

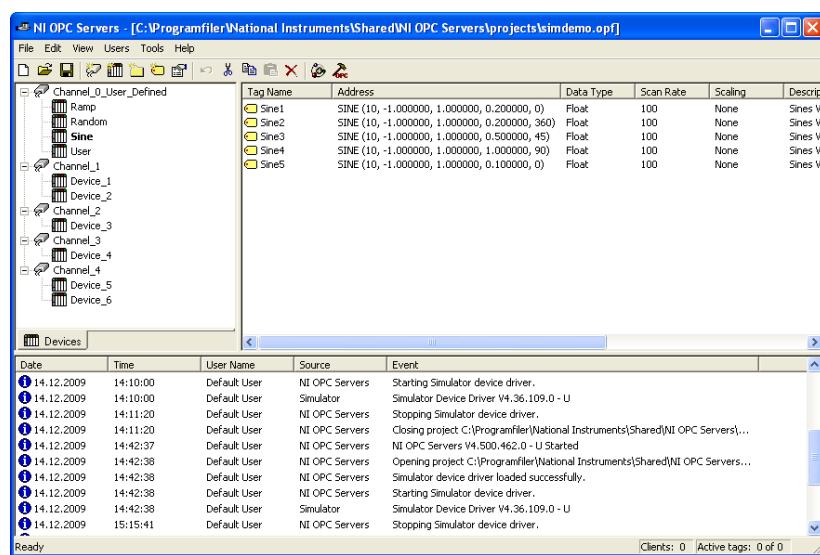
Telemark University College

Department of Electrical Engineering, Information Technology and Cybernetics

Tutorial

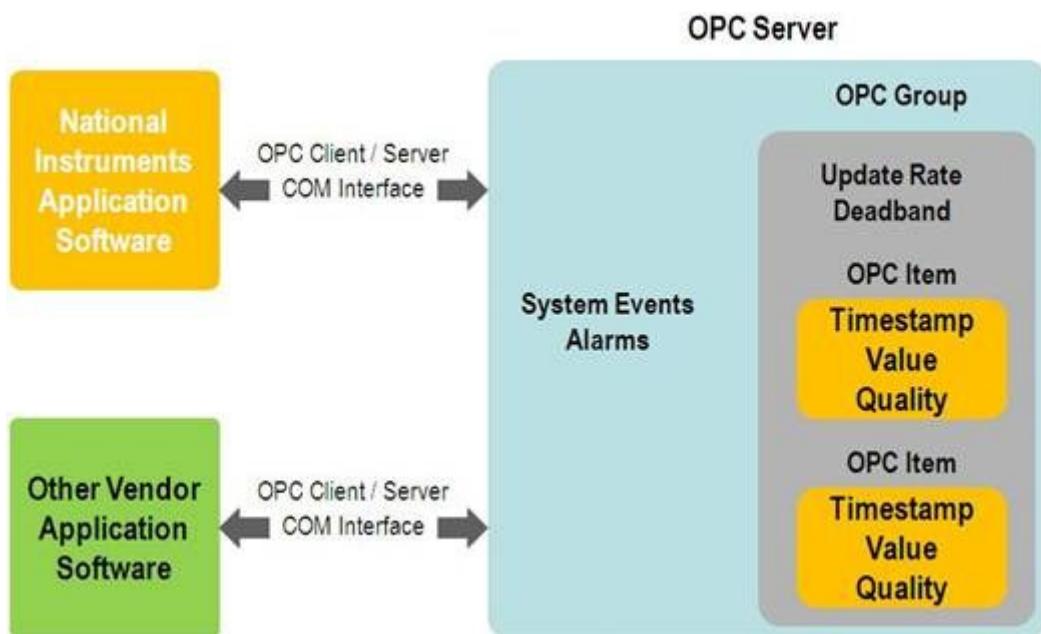
OPC and Real-Time Systems in LabVIEW

HANS-PETTER HALVORSEN, 2012.01.11



Preface

OPC (OLE for process control) is a standard interface between numerous data sources, – such as programmable logic controllers (PLCs), remote terminal units (RTUs), and sensors on a factory floor – to HMI/SCADA applications, application tools, and databases. With OPC, your device-side server and application software can communicate without your duplicating device driver development and providing support for hardware feature changes. The OPC Foundation defines the standards that allow any client to access any OPC-compatible device.



Installation Requirements

The following software is used in this Tutorial (you don't necessarily need to install all, it depends if you are interested in everything or just parts of this Tutorial):

- LabVIEW
- LabVIEW Real-Time Module
- FieldPoint
- CompactRIO
- NI-DAQmx
- NI Measurement & Automation Explorer

- LabVIEW Datalogging and Supervisory Control Module

In addition the following software will be needed in this Tutorial:

- **NI OP Servers** (Evaluation version available from www.ni.com)
- **MatrikonOPC Simulation Server** (free download from <http://matrikon.com/>)

Table of Contents

Preface.....	ii
Installation Requirements.....	ii
Table of Contents	iv
1 Introduction to LabVIEW	1
1.1 Dataflow programming	1
1.2 Graphical programming.....	2
1.3 Benefits.....	2
1.4 LabVIEW MathScript RT Module	3
2 Introduction to OPC.....	4
2.1 OPC Server.....	5
2.1.1 MatrikonOPC Server for Simulation	7
2.1.2 NI OPC Servers.....	7
2.2 OPC Client.....	7
2.3 Tags.....	8
2.4 Why Do We Need OPC Standards?	8
3 Introduction to Real-Time Systems	9
3.1 LabVIEW Real-Time Module	9
4 Introduction to Embedded Systems	10
4.1 Some Examples of Embedded systems	10
4.2 Embedded hardware and Real-Time systems from National Instruments	11
5 Real-Time VIs in LabVIEW	12
5.1 Synchronization	12
5.2 Semaphores.....	13
Queue Operations.....	13

5.3	Timed Loop	15
6	Multithreading in LabVIEW.....	16
6.1	Multitasking.....	16
6.2	Multithreading.....	16
7	DataSocket.....	19
7.1	Architecture.....	20
7.2	DataSocket Server	21
7.3	DataSocket in LabVIEW	21
8	OPC Servers from National Instruments.....	23
8.1	NI OPC Servers.....	23
8.1.1	OPC Quick Client.....	24
8.2	Shared Variable Engine.....	27
9	MatrikonOPC Simulation Server	28
9.1	MatrikonOPC Server.....	28
9.2	Aliases.....	29
9.3	Matrikon OPC Explorer (Client)	30
10	Using OPC from LabVIEW	32
10.1	OPC URL.....	32
10.2	Read OPC Data.....	33
10.3	Write OPC Data.....	35
11	LabVIEW Datalogging and Supervisory Control.....	36
11.1	Functions and VIs Palettes	37
11.2	DSC Module Controls Palettes.....	38
11.3	Citadel Database.....	38
11.4	Historical Data Viewer	39
11.5	Distributed System Manager.....	40

11.6 SQL Server	40
Exercises.....	41
11.6.1 Task 1: Open and Run the Example.....	42
11.6.2 Task 2: Creating a New Project Library.....	42
11.6.3 Task 3: Creating a Periodic I/O Server.....	43
11.6.4 Task 4: Deploying the Periodic I/O Server	48
11.6.5 Task 5: Creating Shared Variables	50
11.6.6 Task 6: Configuring Data Logging	52
11.6.7 Task 7: Configuring Alarming.....	53
11.6.8 Task 8: Enabling Logging.....	54
11.6.9 Task 9: Create the LabVIEW application.....	55
11.6.10 Task 10: Use the LabVIEW DSC Functions and VIs	58
11.6.11 Task 11: Viewing Real-Time Data	60
11.6.12 Task 12: Viewing Alarms using the Distributed System Manager	62
12 LabVIEW I/O Server	63
12.1 Connect LabVIEW to OPC Tags by Creating an I/O Server	63
12.2 Create Shared Variables that Connect to the OPC Tags through the I/O Server	64
12.3 Viewing Shared Variables with Distributed System Manager.....	66
12.4 Using OPC Tag Data in LabVIEW	66
13 LabVIEW Real-Time Module	71
13.1 Real-Time Development in LabVIEW.....	72
14 Compact FieldPoint.....	74
14.1 Introduction.....	74
14.2 Development	75
15 CompactRIO	83
15.1 Introduction.....	83

15.2 Development.....	83
-----------------------	----

1 Introduction to LabVIEW

LabVIEW (short for **L**aboratory **V**irtual **I**nstrumentation **E**ngineering **W**orkbench) is a platform and development environment for a visual programming language from National Instruments. The graphical language is named "G". Originally released for the Apple Macintosh in 1986, LabVIEW is commonly used for data acquisition, instrument control, and industrial automation on a variety of platforms including Microsoft Windows, various flavors of UNIX, Linux, and Mac OS X. The latest version of LabVIEW is version LabVIEW 2009, released in August 2009. Visit National Instruments at www.ni.com.

The code files have the extension ".vi", which is an abbreviation for "Virtual Instrument". LabVIEW offers lots of additional Add-Ons and Toolkits.

This paper is part of a series with LabVIEW papers:

- Introduction to LabVIEW
- Data Acquisition in LabVIEW
- Control and Simulation in LabVIEW
- Linear Algebra in LabVIEW
- Database Communication in LabVIEW
- Datalogging and Supervisory Control in LabVIEW
- Intermediate Topics in LabVIEW
- Advanced Topics in LabVIEW
- Etc.

Each paper may be used independently of each other.

1.1 Dataflow programming

The programming language used in LabVIEW, also referred to as G, is a dataflow programming language. Execution is determined by the structure of a graphical block diagram (the LV-source code) on which the programmer connects different function-nodes by drawing wires. These wires propagate variables and any node can execute as soon as all its input data become available. Since this might be the case for multiple nodes simultaneously, G is inherently capable of parallel execution. Multi-processing and multi-threading hardware is automatically exploited by the built-in scheduler, which multiplexes multiple OS threads over the nodes ready for execution.

1.2 Graphical programming

LabVIEW ties the creation of user interfaces (called front panels) into the development cycle. LabVIEW programs/subroutines are called virtual instruments (VIs). Each VI has three components: a block diagram, a front panel, and a connector panel. The last is used to represent the VI in the block diagrams of other, calling VIs. Controls and indicators on the front panel allow an operator to input data into or extract data from a running virtual instrument. However, the front panel can also serve as a programmatic interface. Thus a virtual instrument can either be run as a program, with the front panel serving as a user interface, or, when dropped as a node onto the block diagram, the front panel defines the inputs and outputs for the given node through the connector pane. This implies each VI can be easily tested before being embedded as a subroutine into a larger program.

The graphical approach also allows non-programmers to build programs simply by dragging and dropping virtual representations of lab equipment with which they are already familiar. The LabVIEW programming environment, with the included examples and the documentation, makes it simple to create small applications. This is a benefit on one side, but there is also a certain danger of underestimating the expertise needed for good quality "G" programming. For complex algorithms or large-scale code, it is important that the programmer possess an extensive knowledge of the special LabVIEW syntax and the topology of its memory management. The most advanced LabVIEW development systems offer the possibility of building stand-alone applications. Furthermore, it is possible to create distributed applications, which communicate by a client/server scheme, and are therefore easier to implement due to the inherently parallel nature of G-code.

1.3 Benefits

One benefit of LabVIEW over other development environments is the extensive support for accessing instrumentation hardware. Drivers and abstraction layers for many different types of instruments and buses are included or are available for inclusion. These present themselves as graphical nodes. The abstraction layers offer standard software interfaces to communicate with hardware devices. The provided driver interfaces save program development time. The sales pitch of National Instruments is, therefore, that even people with limited coding experience can write programs and deploy test solutions in a reduced time frame when compared to more conventional or competing systems. A new hardware driver topology (DAQmxBase), which consists mainly of G-coded components with only a few register calls through NI Measurement Hardware DDK (Driver Development Kit) functions, provides platform independent hardware access to numerous data acquisition and instrumentation devices. The DAQmxBase driver is available for LabVIEW on Windows, Mac OS X and Linux platforms.

For more information about LabVIEW, visit my Blog: <http://home.hit.no/~hansha/>

1.4 LabVIEW MathScript RT Module

The LabVIEW MathScript RT Module is an add-on module to LabVIEW. With LabVIEW MathScript RT Module you can:

- Deploy your custom .m files to NI real-time hardware
- Reuse many of your scripts created with The MathWorks, Inc. MATLAB® software and others
- Develop your .m files with an interactive command-line interface
- Embed your scripts into your LabVIEW applications using the MathScript Node

2 Introduction to OPC

[Source: Wikipedia, National Instruments]

OLE for Process Control (OPC), which stands for Object Linking and Embedding (OLE) for Process Control, is the original name for a standard specification developed in 1996. The standard specifies the communication of real-time plant data between control devices from different manufacturers. After the initial release, the **OPC Foundation** was created to maintain the standard. OPC Foundation: <http://www.opcfoundation.org/>.

In 1994 a group of vendors representing a broad spectrum of disciplines in industrial segment formed what is now known as the OPC Foundation. The OPC Foundation put forth the goal of developing a single client/server specification that would allow any vendor to develop software and applications that could share data in a fast, robust fashion, and do it in a way that would eliminate the proprietary schemes that forced these same vendors to duplicate development efforts. The OPC Foundation developed the first specification called Data Access Specification 1.0a that was released in early 1996. Using this specification, vendors were able to quickly develop client server software.

A major goal of the OPC Foundation and the Data Access specification was to eliminate the need of client application vendor's to develop their own proprietary set of communications drivers.

While OPC originally stood for “OLE for Process Control”, the official stance of the OPC Foundation is that OPC is no longer an acronym and the technology is simply known as “OPC”. One of the reasons behind this is while OPC is heavily used within the process industries.

As of June, 2006, "OPC is a series of standards specifications". OPC consists of seven current standards and two emerging standards.

The OPC Specification was based on the OLE, COM, and DCOM technologies developed by Microsoft for the Microsoft Windows operating system family. The specification defined a standard set of objects, interfaces and methods for use in process control and manufacturing automation applications to facilitate interoperability. The most common OPC specification is OPC Data Access, which is used to read and write real-time data. When vendors refer to OPC generically, they typically mean OPC Data Access.

The OPC specifications:

- **OPC Data Access (DA)**
- **OPC Alarms and Events**
- OPC Batch
- OPC Data eXchange
- **OPC Historical Data Access**
- OPC Security
- OPC XML-DA

- OPC Unified Architecture (UA)

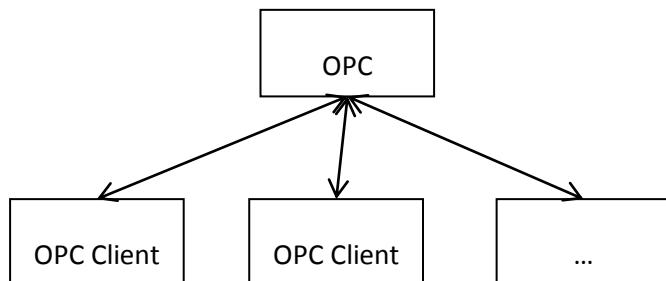
Note! LabVIEW supports only the OPC Data Access specification.

OPC servers provide a method for many different software packages to access data from a process control device, such as a PLC (Programmable Logic Controller) or DCS (Distributed Control System). Traditionally, any time a package needed access to data from a device, a custom interface, or driver, had to be written. The purpose of OPC is to define a common interface that is written once and then reused by any business, SCADA, HMI, or custom software packages.

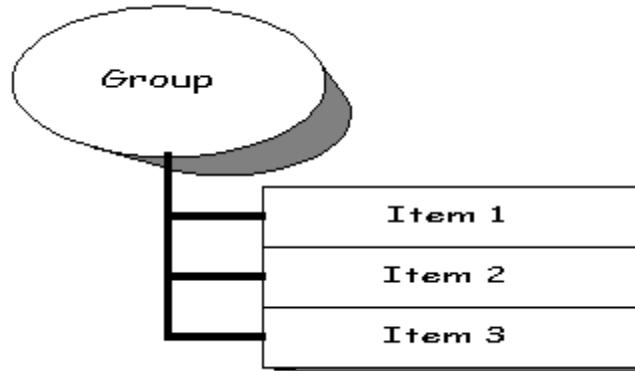
Once an **OPC server** is written for a particular device, it can be reused by any application that is able to act as an **OPC client**. OPC servers use Microsoft's OLE technology (also known as the Component Object Model, or COM) to communicate with clients. COM technology permits a standard for real-time information exchange between software applications and process hardware to be defined.

2.1 OPC Server

The basic concept in OPC is that we have an OPC Server and one or more OPC Clients that communicate with the server in order to write or read data. An OPC server has implemented a set of services, and the clients are using these services.



At a high level, an OPC server is comprised of several objects: the server, the group, and the item. The OPC server object maintains information about the server and serves as a container for OPC group objects. The OPC group object maintains information about itself and provides the mechanism for containing and logically organizing OPC items.

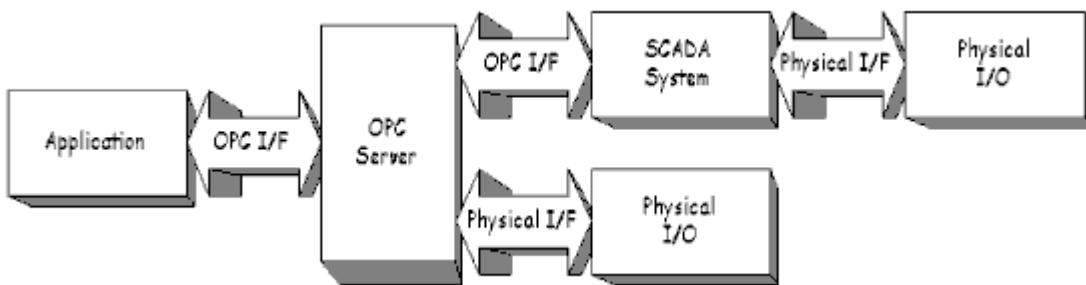


The **OPC Groups** provide a way for clients to organize data. For example, the group might represent items in a particular operator display or report. Data can be read and written.

The **OPC Items** represent connections to data sources within the server. An OPC Item, from the custom interface perspective, is not accessible as an object by an OPC Client. Therefore, there is no external interface defined for an OPC Item. All access to OPC Items is via an OPC Group object that contains the OPC item, or simply where the OPC Item is defined. Associated with each item are a Value, Quality and Time Stamp.

Note! The items are not the data sources, they are just connections to them. For example, the tags in a DCS system exist regardless of whether an OPC client is currently accessing them. The OPC Item should be thought of as simply specifying the address of the data, not as the actual physical source of the data that the address references.

Although OPC is primarily designed for accessing data from a networked server, OPC interfaces can be used in many places within an application. At the lowest level they can get raw data from the physical devices into a SCADA or DCS, or from the SCADA or DCS system into the application. The architecture and design makes it possible to construct an OPC Server which allows a client application to access data from many OPC Servers provided by many different OPC vendors running on different nodes via a single object.



In this Tutorial we will use some OPC Server for test purposes:

Tutorial: OPC and Real-Time Systems in LabVIEW

2.1.1 MatrikonOPC Server for Simulation

MatrikonOPC Simulation is free for non-production use and can be distributed openly. It is a fully functioning application without restriction.

MatrikonOPC Simulation Server is a free utility used to help test and troubleshoot OPC applications (clients) and connections. Testing applications on “live” OPC servers may result in loss of actual production data. The MatrikonOPC Simulation Server creates a simulated environment so that in the event of a problem, no real process data is lost.

MatrikonOPC Explorer is an OPC Client application for testing and configuring OPC connections.

<http://www.matrikonopc.com/downloads/178/index.aspx>

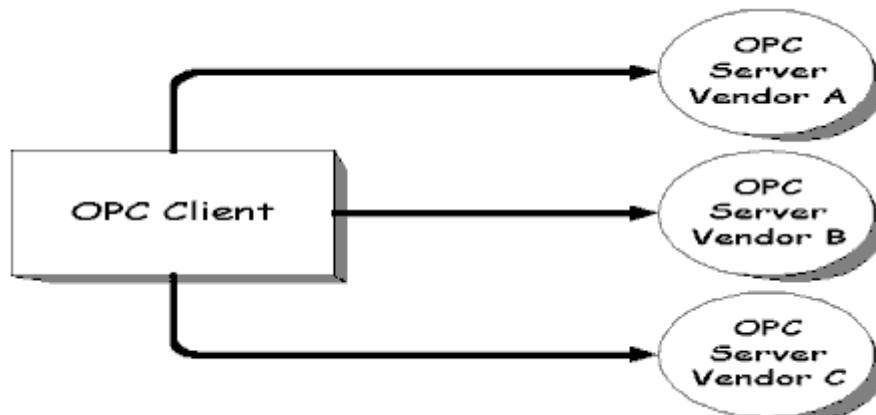
2.1.2 NI OPC Servers

The National Instruments OPC Servers provides a single consistent interface to communicate with multiple devices, saving you from learning new communication protocols or spending time understanding new applications. The combination of NI OPC Servers and LabVIEW provides a single platform for delivering high performance measurements and control to both new and existing industrial systems. NI OPC servers connect through the OPC client in LabVIEW Datalogging and Supervisory Control (DSC) Module to enable you develop a fully fledged HMI/SCADA system with PLCs, PACs and smart sensors.

An evaluation version is available here: <http://sine.ni.com/nips/cds/view/p/lang/en/nid/4584>

2.2 OPC Client

An OPC Client can connect to OPC Servers provided by one or more vendors.



2.3 Tags

Tags are used a lot in the process industry and are normally assigned to a piece of information. A tag consists of a name describing a single point of information so a process system can consist of hundreds and even thousands of tags. The OPC server has one tag for each measurement points and controller points in the plant and it is the responsibility of the OPC server to get the information from the controllers. This is one of the reasons for the complexity of the servers, they need to have drivers for a lot of controllers and measurement systems.

2.4 Why Do We Need OPC Standards?

With hundreds of major hardware and software vendors, you have the difficult task of making sure that your client application can communicate with any device and driver. Before OPC, you had to write separate client application code to communicate with each device because each device driver used a different API. OPC provides an industry-standard interface - a single API - so client applications can retrieve process variables, such as temperature, pressure, flow rate, or position, and set control variables, such as the current or discrete output on an I/O module.

OPC offers the following benefits:

- Single API for all OPC servers so that you can reuse the code for your client application with each device.
- Opportunity to develop client applications in development environments that take advantage of COM and ActiveX, such as Microsoft Visual Basic, Visual C++, Excel, and Internet Explorer.
- Browser through which you can select OPC items available to clients. An OPC item is a channel or variable in a real-world device (normally an I/O point) that a device server monitors or controls.
- Distributed and remote access through DCOM. You can access devices connected to other computers on the network.

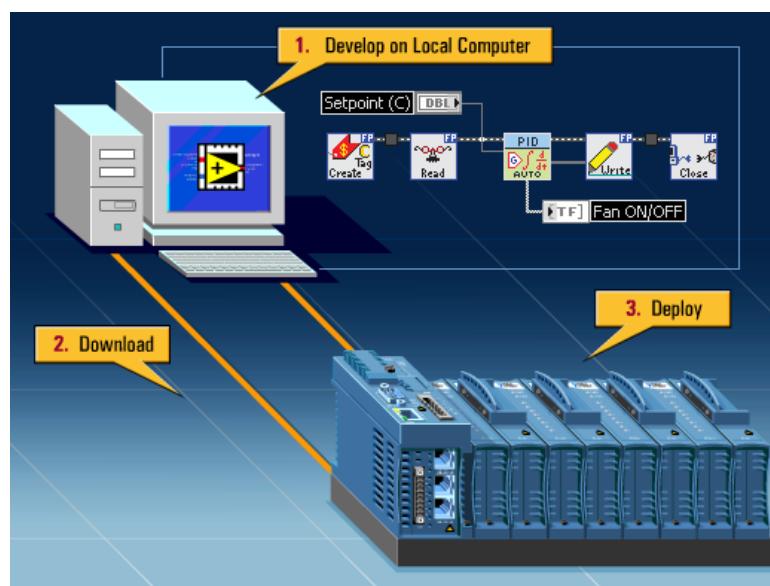
3 Introduction to Real-Time Systems

A real-time system means a computer based system where one or more of the applications must be able to synchronize with a physical process. Real-time means that the computer system is monitoring the states of the physical process and must respond to changes of one or more of these states within a maximum time. A real-time system can then be used for monitoring of different parameters in the physical process for presentation, warnings, alarm situations and for control.

A real-time operating system (**RTOS**) is a multitasking operating system intended for real-time applications. Such applications include embedded systems (programmable thermostats, household appliance controllers), industrial robots, spacecraft, and industrial control.

3.1 LabVIEW Real-Time Module

The National Instruments LabVIEW Real-Time Module is an add-on component for the LabVIEW Development System. When installed, this software compiles NI LabVIEW graphical code and optimizes it for the selected real-time target. Using the LabVIEW Real-Time Module, you can develop and deploy applications to all NI real-time hardware targets including PXI, Compact FieldPoint, FieldPoint, CompactRIO, and standard desktop PCs. The embedded RTOS for these targets is a single dedicated kernel that provides maximum reliability for embedded code.



4 Introduction to Embedded Systems

An embedded system is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer, is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.

Embedded systems are controlled by one or more main processing cores that is typically either a microcontroller or a digital signal processor (DSP). The key characteristic is however being dedicated to handle a particular task, which may require very powerful processors. For example, air traffic control systems may usefully be viewed as embedded, even though they involve mainframe computers and dedicated regional and national networks between airports and radar sites.

Since the embedded system is dedicated to specific tasks, design engineers can optimize it reducing the size and cost of the product and increasing the reliability and performance.

4.1 Some Examples of Embedded systems

Fuji PXG5 PID Controller:



Compact FieldPoint (from National Instruments):



4.2 Embedded hardware and Real-Time systems from National Instruments

Embedded applications have unique requirements such as deterministic behavior. With LabVIEW Real-Time, real time controllers, and data acquisition hardware from NI you can create applications with deterministic, real-time performance.

You can develop and debug your application using LabVIEW graphical programming, and download time-critical application components to run embedded on Real-Time hardware.

5 Real-Time VIs in LabVIEW

A real-time system means a computer based system where one or more of the applications must be able to “synchronize” with a physical process. “Real-time” means that the computer system is monitoring the states of the physical process and must respond to changes of one or more of these states within a maximum time. A real-time system can then be used for monitoring of different parameters in the physical process for presentation, warnings, alarm situations and for control.

A real-time system will very often be an embedded system, while an embedded system does not need to be a real-time system.

Important features in such real-time systems are **synchronization**, **semaphores**, **deadlock**, **multithreading** and **scheduling**. In this chapter we will see how we can use these features in LabVIEW when programming our real-time systems.

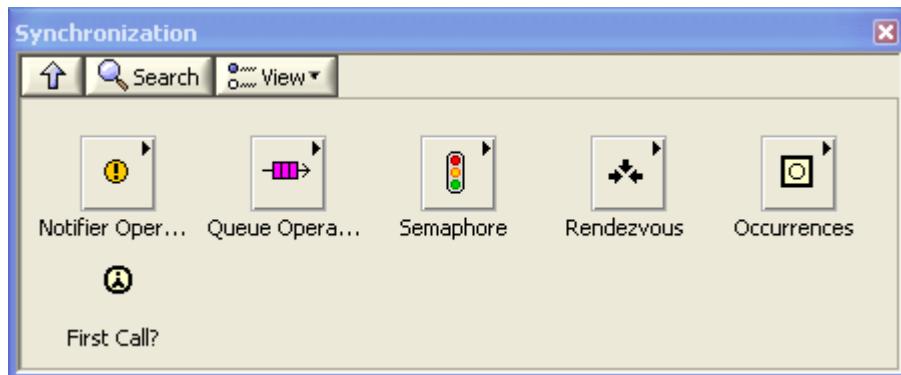
5.1 Synchronization

The applications of the real-time system must run together with the physical process, so the real-time system must be able to manage simultaneity. The solution is often to run several applications on the computer system or on different computers in a distributed system. These solutions require some sort of synchronization between the applications and between the applications and the physical process.

When several applications are running “simultaneous” on a computer system, there must also be some synchronization of the usage of the resources in the computer system. Resources can be both hardware and software, like I/O global variables in the software, the CPU, memory, disk etc.

LabVIEW offers different mechanisms for synchronization we will use in this task.

Below we see the Synchronization palette in LabVIEW:



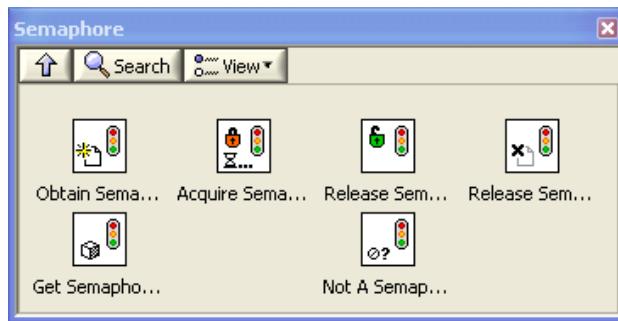
We can, e.g., use the Synchronization VIs and functions to synchronize tasks executing in parallel and to pass data between parallel tasks.

5.2 Semaphores

A semaphore is the simplest form of synchronization and has two basic functions. These functions are:

- **Request;** the system will move the task to wait queue if the semaphore is already “occupied” by another task.
- **Release;** releases a semaphore, the system will move the “blocked” tasks from the wait queue to the ready queue.

Below we see the Semaphore palette in LabVIEW:



Use the Semaphore VIs to limit the number of tasks that can simultaneously operate on a shared (protected) resource. A protected resource or critical section of code might include writing to global variables or communicating with external instruments.

