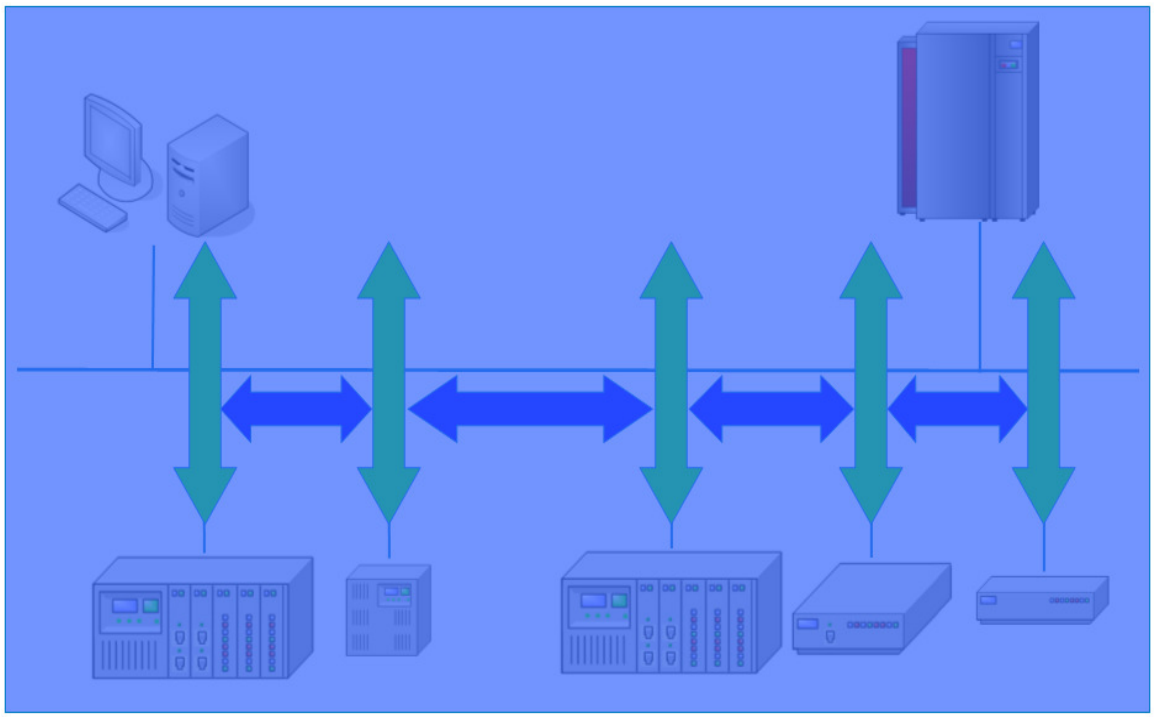

deviceWISE 2.3 Platform

C# Application Programming Interface

Version 1.0.

Table of Contents

Introducing the deviceWISE C# Client 3
deviseWISE C# Functional Components 3
Secure Access Operations 4
Project Operations 9
Trigger Operations 12
Device Operations 16
Variable Operations 19
Notes on Reading Variables 24
Sample Application: Projects 25
Sample Application: Triggers 27
Sample Application: Devices 29
Sample Application: Variables 32
Sample Application: Reading/Writing Lists of Variables 36

Introducing the deviceWISE C# Client

The deviceWISE C# Client library (dwclient.dll) allows programmers to develop applications which function in a closely coupled manner with deviceWISE enabled embedded, edge and enterprise systems.

The C# dwclient library allows programmers to connect locally or remotely in a secure encrypted manner to deviceWISE systems using managed authorized user/password combinations.

The C# dwclient is delivered as a Microsoft .NET V2.0 component module to support both Win32/Win64 based systems.

deviseWISE C# Functional Components

The deviceWISE C# API (Application Programming Interface) is a collection of application methods based upon five functional groups which support closely coupled client application development.

