

Example 1:

```
# This code would gather some user entered text and insert it into the database.  
  
userText = event.source.parent.getComponent("TextArea").text  
userName = fpmi.security.getUsername()  
fpmi.db.runPrepStmt("INSERT INTO Comments (Name, UserComment) VALUES (?,?)", [userName, userText])
```

Example 2:

```
# This code would read a file and upload it to the database  
  
filename = fpmi.file.openFile() # Ask the user to open a file  
if filename != None:  
    filedatalist = fpmi.file.readFileAsBytes(filename)  
    fpmi.db.runPrepStmt("INSERT INTO Files (file_data) VALUES (?)", [filedata])
```

runPreStmtGetKey(Query, [Datasource], Parameters[])

Identical to `runPreStmt`, except that it returns the autogenerated key that the query created. This is useful for finding the key created by `INSERT` statements into tables with auto incrementing keys. Note that not all databases support this feature. `None` is returned if no key is created.

runQuery(Query, [DataSource])

Parameters

Query

The SQL statement to execute.

DataSource (optional)

The datasource to use in executing the query. If none is specified, the project's default will be used.

Runs the SQL query against the project's default datasource, or against the specified datasource if present. Returns the results as a "PyDataSet", which is a Jython object that can be referenced like a tuple(sequence) of rows. Each row can be treated as a tuple or as a dictionary of columns. For example, suppose you had the table TEST that consisted :

ID	Value
0	3.55
1	67.2
2	9.87

If you executed the following code:

```
table = fpmi.db.runQuery("SELECT * FROM TEST")
then table[2] would access the third row (rows are zero-indexed), and both table[2][0] and table[2]["ID"] would access the ID value of the third row. As further example of how to use the results of runQuery, here are seven different ways to print out the table, and their results follow. Note that some of the later methods exercise some more advanced Jython concepts such as list comprehensions and string formatting, but their intent should be obvious. Generally speaking, the more concise Jython code becomes, the more readable it is.
```

```
table = fpmi.db.runQuery("SELECT * FROM Test")
print "Printing TEST Method 1..."
for row in table:
    for col in row:
        print col,
    print ""
print ""

print "Printing TEST Method 2..."
for row in table:
    print row[0], row[1]
print ""

print "Printing TEST Method 3..."
for row in table:
    print row["ID"], row["VALUE"]
print ""

print "Printing TEST Method 4..."
for rowIdx in range(len(table)):
    print "Row ",str(rowIdx)+": ", table[rowIdx][0], table[rowIdx][1]
print ""

print "Printing TEST Method 5..."
print [str(row[0])+", "+ str(row[1]) for row in table]
print ""

print "Printing TEST Method 6..."
print ["%s, %s" % (row["ID"],row["VALUE"]) for row in table]
print ""

print "Printing TEST Method 7..."
print [[col for col in row] for row in table]
print ""
```

The results printed out would be:

```
Printing TEST Method 1...
0 3.55
1 67.2
2 9.87
```

```
Printing TEST Method 2...
```

```
0 3.55  
1 67.2  
2 9.87
```

```
Printing TEST Method 3...
```

```
0 3.55  
1 67.2  
2 9.87
```

```
Printing TEST Method 4...
```

```
Row 0: 0 3.55  
Row 1: 1 67.2  
Row 2: 2 9.87
```

```
Printing TEST Method 5...
```

```
['0', '3.55', '1', '67.2', '2', '9.87']
```

```
Printing TEST Method 6...
```

```
['0', '3.55', '1', '67.2', '2', '9.87']
```

```
Printing TEST Method 7...
```

```
[[0, 3.55], [1, 67.2], [2, 9.87]]
```

runScalarQuery(Query, [DataSource])

Runs the SQL query query against the project's default datasource, or against the specified datasource if present. Returns the value of the first column of the first row in the results returned. Equivalent to: fpmi.db.runQuery("SELECT ...")[0][0]. Returns None if no results are found.

Parameters

Query

The SQL statement to execute.

DataSource (optional)

The datasource to use in executing the query. If none is specified, the project's default will be used.