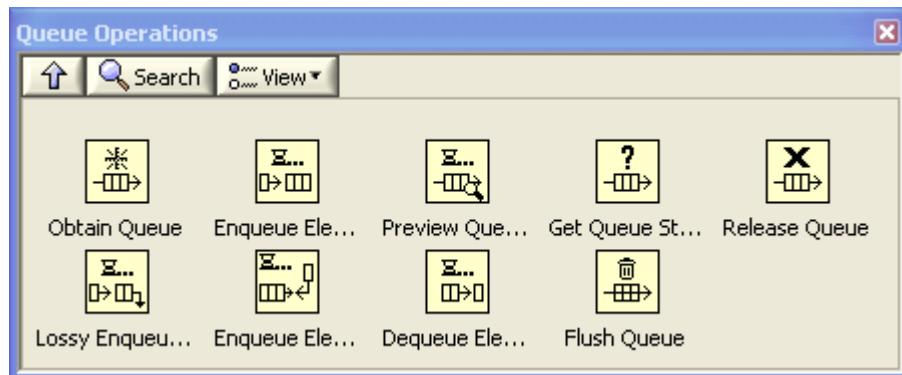
You can use the Semaphore VIs to synchronize two or more separate, parallel tasks so that only one task at a time executes a critical section of code protected by a common semaphore. In particular, use these VIs when you want certain VIs or parts of a block diagram to wait until another VI or part of a block diagram is finished with the execution of a critical section.

A typical example could be where two resources want to write to the same file, but only one can access to file at a time. We can use the Semaphore VIs to synchronize this operation. The application could have 2 while loops in parallel. Each while loop can then write to the same file if you synchronize the operations using semaphores.

Queue Operations

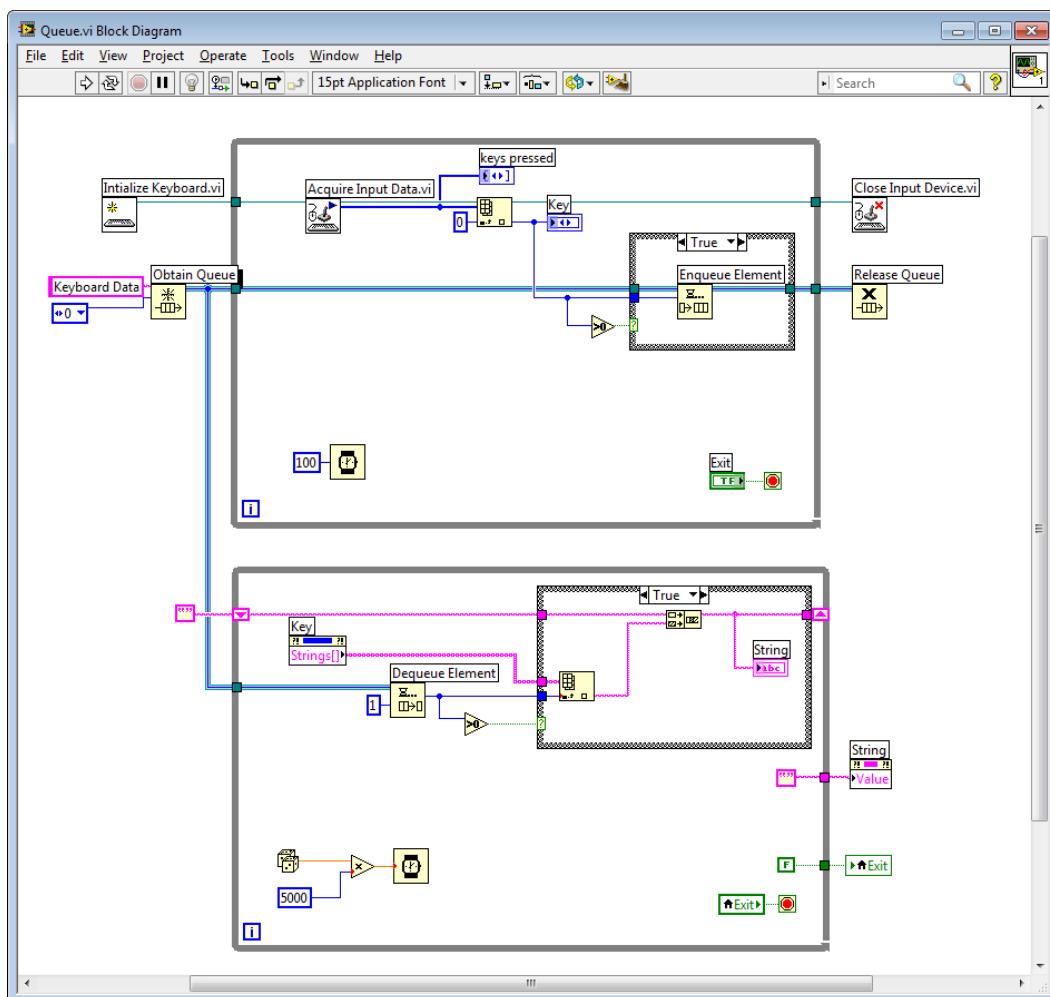
You will use the Queue Operations functions to create a queue for communicating data between sections of a block diagram or from another VI.

The Queue Operations palette in LabVIEW:

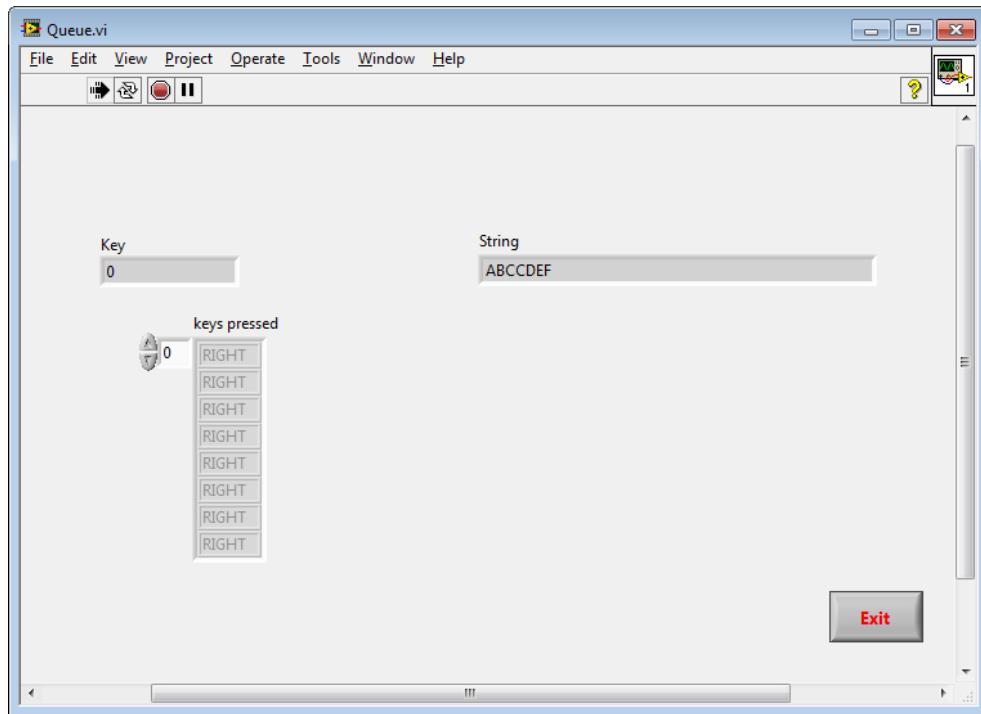


Example:

You typically use the Queue VIs for exchange of messages. E.g., a **Sender** activity shall read letters from the keyboard and put them into the Queue. The **Receiver** activity shall read the letters from the Queue and display them on a screen in a String indicator. The **Sender** and the **Receiver** activity could operate at different speed (delay). Below we see the LabVIEW code:



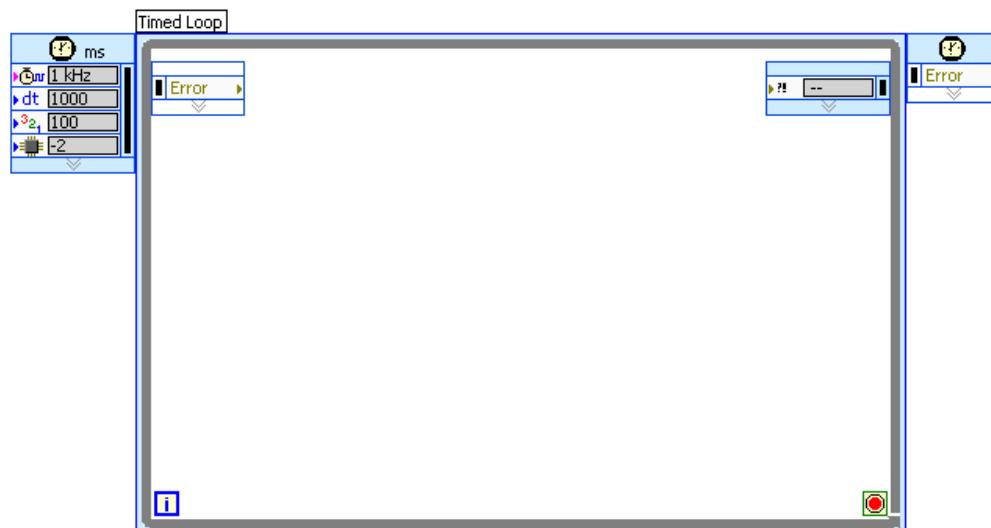
We type inn some letters using the keyboard. The Receiver is much slower, but to be sure to not miss any letters, we use a Queue to handle this. Below we see the Front Panel:



[End of Example]

5.3 Timed Loop

In LabVIEW we can also use “**Timed Loop**” structure instead of ordinary While Loops.



6 Multithreading in LabVIEW

A **multicore** system is a single-processor CPU that contains two or more cores, with each core housing independent microprocessors. A multicore microprocessor performs multiprocessing in a single physical package. Multicore systems share computing resources that are often duplicated in multiprocessor systems, such as the L2 cache and front-side bus.

Multicore systems provide performance that is similar to multiprocessor systems but often at a significantly lower cost because a motherboard with support for multiple processors, such as multiple processor sockets, is not required.

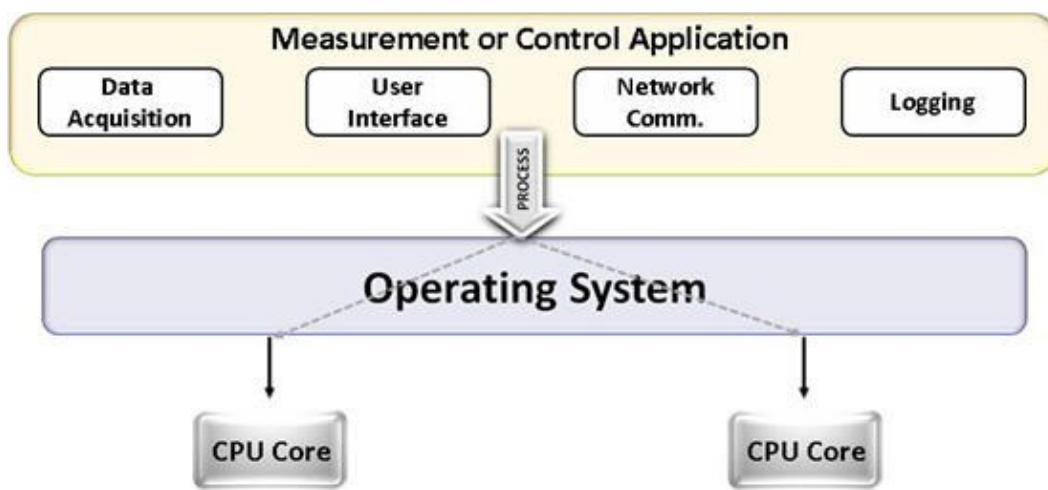
6.1 Multitasking

In computing, **multitasking** is a method by which multiple tasks, also known as processes, share common processing resources such as a CPU. With a multitasking OS, such as Windows XP, Vista, etc., you can simultaneously run multiple applications. Multitasking refers to the ability of the OS to quickly switch between each computing task to give the impression the different applications are executing multiple actions simultaneously.

6.2 Multithreading

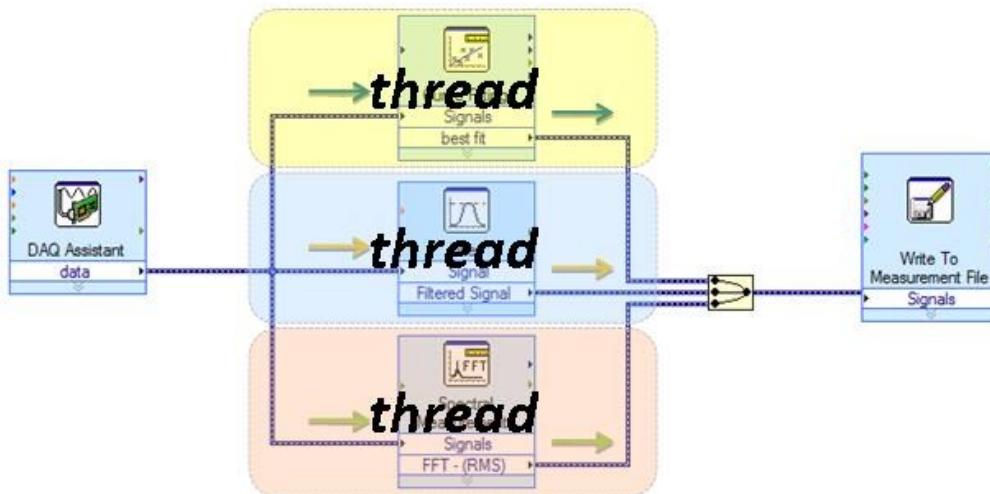
Multithreading extends the idea of multitasking into applications, so you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

In a multithreaded National Instruments LabVIEW program, an example application might be divided into four threads - a user interface thread, a data acquisition thread, network communication, and a logging thread. You can prioritize each of these so that they operate independently. Thus, in multithreaded applications, multiple tasks can progress in parallel with other applications that are running on the system.

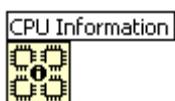


Multithreading in LabVIEW:

LabVIEW automatically divides each application into multiple execution threads. The complex tasks of thread management are transparently built into the LabVIEW execution system.



You can also use the “CPU Information” function in LabVIEW to find CPU information about your PC.



Multitasking in LabVIEW:

LabVIEW uses preemptive multithreading on OSs that offer this feature. LabVIEW also uses cooperative multithreading. OSs and processors with preemptive multithreading employ a limited number of threads, so in certain cases, these systems return to using cooperative multithreading.

The execution system preemptively multitasks VIs using threads. However, a limited number of threads are available. For highly parallel applications, the execution system uses cooperative multitasking when available threads are busy. Also, the OS handles preemptive multitasking between the application and other tasks.

7 DataSocket

DataSocket is a technology for sharing data between applications or different data sources. The DataSocket control provides a simple interface through which it can interact with OPC servers, such as National Instruments FieldPoint, from any ActiveX container, including Visual Basic and Visual C++. You can connect to an OPC server with DataSocket using an OPC URL, which is similar to the URLs used in a Web browser. URLs provide a standard mechanism for referring to locations. You already know how to use URLs to locate things on the Web, and you can locate OPC data items with DataSocket using a similar URL model:

`opc://machine_name/server_name/item_name`

With DataSocket, you can share live data with one or more client applications on a network without worrying about data formats and network protocols. Your LabVIEW applications can easily share live data with a variety of clients, including Visual Basic applications, Web browsers, Visual C++, Microsoft Excel, LabWindows/CVI, and other LabVIEW applications. Using DataSocket technology, you can publish and receive data from any application in the same way, giving you the power to connect diverse applications easily.

Many typical instrumentation solutions involve a single local application for acquisition, logging, analysis, and presentation. However, because of the rise in popularity of the Internet and company intranets, and the need to remotely monitor and control your data acquisition, you often need to exchange live data with other applications on different computers around the world. DataSocket for LabVIEW simplifies live data exchange between different applications on one computer or between computers connected through a network.

Although a variety of different technologies exist today to share data between applications, including TCP/IP and dynamic data exchange (DDE), most of these tools are not targeted for live data transfer to multiple clients. With TCP/IP, you have to convert your data into an unstructured stream of bytes in the broadcasting application and then parse the stream of bytes back into its original format in subscribing applications. DataSocket, however, simplifies live data transfer. It implements an easy-to-use, high-performance programming interface that is designed specifically for sharing and publishing live data in measurement and automation applications. In addition, DataSocket features interapplication connectivity, rich data types, and security to make sharing data easy. DataSocket is included with LabVIEW.

If you are comfortable programming with COM, you can write programs using either the OPC Custom API or Automation API, depending on the programming environment in which you are developing your OPC client.

If you do not want to program with COM, use the DataSocket control, an ActiveX control for sharing data between applications and devices. DataSocket offers the following benefits:

- Simple API, helpful to developers who don't want to program with COM.
- Access to the OPC server custom interface, which enables your applications to run faster than applications that access the automation interface. DataSocket is not an extra layer but rather a direct connection to OPC servers, and DataSocket always accesses the custom interface, regardless of the ActiveX container in which you are developing.
- Easy integration with National Instruments products such as LabVIEW.

7.1 Architecture

DataSocket is a single, unified, end-user application programming interface (API) for connecting to data from a number of sources – local files, files on FTP or Web servers, and data items on OPC Servers. A DataSocket application specifies the data location by using a familiar networking standard, the URL. Just as a Web browser uses a URL to connect to a Web page, a DataSocket application uses a URL to connect to data. By using an industry-standard URL, you can quickly and easily bring data into or share data from your DataSocket applications.

opc://machine_name/server_name/item_name

machine_name [optional] - Computer on which the OPC server is installed. DataSocket can access OPC servers on other computers using DCOM. If the machine name is omitted, DataSocket directly connects to the OPC server on the computer on which it is running.

server_name - OPC server (provided with the hardware) to connect to.

item_name - OPC item on the specific OPC server. An item is a channel or variable (normally an I/O point) in a real-world device that a device server monitors or controls. Example FieldPoint item: FP Res\FP-DI-330 @1\Channel 1

UpdateRate=n in milliseconds [optional] - Maximum rate at which the OPC server will indicate that an item's value has changed. The server should use an update rate as close as possible to the rate requested by the client. If this parameter is omitted, the default is 100 ms. The following example sets the update rate to 1000 ms: UpdateRate=1000

DeadBand=n in % of range [optional] - Percentage change required before the server notifies your application of a value change. Not all servers support this option. If omitted, this parameter defaults to 0%. The following example sets the deadband to 10%: DeadBand=10

For example, the following URL connects to the sine item on the National Instruments OPCDemo server on the local machine. The options at the end of the URL specify an update rate of 1000 ms and a deadband of 10%.

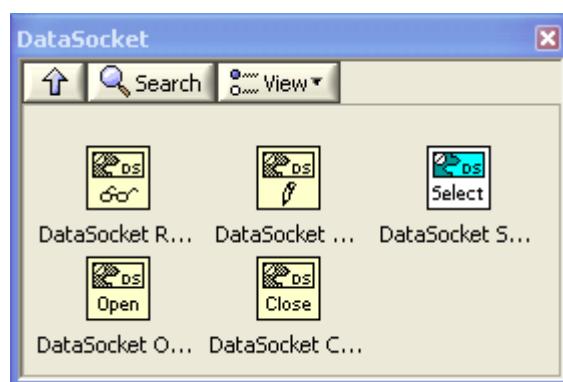
opc:/National Instruments.OPCDemo/sine?UpdateRate=1000&DeadBand=10

7.2 DataSocket Server

With the DataSocket Server, a lightweight, stand-alone component, programs using DataSocket can broadcast live measurement data at high rates across the Internet to multiple remote clients concurrently. These client applications use DataSocket to subscribe to the live measurement data. Because the DataSocket Server is a stand-alone component, it simplifies network (TCP/IP) programming by automatically managing connections to clients and automatically converting your measurement data to and from the stream of bytes sent across the network. You do not have to write the parsing code. And because the DataSocket Server can run on any machine on your network, it also improves performance and provides security by isolating the Web connections from your acquisition application.

7.3 DataSocket in LabVIEW

The DataSocket palette in LabVIEW:



Description of the DataSocket VIs in LabVIEW.

 **DataSocket Select URL** Displays a dialog box for the user to select a data source and returns the URL to that data.

 **DataSocket Open** Opens a data connection you specify in URL.

 **DataSocket Read** Dequeues the next available data value from the client-side buffer associated with the connection you specify in connection in and returns the data.

 **DataSocket Write** Writes data to the connection you specify in connection in.



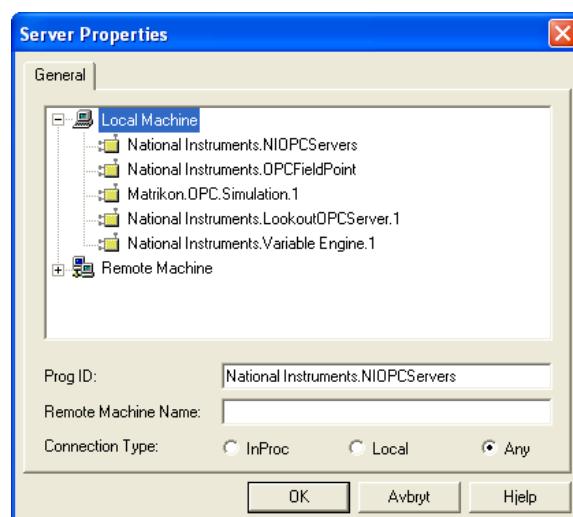
DataSocket Close Closes a data connection you specify in connection id.

8 OPC Servers from National Instruments

National Instruments offers different OPC Servers, such as:

- NI OPC Servers
- FieldPoint OPC Server
- Variable Engine
- Etc.

If you browse for OPC servers and OPC Items on your computer it probably looks something like this:

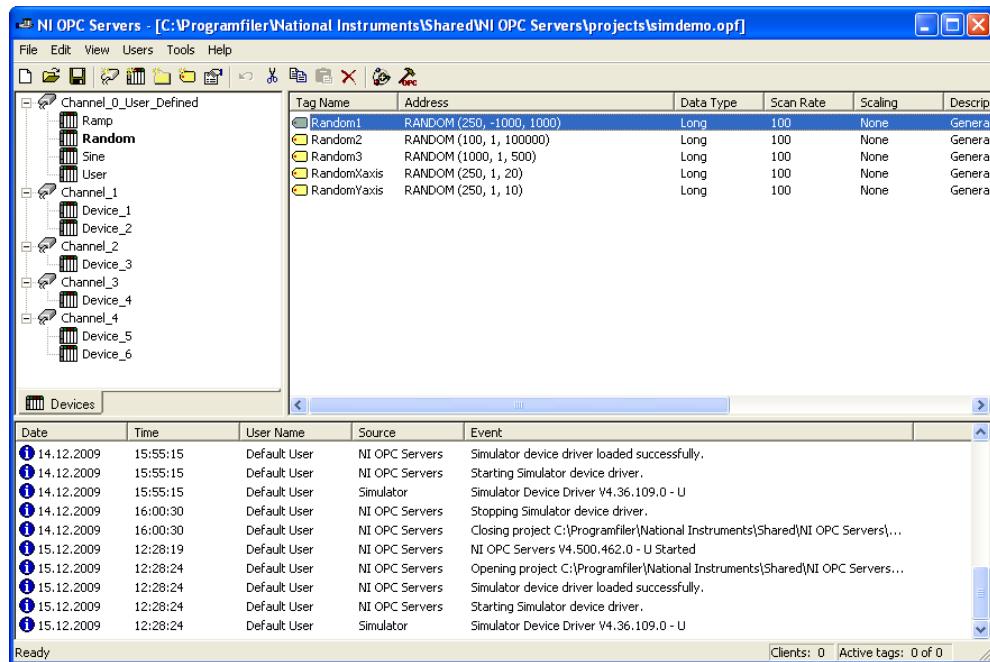


Below we go through the basic functionality in these OPC Servers.

8.1 NI OPC Servers

The National Instruments OPC Servers provides a single consistent interface to communicate with multiple devices, saving you from learning new communication protocols or spending time understanding new applications. The combination of NI OPC Servers and LabVIEW provides a single platform for delivering high performance measurements and control to both new and existing industrial systems. NI OPC servers connect through the OPC client in LabVIEW Datalogging and Supervisory Control (DSC) Module to enable you develop a full fledged HMI/SCADA system with PLCs, PACs and smart sensors.

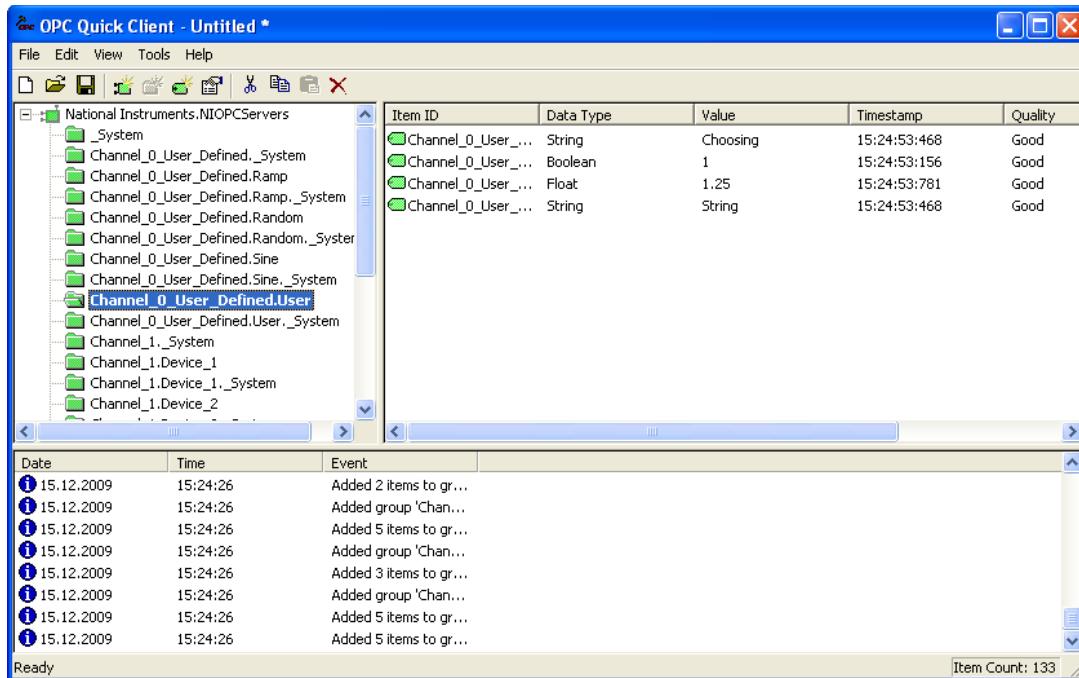
NI OPC Servers are a 32-bit windows application that provides a means of bringing data and information from a wide range of industrial devices and systems into client applications on your windows PC. Our OPC Server application enables the sharing of manufacturing or production data between a variety of applications ranging from human machine interface software and data historians, to large MES and ERP applications.



8.1.1 OPC Quick Client

The **OPC Quick Client** has been developed to assist in the test and development of the OPC Data Access 1.0 and 2.0 Servers. The OPC Quick Client does not fully support OPC DA 3.0.

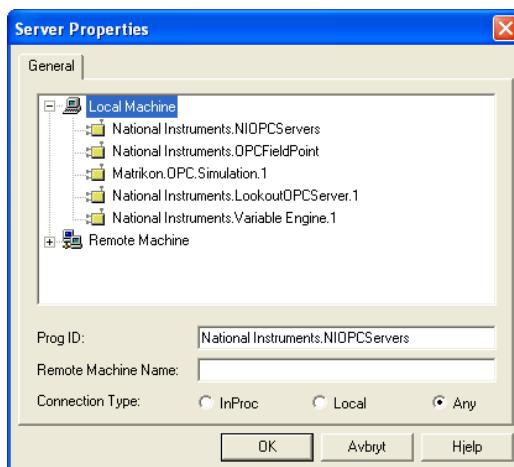
The OPC Quick Client supports both local and remote OPC server connections. Remote connections are handled through the operating system's DCOM interface.



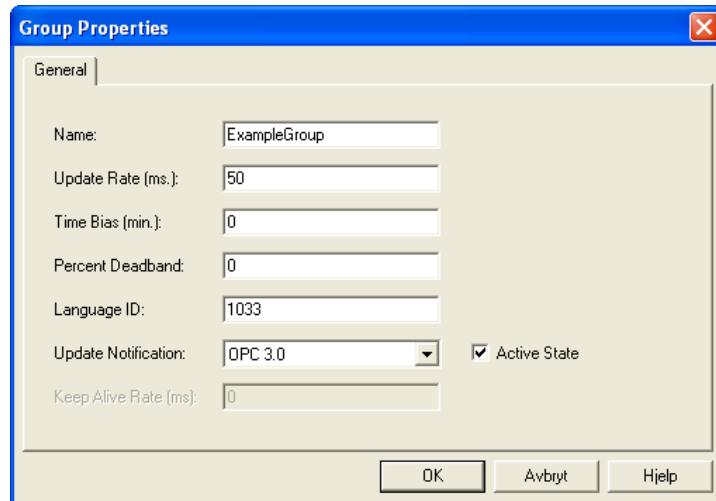
To start using the OPC Quick Client, do the following:

1. Create a **server connection** which will be used to connect to an OPC server.
2. Next, add a **group** to the connection which contains varying properties (such as update rate, dead band and time bias).
3. Finally, add **items** to the individual groups (which contain properties such as an initial active state, data type and access path).

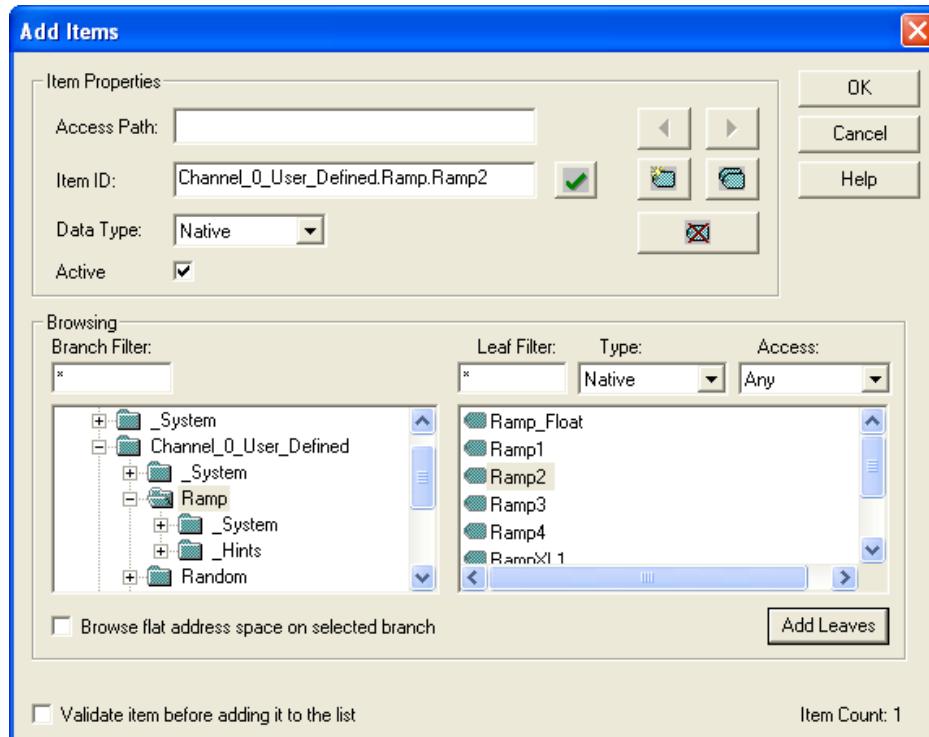
A **server connection** provides a link between an OPC server and this client. Groups are added through this connection. To create a new server connection, click **Edit | New Server Connection....**. Alternatively, click the **New Server** toolbar button.