Operation	Description
Secure Access	Support for Secure Connect/Disconnect
Project Operations	Support for Project Start/Stop
Trigger Operations	Support for Trigger Start/Stop/Execute
Device Operations	Support for Device Start/Stop
Variable Operations	Support for Variable Read/Write

Secure Access Operations

The deviceWISE C# API Secure Access functional group allows programmers to develop client applications which securely connect to deviceWISE enabled embedded, edge and enterprise systems.

The four exposed methods for Secure Access functionality are:

Connect( )	Connect to a deviceWISE enabled System
Login( )	Login to a deviceWISE enabled System
InitializeWorker( )	Initialize environment to perform deviceWISE actions
Disconnect( )	Disconnect from a deviceWISE enabled System

The following pages provide examples of using the deviceWISE C# API Secure Access functional group methods.

Connect(url);

Obtain a secure connection to a deviceWISE system.

Input Parameters

Parameter	Type	Description
url	string	The hostname or ipAddress of the target System

Output Parameters

none

Calling Example

```
:
// create a local instance of the DW worker
DwWorker worker = new DwWorker( );
:
:
:
// get a secure connection to the module
try { worker.Connect("secure://192.168.2.176"); }
catch ( Exception e  )
{
System.Console.WriteLine("Error connecting to the system"  );
System.Console.WriteLine( e );
System.Environment.Exit( -4 );
}
:
:
:
```

Login(userID, password);

Login to the connected deviceWISE system.

Input Parameters

Parameter	Type	Description
userID	string	A defined userid within the target System
password	string	Valid password for the userid provided

Output Parameters

none

Calling Example

```
:
// create a local instance of the DW worker
DwWorker worker = new DwWorker( );
:
:
:
// login to deviceWISE
try { worker.Login("admin", "admin"); }
catch ( DwException e )
    {
    System.Console.WriteLine("Error logging on to system - deviceWISE Status=" + e.Status );
    System.Environment.Exit( -4 );
    }
:
:
:
```

InitializeWorker( );

Set up process initialization on this connection to perform additional actions. The InitializeWorker method requires no parameters.

Calling Example

```
// create a local instance of the DW worker
DwWorker worker = new DwWorker( );
:
:
:
:
worker.Connect("secure://192.168.2.176");
worker.Login("admin", "admin");

worker.InitializeWorker( );
:
:
:
```

Disconnect( );

Disconnect from the deviceWISE system.
The Disconnect method requires no parameters.

Calling Example

```
// create a local instance of the DW worker
DwWorker worker = new DwWorker( );
:
:
:
worker.Connect("secure://192.168.2.176");
worker.Login("admin", "admin");
worker.InitializeWorker( );
:
:
:
worker.Disconnect( );
:
```

Project Operations

The deviceWISE C# API Project Operations functional group allows programmers to develop client applications which closely integrate to deviceWISE enabled embedded, edge and enterprise systems.

The two exposed methods for Project Operations functionality are:

Start( ) Start a deviceWISE Project on the connected server
Stop( ) Stop a deviceWISE Project on the connected server

The following pages provide examples of using the deviceWISE C# API Project Operations functional group methods.

Start(projectName, timeout);

Start a deviceWISE project.

Input Parameters

Parameter	Type	Description
projectName	string	The name of a project
timeout	int	Timeout in milliseconds for operation to complete

Output Parameters

Parameter	Type	Description
return	int	The completion status of the call

Completion Status

Status	Description
0	Success
-1	Timeout
-5302	Project does not exist
-5304	Project is already started

Calling Example

```
// create a local instance of the DW worker
DwWorker worker = new DwWorker( );
int       timeout = 5000;
:
:
worker.Connect("secure://192.168.2.176");
worker.Login("admin", "admin");
worker.InitializeWorker( );
:
:
:
:
// set the project name from the command line and then start it
project_name = String.Copy( args[0] );
rc = worker.Project.Start( project_name, timeout );
if ( rc == 0 ) Console.WriteLine("Project=" + project_name + " start operation, rc = " + rc);
```

Stop(projectName, timeout);

Stop a deviceWISE project.