Example 1:

```
# This code would count the number of active alarms, and acknowledge them all if there is at least one.
```

```
numAlarms = fpmi.db.runScalarQuery("SELECT COUNT(*) FROM alarmstatus WHERE unacknowledged = 1")  
if numAlarms > 0:  
    # There are alarms - acknowledge all of them  
    fpmi.db.runUpdateQuery("UPDATE alarmstatus SET unacknowledged = 0")
```

Example 2:

```
# This code would read a single value from a table and show it to  
# the user an a popup box
```

```
lakeLevel = fpmi.db.runScalarQuery("SELECT Level FROM LakeInfo WHERE LakeId='Tahoe'")  
fpmi.gui.messageBox("The lake level is: %d feet" % lakeLevel)
```

runUpdateQuery(Query, [DataSource])

Runs the SQL query against the project's default datasource, or against the specified datasource if present. Query should be an **UPDATE**, **INSERT**, or **DELETE** query. Returns the number of rows affected.

Parameters

Query

The SQL statement to execute (normally an UPDATE, INSERT, or DELETE statement).

DataSource (optional)

The datasource to use in executing the query. If none is specified, the project's default will be used.

Example 1:

```
# This code would acknowledge all unacknowledged alarms  
# and show the user how many alarms were acknowledged.
```

```
rowsChanged = fpmi.db.runUpdateQuery("UPDATE alarmstatus SET unacknowledged = 0")  
fpmi.gui.messageBox("Acknowledged %d alarms" % rowsChanged)
```

Example 2:

```
# This code would insert a new recipe step into a recipe table,
```

```
# after asking the user how many gallons of syrup should be added
# on this recipe step.

inputText = fpmi.db.inputBox("How many gallons?", "12.3")
# Make sure the user didn't hit cancel
if inputText != None:
    # Make sure the input is a number
    gallons = float(inputText)
    # Detect the next step number by adding 1 to the last step number
    nextStepNum = fpmi.db.runScalarQuery("SELECT MAX(StepNum) + 1 FROM RecipeSteps")
    # Insert recipe step
    fpmi.db.runUpdateQuery("INSERT INTO RecipeSteps (StepNum, Gallons) VALUES (%d, %f)" % (nextStepNum, gallons))
```

runUpdateQueryGetKey(Query, [DataSource])

Identical to `runUpdateQuery`, except that it returns the autogenerated key that the query created. This is useful for finding the key created by `INSERT` statements into tables with auto incrementing keys. Note that not all databases support this feature. `None` is returned if no key is created.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Modules - dataset

The fpmi.dataset module provides functions that allow for easy manipulation of FactoryPMI DataSet objects.

addRow(DataSet, [row], Sequence)

Takes an existing DataSet object, and inserts a new row into it, returning the resulting dataset. If *row* is omitted, the row will be appended to the end of the dataset, otherwise it will be inserted at the given row index.

Parameters

DataSet

A DataSet into which the new row will be added.

row (optional)

A row index, ranging from 0 to the length of the DataSet. If omitted, the row will be appended to the end.

Sequence

A python sequence (list or tuple) that contains the row to add. Must contain values that match up to the number and types of the columns in the dataset.

Example:

```
# This would add a new option into a Dropdown component.
```

```
dropdown = event.source.parent.getComponent("Dropdown")
newRow = [5, "New Option"]
dropdown.data = fpmi.dataset.addRow(dropdown.data, newRow)
```

```
# This would add a new option into a Dropdown component, but at the beginning
```

```
dropdown = event.source.parent.getComponent("Dropdown")
newRow = [5, "New Option"]
dropdown.data = fpmi.dataset.addRow(dropdown.data, 0, newRow)
```

deleteRow(DataSet, row)

Takes an existing DataSet object, deletes the row at the given index, and returns the resulting dataset.

Parameters

DataSet

A DataSet from which the specified row will be deleted.

row

A row index, ranging from 0 to the length of the DataSet minus one.

Example:

```
# This would remove the selected row from a List component's data
```

```
list = event.source.parent.getComponent("List")
row = list.selectedIndex
if row != -1:
    list.data = fpmi.dataset.deleteRow(list.data, row)
```

exportCSV(Filename, HeaderRow, Data)

This function prompts the user to save the given Data as a CSV (Comma Separated Values) file. A spreadsheet program (like Excel) can then be used to view and print the data. This function returns the path to the file created,

or None if the save as dialog box was canceled by the user.

Parameters

Filename

The suggested name of the file to save to.

HeaderRow

Boolean (0 or 1) value of whether or not an initial row of column names should be included in the CSV file.

Data

The DataSet to export. This is typically the return value of `fpmi.db.runQuery`, or the `data` property of a Table component.