A **group** is used to organize a collection of items with a common set of properties. The group also specifies the following properties: group name, update rate, time bias, percent dead band, Language ID, active state and the data connection type. To create a new group, click **Edit | New Group**. Alternatively, click **New Group** in the toolbar.



Items represent data that may be accessed via an OPC server. An item specifies the following properties: Item ID, access path, requested data type and active state. To define an item using the Item Editor dialog, click **Edit | New Item**. Alternatively, click **New Item** on the toolbar.



8.2 Shared Variable Engine

LabVIEW contains an OPC server called the Shared Variable Engine. The Shared Variable Engine supports OPC Data Access 2.x and OPC Data Access 3.0. You can publish data from the Shared Variable Engine using LabVIEW shared variables.

To connect to the Shared Variable Engine from a third-party OPC client, use the ProgID National Instruments.Variable Engine. If the OPC client allows you to browse for OPC servers, you can locate the National Instruments.Variable Engine under OPC version 2.x or 3.0, depending on which versions the client supports.

9 MatrikonOPC Simulation Server

MatrikonOPC Simulation is free for non-production use and can be distributed openly. It is a fully functioning application without restriction.

MatrikonOPC Simulation Server is a free utility used to help test and troubleshoot OPC applications (clients) and connections. Testing applications on “live” OPC servers may result in loss of actual production data. The MatrikonOPC Simulation Server creates a simulated environment so that in the event of a problem, no real process data is lost.

MatrikonOPC Explorer is an OPC Client application for testing and configuring OPC connections.

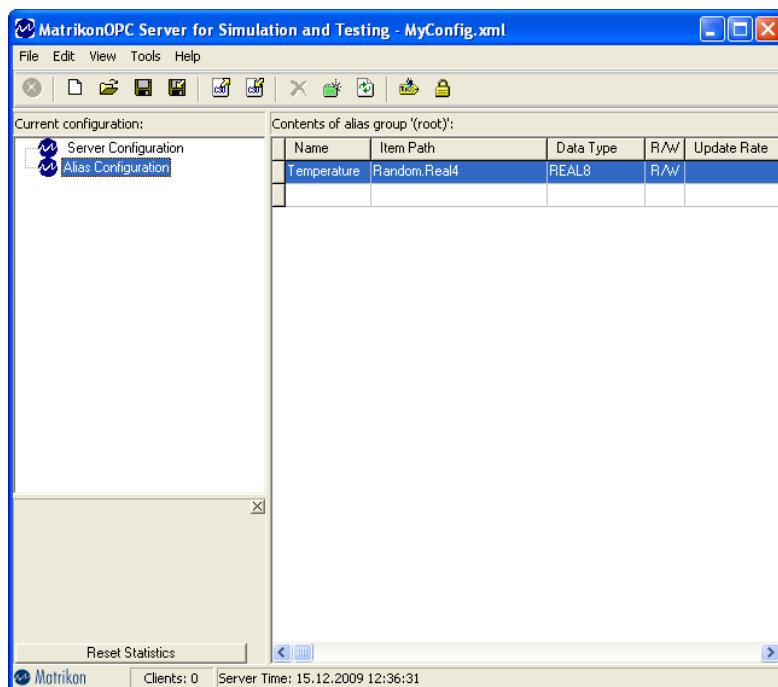
Download MatrikonOPC Simulation Server and MatrikonOPC Explorer her:

<http://www.matrikonopc.com/downloads/178/index.aspx>

Watch this video: http://www.matrikonopc.com/training/opc-multimedia-tutorial/opcda_pop.html

9.1 MatrikonOPC Server

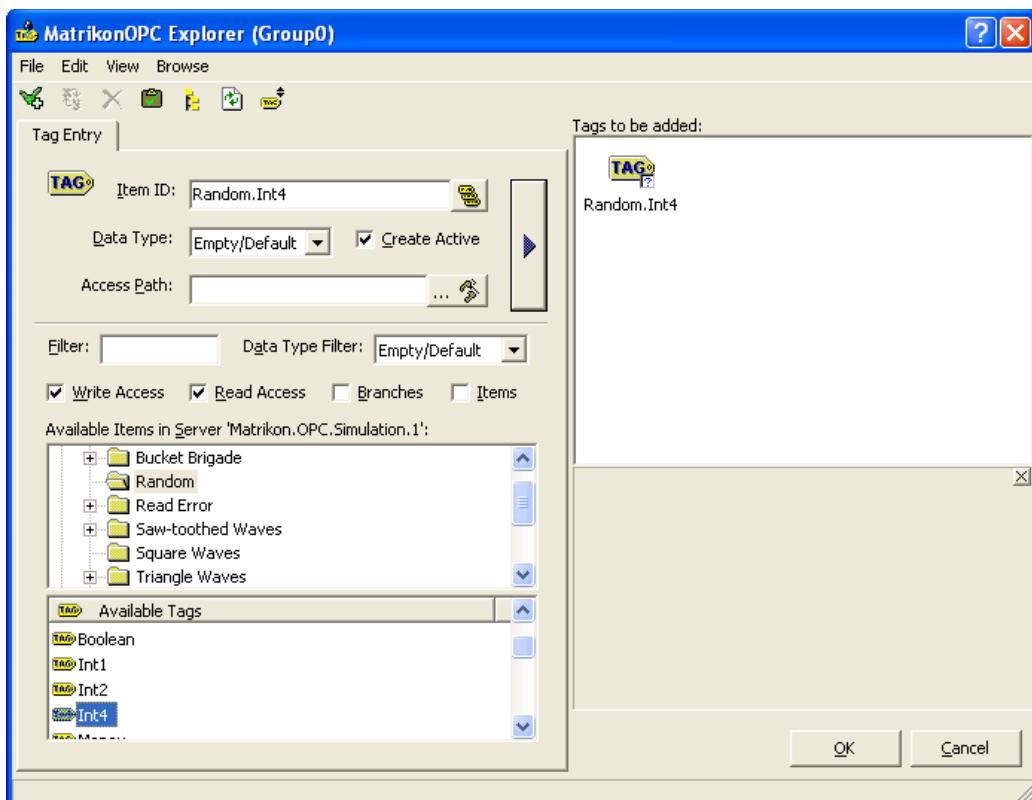
Below we see the MatrikonOPC Server for Simulation:



View Tags:



Open the MatrikonOPC Explorer where you browse for available Items (Tags).



9.2 Aliases

MatrikonOPC Servers, including this one, provide the ability to create user-defined aliases that can be used in place of regular OPC items. This feature is particularly useful when the item path for a given server is very complex or difficult to remember, for example: Com1.Radio1.Unit1.41.4.123. Servers can also be configured so that client applications have access to configured aliases only, rather than every available item.

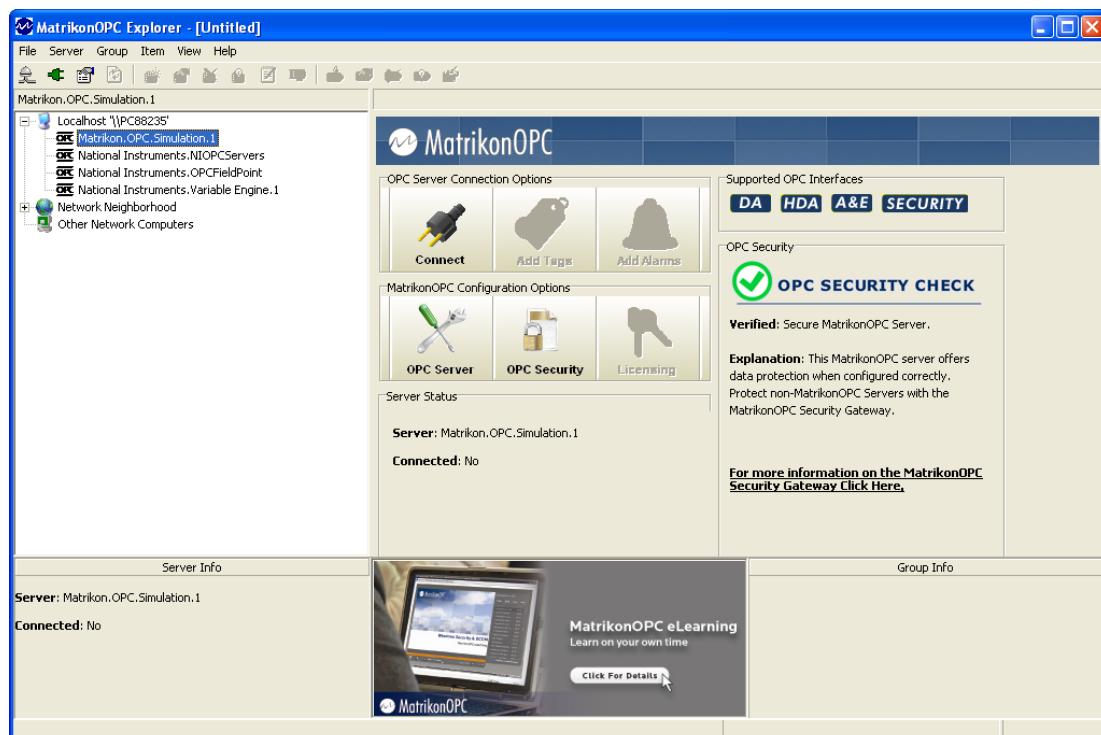
To insert a new alias, perform the following steps:

1. Open the MatrikonOPC Server for Simulation Configuration window.
2. Click on the Alias Configuration node in the Current Configuration panel to open the Alias Configuration panel.
3. Right-click your mouse and select the Insert New Alias option, or Press the Insert button on the keyboard, to launch the Insert New Alias window.
4. Fill in a name for the alias, and enter a valid item path. The item path can be found by browsing the server's address space (use the ellipsis button next to this field).
5. Save this alias when finished. It will now be visible to OPC clients under the Configured Aliases heading.

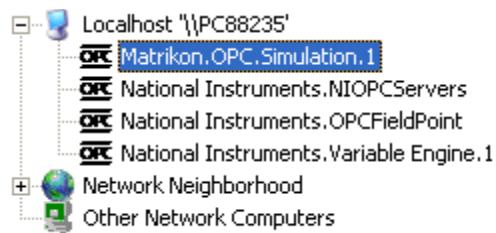
For more detailed scaling options, please consult the MatrikonOPC Server for Simulation User's Manual.

9.3 Matrikon OPC Explorer (Client)

MatrikonOPC Explorer is an OPC Client application for testing and configuring OPC connections.



The MatrikonOPC Explorer lists all the available OPC Servers in the upper left corner.



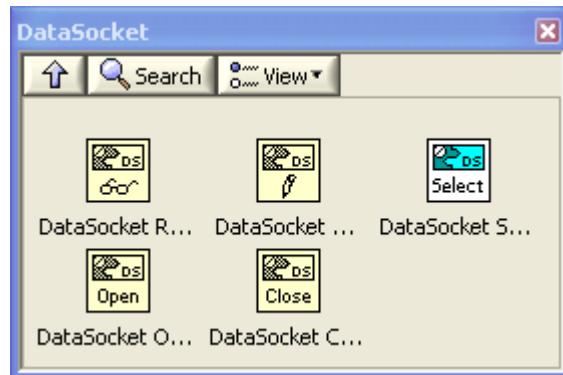
MatrikonOPC Explorer lists OPC servers on your computer (localhost) or in the network.

Click Connect on order to connect to the server.

10 Using OPC from LabVIEW

You can use LabVIEW as an OPC client by connecting to an OPC server through a DataSocket connection.

The DataSocket palette in LabVIEW:



Description of the DataSocket VIs in LabVIEW:



DataSocket Select URL Displays a dialog box for the user to select a data source and returns the URL to that data.



DataSocket Open Opens a data connection you specify in URL.



DataSocket Read Dequeues the next available data value from the client-side buffer associated with the connection you specify in connection in and returns the data.



DataSocket Write Writes data to the connection you specify in connection in.



DataSocket Close Closes a data connection you specify in connection id.

10.1 OPC URL

A typical URL for an OPC Item could be:

opc://localhost/Matrikon.OPC.Simulation/Random.Int4

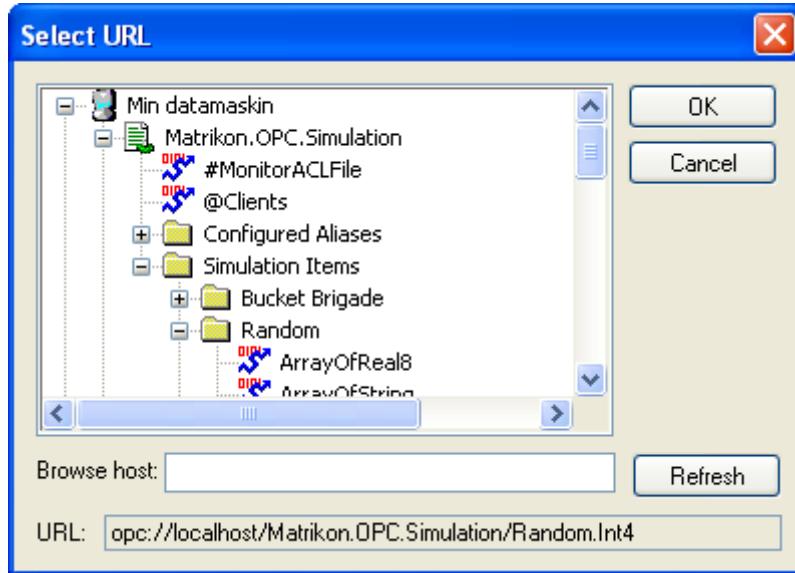
or in general:

`opc://machine_name/server_name/item_name`



You may use the “**DataSocket Select URL**” VI in order to find an OPC Item.

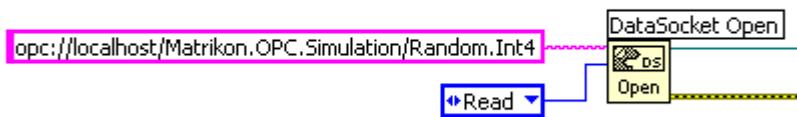
Using the “DataSocket Select URL” displays the “Select URL” window:



This VI is nice to have when you don't know the exact name of the OPC URL.



If you know the URL in advance you use the “**DataSocket Open**” VI. This VI opens a data connection you specify in the input URL. Like this:

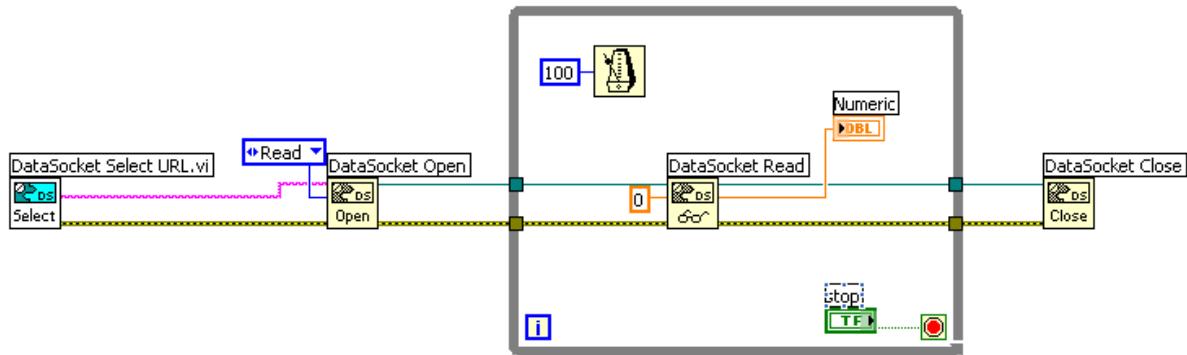


10.2 Read OPC Data

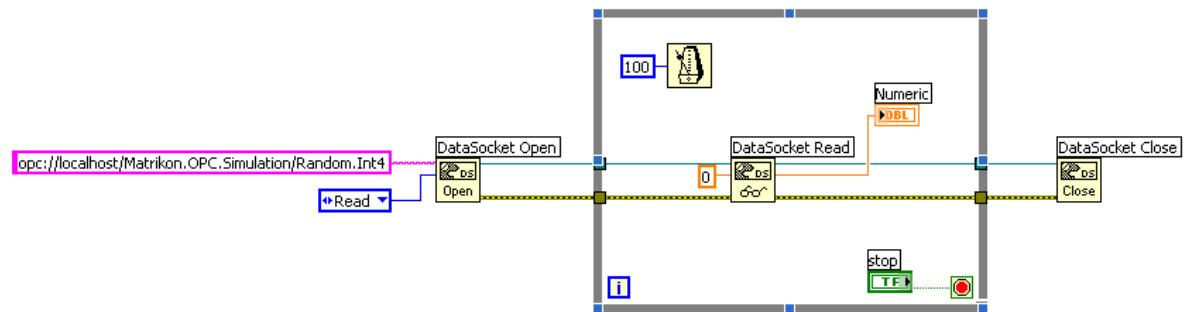


You use **DataSocket Read** in order to get data from a specific item in the OPC server.

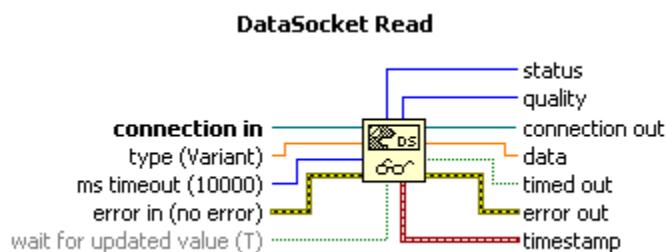
Example:



or using the specific URL directly like this:



The DataSocket Read VI have several outputs in addition to the Value (data).



All OPC Items have the following properties:

- Value
- Timestamp
- Status
- Quality

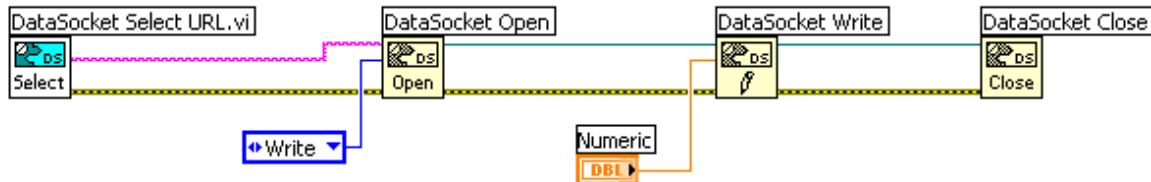
10.3 Write OPC Data



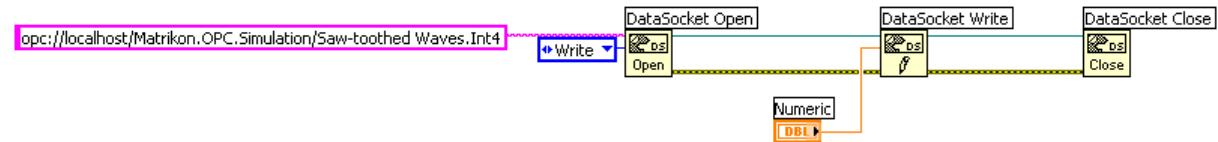
You use **DataSocket Write** in order to write data to a specific Item in the OPC server.

Note! In order to write data to an OPC Item it must have the Write property set.

Example:



or using the specific URL directly like this:



11 LabVIEW Datalogging and Supervisory Control

The LabVIEW Datalogging and Supervisory Control (DSC) Module extends the LabVIEW graphical development environment with additional functionality for the rapid development of distributed measurement, control, and high-channel-count monitoring applications.

The DSC Module also enhances the LabVIEW shared variable. Use the shared variable to access and pass data among several VIs in a LabVIEW project or across a network. A shared variable can represent a value or an I/O point. With the DSC Module, you can log data automatically; add alarming, scaling, and security to the shared variable; and configure the shared variable programmatically.

The DSC Module also provides tools for graphing historical or real-time trends, enhancing the security of front panels, and writing custom I/O servers. You can read or write to OLE for Process Control (OPC) connections, programmable logic controllers (PLC), or custom I/O servers that you write.

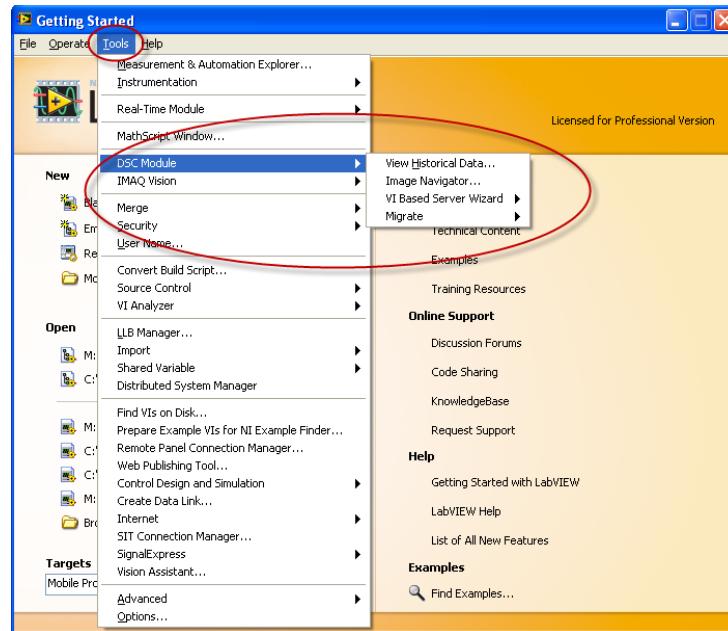
To run applications built with LabVIEW, the DSC Module on a computer without the DSC Module installed, you must install the “DSC Module Run-Time System” on that computer. The DSC Module Run-Time System contains components that enable the DSC Module features in the built applications.

The DSC Module includes the following components:

- Functions and VIs Palettes
- DSC Module Controls Palettes
- Citadel Database
- Historical Data Viewer
- Distributed System Manager

These components are described in more detail below.

Most of the LabVIEW DSC functionality is available from the Tools menu (“Tools → DSC Module”).

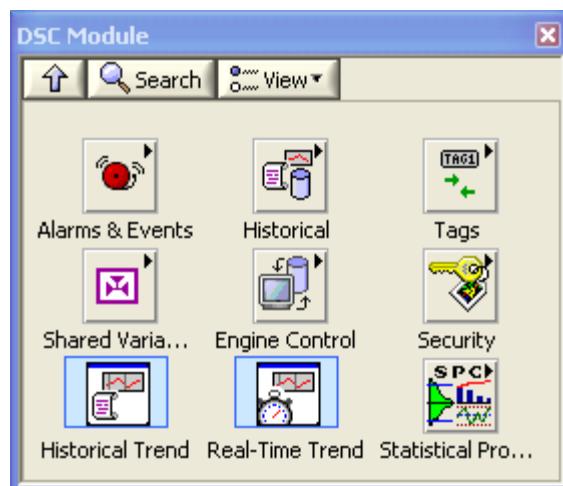


11.1 Functions and VIs Palettes

The DSC Module installs the following palettes:

- Alarms & Events
- Historical
- Tags
- Shared Variables
- Engine Control
- Security

The DSC Module also includes the Historical Trend Express VI and the Real-Time Trend Express VI.

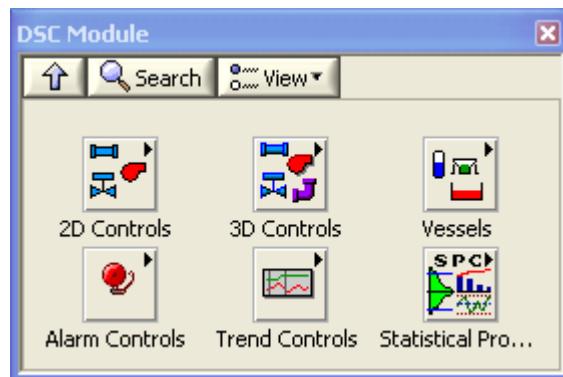


11.2 DSC Module Controls Palettes

The DSC Module installs the following controls palettes to help you build user interfaces that resemble a plant or system and to view real-time data from the plant or system:

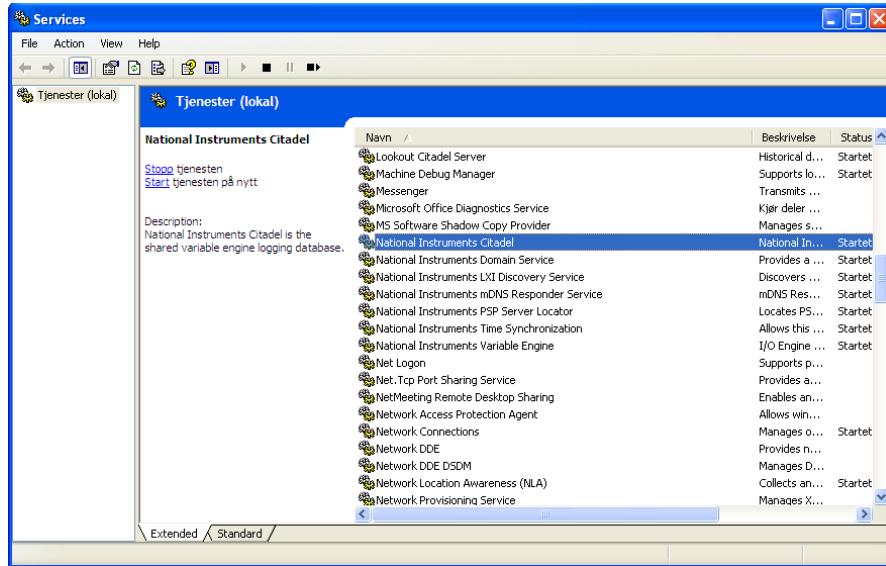
- 2D Controls
- 3D Controls
- Vessels
- Alarm Controls
- Trend Controls

The DSC Module also includes the Historical Trend control and the Real-Time Trend control.



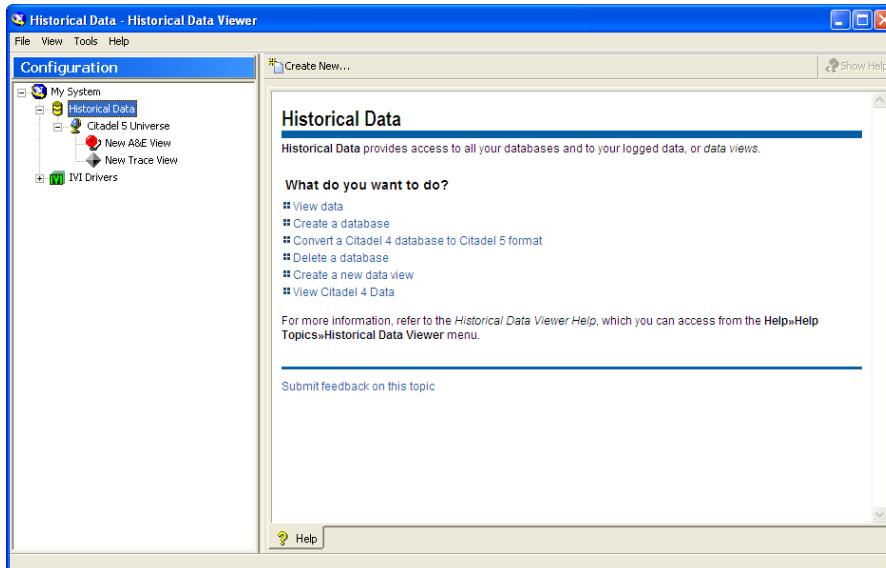
11.3 Citadel Database

The DSC Module logs shared variable data to the Citadel database. **The Citadel database stores historical data, alarms, and events.** You can access and view Citadel data using the Historical Data Viewer and using the “Historical VIs”.

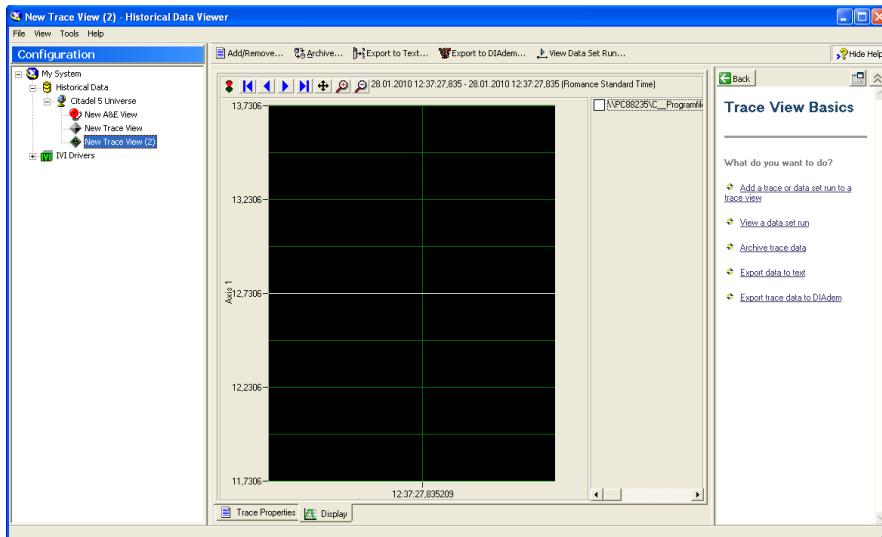


11.4 Historical Data Viewer

Use the Historical Data Viewer to view data stored in the Citadel database. Select “Tools → DSC Module → View Historical” Data to launch the Measurement & Automation Explorer (MAX). Expand the “Historical Data” category to select a database that appears under “Citadel 5 Universe”. You also can use the Call HDV VI to launch the Historical Data Viewer programmatically.