Input Parameters

Parameter	Type	Description
projectName	string	The name of a project
timeout	int	Timeout in milliseconds for operation to complete

Output Parameters

Parameter	Type	Description
return	int	The completion status of the call

Completion Status

Status	Description
0	Success
-1	Timeout
-5302	Project does not exist
-5303	Project is not started

Calling Example

```
// create a local instance of the DW worker
DwWorker worker = new DwWorker( );
int       timeout = 5000;
:
:
worker.Connect("secure://192.168.2.176");
worker.Login("admin", "admin");
worker.InitializeWorker( );
:
:
:
:
// set the project name from the command line
project_name = String.Copy( args[0] );
rc = worker.Project.Stop( project_name, timeout );
if ( rc == 0 ) Console.WriteLine("Project=" + project_name + " stop operation, rc = " + rc);
```

Trigger Operations

The deviceWISE C# Trigger Operations functional group allows programmers to develop client applications which integrate closely to deviceWISE enabled embedded, edge and enterprise systems.

The three exposed methods for Trigger Operations functionality are:

Start( ) Start a Named deviceWISE Trigger
Stop( ) Stop a Named deviceWISE Trigger
FireNow( ) Execute a Named deviceWISE Trigger

The following pages provide examples of using the deviceWISE C# API Trigger Operations functional group methods.

Start(projectName, triggerName, timeout);

Start a deviceWISE trigger.

Input Parameters

Parameter	Type	Description
projectName	string	The name of the project
triggerName	string	The name of the trigger
timeout	int	Timeout in milliseconds for operation to complete

Output Parameters

Parameter	Type	Description
return	int	The completion status of the call

Completion Status

Status	Description
0	Success
-1	Timeout
-5402	Trigger does not exist
-5404	Trigger is already started

Calling Example

```
// create a local instance of the DW worker
DwWorker worker = new DwWorker( );
int       timeout = 5000;
:
:
// set the project name from the command line and then start it
project_name = String.Copy( args[0] );
rc = worker.Project.Start( project_name, timeout );
if ( rc == 0 ) Console.WriteLine("Project=" + project_name + " start operation, rc = " + rc);

// set the trigger name from the command line and then start it
trigger_name = String.Copy( args[1] );
rc = worker.Trigger.Start( project_name, trigger_name, timeout );
if ( rc == 0 ) Console.WriteLine("Trigger=" + trigger_name + " start operation, rc = " + rc);
```

Stop(projectName, triggerName, timeout);

Stop a deviceWISE trigger.

Input Parameters

Parameter	Type	Description
projectName	string	The name of the project
triggerName	string	The name of the trigger
timeout	int	Timeout in milliseconds for operation to complete

Output Parameters

Parameter	Type	Description
return	int	The completion status of the call

Completion Status

Status	Description
0	Success
-1	Timeout
-5402	Trigger does not exist
-5403	Trigger is not started

Calling Example

```
// create a local instance of the DW worker
DwWorker worker = new DwWorker( );
int       timeout = 5000;
:
:
:
// set the trigger name from the command line and then stop it
project_name = String.Copy( args[0] );
trigger_name = String.Copy( args[1] );
rc = worker.Trigger.Stop( project_name, trigger_name, timeout );
if ( rc == 0 ) Console.WriteLine("Trigger=" + trigger_name + " stop operation, rc = " + rc);
```

FireNow(projectName, triggerName, timeout);

Execute the named trigger.

Input Parameters

Parameter	Type	Description
projectName	string	The name of the project
triggerName	string	The name of the trigger
timeout	int	Timeout in milliseconds for operation to complete

Output Parameters

Parameter	Type	Description
return	int	The completion status of the call

Completion Status

Status	Description
0	Success
-1	Timeout
-5402	Trigger does not exist
-5403	Trigger is not started

Calling Example

```
// create a local instance of the DW worker
DwWorker worker = new DwWorker( );
int       timeout = 5000;
:
:
:
// set the trigger name from the command line and then start it
trigger_name = String.Copy( args[1] );
rc = worker.Trigger.Start( project_name, trigger_name, timeout );
if ( rc == 0 ) Console.WriteLine("Trigger=" + trigger_name + " start operation, rc = " + rc);

// let the trigger start up completely
Thread.Sleep( 5000 );

// execute the trigger now
rc = worker.Trigger.FireNow( project_name, trigger_name, timeout );
if ( rc == 0 ) Console.WriteLine("Trigger=" + trigger_name + " executed, rc = " + rc);
```

Device Operations

The deviceWISE C# Device Operations functional group allows programmers to develop client applications which closely integrate to deviceWISE enabled embedded, edge and enterprise systems.