Example 1:

```
# This code, which would probably run in a button,  
# would export a table component's data to a CSV file
```

```
table = event.source.parent.getComponent("Table (1)")  
fpmi.dataset.exportCSV("mydata.csv", 1, table.data)
```

Example 2:

```
# This code would export the results of a query to a CSV file
```

```
results = fpmi.db.runQuery("SELECT * FROM MyTable")  
fpmi.dataset.exportCSV("mydata.csv", 1, results)
```

Example 3:

```
# Tip: you can open the file afterwards by using the fpmi.net.openURL() function
```

```
table = event.source.parent.getComponent("Table (1)")  
filename = fpmi.dataset.exportCSV("mydata.csv", 1, table.data)  
if filename != None:  
    fpmi.net.openURL("file://" + filename)
```

exportExcel(Filename, HeaderRow, Data)

This function prompts the user to save the given `Data` as an excel spreadsheet. This function returns the path to the file created, or None if the save as dialog box was canceled by the user.

Note that the [EasyChart](#) component has a built in function for exporting to Excel.

Parameters

Filename

The suggested name of the file to save to.

HeaderRow

Boolean (0 or 1) value of whether or not an initial row of column names should be included in the Excel spreadsheet.

Data

The DataSet to export. This is typically the return value of `fpmi.db.runQuery`, or the `data` property

of a Table component. Note that if this is instead a list of datasets, each dataset will be put in a different worksheet inside the spreadsheet.

Example 1:

```
# This code, which would probably run in a button,  
# would export a table component's data to an Excel spreadsheet  
  
table = event.source.parent.getComponent("Table")  
fpmi.dataset.exportExcel("mydata.xls", 1, table.data)
```

Example 2:

```
# This code would export the results of a query to an Excel spreadsheet  
  
results = fpmi.db.runQuery("SELECT * FROM MyTable")  
filename = fpmi.dataset.exportExcel("mydata.xls", 1, results)  
  
# Tip: you can open the file afterwards by using the fpmi.net.openURL() function  
  
if filename != None:  
    fpmi.net.openURL("file://" + filename)
```

Example 3:

```
# This code would export the 3 datasets that make up a chart's data to Excel  
# Each dataset will be in its own worksheet (a.k.a. tab)  
  
chart = event.source.parent.getComponent("Chart")  
fpmi.dataset.exportExcel("mydata.xls", 1, [chart.temperatures, chart.pressures, chart.digital])
```

exportHTML(Filename, HeaderRow, Data, Title)

This function prompts the user to save the given `Data` as an HTML (Web Page) file. A web browser program like Internet Explorer can then be used to view the data. This function returns the path to the file created, or `None` if the save as dialog box was canceled by the user.

Note: The table component has a function in it called `exportHTML` as well. The table's version of this function will render the web page to look very much like the actual table on the screen, including colors and number formatting.

Parameters

Filename

The suggested name of the file to save to.

HeaderRow

Boolean (0 or 1) value of whether or not an initial row of column names should be included in the web page.

Data

The `DataSet` to export. This is typically the return value of `fpmi.db.runQuery`, or the `data` property of a Table component.

Title

The title of the web page that is generated.

Example 1:

```
# This code, which would probably run in a button,  
# would export a table component's data to an HTML file  
  
table = event.source.parent.getComponent("Table (1)")
```

```
fpmi.dataset.exportHTML("mydata.html", 1, table.data, "Exported data")
```

Example 2:

```
# This code would export the results of a query to an HTML file

results = fpmi.db.runQuery("SELECT * FROM MyTable")
fpmi.dataset.exportHTML("mydata.html", 1, results, "Exported data")
```

Example 3:

```
# Tip: you can open the file afterwards by using the fpmi.net.openURL() function

table = event.source.parent.getComponent("Table (1)")
filename = fpmi.dataset.exportHTML("mydata.html", 1, table.data, "Exported data")
if filename != None:
    fpmi.net.openURL("file://" + filename)
```

setValue(DataSet, row, column, value)

Takes an existing DataSet object, sets a single value at a row/column location, and returns the resulting dataset.