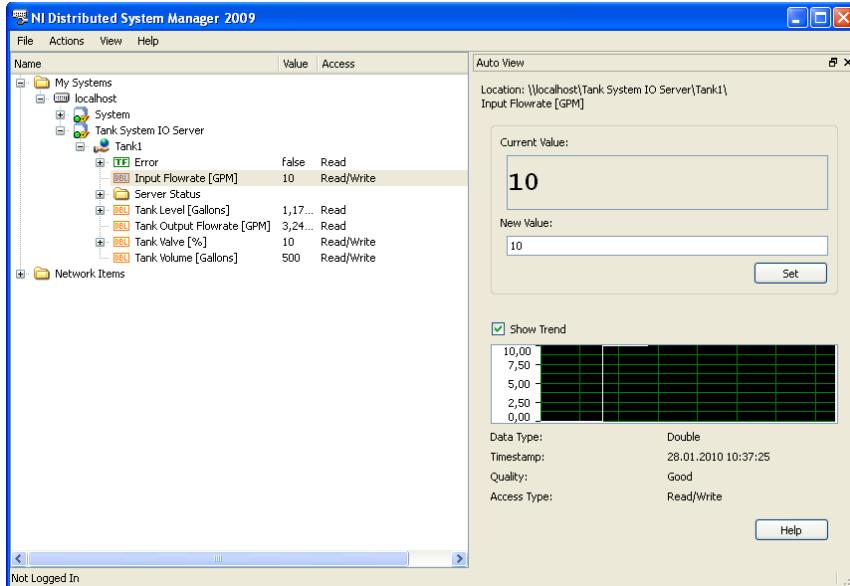


Data Logger:



11.5 Distributed System Manager

Use the Shared Variable Monitor to view the current value of a shared variable and its status and alarm state. Select “**Tools → Distributed System Manager**” to launch the **Distributed System Manager**.



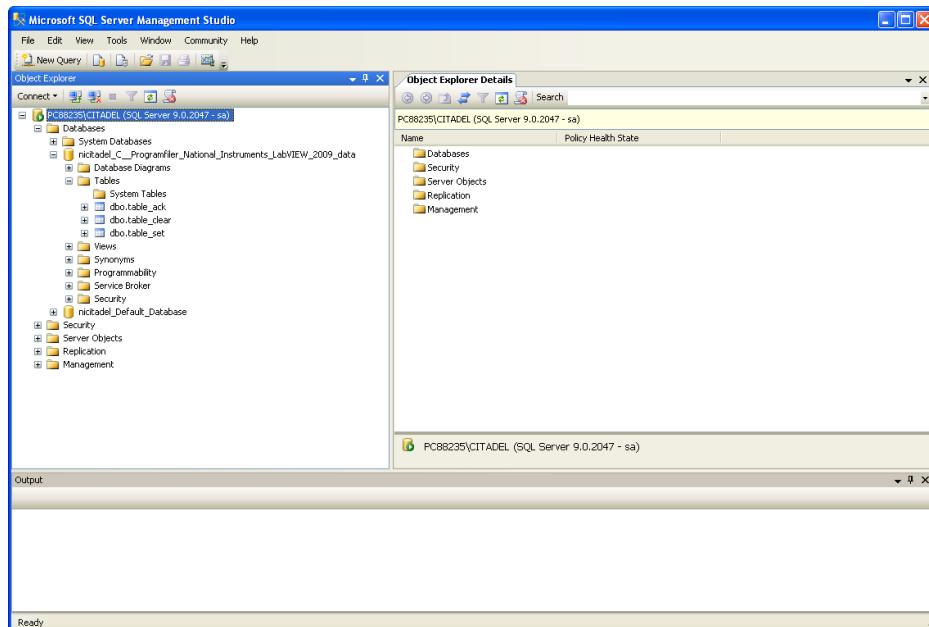
11.6 SQL Server

The DSC Module requires the Microsoft SQL Server 2005 Express Edition (SQL Express). This component is installed by default when you install the DSC Module. During the installation process,

the DSC Module installer creates an instance of SQL Express named CITADEL. To prevent unauthorized access to SQL Express, the installer also generates a password for the default SQL Express administrator sa. The default password is the computer ID. Complete the following steps to find the computer ID using the NI License Manager.

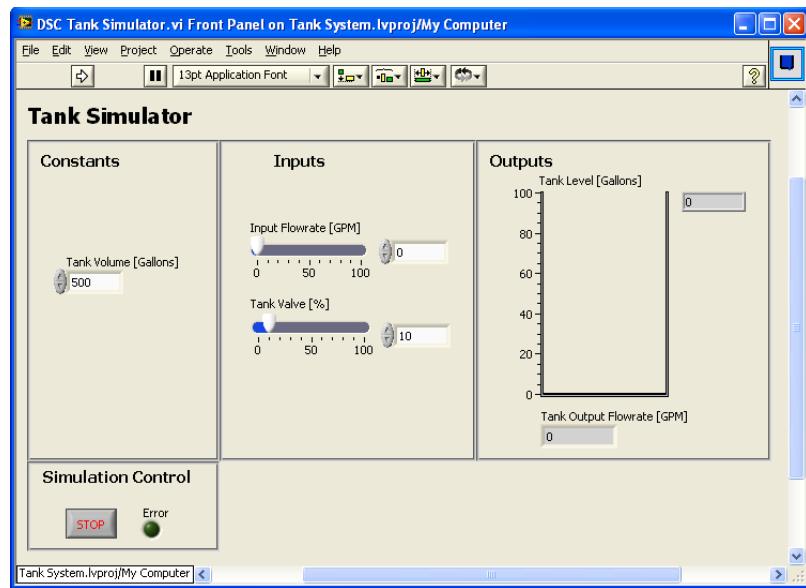
1. Launch the NI License Manager by selecting “Start → All Programs → National Instruments → NI License Manager”.
2. Click the Display Computer Information button on the toolbar.

The SQL Server database is used for configurations, etc., while the Citadel database is used for Datalogging.



Exercises

In the following exercises we will build an application in LabVIEW using the functionality from the LabVIEW DSC Module.



The Tasks are as follows:

- Task 1: Open and Run the Example
- Task 2: Creating a New Project Library
- Task 3: Creating a Periodic I/O Server
- Task 4: Deploying the Periodic I/O Server
- Task 5: Creating Shared Variables
- Task 6: Configuring Data Logging
- Task 7: Configuring Alarming
- Task 8: Enabling Logging
- Task 9: Create the LabVIEW Application
- Task 10: Use the LabVIEW DSC Functions and VIs
- Task 11: Viewing Real-Time Data
- Task 12: Viewing Alarms using the Distributed System Manager

11.6.1 Task 1: Open and Run the Example

Open **DSC Tank Simulator.vi**. The Example is available from: <http://home.hit.no/~hansha/>

The example does not use any DSC functionality so far. In the next exercise you will create a DSC application based on this example.

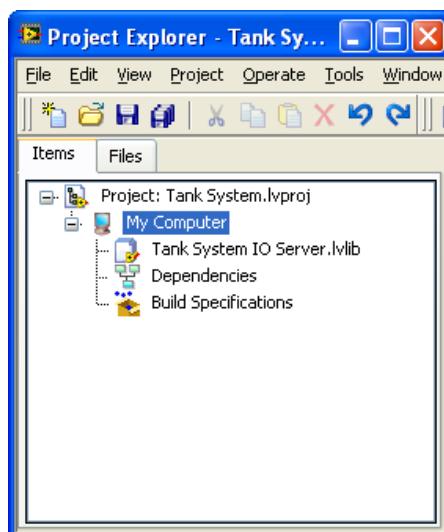
11.6.2 Task 2: Creating a New Project Library

In this exercise you will learn how to create a new LabVIEW project and a project library. The project allows you to manage shared variables, project libraries, and VIs in one window. LabVIEW project

libraries are collections of VIs, type definitions, shared variables, palette menu files, and other files, including other project libraries.

Complete the following steps to create a LabVIEW project library.

- Click the **Empty Project** link in the **Getting Started** window. The **Project Explorer** window appears.
- Right-click **My Computer** in the **Project Explorer** window and select **New → Library** from the shortcut menu.
- Select **File → Save All**. The **Name the Project** dialog box appears.
- Enter “Tank System” in the **File name** text box.
- Click the **OK** button. The **Name the Library** dialog box appears.
- Enter “Tank System IO Server” in the **File name** text box.
- Click the **OK** button.



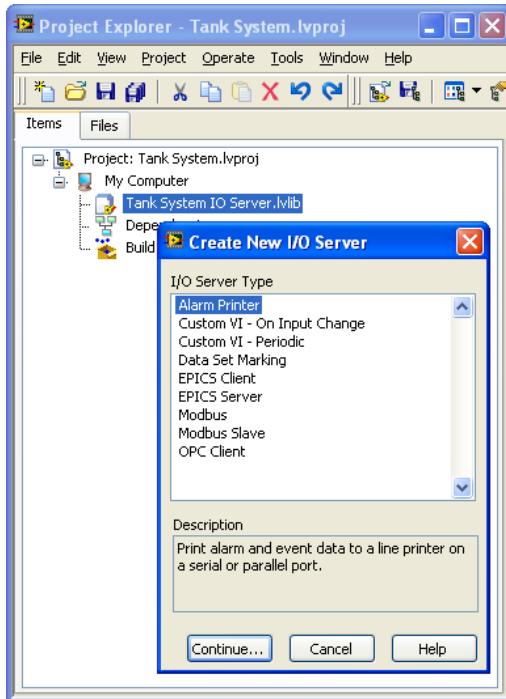
You now have a project containing a project library. In the following exercise you will use the project library in the project to create a periodic I/O server.

11.6.3 Task 3: Creating a Periodic I/O Server

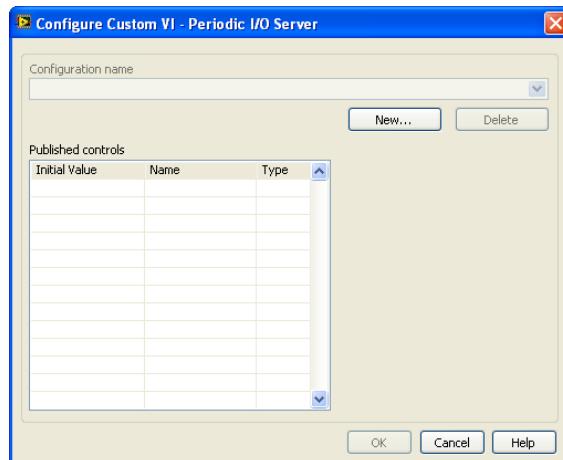
A server is an application that communicates with and manages input/output devices such as PLCs, remote input/output devices, remote Shared Variable Engines, and data acquisition (DAQ) plug-in devices. These servers read selected input items and write to them on demand. The DSC Module can connect to any OPC-compliant server and to many third-party device servers. You also can create custom I/O servers. You will build a periodic I/O server in the following exercise. The periodic I/O server will run as a service and publish NI Publish-Subscribe Protocol (NI-PSP) data items to the network.

Complete the following steps to add the periodic I/O server to the project.

Right-click the **Tank System IO Server.lvlib** project library in the **Project Explorer** window and select **New → I/O Server** from the shortcut menu. The **Create New I/O Server** dialog box appears.



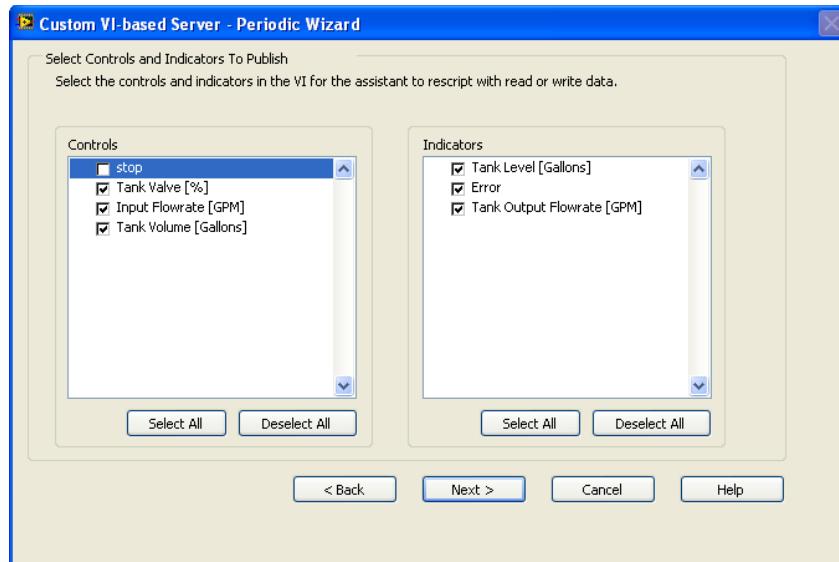
Select **Custom VI – Periodic** from the **I/O Server Type** list and click the **Continue** button. The **Configure Custom VI – Periodic I/O Server** dialog box appears.



Click the **New** button to display the **Select VI** step of the Custom VI-based Server – Periodic Wizard.

Select the **DSC Tank Simulator.vi**.

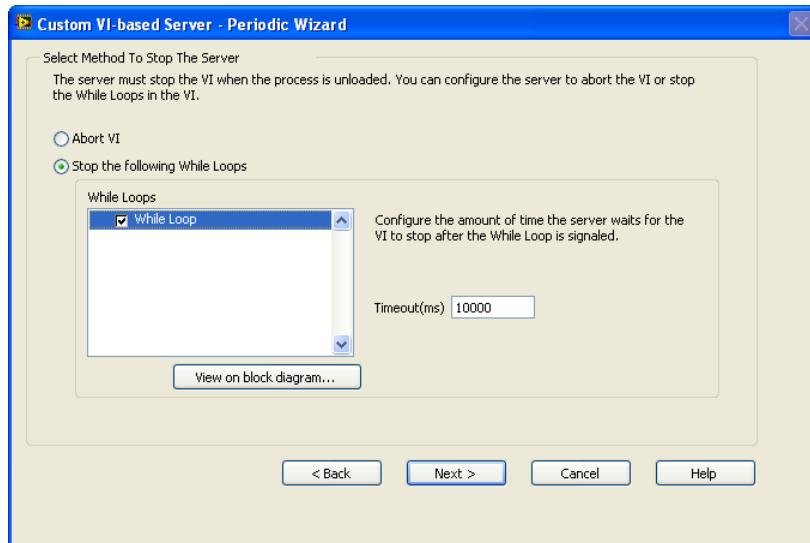
Click the Next button in the Custom VI-based Server – Periodic Wizard to advance to the **Select Controls and Indicators To Publish** step.



Here you will select Controls and Indicators you want to publish as shared variables.

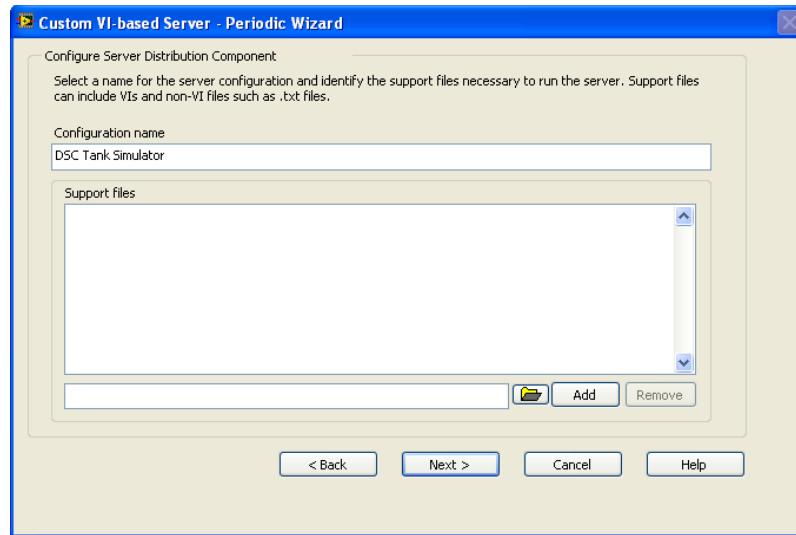
Remove the checkmark from the **stop** checkbox in the **Controls** list. You will publish the remaining controls and indicators.

Click the Next button to advance to the **Select Method To Stop The Server** page.



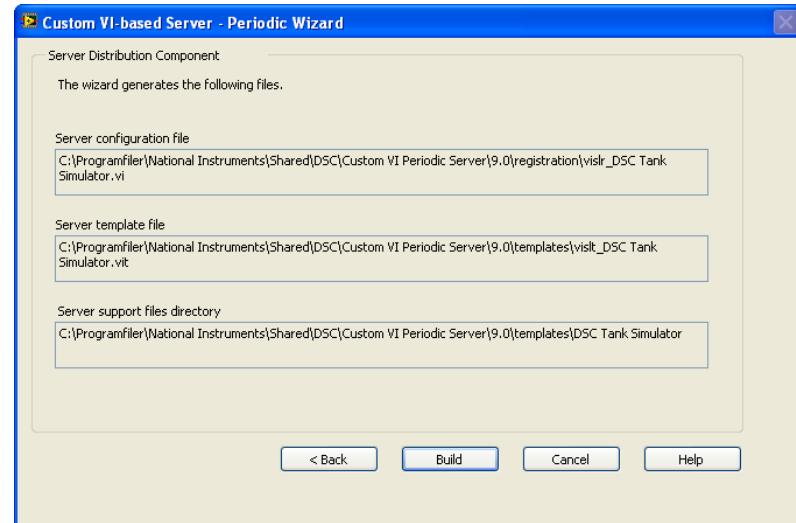
Select **Stop the following While Loops** and Place a checkmark in the **While Loop** checkbox.

Click the Next button to advance to the **Configure Server Distribution Component** step.

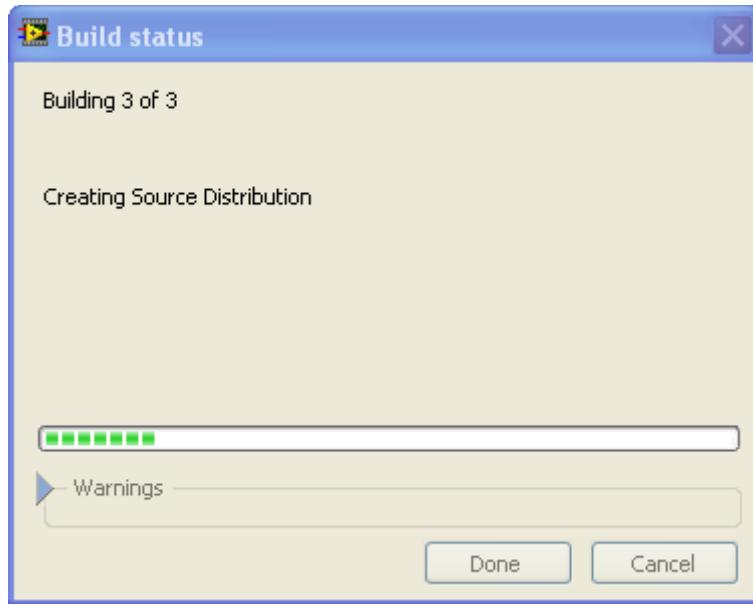


Leave the default options and click **Next**.

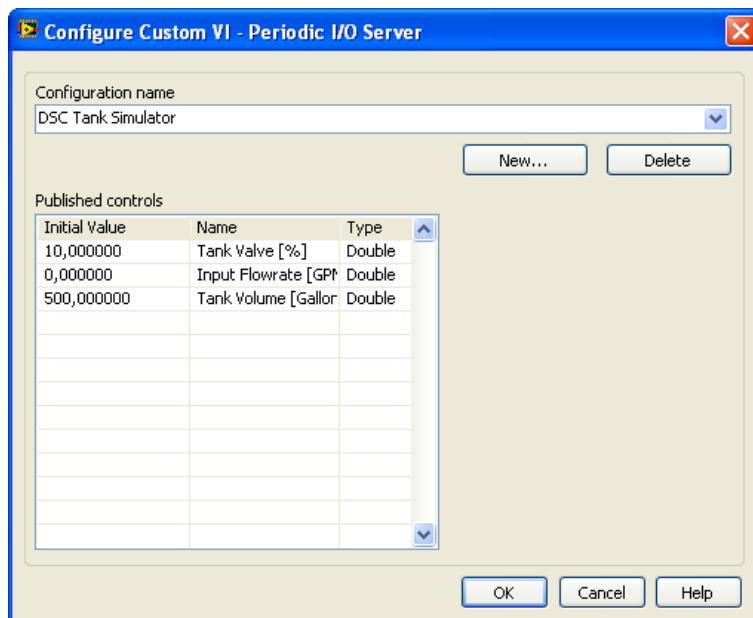
The **Server Distribution Component** page appears



The DSC Module displays a summary of the files that the Custom VI-based Server – Periodic Wizard will create from the **Server Distribution Component** page. Click the **Build** button. The wizard displays the **Build status** dialog box as it creates a VI template file, a registration VI, and a support DLL and VIs.

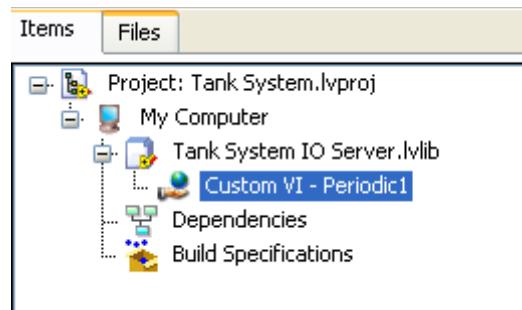


After the wizard creates the periodic I/O server, the **Configure Custom VI – Periodic I/O Server** dialog box appears with the name of the periodic I/O server and the data items it contains.

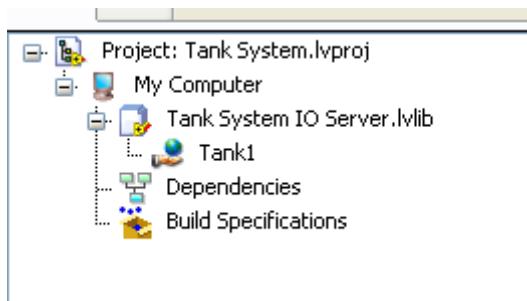


Click the OK button.

LabVIEW adds the periodic I/O server to the **Tank System IO Server** project library.



Right-click the **Custom VI – Periodic1** item in the **Project Explorer** window and select **Rename** from the shortcut menu. Rename the periodic I/O server “Tank1”. The **Project Explorer** window appears as shown below.



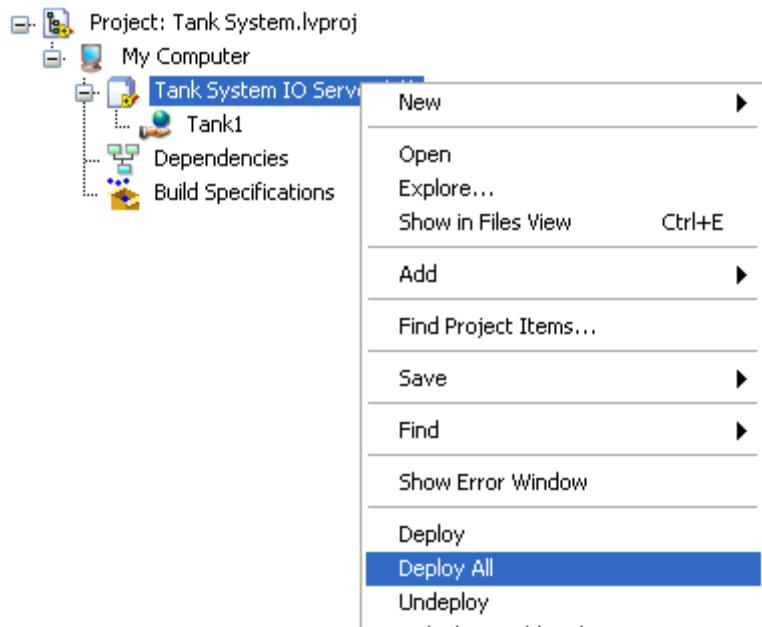
11.6.4 Task 4: Deploying the Periodic I/O Server

Now you must deploy the periodic I/O server so that the data items in the I/O server are available for use in other VIs and across the network. In this exercise you will deploy the periodic I/O server and view the I/O server data in the Shared Variable Monitor.

Note! The periodic I/O server runs continuously in the background until you **undeploy** the library in the Project Explorer window that contains the I/O server.

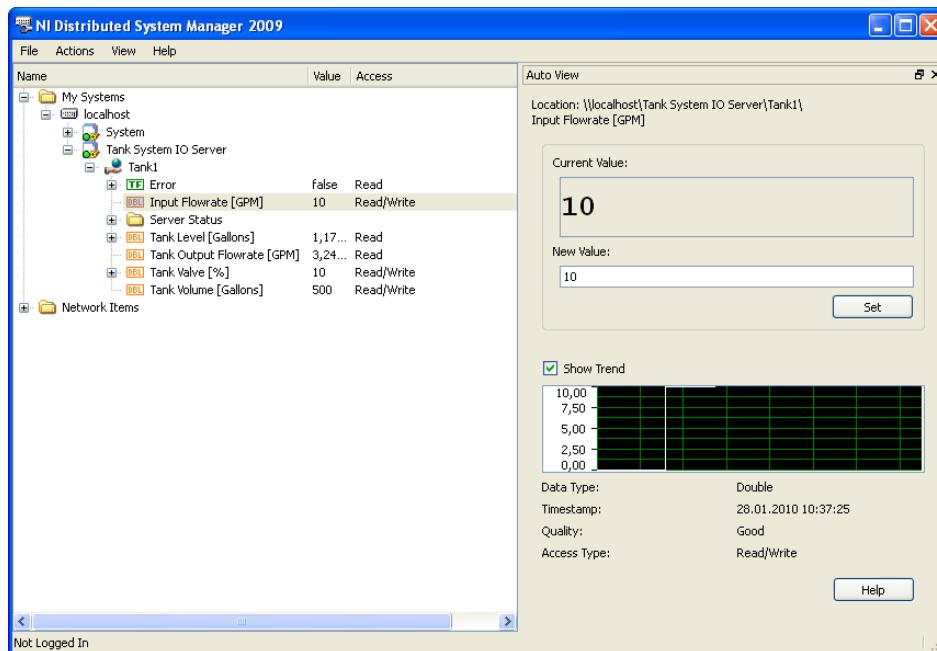
Complete the following steps to deploy the Tank1 periodic I/O server and view the data.

Right-click the **Tank System IO Server.lvlib** project library under the **My Computer** item and select **Deploy All** from the shortcut menu to deploy the project library.



Click the **Done** button to close the **Deploy** dialog box when the deployment is complete.

Select **Tools**→**Distributed System Manager**.



Expand **Tank System IO Server** → **Tank1** in the left pane. Notice that the controls and indicators of the I/O server appear under **Tank1**. Because you have deployed the project library, the I/O server is running and each control and indicator is an I/O data item.

Click the **Tank1.Input Flow Rate [GPM]** control and enter a value of 10 and click the OK button. Notice that the values of **Tank1.Tank Level [Gallons]** and **Tank1.Tank Output Flowrate [GPM]** begin increasing.

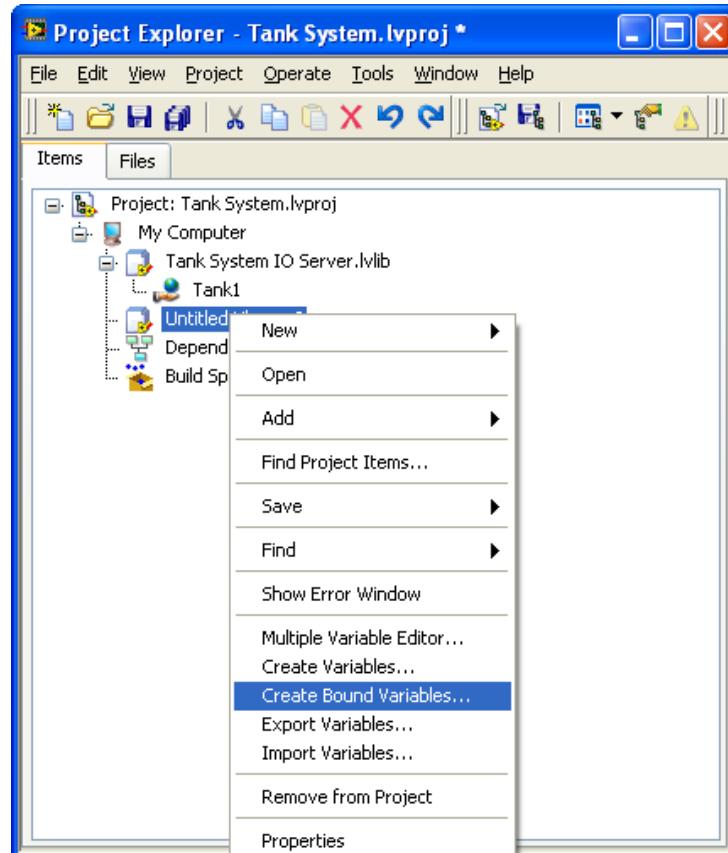
Close **Distributed System Manager**. The periodic I/O server continues to run.