The two exposed methods for Device Operations functionality are:

Start( ) Start a deviceWISE Device
Stop( ) Stop a deviceWISE Device

The following pages provide examples of using the deviceWISE C# Device Operations functional group methods.

Start(deviceName, timeout);

Start a deviceWISE device.

Input Parameters

Parameter	Type	Description
deviceName	string	The name of the device
timeout	int	Timeout in milliseconds for operation to complete

Output Parameters

Parameter	Type	Description
return	int	The completion status of the call

Completion Status

Status	Description
0	Success
-1	Timeout
-6202	Device does not exist
-6208	Device is not stopped

Calling Example

```
:
:
// create a local instance of the DW worker
DwWorker worker = new DwWorker( );
string   device_name = "Local CPU 1";
int      timeout = 5000;
:
:

// *** start the device
rc = worker.Device.Start( device_name, timeout );
if ( rc == 0 ) Console.WriteLine("Device=" + device_name +  " started, rc = " + rc);

// *** let the device start operation complete
Thread.Sleep( 3000 );
```

Stop(deviceName, timeout);

Stop a deviceWISE device.

Input Parameters

Parameter	Type	Description
deviceName	string	The name of the device
timeout	int	Timeout in milliseconds for operation to complete

Output Parameters

Parameter	Type	Description
return	int	The completion status of the call

Completion Status

Status	Description
0	Success
-1	Timeout
-6202	Device does not exist
-6207	Device is not started

Calling Example

```
:
:
// create a local instance of the DW worker
DwWorker worker = new DwWorker( );
string   device_name = "Local CPU 1";
int      timeout = 5000;
:
:

// *** stop the device
rc = worker.Device.Stop( device_name, timeout );
if ( rc == 0 ) Console.WriteLine("Device=" + device_name +  " stopped, rc = " + rc);
```

Variable Operations

The deviceWISE C# Variable Operations functional group allows programmers to develop client applications which closely integrate to deviceWISE enabled embedded, edge and enterprise systems.

The two exposed methods for Variable Operations functionality are:

Write( ) Write a Named deviceWISE Variable on the Target Server
Read( ) Read a Named deviceWISE Variable on the Target Server

The following pages provide examples of using the deviceWISE C# Variable Operations functional group methods.

Write(out vwe, device_name, var_name, data_count, data_length, DwDataType.____, data_value, timeout);

Write a value to a device.

Input Parameters

Parameter	Type	Description
vwe	out	Name of a **variable write entry** object
device_name	string	The name of the device
var_name	string	Name of the variable to write
data_count	int	The number of elements to write (1=scalar, >1=array)
data_length	int	Length of string element, otherwise = –1
DwDataType.___		Data type of element being written: BOOL, INT2, INT4, FLOAT4, STRING
data_value	string	The value to be written expressed as a string
timeout	int	Timeout in milliseconds for operation to complete

Output Parameters

Parameter	Type	Description
return	int	The completion status of the call

Completion Status

Status	Description
0	Success
-1	Timeout
-6202	Device does not exist
-6207	Device is not started

Calling Example

```
DwWorker.DwVariable.VariableWriteEntry vwe = null;

// *** write value for INT2
string  device_name = "Local CPU 1";
data_count  = 1;
data_length = -1;
var_name = "D[10]";
data_value = “123”;

rc = worker.Variable.Write( out vwe,  device_name, var_name, data_count, data_length,
DwDataType.INT2, data_value, timeout );

if ( rc == 0 ) System.Console.WriteLine("Wrote  " + var_name +  "  Value = " + data_value );
```

Read(out vre, device_name, var_name, data_count, data_length, DwDataType.____, timeout);

Read a value from a device.