Parameters

DataSet

A DataSet into which the new value will be set.

row

A row index, ranging from 0 to the length of the DataSet minus one.

column

A column index or name.

value The new value for the location. Example:

```
# This would replace the selected list value with the value 15.
```

```
list = event.source.parent.getComponent("List")
row = list.selectedIndex
if row != -1:
    list.data = fpmi.dataset.setValue(list.data, row, "ColName", 15)
```

toDataSet(PyDataSet)

This function takes a PyDataSet (the kind of object returned by [fpmi.db.runQuery\(\)](#)), and turns it into a DataSet, the kind of object used in FactoryPMI components. Note that there are two versions of this function, see below for the other one.

Parameters

PyDataSet

A PyDataSet object, likely returned by [fpmi.db.runQuery\(\)](#) or [fpmi.dataset.toPyDataSet\(\)](#).

Example:

```
# This code would programmatically set the data of a pie
# chart to the results of a query

data = fpmi.db.runQuery("SELECT Country, Population FROM MyTable")
```

```
piechart = event.source.parent.getComponent("PieChart")
piechart.data = fpmi.dataset.toDataSet(data)
```

toDataSet(Headers, Data)

This function converts a Jython sequence of column headers, and a Jython sequence of Jython sequences (a two-dimensional array) into a DataSet object that can be assigned to things like a Table's data property.

Parameters

Headers

A Jython sequence of strings representing column headers.

Data

A Jython sequence of Jython sequences representing a two-dimensional array of data, any number of rows long, and the same width as the Headers sequence.

Example:

```
# This code would populate a pie chart with some imaginary data.
# In the real world, this data would be pulled from a database. This technique is useful
# for sets of data that cannot easily be retrieved in a single SQL query.
```

```
headers = ["Country", "Population"]
data = []
data.append(["Japan", 126000000])
data.append(["Thailand", 60000000])
data.append(["Italy", 56700000])
data.append(["Spain", 39000000])

piechart = event.source.parent.getComponent("PieChart")
piechart.data = fpmi.dataset.toDataSet(headers, data)
```

toPyDataSet(DataSet)

This function converts a DataSet object, which you would probably get from the data property of a Table component, or one of the DataSets on a graph, into a PyDataSet, which is an object that is more easily used in Jython. See [runQuery\(\)](#) for a detailed example of how to use a PyDataSet object.

Parameters

DataSet

A DataSet object, probably retrieved from a Table or Graph property.

Example:

```
# This code would calculate the average value in the "Temperature" column of a table..
```

```
table = event.source.parent.getComponent("Table")
data = fpmi.dataset.toPyDataSet(table.data)
total = 0.0
for row in data:
    total += row["Temperature"]
average = total / len(data)
fpmi.gui.messageBox("The average temperature is %f" % average)
```

updateRow(DataSet, row, changes)

Takes an existing DataSet object, updates multiple values of a given row, and returns the resulting DataSet.

Parameters

DataSet

A DataSet onto which the changes will be applied.

row

A row index, ranging from 0 to the length of the DataSet minus one.

changes

A python dictionary, whose keys are column indexes or names, and whose values are the new values for that column.

Example:

```
# This would update various column values on row 89 of a report's dataset.
```

```
report = event.source.parent.getComponent("Report")
changes = {"Equipment":"filler", "UserID":23, "Duration":29.2}
report.data = fpmi.dataset.updateRow(report.data, 89, changes)
```

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Modules - security

The fpmi.security module provides some basic functions to interact with the FactoryPMI security model.

getUsername()

Returns the name of the user currently logged in. If you're looking to *display* the currently logged in user, consider simply binding a label's text to the [System]Client/User/Username system tag.

Example:

```
# This code would run on a startup script and do special logic based upon
# who was logging in
name = fpmi.security.getUsername()
if name == 'Bob':
    fpmi.nav.openWindow("BobsHomepage")
else:
    fpmi.nav.openWindow("NormalHomepage")
```

getRoles()

Returns a Jython tuple of the roles that the current user possesses.

Example:

```
# This would run on a button to prevent certain users from opening a window
if "Supervisor" in fpmi.security.getRoles():
    fpmi.nav.openWindow("ManagementOnly")
else:
    fpmi.gui.errorBox("You don't have sufficient privileges to continue")
```

isScreenLocked()

Returns true or false (1 or 0) indicating whether or not the screen is currently locked.

lockScreen([Obscure?])

Locks the application's screen so that no interaction can occur with the running project. The screen is unlocked when the user that locked it enters his/her password. By default, the running project is still partially visible. If you want to totally obscure the screen, set the optional parameter Obscure? to true.