11.6.5 Task 5: Creating Shared Variables

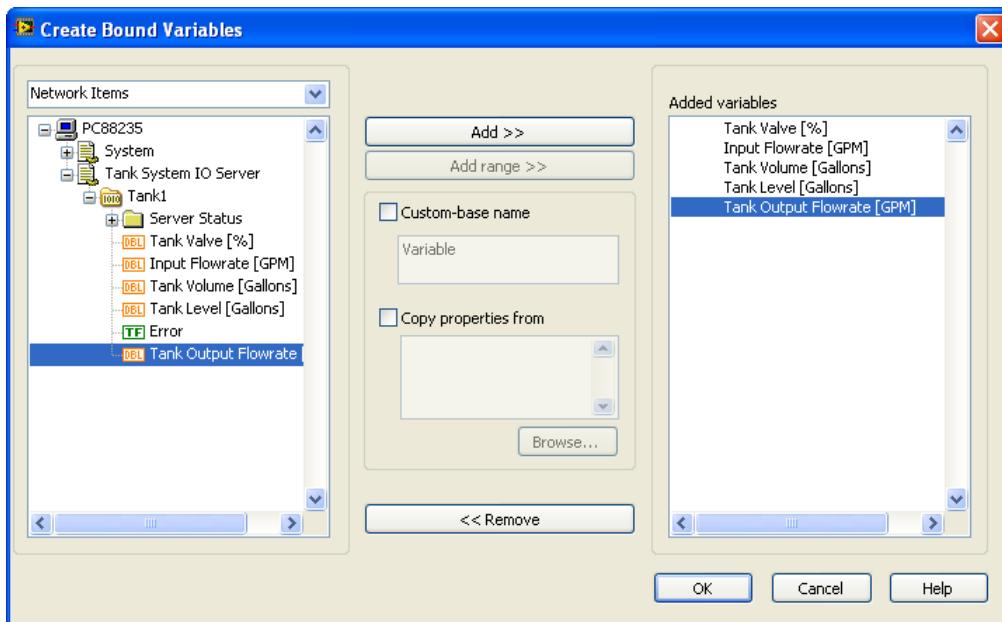
In this exercise you will add the network-published shared variables that represent the data items in the periodic I/O server to the Tank System Shared Variables project library.

Complete the following steps to add the Tank System Shared Variables project library to the Tank System project.

Right-click **My Computer** in the **Project Explorer** window and select **New → Library** from the shortcut menu.



Right-click the new project library you created and select **Create Bound Variables** from the shortcut menu to display the **Create Bound Variables** dialog box.



Select **Network Items**.

Expand **Tank System IO Server → Tank1** in the Network tree. The shared variables appear under **Tank1**.

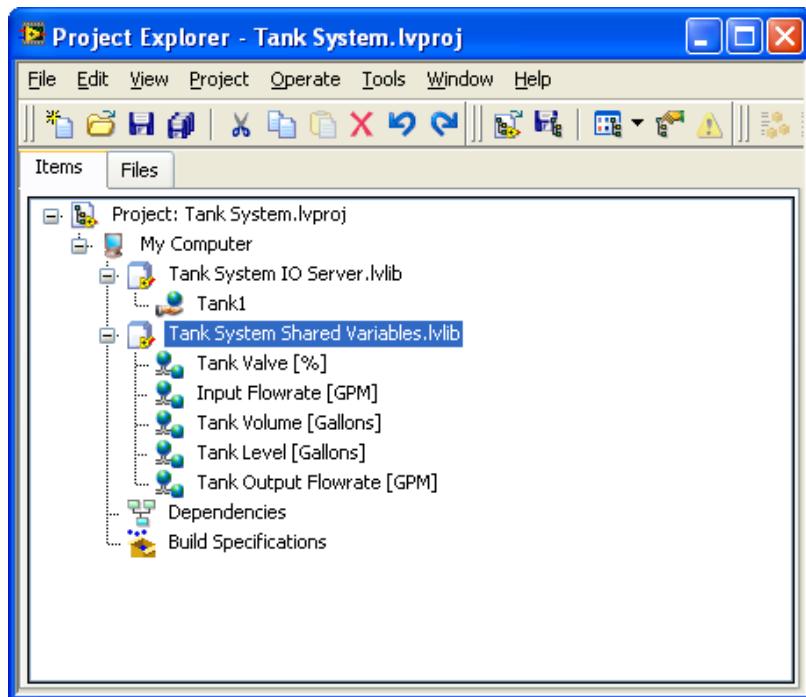
Select each shared variable with the data type **DBL** and click the **Add** button to add each variable to the **Added variables** list. Click the **OK** button. The **Create Bound Variables** dialog box closes and the shared variables appear in the **Multiple Variable Editor** window.

	Path	Name	Var Type	Data Type	Network-Published: Buffering	N
Tank Valve [%]	...vproj/My Computer/Untitled Library 2/	Tank Valve [%]	Network-Published	Double	<input checked="" type="checkbox"/>	50
Input Flowrate [GPM]	...vproj/My Computer/Untitled Library 2/	Input Flowrat...	Network-Published	Double	<input checked="" type="checkbox"/>	50
Tank Volume [Gallons]	...vproj/My Computer/Untitled Library 2/	Tank Volume [...]	Network-Published	Double	<input checked="" type="checkbox"/>	50
Tank Level [Gallons]	...vproj/My Computer/Untitled Library 2/	Tank Level [G...	Network-Published	Double	<input checked="" type="checkbox"/>	50
Tank Output Flowrate [GPM]	...vproj/My Computer/Untitled Library 2/	Tank Output F...	Network-Published	Double	<input checked="" type="checkbox"/>	50

Click the **Done** button to close the **Multiple Variable Editor** window.

Select **File → Save All** in the **Project Explorer**. The **Name the Library** dialog box appears.

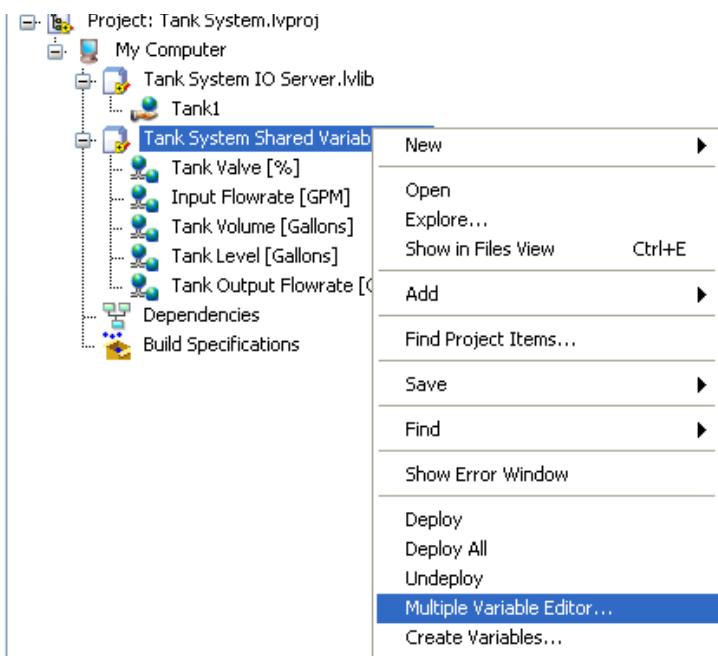
Enter “Tank System Shared Variables” in the File name text box. Click the OK button. LabVIEW binds the shared variables in the **Tank System Shared Variables** project library to the corresponding items on the network.



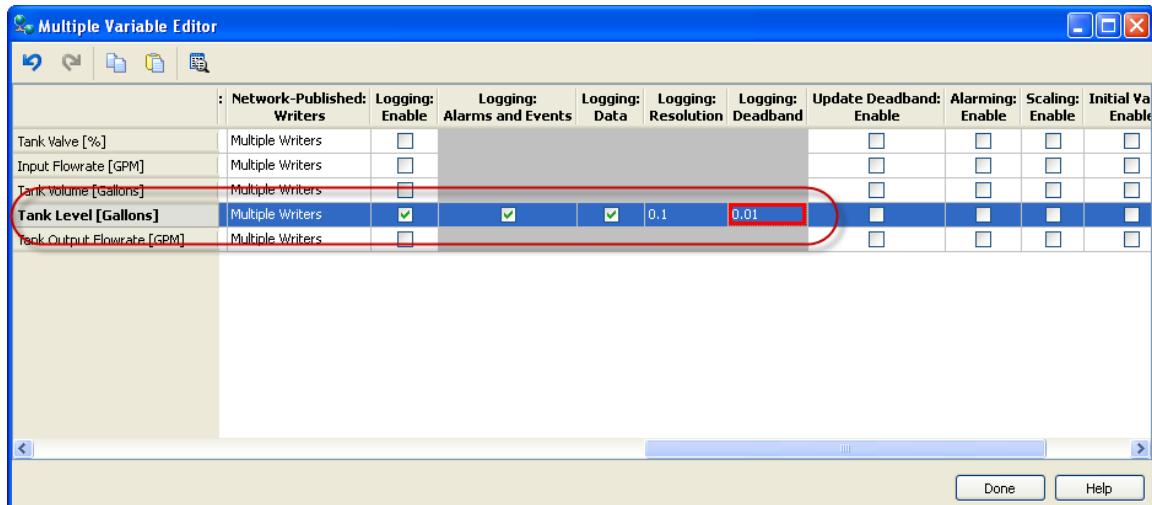
11.6.6 Task 6: Configuring Data Logging

When you add logging to a **shared variable**, the DSC Module logs shared variable data, including the shared variable value, timestamp, whether the value is in an alarm state, and the quality of the value. The DSC Module logs all data to the **Citadel** database. Complete the following steps to add logging for the **Tank Level [Gallons]** shared variable.

Right-click the **Tank System Shared Variables.lvlib** project library and select **Multiple Variable Editor**.



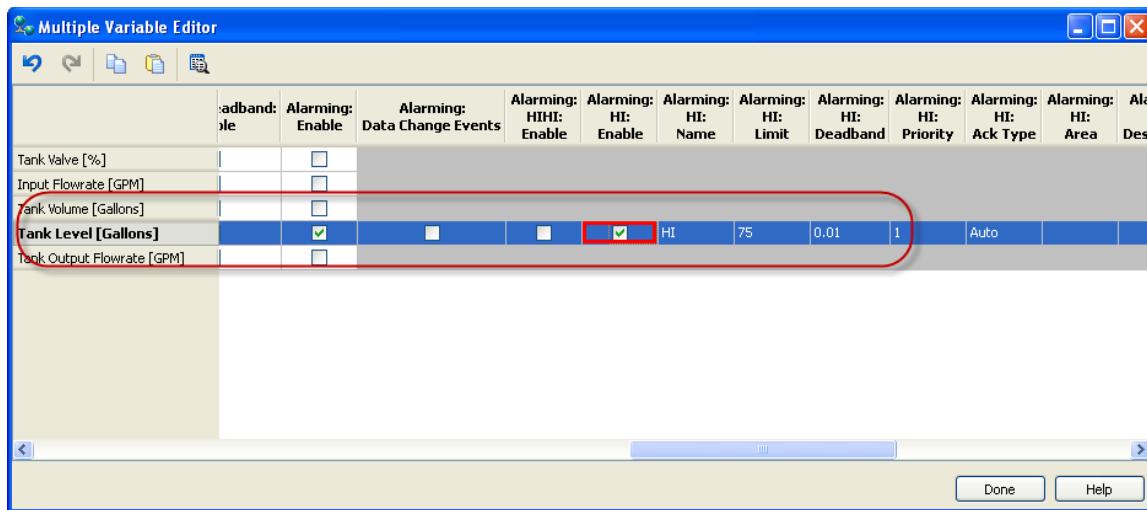
Set the following for the **Tank Level [Gallons]** shared variable in the **Multiple Variable Editor**:



11.6.7 Task 7: Configuring Alarming

An alarm is an abnormal condition on a shared variable or a user-defined condition. **An alarm occurs if a shared variable value goes out of its defined alarm limits or if a shared variable has bad status.** In this exercise you will add an alarm for the **Tank Level [Gallons]** shared variable.

Set the following for the **Tank Level [Gallons]** shared variable in the **Multiple Variable Editor**:

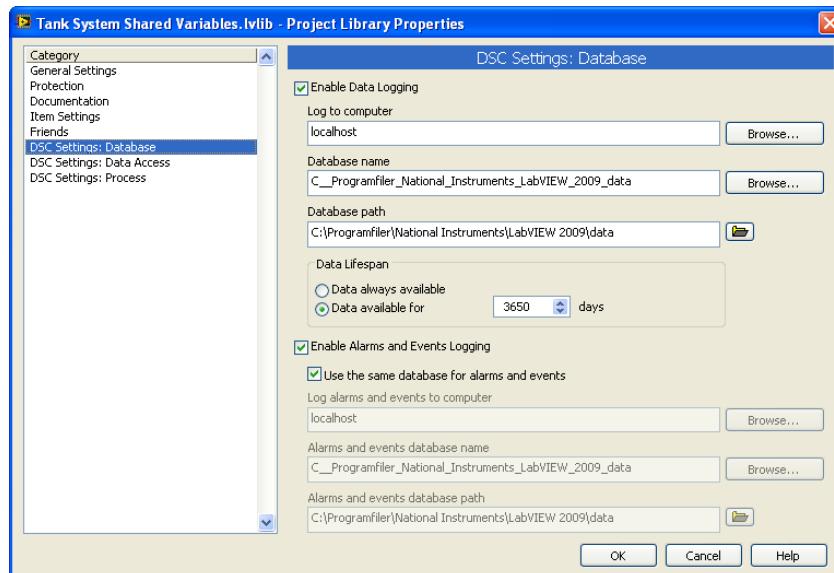


11.6.8 Task 8: Enabling Logging

Complete the following steps to enable data logging and alarm and event logging for the Tank System Shared Variables project library.

Right-click the **Tank System Shared Variables.lvlib** project library in the **Project Explorer** window and select **Properties** from the shortcut menu. The **Project Library Properties** dialog box appears.

Select **DSC Settings: Database** from the Category list. Verify that the options in the DSC Settings: Database page appear similar to the Figure below.



The **Enable Data Logging** option turns on data logging for the project library. **localhost** specifies the local computer. Use localhost instead of the name of the computer to reduce the changes you must make if you move this project to another computer. The **Enable Alarms and Events Logging** option turns on event logging for the project library. The **Use the same database for alarms and events**

option ensures that the DSC Module logs alarms and events for this project library to the same database that it logs data.

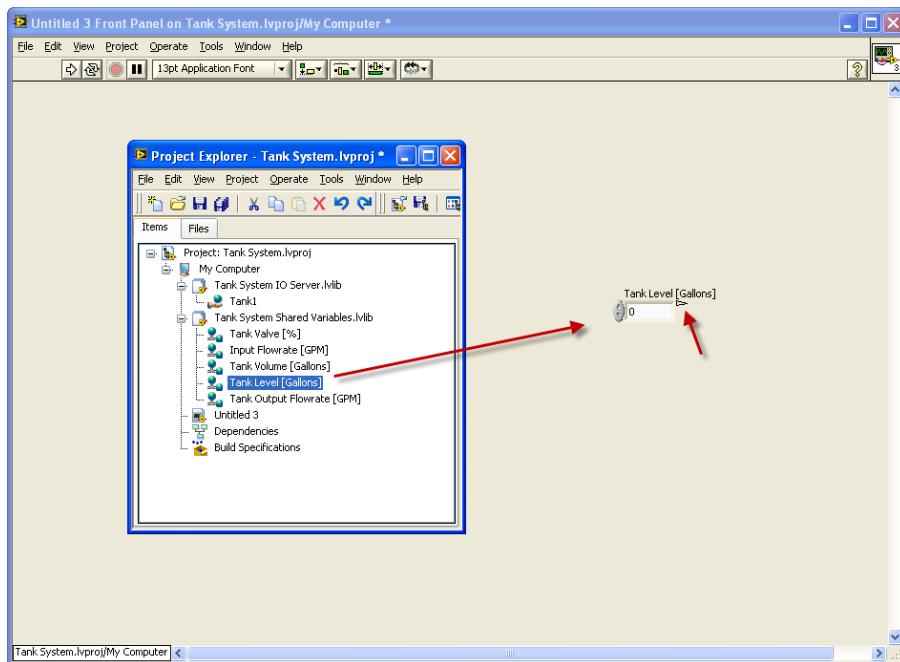
You can change the **Database** name that appears in the **Project Library Properties** dialog box to a more useful or descriptive name.

11.6.9 Task 9: Create the LabVIEW application

In this exercise you will create a VI to display data on a front panel. You do not need to add any code to the block diagram. Complete the following steps to create a front panel to display the data items in the periodic I/O server.

Right-click **My Computer** in the Project Explorer window and select **New→VI** from the shortcut menu. A new VI front panel and block diagram appear.

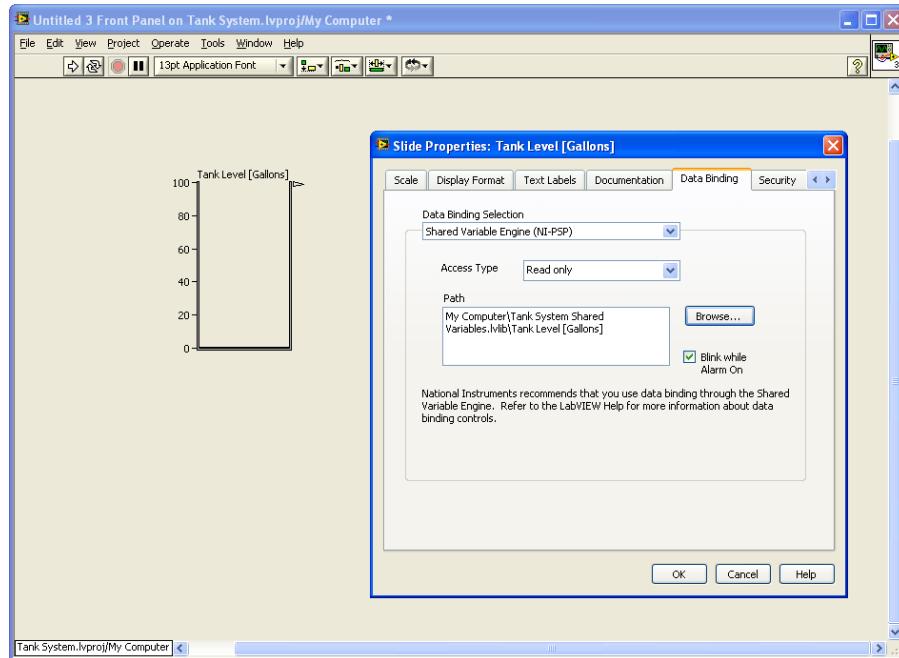
Select the **Tank Level [Gallons]** shared variable from the Tank System Shared Variables.lvlib project library in the **Project Explorer** window and drag the shared variable onto the front panel. The shared variable appears as a numeric control. Notice the triangle that appears next to the control. The triangle indicates that this control has been configured for data binding.



Right-click the **Tank Level [Gallons]** control and select **Change to Indicator** from the shortcut menu.

Right-click the **Tank Level [Gallons]** indicator and select **Replace → DSC Module → Vessels → Open Tank** from the shortcut menu.

Right-click the **Tank Level [Gallons]** indicator and select Properties from the shortcut menu. The Properties dialog box appears. Select the **Data Binding** tab in the **Properties** dialog box.



Place a checkmark in the **Blink while Alarm On** checkbox to configure the control to blink when the water level reaches **75**, the default alarm you set in the Configuring Alarming section of this document.

For the **Input Flowrate [GPM]** shared variable:

- Select the **Input Flowrate [GPM]** shared variable from the **Tank System Shared Variables.lvlib** project library in the **Project Explorer** window and drag the shared variable onto the front panel.
- Right-click the **Input Flowrate [GPM]** control and select **Replace → Num Ctrl → Pointer Slide**.

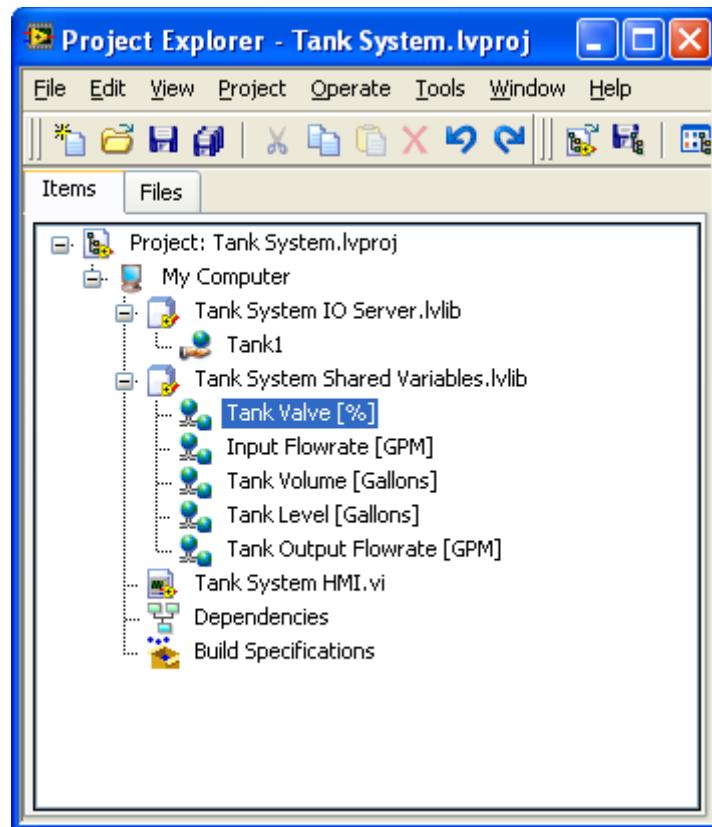
For the **Tank Valve [%]** shared variable:

- Select the **Tank Valve [%]** shared variable from the **Tank System Shared Variables.lvlib** project library in the Project Explorer window and drag the shared variable onto the front panel.
- Right-click the **Tank Valve [%]** control and select **Replace → Num Ctrl → Pointer Slide**.

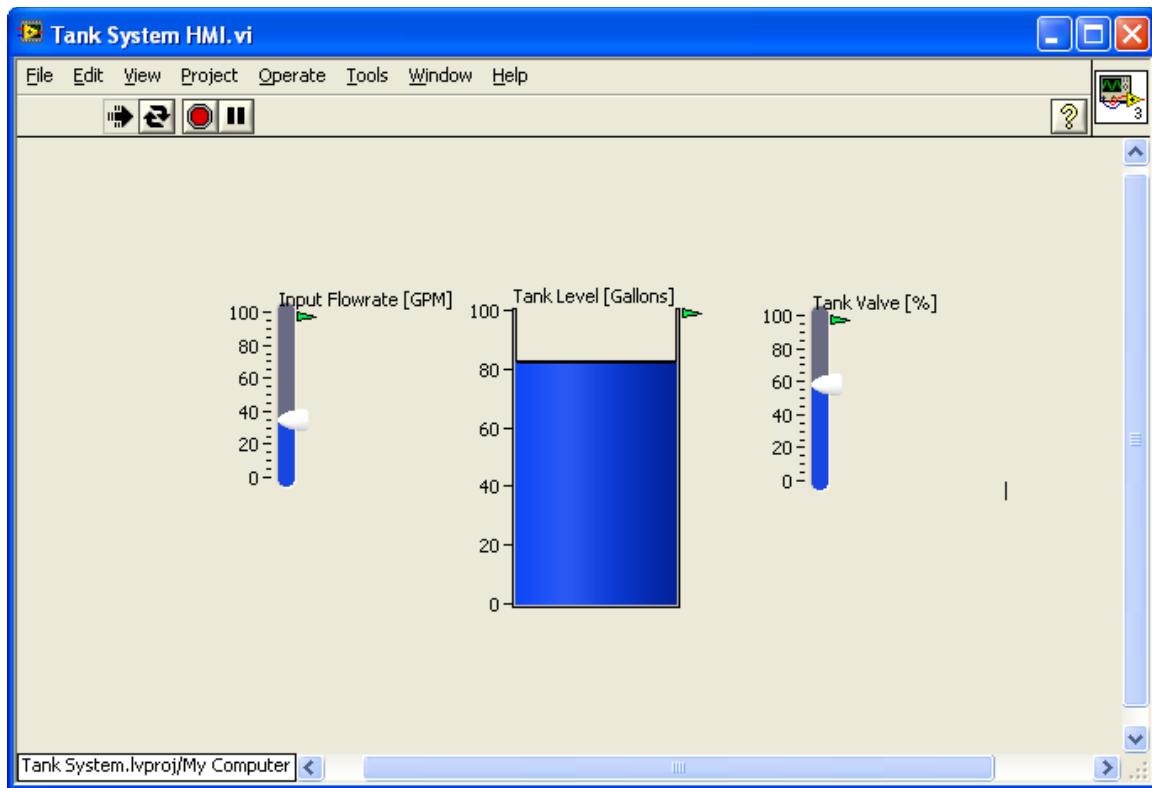
Select **File → Save As**. The Name the VI dialog box appears. Enter “**Tank System HMI**” in the File name text box. Click the **OK** button.

Click the **Run Continuously** button to run the VI. The VI should function in the same way as the example you ran in the first exercise.

Below we see the final **Project Explorer**:



Below we see the final application:

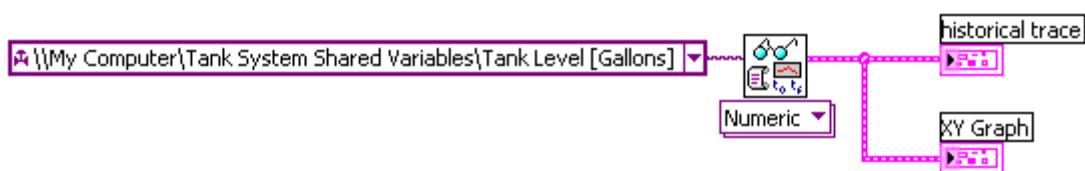


11.6.10 Task 10: Use the LabVIEW DSC Functions and VIs

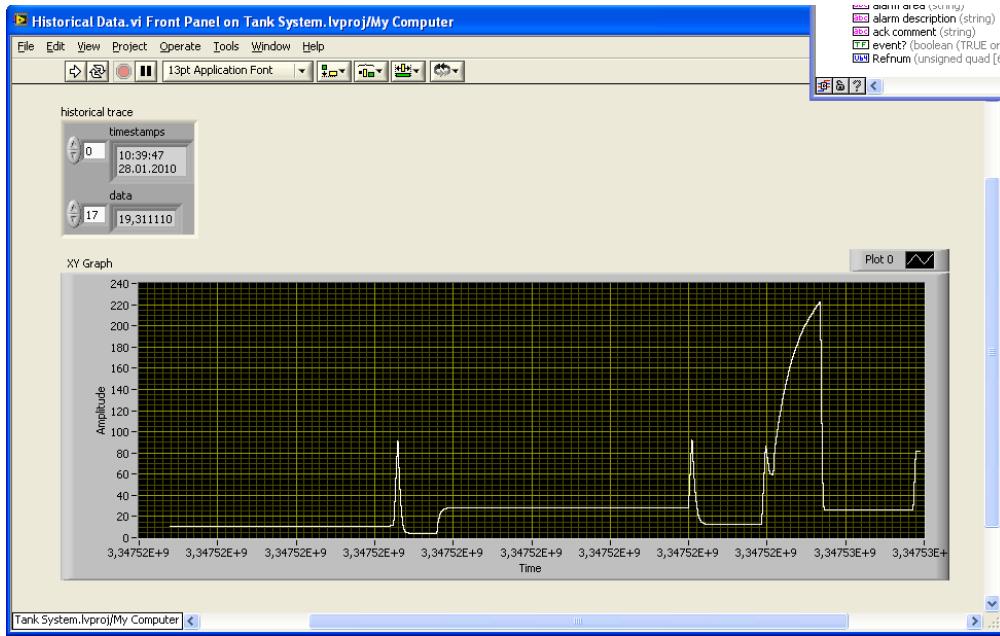
In this task you will use the LabVIEW DSC Functions and VIs palette.

Read Historical Data:

Block Diagram:



Front Panel:

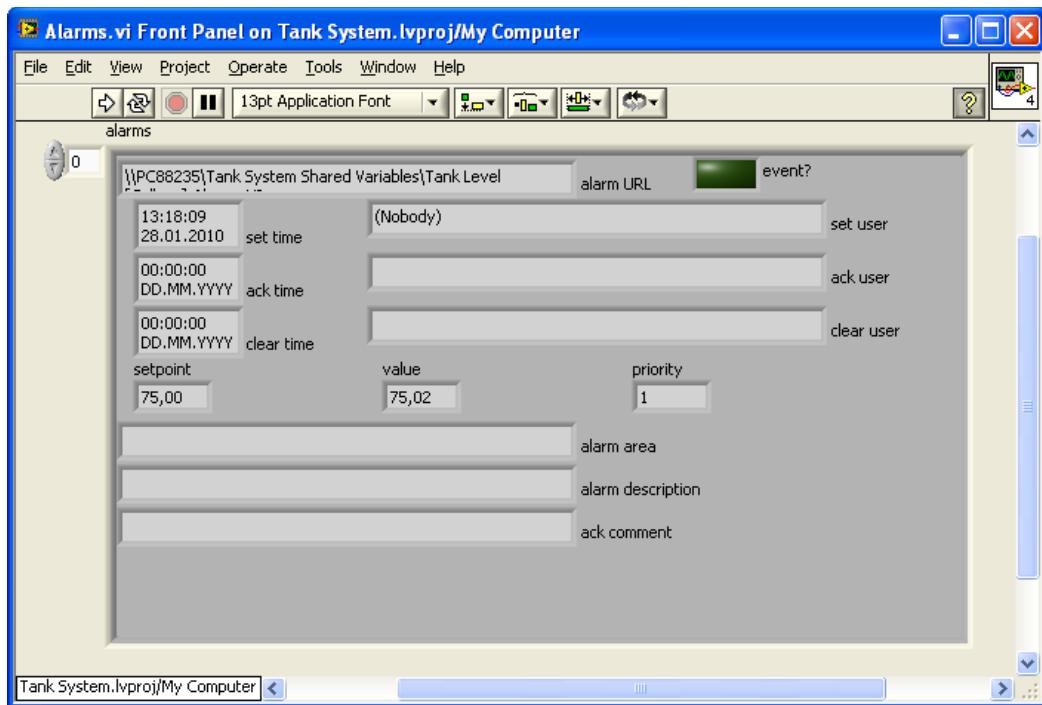


Read Alarm Data:

Block Diagram:

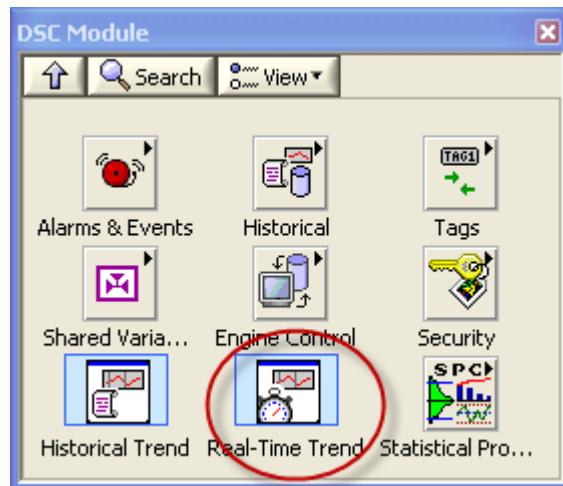


Front Panel:

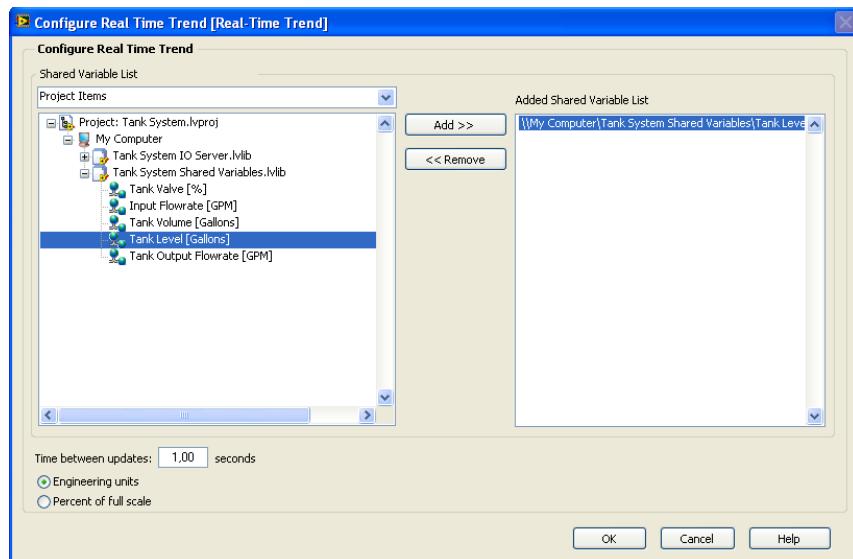


11.6.11 Task 11: Viewing Real-Time Data

You can use the DSC Module to view live data. The **Real-Time Trend** Express VI displays live data from a shared variable on an XY graph. In the following exercise you will add the ability to view live data to the Tank System VI.