Input Parameters

Parameter	Type	Description
vre	out	Name of a **variable read entry** object
device_name	string	The name of the device
var_name	string	Name of the variable to read
data_count	int	The number of elements to read (1=scalar, >1=array)
data_length	int	Length of string element, otherwise = –1
DwDataType.___		Data type of element being read: BOOL, INT2, INT4, FLOAT4, STRING
timeout	int	Timeout in milliseconds for operation to complete

Output Parameters

Parameter	Type	Description
return	int	The completion status of the call

Completion Status

Status	Description
0	Success
-1	Timeout
-6202	Device does not exist
-6207	Device is not started

Calling Example

```
DwWorker.DwVariable.VariableReadEntry vre1 = null;

// *** read INT2
data_count  = 1;
data_length = -1;
var_name = "D[10]";

rc = worker.Variable.Read( out vre1, device_name, var_name, data_count, data_length,
DwDataType.INT2, timeout );

if ( rc == 0 ) System.Console.WriteLine(" Read  " + var_name +  "  Value = " +
              vre1.toINT2( ).ToString( ) );
```

Notes on Reading Variables

1. After you read a variable and wish to access it within the program, you must use one of the defined methods to extract it out of the **VariableReadEntry** object.

   For example, if you read an INT2 from the device into **vre1**, you can use the following to extract the value:

   ```
   short  x;
   x = vre1.toINT2( );
   ```

2. Similarly, if you read an INT4 from the device into **vre1**, you can use the following to extract the value:

   ```
   int   x;
   x = vre1.toINT4( );
   ```

3. If you read an INT2 array from the device into **vre1**, you can use the following to extract the value:

   ```
   short[ ]  x;
   x = vre.toINT2Array( );
   foreach ( short i in x )  {  System.Console.WriteLine("    {0} ", i );   }
   ```

4. Refer to **class VariableReadEntry** in Microsoft Visual Studio for all methods available.

5. There are other properties of the object that may be useful, for example:

   ```
   vre1.Count          = the number of elements
   vre1.DataType       = the data type of the object
   vre1.Name           = the name of the variable read
   ```

Sample Application: Projects

```
//=====================================================================
//
// Program Name:  SampleStartProject.cs
//                      C# Example Application for dwclient.dll (Library Component)
//
// Description:   This CSharp application demonstrates starting and stopping a project.
//
// Command Line Parameters:
//
//              The single parm on the command line is the name of the project
//
// To build this sample application, execute the CSharp command line compiler as shown
//
//               csc  /t:exe  /r:dwclient.dll   SampleStartProject.cs
//
// This Application and the associated dwclient library requires the
// Microsoft .NET Framework 2.0 or above.
// The libraries were built using the Microsoft .NET SDK 2.0 SP2 platform.
//
// Copyright(c) 2009.  ILS Technology, LLC.  All Rights Reserved.
//
//=====================================================================
//
// The sample program is provided to you on an "AS IS" basis, without warranty of any kind.
// ILS TECHNOLOGY HEREBY  EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER
// EXPRESS
// OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
// MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
// Some jurisdictions do not allow for the exclusion or limitation of implied warranties,
// so the above limitations or exclusions may not apply to you.
// ILS TECHNOLOGY shall not be liable for any damages you suffer as a result of using,
// modifying or distributing the sample program or its derivatives.
//
//=====================================================================

using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using DeviceWISE;

namespace sample1
{
   class Program
   {
      static void Main(string[] args)
      {
         int       rc        = 0;
         int       timeout = 5000;
         string   project_name;

          // create a local instance of the DW worker
```

```
        DwWorker worker = new DwWorker();

        // get a secure connection to the module
        try { worker.Connect("secure://192.168.2.176"); }
        catch ( Exception e  )
            {
            System.Console.WriteLine("Error connecting to the system"  );
            System.Console.WriteLine( " " );
            System.Console.WriteLine( e );
            System.Environment.Exit( -4 );
            }

        // login to deviceWISE
        try { worker.Login("admin", "admin"); }
        catch ( DwException e )
            {
            System.Console.WriteLine("Error logging on to system - Status=" + e.Status );
            System.Environment.Exit( -4 );
            }

        // set up for additional DW actions
        worker.InitializeWorker();

        // set the project name from the command line and then start it
        project_name = String.Copy( args[0] );
        rc = worker.Project.Start( project_name, timeout );
        if ( rc == 0 )
           Console.WriteLine("Project=" + project_name + " start operation, rc = " + rc);
        else
           Console.WriteLine("Project =" + project_name + " start operation failed, rc = " + rc);

        // go to the workbench and verify that the project is started - waiting 10 seconds
        Thread.Sleep( 10000 );

        rc = worker.Project.Stop( project_name, timeout );
        if ( rc == 0 )
           Console.WriteLine("Project=" + project_name + " stop operation, rc = " + rc);
        else
           Console.WriteLine("Project =" + project_name + " stop operation failed, rc = " + rc);

        worker.Disconnect();
     }
  }
}
```