Parameters

Obscure? (optional)

Controls whether or net the running application can be seen while the screen is obscured.

```
# This would run in a timer script to lock the screen after 30 seconds
# of inactivity, and then log the user out after 1 minute of inactivity.
if fpmi.system.getInactivitySeconds() > 15 and not fpmi.security.isScreenLocked():
    fpmi.security.lockScreen()
elif fpmi.system.getInactivitySeconds() > 30:
    fpmi.security.logout()
```

logout()

Closes down the application and presents the login screen.

switchUser(*Username*, *Password*, [*eventObject*])

Attempts to switch the current user on the fly. If the given username and password fail, this function will return false. If it succeeds, then all currently opened windows are closed, the user is switched, and windows are then re-opened in the states that they were in. If an event object is passed to this function, the parent window of the event object will *not* be re-opened after a successful user switch. To download a pre-made user switching dialog window, visit the [FactoryPMI goodies page](#).

unlockScreen()

Programmatically unlocks the screen if it is currently locked. Has no effect if the screen is not currently locked.

Copyright © 2001-2005 Inductive Automation, Inc. All Rights Reserved.



Modules - system

The fpmi.system module provide functions to interact with the system.

beep()

Tells the computer to make its default 'beep' sound..

execute(commands [])

Executes the given commands via the operating system. The commands argument is an array of strings. The first string is the program to execute, with subsequent strings being the arguments to that command.

Example:

```
# This code would work on a Windows system to play a sound file.
```

```
fpmi.system.execute(["sndrec32", "/play", "/close", "/embedding", "C:\\somethingwrong.wav"])
```

exit([force])

Exits the application, as long as the **Shutdown Interception** script doesn't cancel the shutdown event. Set force to **true** to not give the **Shutdown Interception** script a chance to cancel the exit.

Example:

```
# This code would exit the FactoryPMI Runtime client after confirming with the user.
```

```
if fpmi.gui.confirm("Are you sure you want to exit?"):
    fpmi.system.exit()
```

getClientId()

Returns a hex-string that represents a number unique to the running client's session. You are guaranteed that this number is unique between all running clients.

Example:

```
# This code would print the current client's id to the debug console.
```

```
id = fpmi.system.getClientId()
print id
```

getConnectTimeout()

Returs the connect timeout in milliseconds for all client to gateway communication. The defaultis 10,000 (10 seconds).

Example:

```
# This code would print out the current connect timeout
```

```
print fpmi.system.getConnectTimeout();
```

getGatewayAddress()

Returns the address (IP address and port) of the gateway that the client is connected to.

Example:

```
# This code would open up the FactoryPMI gateway configuration page.
```

```
address = fpmi.system.getGatewayAddress()
fpmi.net.openURL("http://%s/setup" % address)
```

getInactivitySeconds()

Returns the number of seconds the client has been inactive (no mouse or keyboard input). Note: only works in the Client - will always return zero in the Designer.

Example:

```
# This code could run in a global timer script.  
# After a 5-minute timeout, navigate back to the home screen  
if fpmi.system.getInactivitySeconds() > 300 and app.nav.getCurrentWindow() != "HomeScreen":  
    app.nav.swapTo("HomeScreen")
```

getProjectName()

Returns the name of the project that is currently being run.

Example:

```
# This code would display the name of the currently running project  
fpmi.gui.messageBox("You are running project: %s" % fpmi.system.getProjectName())
```

getProperty(name)

Retrieves the value of a named system property. Available properties are:

- **file.separator**. The system file separator character. (for example, "/" (unix) or "\" (windows))
- **line.separator**. The system line separator string. (for example, "\r\n" (carriage return, newline))
- **os.arch**. Operating system architecture. (for example, "x86")
- **os.name**. Operating system name. (for example, "Windows XP")
- **os.version**. Operating system version. (for example, "5.1")
- **user.home**. User's home directory.
- **user.name**. User's account name.

getSessionInfo([usernameFilter], [projectFilter])

Returns a PyDataSet holding information about all of the sessions (logged in users) on the FactoryPMI Gateway. Optional regular-expression based filters can be provided to filter the username or the username and the project returned. The PyDataSet returned has these columns:

- **username** (String)
- **project** (String)
- **address** (String)
- **isDesigner** (Boolean)
- **clientId** (String)

Note that this function will not return all sessions across a cluster - only the FactoryPMI cluster node that is being communicated with by the client who makes the call.