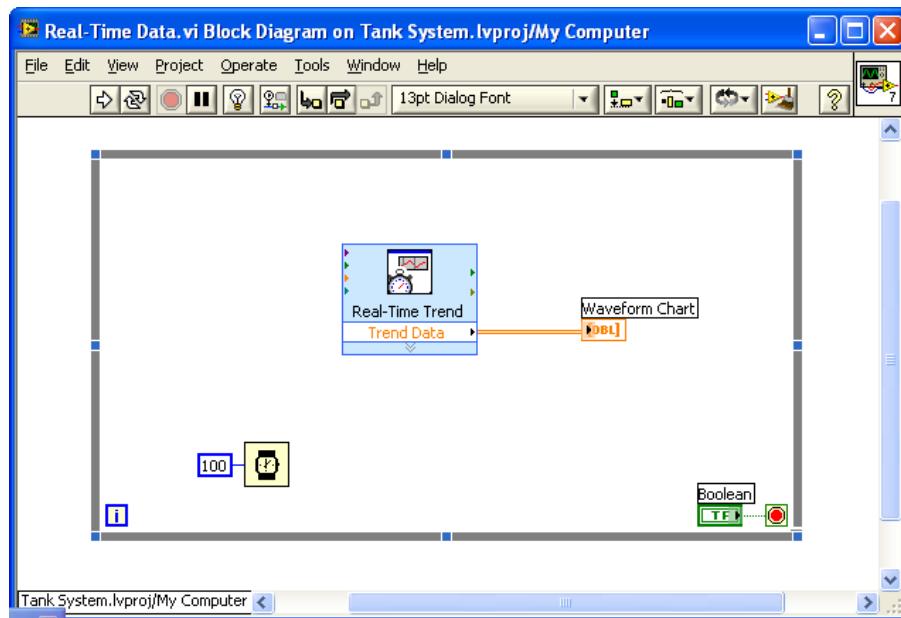


Place the **Real-Time Trend** Express VI, available on the DSC Module palette, on the block diagram. The **Configure Real-Time Trend** dialog box appears.

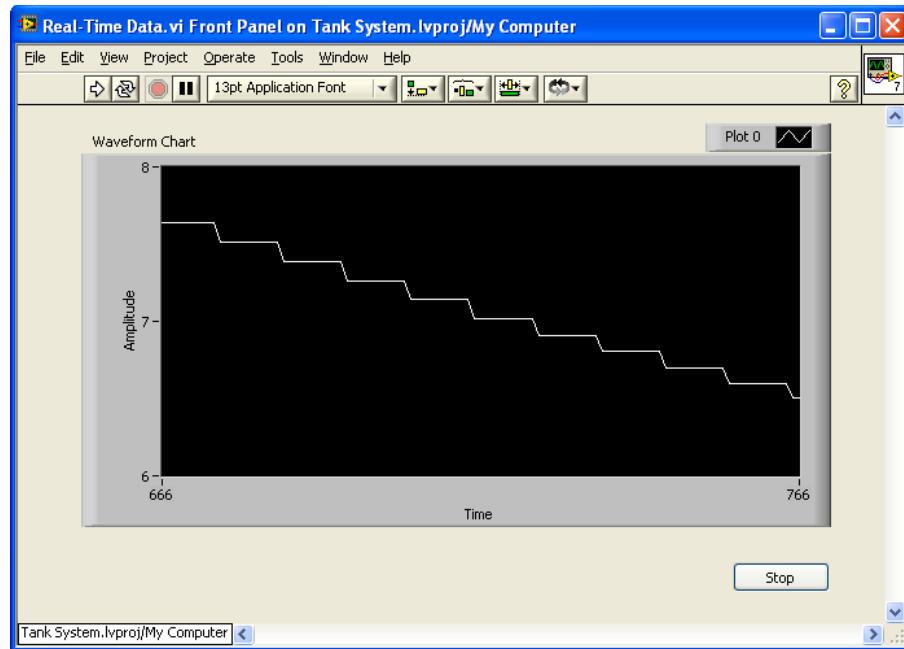


Select the **Tank Level [Gallons]** shared variable and select **Add**.

Place a **waveform** chart on the front panel. Wire the trend data output of the **Real-Time Trend** Express VI to the waveform chart on the block diagram. Place a While Loop around the **waveform** chart and Express Real-Time Trend control. Run the VI.

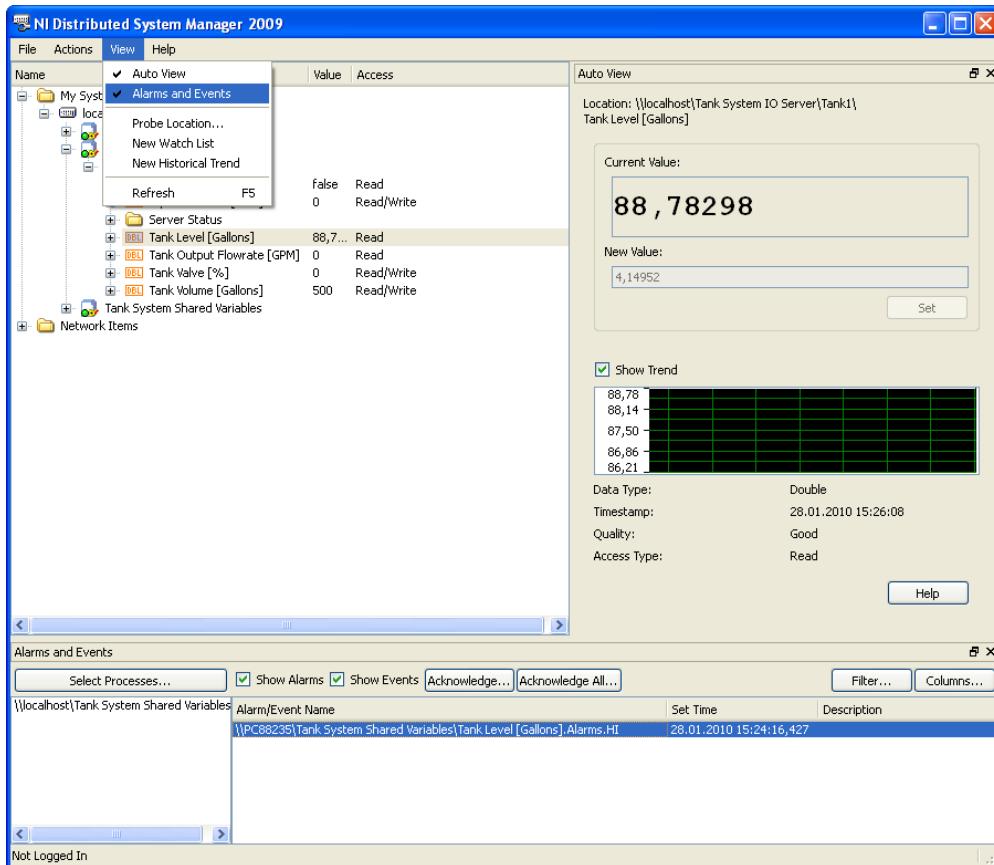


Front Panel



11.6.12 Task 12: Viewing Alarms using the Distributed System Manager

Open the Distributed System Manager from Tools → Distributed System Manager.



12 LabVIEW I/O Server

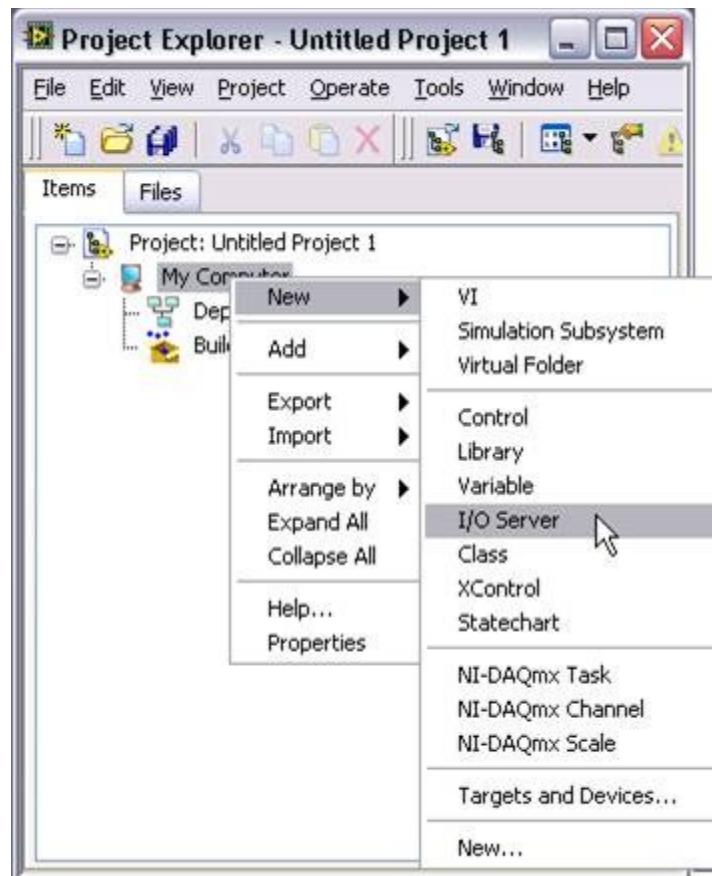
12.1 Connect LabVIEW to OPC Tags by Creating an I/O Server

In this section, create a LabVIEW interface to the OPC tags called an I/O Server. The I/O Server automatically updates LabVIEW with the current tag values at a rate you specify.

In the **Getting Started** window of LabVIEW, click “File → New Project”. This opens a new LabVIEW Project.

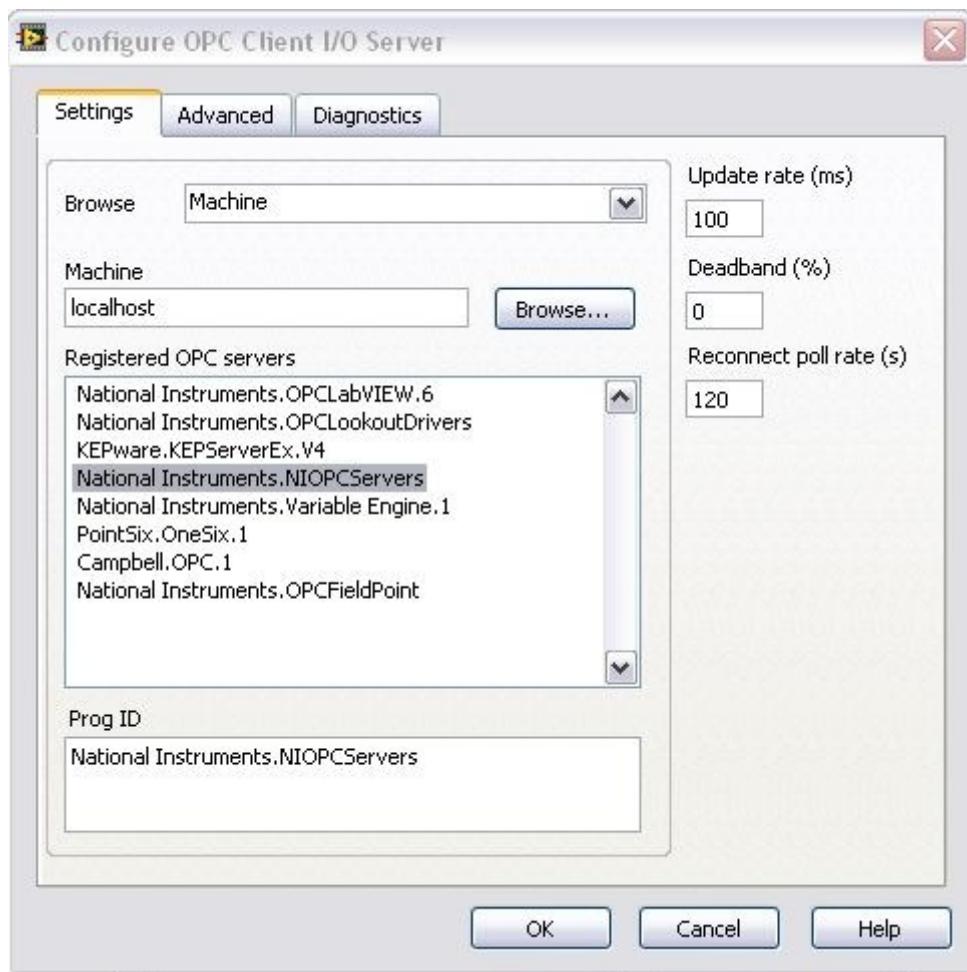
If the **Context Help** window is not visible, press “Ctrl+H” to display the window. Keep this window open for helpful information about items under your cursor.

In the LabVIEW Project window, right-click **My Computer** and select “New → I/O Server”, as shown in the Figure below.



Select **OPC Client** in the “Create New I/O Server” window and click **Continue**.

Choose “**National Instruments.NIOPCServers**” from the Registered OPC servers field and set **Update rate (ms)** to 100. This creates a connection from LabVIEW to the OPC tags, which updates every 100 ms.



Select **OK**. A library is automatically created in your project explorer window to manage the I/O Server.

Save the project as “OPCDemoProject” and the library as “OPCDemoLibrary” by selecting “File → Save All” from the project explorer window.

12.2 Create Shared Variables that Connect to the OPC Tags through the I/O Server

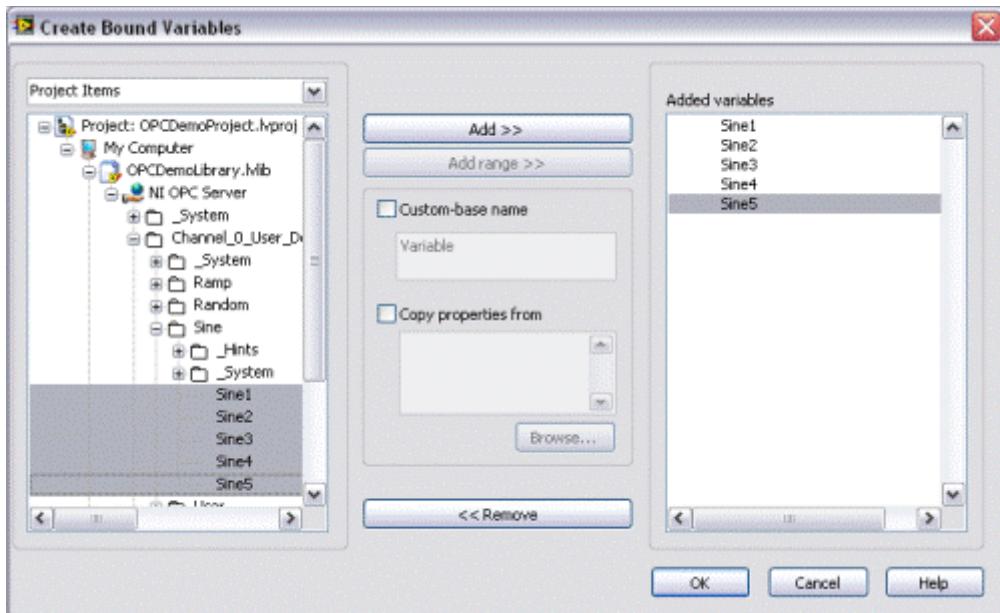
In this section, create shared variables, which are bound to the OPC tags, giving you native access in LabVIEW to OPC data. With the shared variable, you can share data across LabVIEW applications on a single computer or across the network.

Create new shared variables that are bound to the OPC tags.

In the LabVIEW Project window, right-click My Computer and select “**New → Library**”. This creates a new library for the shared variables, which are used to connect to the PLCs’ OPC tags.

Right-click the newly created library and select “Create Bound Variables...”.

In the Create Bound Variables window, select the OPC tags to bind the shared variables to by browsing down to the simulated sine data from the OPC server as shown in the Figure below.



Select all the sine items and click **Add** and **OK**. This creates shared variables that are bound to the OPC tags and loads them into the Multiple Variable Editor.

In the Multiple Variable Editor, select **Done**. This adds the new shared variables to the library that was created earlier.

Note! The LabVIEW DSC Module enhances shared variables by adding the ability to log data, alarms, and events directly to a database without ever writing a LabVIEW application.

Save the new library as “OPCItems.lvlib” in the project explorer window by right-clicking the library and selecting **Save As**.

Deploy the shared variables by right-clicking the “OPCItems” library and selecting **Deploy**. This publishes the shared variables, making them available on the network to other computers, OPC clients, and the LabVIEW Real-Time PAC.

You now have access to OPC data natively in LabVIEW through the shared variables.

12.3 Viewing Shared Variables with Distributed System Manager

From the Project Explorer window, select “**Tools → Distributed System Manager**”. This opens a window that you can use to manage your shared variables in various ways (view, deploy, undeploy, etc.).

In the Tree pane of the Variable Manager, expand the “localhost” item under the “My Systems” category. Right-click the “**OPCItems**” library, and select “**Watch List**” to display the shared variables, which are bound to the PLCs’ OPC tags.

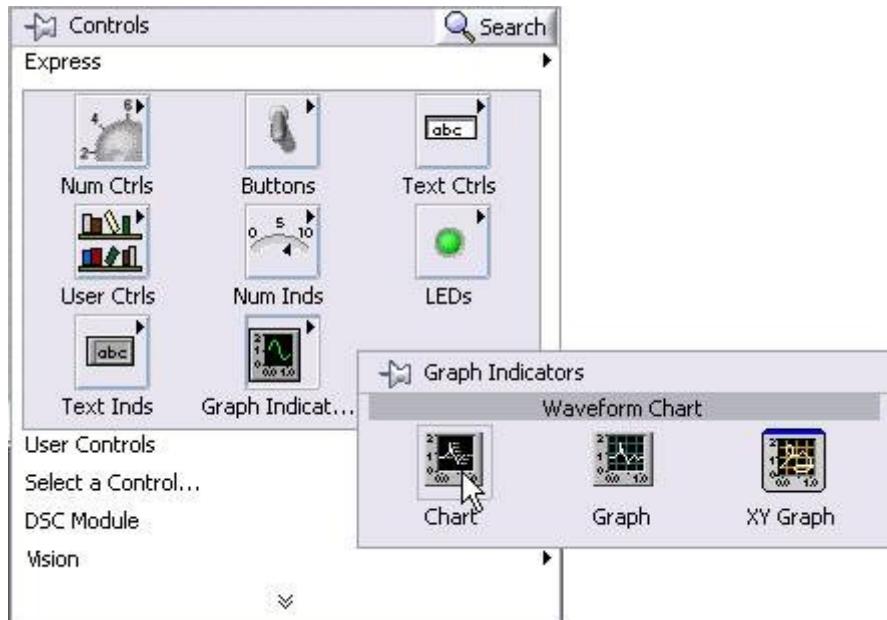
The shared variables will be updating with the simulated sine data.

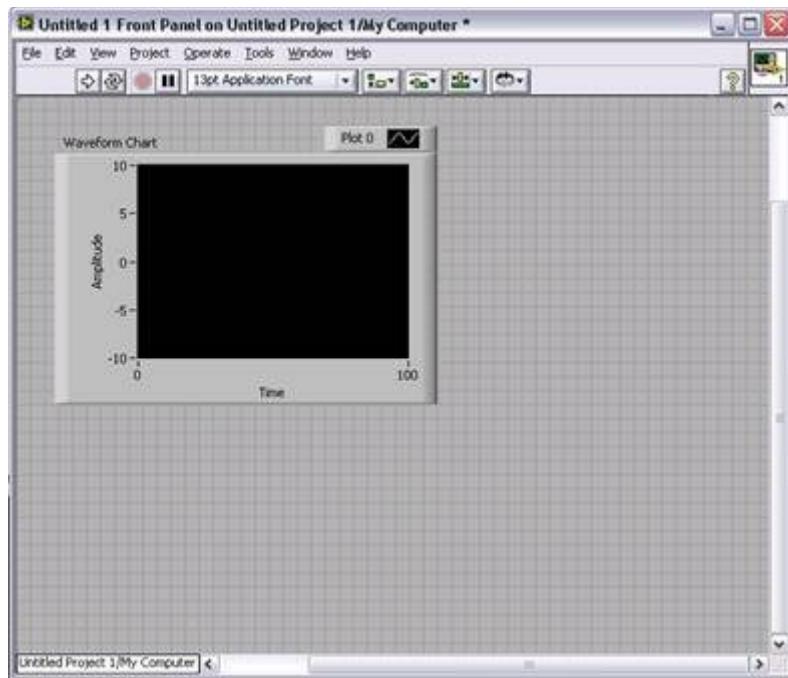
12.4 Using OPC Tag Data in LabVIEW

From the project explorer, right-click My Computer and select “**New → VI**”. This creates a new virtual instrument or VI. A VI is used to create a user interface and executable graphical code.

By default, you see the Front Panel, which is the user interface of the VI. Select “**View → Controls Palette**” or right-click anywhere on the Front Panel to bring up the Controls palette.

Select a waveform chart from the Controls palette by selecting “**Express → Graph Indicators → Chart**”, and place it on the Front Panel, as shown in the Figure below.





In the VI, select “**Window → Show Block Diagram**” or press **Ctrl+E** to show the Block Diagram. The Block Diagram is where you build the behavior of your application. Notice the icon on the Block Diagram, which represents the chart on the Front Panel. By passing data into this terminal, you can display it in the chart on the Front Panel.

In the project explorer, expand the “**OPCItems**” library and select the **Sine1** shared variable.

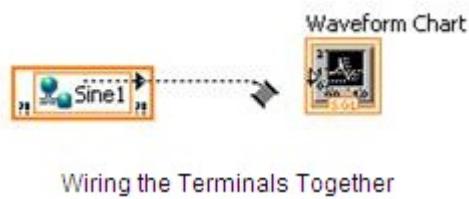
Drag and drop the **Sine1** shared variable from the project explorer to the Block Diagram of the VI. The shared variable acts as a source of data to other terminals on the Block Diagram.

Select “**View → Tools Palette**” or press Shift+right-click to show the Tools palette, which contains various tools for building the Block Diagram. By default you use the **Automatic Tool Selection** tool, which selects the appropriate tool based on the location of the cursor.

Select the **Connect Wire** tool as shown in the Figure below. This tool is used to wire terminals together on the Block Diagram.



Use the Connect Wire tool to wire the **Sine1** shared variable to the **waveform chart** by clicking on the Sine1 shared variable and then on the waveform chart, as shown in the Figure below.



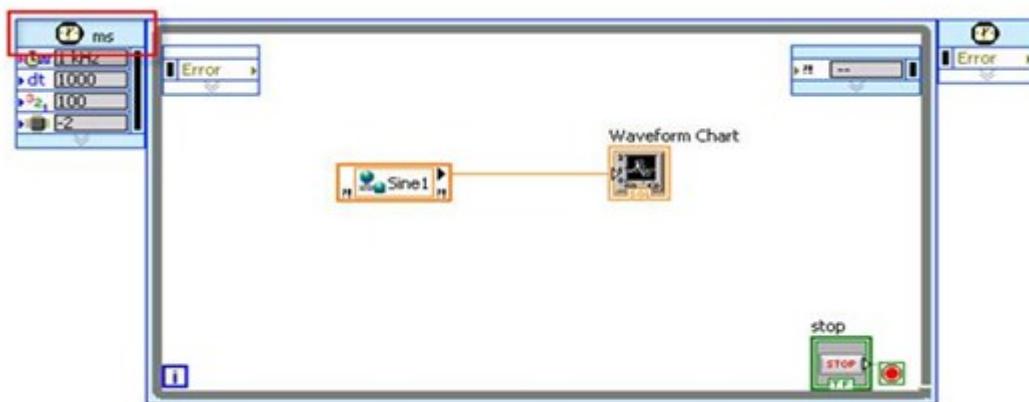
Open the Functions palette by selecting “**View → Functions Palette**” or right-clicking anywhere on the Block Diagram. The Functions palette contains hundreds of analysis functions, control functions, and structures for graphical programming.

Select a while loop from the Functions palette by navigating to “**Express → Execution Control → While Loop**”. This allows you to wrap a while loop around a section of code.

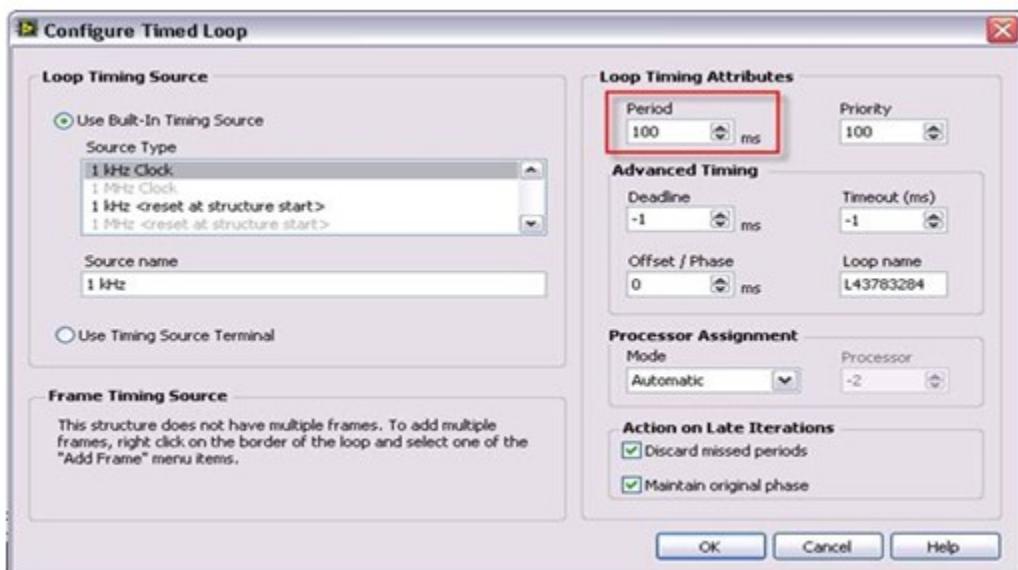
The while loop causes the code within it to execute continuously until stopped by the user or additional logic in the VI.

A timed loop, which is an advanced while loop, contains additional configuration options for timing and execution control. Convert the while loop into a timed loop by right-clicking the **while loop** and selecting **Replace with Timed Loop**.

To configure the timed loop, double-click the timed loop input node

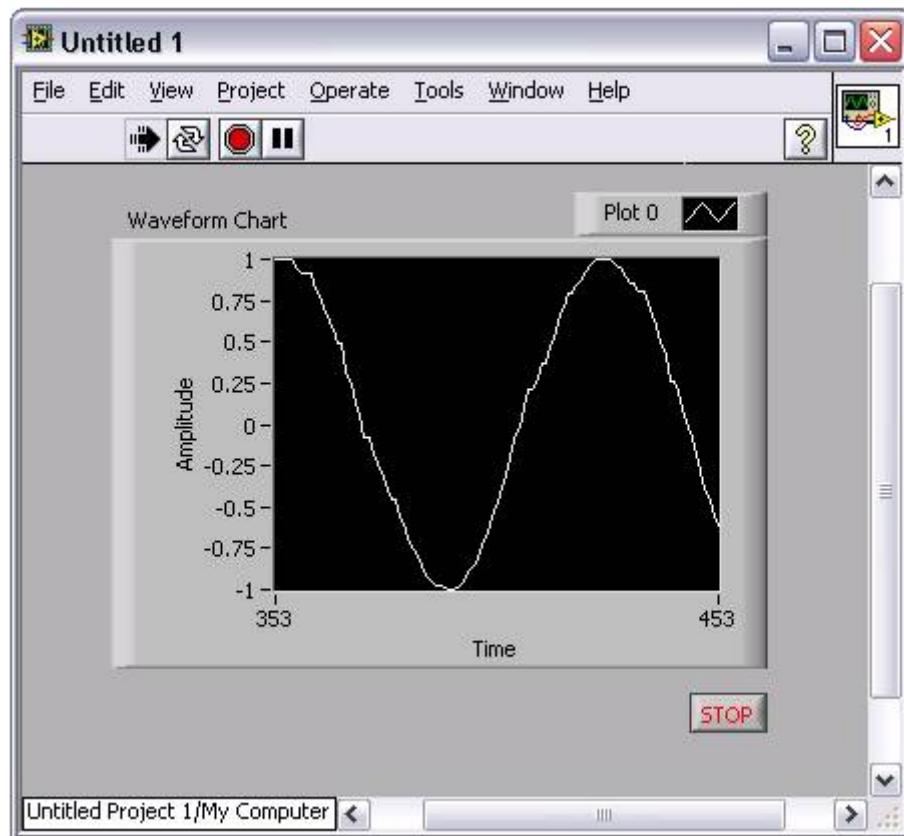


In the Loop Timing Attributes field, set **Period** to 100 ms and click **OK**. This configures the timed loop to execute the contained code every 100 ms.



Click the **Run** button on the toolbar to execute the VI.

Click **Close** on the **Deploy...** window once the deployment completes. When the application begins executing, you see the Sine1 sine wave displayed on the waveform chart.

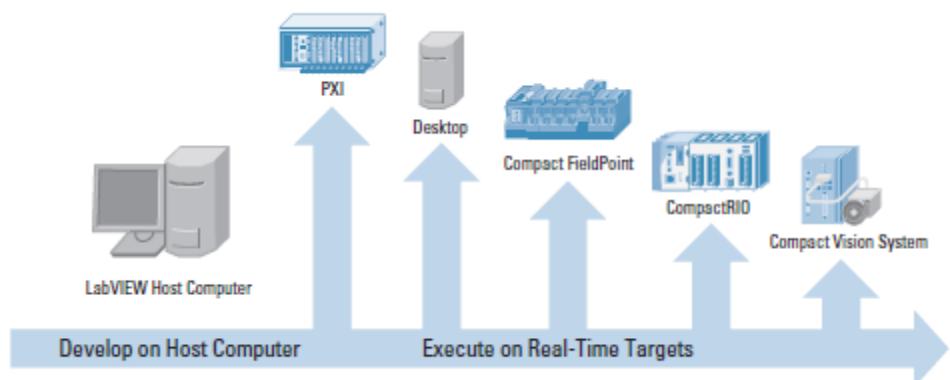


Congratulations! You successfully accessed PLC data in your LabVIEW application, so you can incorporate powerful analysis and control functions in your solution.

13 LabVIEW Real-Time Module

The National Instruments LabVIEW Real-Time Module extends the LabVIEW development environment to deliver deterministic, real-time performance. Develop your application on a host computer using graphical programming and then download the application to run on an independent hardware target with a real-time OS (RTOS).

The National Instruments LabVIEW Real-Time Module is an add-on component for the LabVIEW Development System. When installed, this software compiles NI LabVIEW graphical code and optimizes it for the selected real-time target. Using the LabVIEW Real-Time Module, you can develop and deploy applications to all NI real-time hardware targets including PXI, Compact FieldPoint, FieldPoint, CompactRIO, and standard desktop PCs. The embedded RTOS (Real-Time Operating System) for these targets is a single dedicated kernel that provides maximum reliability for embedded code.

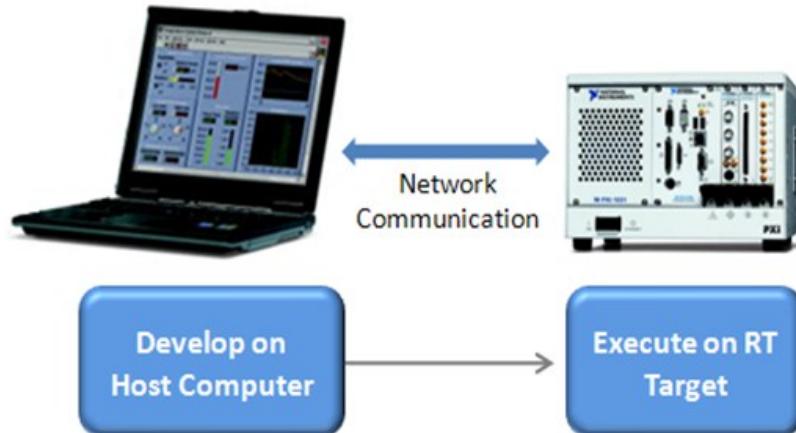


With the LabVIEW Professional Development System and LabVIEW Real-Time Module, you can create a stand-alone executable and download it to a hardware target with one simple step. You can permanently embed the code on the nonvolatile memory of the real-time system so it starts automatically when the system boots for autonomous field applications.

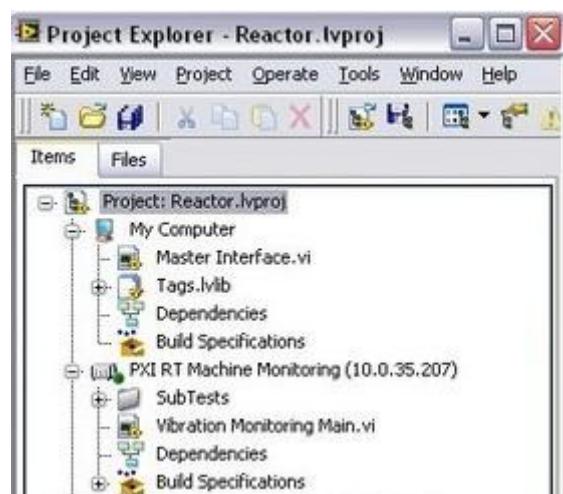
Each hardware target contains an embedded processor running an RTOS. The LabVIEW Real-Time Module embeds compiled code on the hardware target and runs it independently of the host computer. Assign the appropriate execution priority to each real-time task. The embedded RTOS then uses a combination of round-robin and preemptive scheduling to ensure deterministic execution of your time-critical tasks.

13.1 Real-Time Development in LabVIEW

Developing real-time programs in LabVIEW is nearly identical to developing standard LabVIEW applications for your PC. When developing real-time (RT) programs in LabVIEW, you develop on a host Windows computer, and then download and run them on a real-time hardware target.



When developing real-time applications in LabVIEW, you use the LabVIEW **Project Explorer** to organize your programs (VIs) and categorize them by the hardware platform that they will run on. Note that you develop your code on a general-purpose Windows computer, and then connect to your real-time hardware using Ethernet.

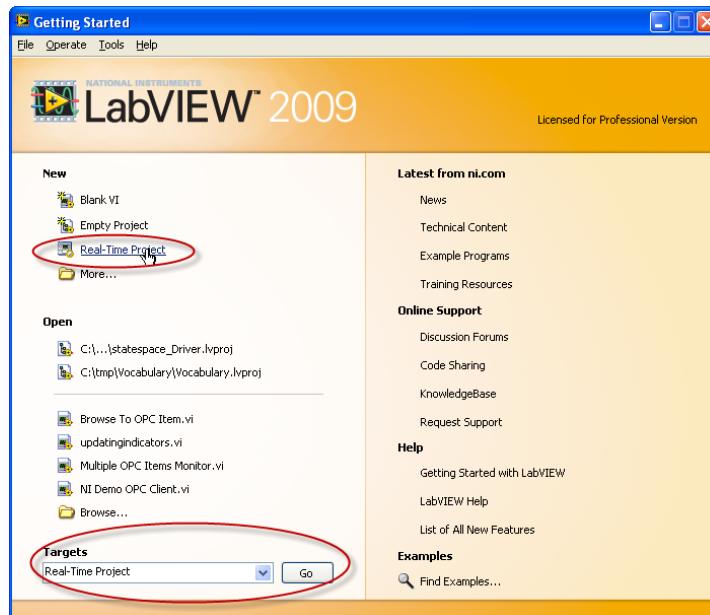


To test your LabVIEW Real-Time program on your hardware platform, simply click on the run arrow and your application will automatically be transferred to your real-time hardware via Ethernet and begin running. You can use standard NI debugging tools such as highlight execution, single stepping,

and breakpoints from your development system even though your real-time program is actually running on a separate system.

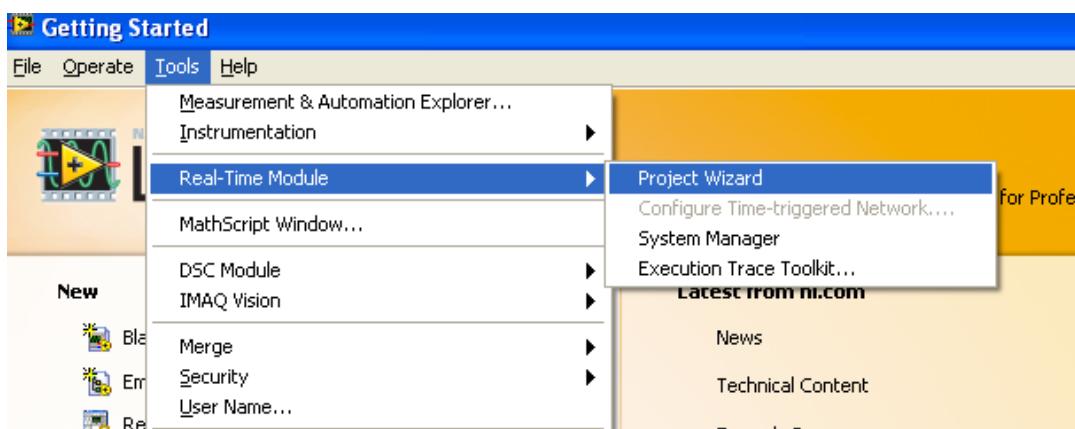
When you have finalized your real-time program, you can build an EXE in LabVIEW and download it to your real-time hardware as a startup application. After rebooting your real-time hardware, your program will automatically run in a reliable, stand-alone fashion.

You start a new Real-Time Project from your Getting Started window.



Select either “Real-Time Project” in the New group or select “Real-Time Project” in the Targets selector and then “Go”.

The Real-Time project is also available from the Tools menu.



14 Compact FieldPoint

14.1 Introduction

Compact FieldPoint is an easy-to-use, highly expandable industrial control and measurement system composed of rugged and dependable I/O modules and intelligent communication interfaces.

Download your NI LabVIEW application onto the embedded real-time controller for stand-alone data logging, monitoring, and advanced control. Plus, connect to virtually any sensor type with the wide variety of I/O modules, such as thermocouples, resistance temperature detectors (RTDs), strain gages, 4-20 mA sensors, and a variety of digital signals from 5-30 VDC and 0-250 VAC. The Compact FieldPoint I/O modules filter, calibrate, and scale raw sensor signals to engineering units and perform self-diagnostics to look for problems, such as an open thermocouple.



With Compact FieldPoint you can:

- Deploy real-time embedded controllers for stand-alone data logging, advanced analysis, and process control.
- Access I/O points nearby or miles away on the network using the same simple read and write software framework.
- Connect virtually any sensor directly to the wide variety of high-accuracy analog and discrete I/O modules.
- Download your NI LabVIEW application to the embedded controller for reliable, stand-alone operation.

Compact FieldPoint is very easy to use and can help you quickly begin performing industrial control and measurement. Getting started is a simple three-step process:

1. Install the hardware
2. Configure the controller
3. Read or write the I/O channels

Tight integration between the LabVIEW software environment and real-time hardware targets, such as Compact FieldPoint, make your control development process quick and easy. You can take advantage of powerful graphical development using industry-standard LabVIEW to rapidly implement your control system.

1. First, develop the program on a Windows host computer using graphical programming blocks.
2. Next, download the application to an embedded controller such the NI cFP-2000/cFP-2220.
3. Finally, deploy your application for dependable long-term execution that performs reliably both with and without a network connection. Once your application is deployed, you can access the embedded application through any Web browser by connecting to the embedded Web server of the real-time Compact FieldPoint controller.

14.2 Development

In order to configure and use your Compact FieldPoint you need to do the following steps.

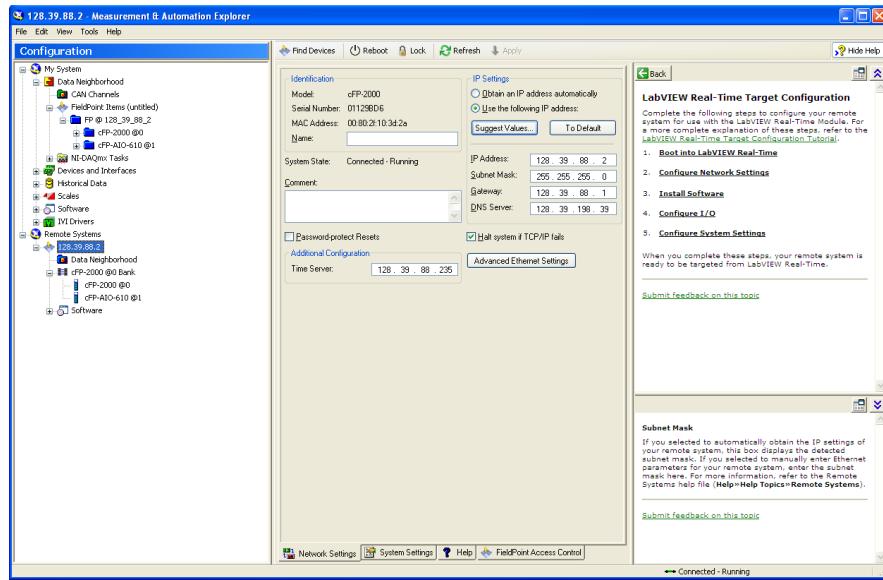
Step 1: Connect the Compact FieldPoint to your PC

Connect your Compact FieldPoint system to your PC using an Ethernet **cross-over cable**. If you use a Switch or Hub you may use a standard Ethernet cable.

Step 2: Install and Configure the Compact FieldPoint

You use the **Measurement & Automation Explorer (MAX)** to configure your Compact FieldPoint system.

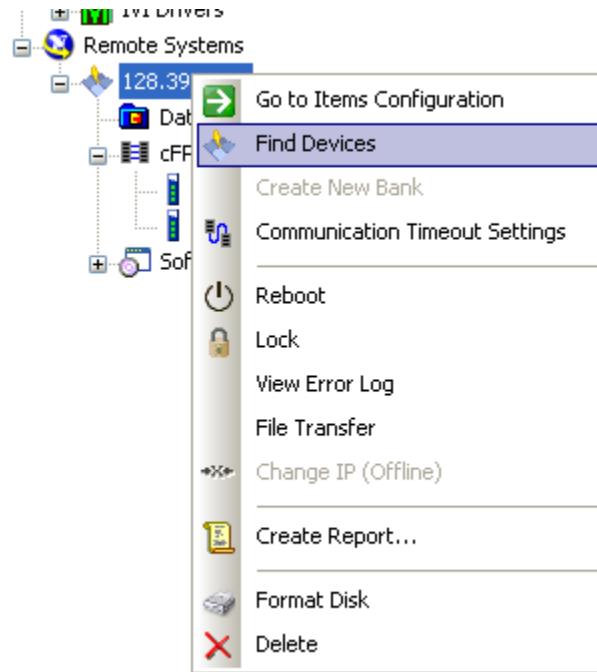
Go to “Remote Systems”. The system should now be automatically detected.



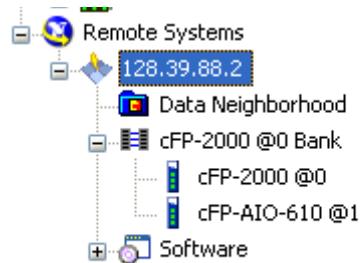
You may need to configure the IP settings.

Step 3: Find your Devices

Click on your Compact FieldPoint node and right-click and select “**Find Devices**”.



All your Compact FieldPoint modules (controller, I/modules) should now appear.



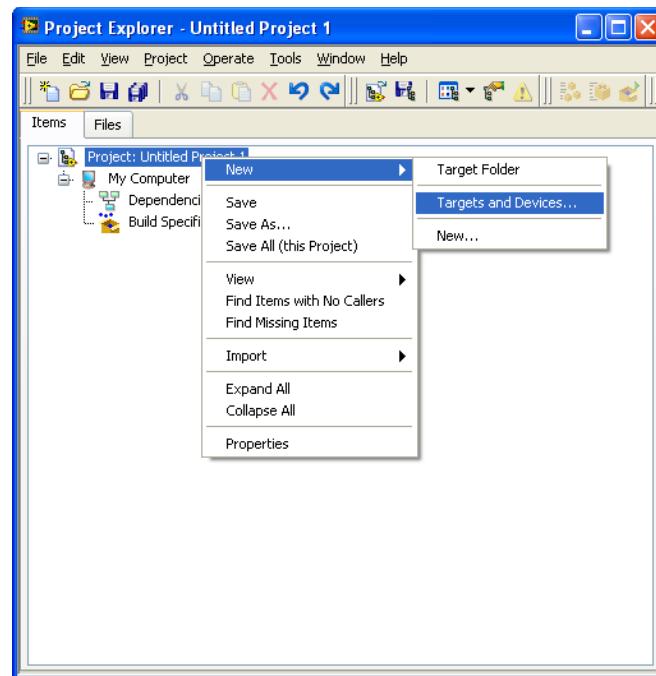
In our case we have acFP-2000 controller and a cFP-AIO-610 I/O module.

Step 4: Start an Empty Project in LabVIEW

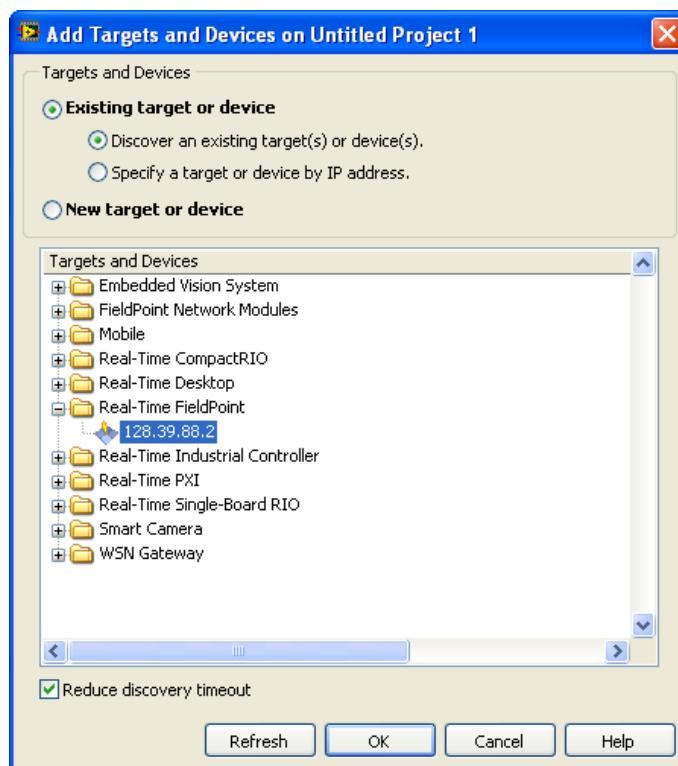
Start LabVIEW and select “Empty Project” in the Getting Started window.



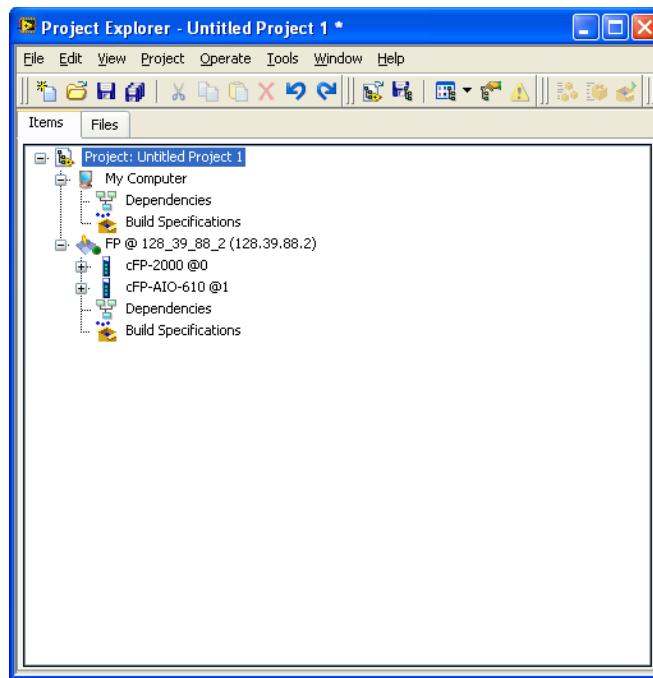
Right-click on the Project node and select “Targets and Devices...”



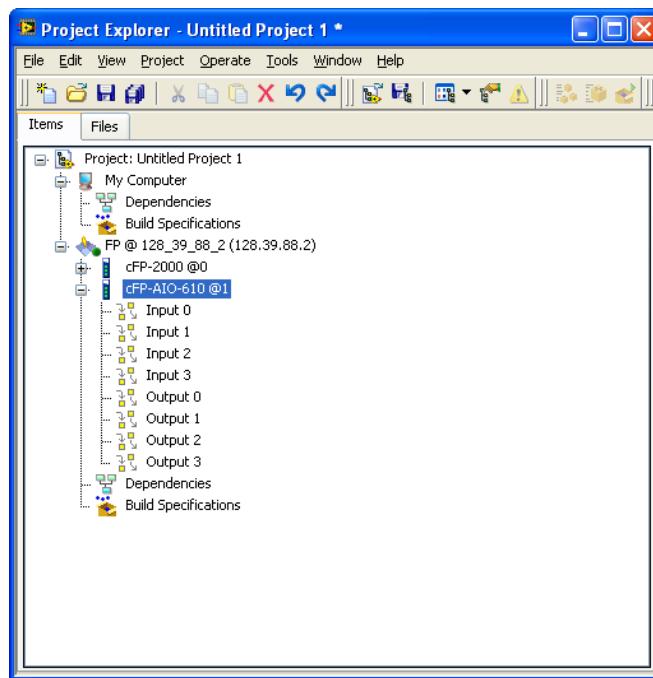
Select your Compact FieldPoint device from the list (**Real-Time FieldPoint**):



When you click OK, all your modules will be automatically inserted into your Project.

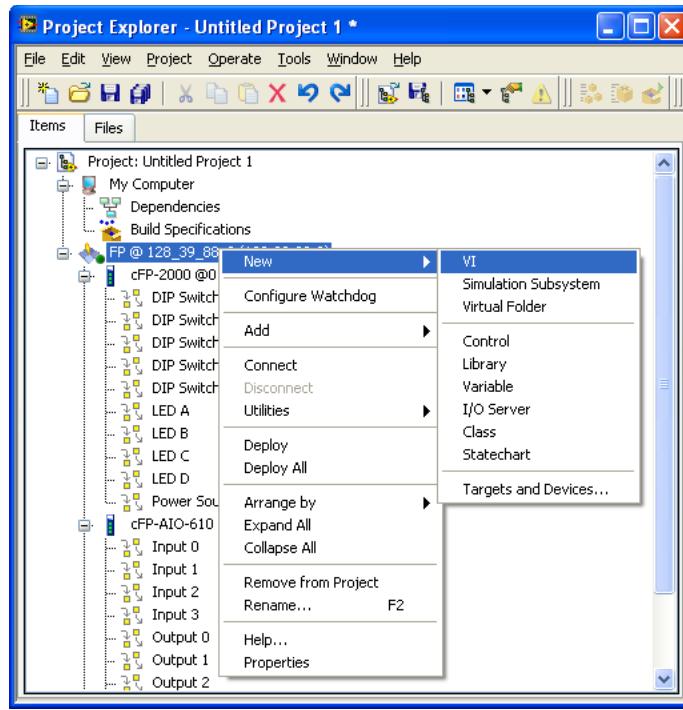


If you click on your I/O module, you will see all your I/O channels for that module:

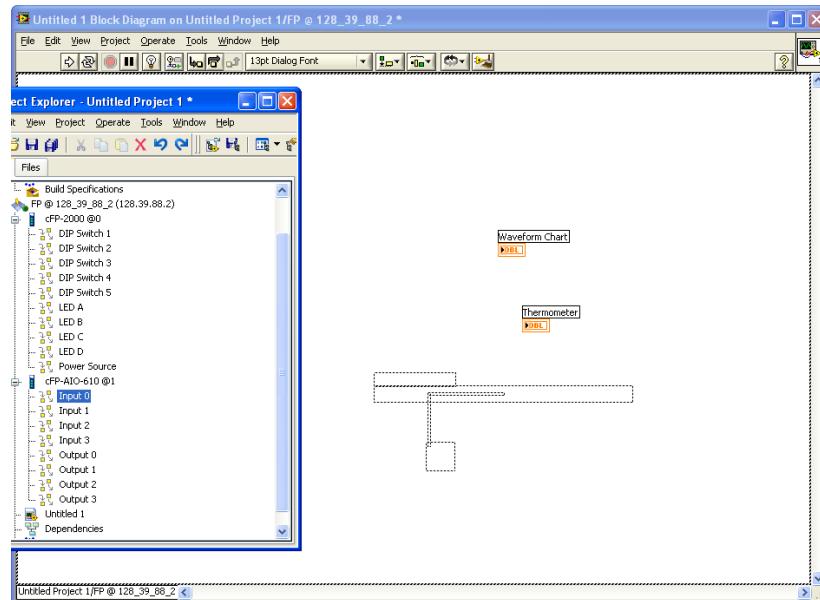


Step 5: Create your VI

Right-click on your controller and select New→VI



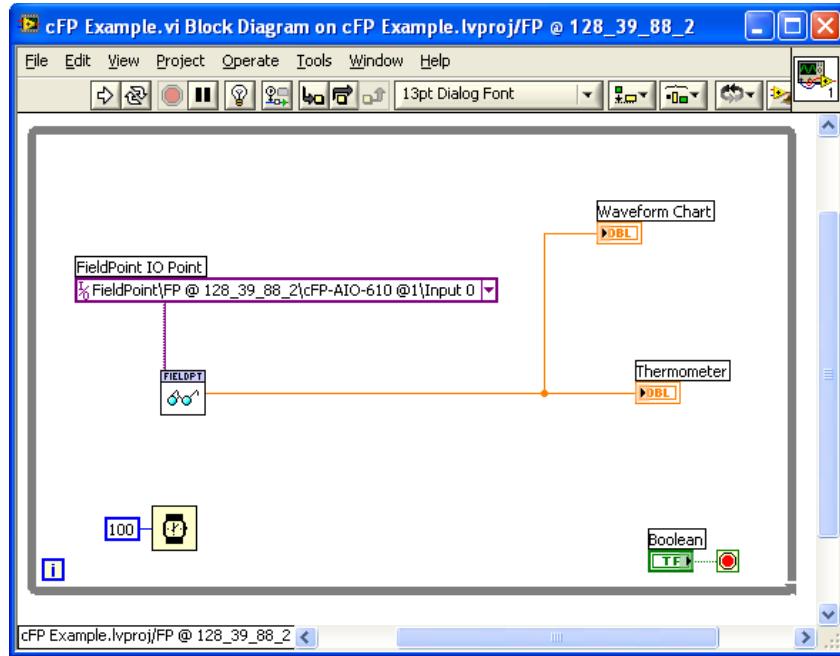
Go to your I/O module in the project tree and select your channel, e.g., Input 0. Drag the channel to your Block Diagram and the code for that channel will be automatically created.



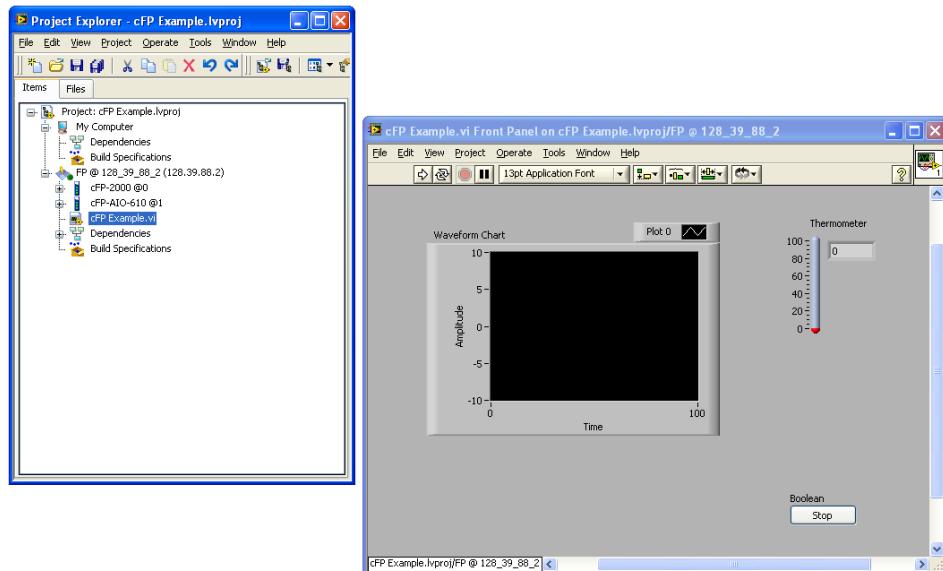
Step 6: Finish your Program

Finish your program, e.g., read Input 0 and plot the values in a Chart.

Block Diagram:

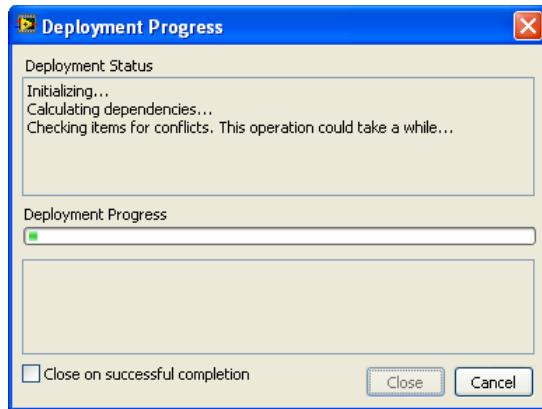


Here are your Project Explorer and the Front Diagram:



Step 6: Deployment

Click the Run button to start test and deploy your program. LabVIEW then automatically starts to check your could and download it to the controller.



Step 6: Finish

Your system should now be ready to use.

15 CompactRIO

15.1 Introduction

CompactRIO (cRIO) is a real time industrial controller made by National Instruments. The CompactRIO is a combination of a Real Time Controller, reconfigurable IO Modules (RIO), FPGA Module and an Ethernet expansion chassis.

Below we see a CompactRIO system:



You develop your CompactRIO software on your computer using LabVIEW then you download your code to the system. The **LabVIEW Real-Time Module** is used for this.

If you want to use the FPGA, you need to install the LabVIEW FPGA Module

15.2 Development

Tight integration between the LabVIEW software environment and real-time hardware targets, such as CompactRIO, make your control development process quick and easy. You can take advantage of powerful graphical development using industry-standard LabVIEW to rapidly implement your control system.