Example 1:

```
# This code would get the entire table of sessions and put it in an adjacent table  
table = event.source.parent.getComponent("Table")  
sessions = fpmi.system.getSessionInfo()  
table.data = fpmi.db.toDataSet(sessions)
```

Example 2:

```
# This code would count the number of times a user named "billy" is logged in  
sessions = fpmi.system.getSessionInfo("billy")  
fpmi.gui.messageBox("Billy has %d sessions" % len(sessions))
```

getSystemFlags()

Returns an integer that represents a bit field containing information about the currently running FactoryPMI system. Each bit corresponds to a public bitmask as defined below. See the examples for tips on how to extract the information in this bit field are in the examples. Note that the tag [System]Client/System/SystemFlags contains the same value.

- **fpmi.system.DESIGNER_FLAG**. Set if running in the Designer. (1)
- **fpmi.system.PREVIEW_FLAG**. Set if running in the Designer, and the Designer is in preview mode. (2)
- **fpmi.system.CLIENT_FLAG**. Set if running as a Client. (4)
- **fpmi.system.WEBSTART_FLAG**. Set if running as a Client in Web Start mode. (8)
- **fpmi.system.APPLET_FLAG**. Set if running as a Client in Applet mode. (16)
- **fpmi.system.FULLSCREEN_FLAG**. Set if running as a Client in full-screen mode. (32)
- **fpmi.system.SSL_FLAG**. Set if communication to the Gateway is encrypted with SSL. (64)

Example 1:

```
# This script would run the code inside the if block only if
# running as a client.
isClient = fpmi.system.getSystemFlags() & fpmi.system.CLIENT_FLAG
if isClient:
    # Do something only the client should do
    ...
```

Example 2:

```
# This code would only open a certain screen if SSL mode was enabled
ssl = fpmi.system.getSystemFlags() & fpmi.system.SSL_FLAG
if ssl:
    fpmi.nav.openWindow("SensitiveDataEntry")
else:
    fpmi.gui.warningBox("This screen requires SSL encryption")
```

invokeAsynchronous(function)

This is an advanced scripting function. Invokes (calls) the given Jython function on a different thread. This means that calls to `invokeAsynchronous` will return immediately, and then the given function will start executing asynchronously on a different thread. This is useful for long-running data intensive functions, where running them synchronously (in the GUI thread) would make the GUI non-responsive for an unacceptable amount of time.

CRITICAL WARNING: Under no circumstances should you ever do anything in the function that is invoked asynchronously that interacts with the GUI. This means things like window navigation, setting and getting component properties, showing error/message popups, etc. If you need to do something with the GUI in this function, this must be achieved through a call to `fpmi.system.invokeLater`.

Example:

```
# This code would do some data-intensive processing, and then call
# back to the GUI to let it know that it is finished.
# We use default function parameters to pass the root container into these
# functions. (See a Python reference if you don't understand this)

def longProcess(rootContainer = event.source.parent):
    import fpmi
    # Do something here with the database that takes a long time
    results = ... ( something )
    # Now we'll send our results back to the UI
```

```
def sendBack(results = results, rootContainer = rootContainer):
    rootContainer.resultsProperty = results
    fpmi.system.invokeLater(sendBack)

fpmi.system.invokeAsynchronous(longProcess)
```

invokeLater(*function*, [*time*])

This is an advanced scripting function. Invokes (calls) the given Jython function object after all of the currently processing and pending events are done being processed. This is useful for events like propertyChangeEvents, where the Jython script is called before any bindings are evaluated.

If you specify an optional time argument (number of milliseconds), the function will be invoked after all currently processing and pending events are processed plus the duration of that time.

Example:

```
# This code is in the historical graph goodie found on Inductive Automation's website.

# The code in the update/refresh button uses the 'date' property on the two calendar components,
# which are bound to the current_timestamp property on their parent. We want to simulate a button
# press when the window opens, but only after the date properties' bindings have been evaluated.

if event.propertyName == 'current_timestamp':
    # Define a function to click the button
    def clickButton(button = event.source.parent.getComponent('Refresh')):
        button.doClick()

    # Tell the system to invoke the function after
    # the current event has been processed
    fpmi.system.invokeLater(clickButton)
```

retarget(*projectName*, [*serverAddress*], [*parameters*], [*windows*])

One of the more powerful features of FactoryPMI, this function allows you to programatically 'retarget' the FactoryPMI Runtime client. You can have it switch to another project on the same Gateway, or another gateway entirely, even across a WAN. This feature makes the vision of a seamless, enterprise-wide SCADA application a reality.

The retarget feature will attempt to transfer the current user credentials over to the new project / Gateway. If the credentials fail on that project, the user will be prompted for a valid username and password, with an option to cancel the retargeting and return to the original project. One valid authentication has been achieved, the currently running project is shut down, and the new project is loaded.

Parameters

projectName

Required. The name of the project to retarget to.

serverAddress

Optional. The address (ip address and port) of the Gateway that the project resides on. Omit, use None or "" (empty string) to specify that the project resides on the same Gateway as the currently running project.

parameters

Optional. A Jython Dictionary listing parameter names and values that will be set in the new project

as global Jython variables. Two parameters will automatically be set:

- `_RETARGET_FROM_PROJECT` The name of the project that was running before the retargeting occurred.
- `_RETARGET_FROM_GATEWAY` The name of the Gateway in use before the retargeting occurred.

windows

Optional. A Jython sequence of window names. If this is included and has at least one window specified, this list of windows will be opened when the new project is started up, *instead of* that project's startup windows.

Example #1:

```
# This code would switch to a project named 'TankControl' on the same Gateway  
# as the currently running project  
fpmi.system.retarget("TankControl")
```

Example #2:

```
# This code would switch to a project named 'TankControl' on a  
# Gateway located at a different IP address running on port 8080  
fpmi.system.retarget("TankControl", "10.30.2.33:8080")
```

Example #3:

```
# This code would switch to a project named 'TankControl' on a  
# Gateway located at a different IP address running on port 8080, and  
# would open the window named "Graph", and set a global jython variable in the new  
# project named "retargetOccured" to the value 1 (one).  
fpmi.system.retarget("TankControl", "10.30.2.33:8080", {"retargetOccured":1}, ["Graph"])
```

Back Button Example:

```
# This code would be put in a button in the target that was retargetted to,  
and act as a 'back' button, that would re-target back to the original project.
```

```
global _RETARGET_FROM_PROJECT  
global _RETARGET_FROM_GATEWAY  
  
fpmi.system.retarget(_RETARGET_FROM_PROJECT, _RETARGET_FROM_GATEWAY)
```

setConnectTimeout(timeout)

Sets the connect timeout (in milliseconds) for all communication with the Gateway. It can be useful to increase the connect timeout (from the default of 10 seconds) when a client is connecting over a WAN with a high latency to avoid unnecessary disconnect messages.

Example:

```
# This code would set the current connect timeout to 30 seconds  
fpmi.system.setConnectTimeout(30000);
```



Modules - tag

The `fpmi.tag` module provides functions interacting with SQLTags.

`getTagValue(TagPath)`

Returns the value of the tag at the given path.

Parameters

`TagPath`

The path of the tag to write to.

Example 1:

```
val = fpmi.tag.getTagValue("[]EastSection/ValveG/HOA_bit")
fpmi.gui.messageBox("The value is %d" % val)
```

`writeToTag(TagPath, Value, [SuppressErrors])`

Writes the given value to the tag at the given path. If `SuppressErrors` is true, error messages on failed writes will not be shown.

Parameters

`TagPath`

The path of the tag to write to.

`Value`

The value to write to the tag.

`SuppressErrors (optional)`

If true, error messages for failed writes will not be shown. Default is 0 (false).

Example 1:

```
fpmi.tag.writeToTag("Tanks/tankHiSP", 25)
```

`writeToTagSynchronous(TagPath, Value)`

Writes the given value to the tag and property specified by the tag path. Does not return until the write succeeds. Throws an error if the write failed. This function cannot be called from the event dispatch thread, which means that it cannot be called directly from a GUI event like a button press, without wrapping it in a `fpmi.system.invokeAsynchronous`. See the examples.

Parameters

`TagPath`

The path of the tag to write to.

`Value`

The value to write to the tag.

Example 1:

```
# These writes will happen in order, each one waiting for the one before it to complete.
# This script would be called from a background thread like a global timer event.
fpmi.tag.writeToTagSynchronous("Tanks/tankHiSP", 25)
fpmi.tag.writeToTagSynchronous("Tanks/tankLoSP", 3)
fpmi.tag.writeToTagSynchronous("Tanks/tankTempSP", 280)
```