1. First, develop the program on a Windows host computer using graphical programming blocks.
2. Next, download the application to an embedded controller such the NI cFP-2000/cFP-2220.
3. Finally, deploy your application for dependable long-term execution that performs reliably both with and without a network connection. Once your application is deployed, you can access the embedded application through any Web browser by connecting to the embedded Web server of the real-time CompactRIO controller.

In order to configure and use your CompactRIO you need to do the following steps.

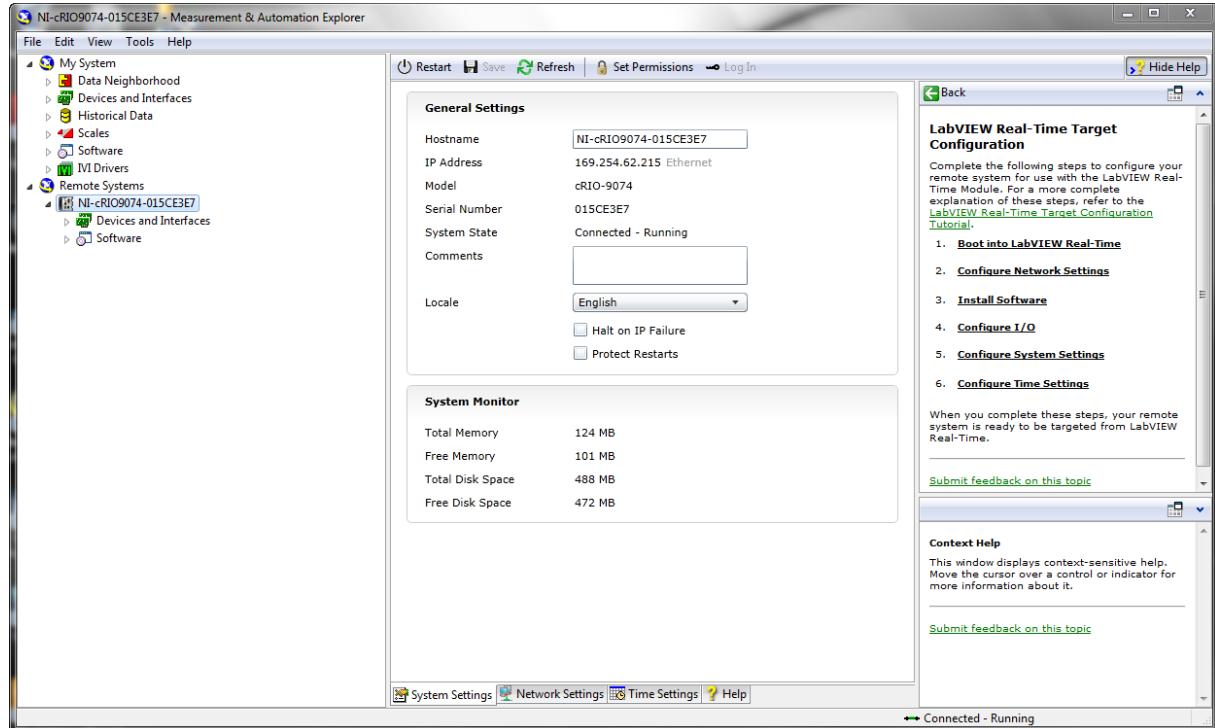
Step 1: Connect the CompactRIO to your PC

Connect your CompactRIO system to your PC using an Ethernet cable.

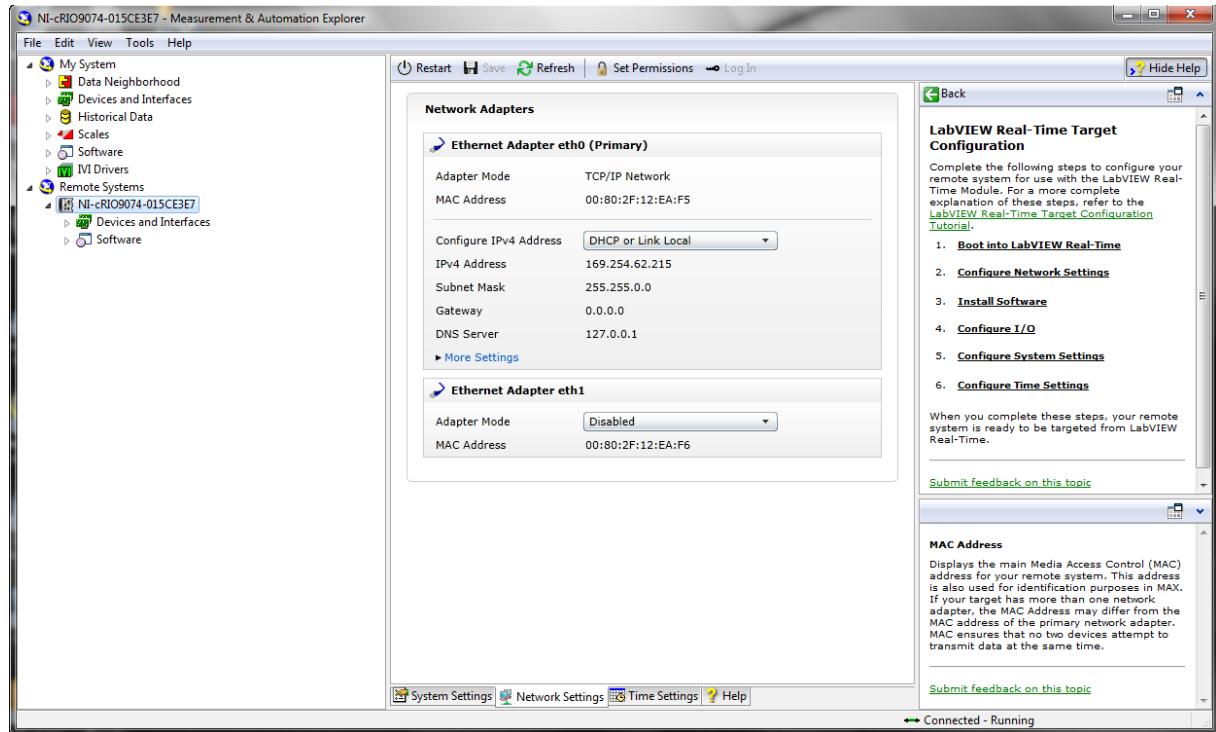
Step 2: Install and Configure the CompactRIO

You use the **Measurement & Automation Explorer (MAX)** to configure your CompactRIO system.

Go to “Remote Systems”. The system should now be automatically detected.



You may need to configure the IP settings, etc. (Network Settings).



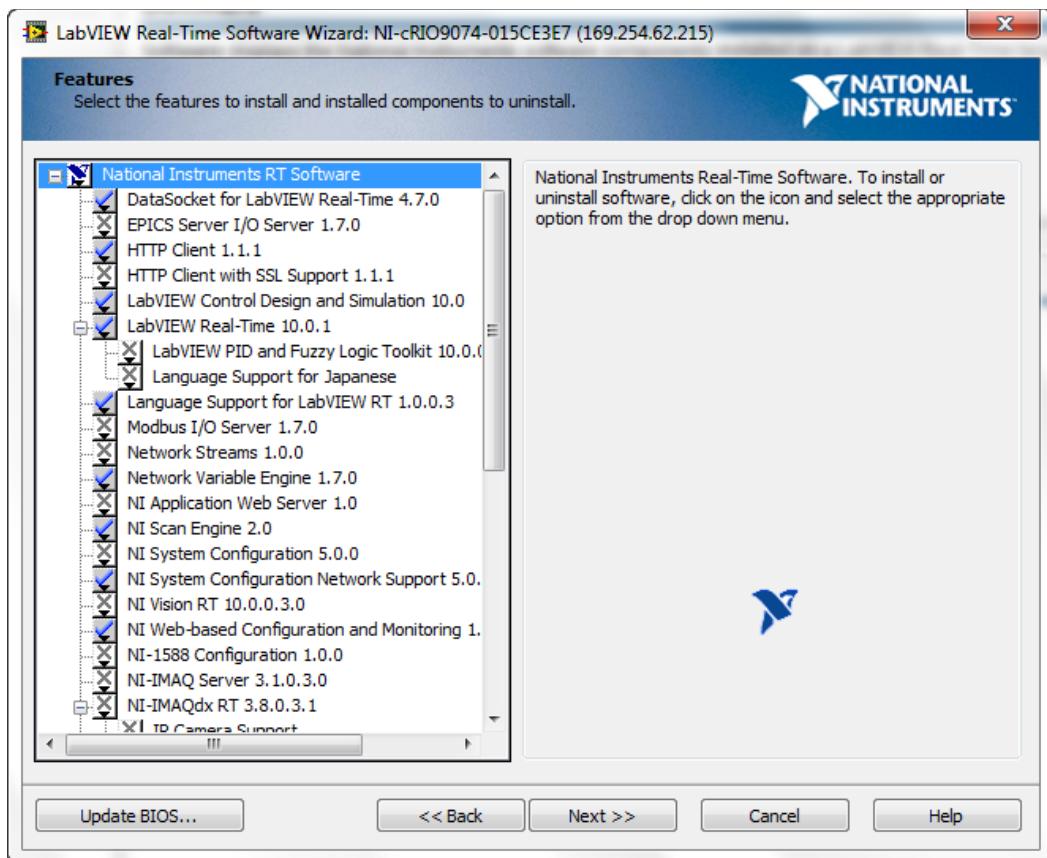
Step 3: Find your Devices

All your CompactRIO modules (controller, I/modules) should now appear under Devices and Interfaces.

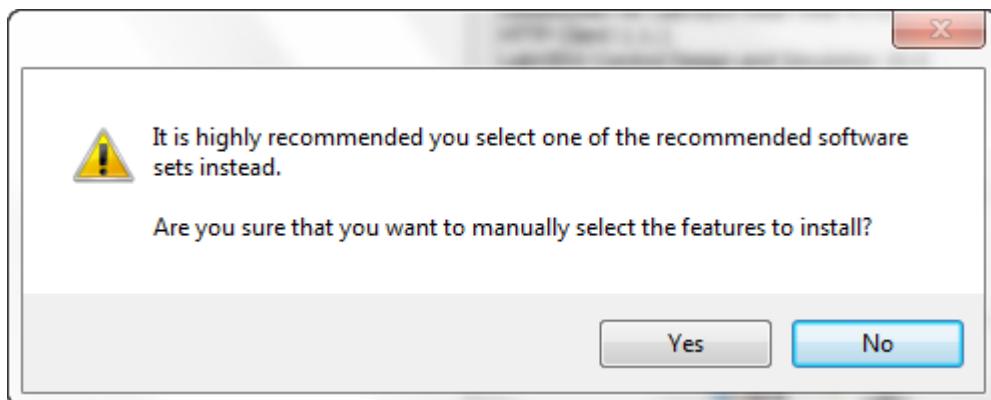
In our case we have a NI cRIO-9074 controller, a NI 9263 Analog Out I/O module and a NI 9201 Analog In I/O module.

Step 4: Install Software

You only do this once when you use the CompactRIO for the first time or you want to add or upgrade the existing software.



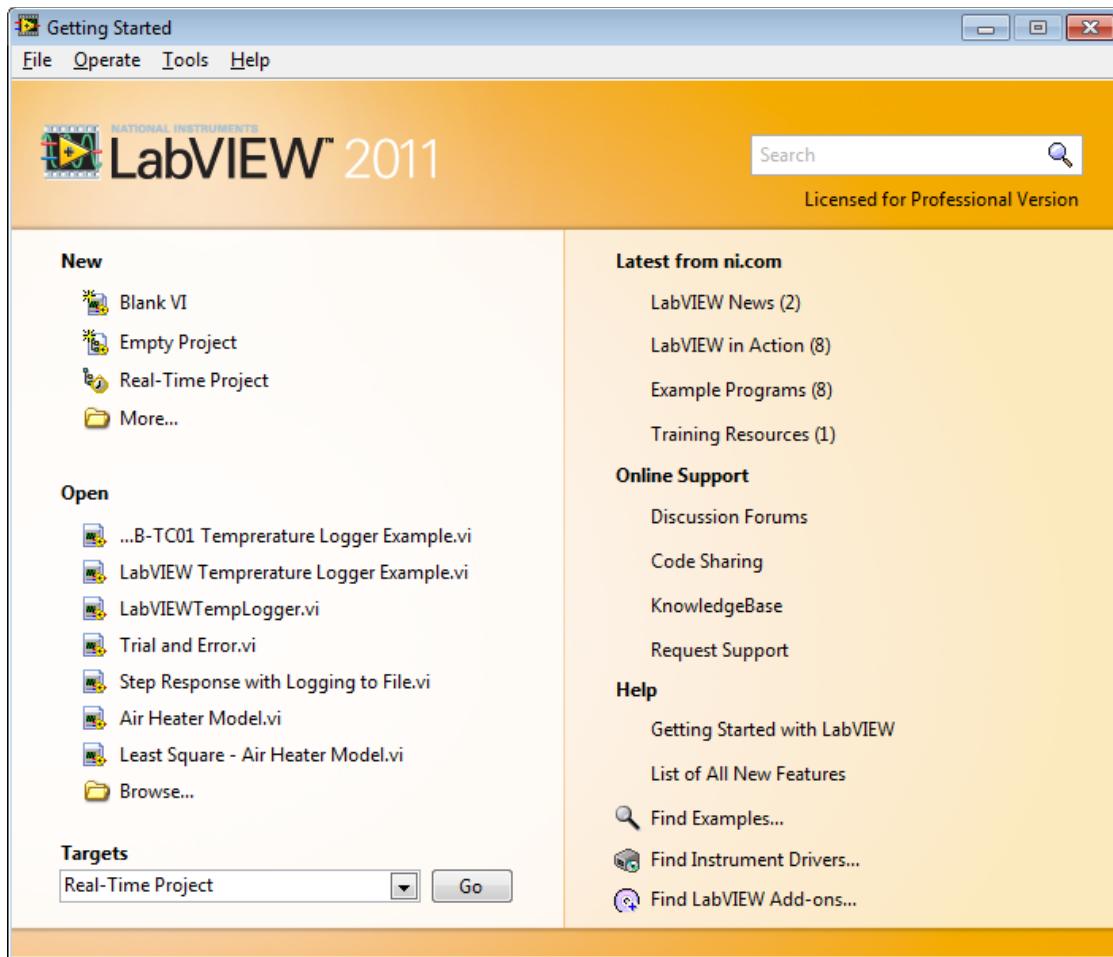
If you select custom installation, you might get a warning like this:



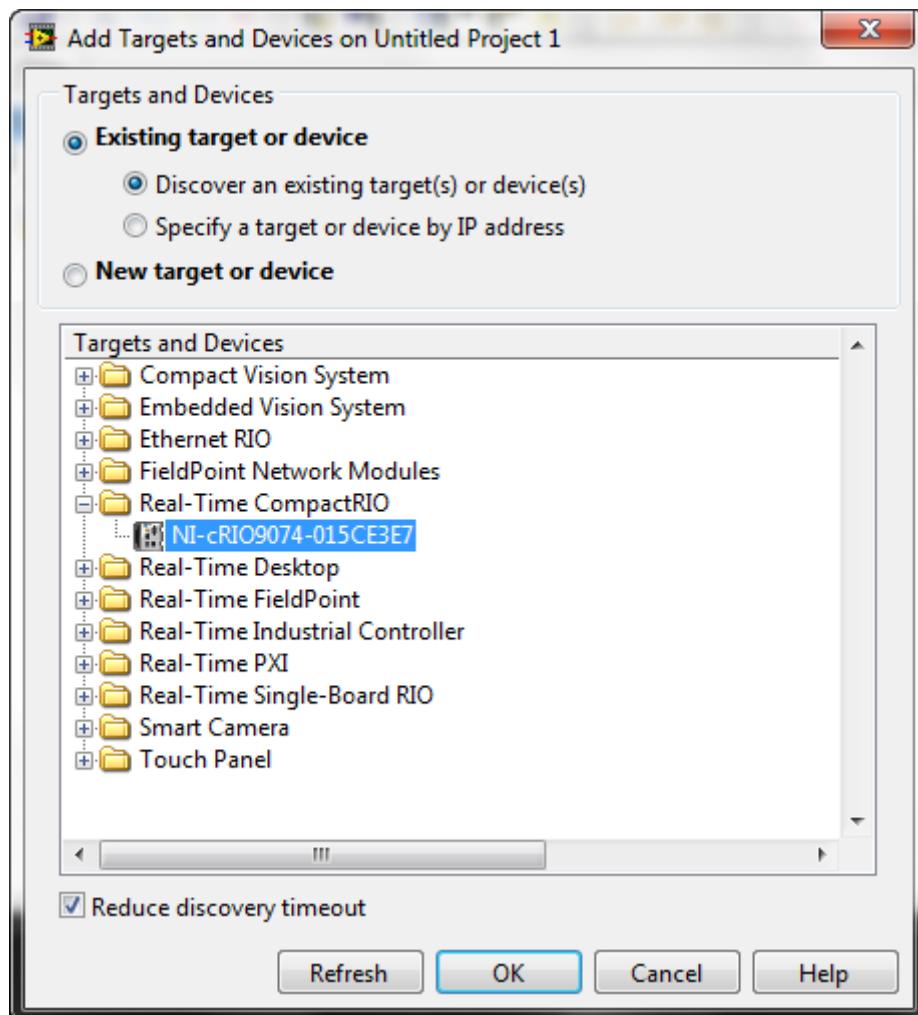
Just discard it!

Step 5: Start a new Project in LabVIEW

Start LabVIEW and select “Real-Time Project” in the Getting Started window.

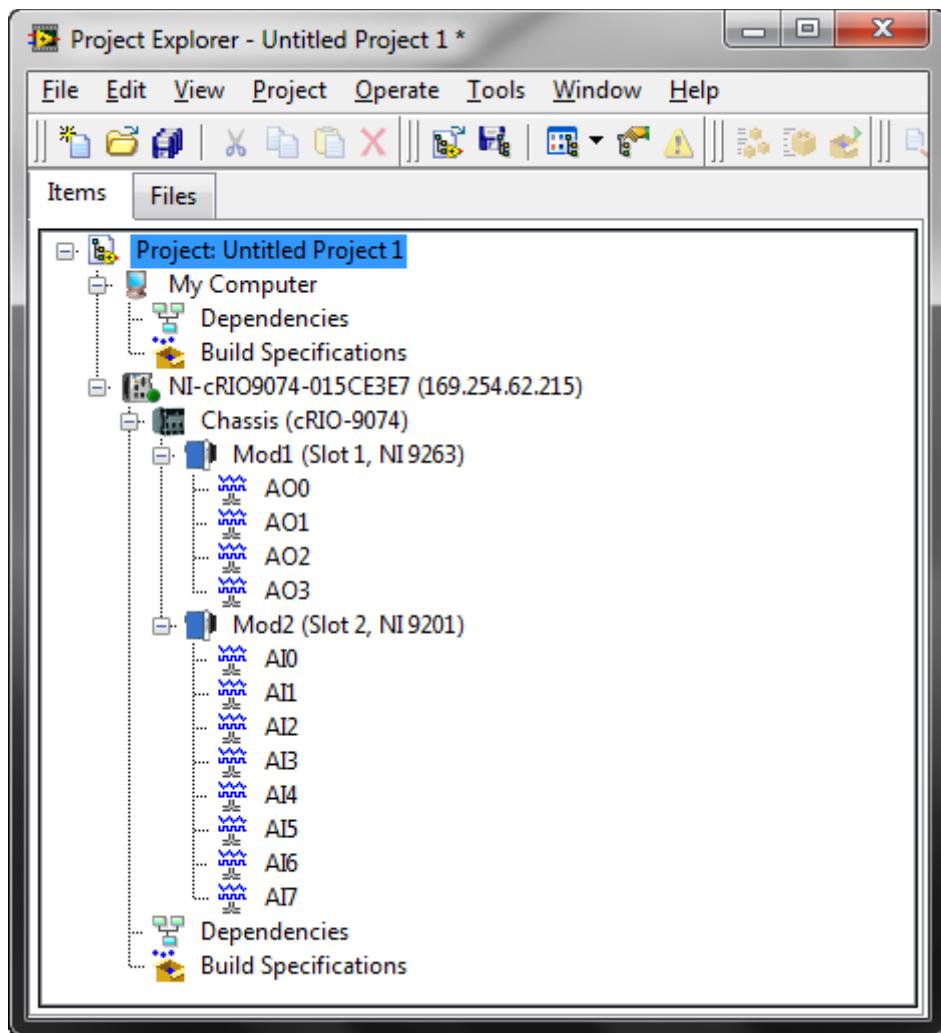


Right-click on the Project node and select “**Targets and Devices...**”



Select your CompactRIO device from the list (**Real-Time CompactRIO**).

When you click OK, all your modules will be automatically inserted into your Project.



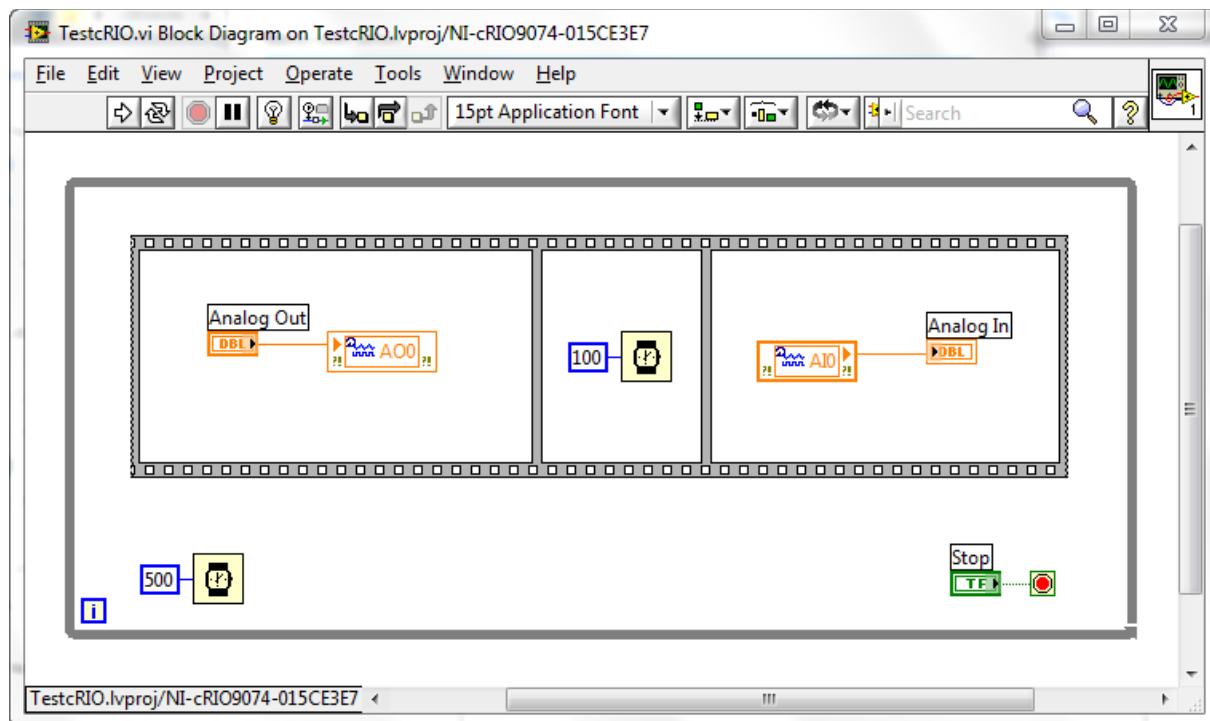
You will now see all your I/O modules that you have and you will see all your I/O channels for these modules.

Step 6: Create your VI

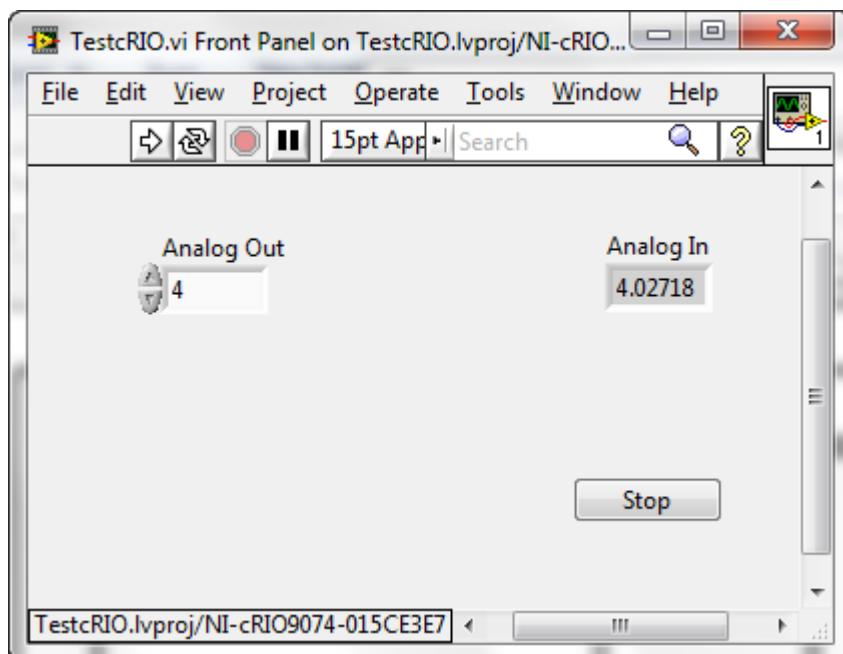
Right-click on your controller and select New→VI

Go to your I/O module in the project tree and select your channel, e.g., Input 0. Drag the channel to your Block Diagram and the code for that channel will be automatically created.

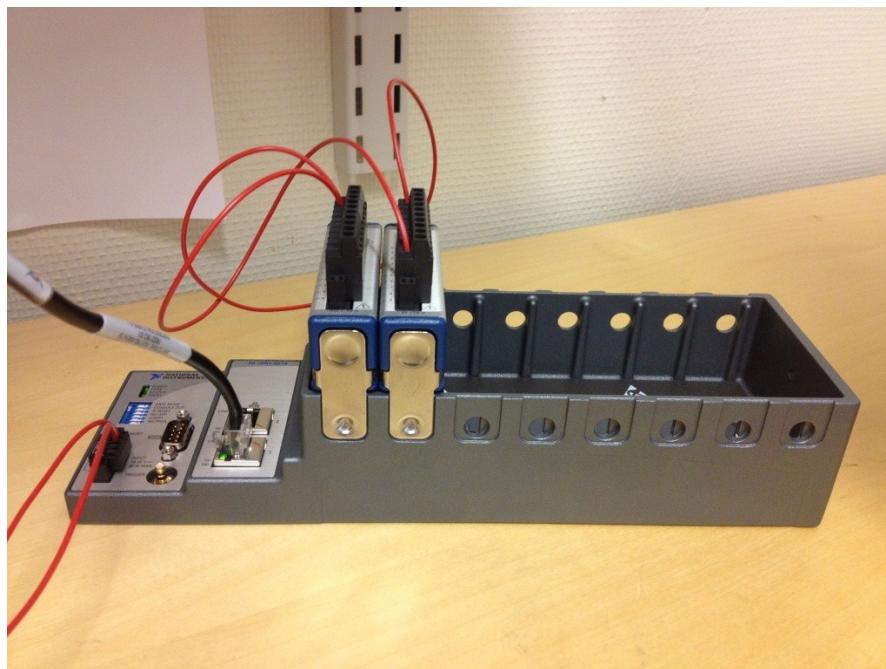
Below we see a simple example with one Analog In and one Analog Out:



The Front Panel:

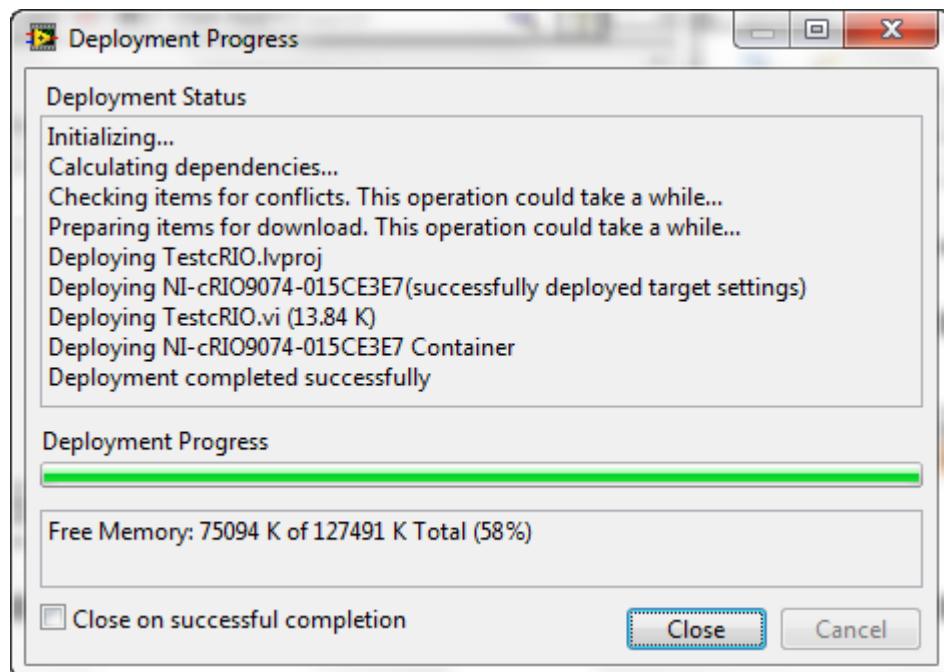


In this simple example we have used a simple loop-back test, meaning we have connected AO0 (Analog Out, Channel 0) and AI0 (Analog In, Channel 0) together, as shown below. This means when we write 4V to the AO0, we should receive 4V on AI0.



Step 7: Deployment

Click the Run button to start test and deploy your program. LabVIEW then automatically starts to check your code and download it to the controller.



Step 8: Finish

Your system should now be ready to use.



Høgskolen i Telemark

Telemark University College

Faculty of Technology

Kjølnes Ring 56

N-3918 Porsgrunn, Norway

www.hit.no

Hans-Petter Halvorsen, M.Sc.

Telemark University College

Department of Electrical Engineering, Information Technology and Cybernetics

Phone: +47 3557 5158

E-mail: hans.p.halvorsen@hit.no

Blog: <http://home.hit.no/~hansha/>

Room: B-237a