Sample Application: Triggers

```
//=========================================================================
//
// Program Name:  SampleStartTrigger.cs
//                        C# Example Application for dwclient.dll (Library Component)
//
// Description:   This CSharp application demonstrates starting,
//                        stopping, and firing a trigger.
//
// Command Line Parameters:
//
//              2 parms on the command line:  project  trigger
//
// To build this sample application, execute the CSharp command line compiler as shown
//
//                csc  /t:exe  /r:dwclient.dll   SampleStartTrigger.cs
//
// This Application and the associated dwclient library requires the
// Microsoft .NET Framework 2.0 or above.
// The libraries were built using the Microsoft .NET SDK 2.0 SP2 platform.
//
// Copyright(c) 2009.  ILS Technology, LLC.  All Rights Reserved.
//
//=========================================================================
//
// The sample program is provided to you on an "AS IS" basis, without warranty of any kind.
// ILS TECHNOLOGY HEREBY  EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER
EXPRESS
// OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
// MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
// Some jurisdictions do not allow for the exclusion or limitation of implied warranties,
// so the above limitations or exclusions may not apply to you.
// ILS TECHNOLOGY shall not be liable for any damages you suffer as a result of using,
// modifying or distributing the sample program or its derivatives.
//
//=========================================================================

using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using DeviceWISE;

namespace sample1
{
   class Program
   {
      static void Main(string[] args)
      {
         int       rc        = 0;
         int       timeout = 5000;
         string   project_name;
         string   trigger_name;
```

```
        // create a local instance of the DW worker
        DwWorker worker = new DwWorker();

        // connect to the module, logon, and initialize the worker
        worker.Connect("secure://192.168.2.176");
        worker.Login("admin", "admin");
        worker.InitializeWorker();

        // set the project name from the command line and then start it
        project_name = String.Copy( args[0] );
        rc = worker.Project.Start( project_name, timeout );
        if ( rc == 0 ) Console.WriteLine("Project=" + project_name + " start operation, rc = " + rc);

        // set the trigger name from the command line and then start it
        trigger_name = String.Copy( args[1] );
        rc = worker.Trigger.Start( project_name, trigger_name, timeout );
        if ( rc == 0 )
           Console.WriteLine("Trigger=" + trigger_name + " start operation, rc = " + rc);
        else
           Console.WriteLine("Trigger=" + trigger_name + " start operation failed, rc = " + rc);

        // let the trigger start up completely
        Thread.Sleep( 5000 );

        // execute the trigger now
        rc = worker.Trigger.FireNow( project_name, trigger_name, timeout );
        if ( rc == 0 ) Console.WriteLine("Trigger=" + trigger_name + " executed, rc = " + rc);

        // stop the trigger
        rc = worker.Trigger.Stop( project_name, trigger_name, timeout );
        if ( rc == 0 )
           Console.WriteLine("Trigger=" + trigger_name + " stop operation, rc = " + rc);
        else
           Console.WriteLine("Trigger=" + trigger_name + " stop operation failed, rc = " + rc);

         // stop the project
         rc = worker.Project.Stop( project_name, timeout );
         if ( rc == 0 ) Console.WriteLine("Project=" + project_name + " stop operation, rc = " + rc);

        worker.Disconnect();
      }
   }
}
```

Sample Application: Devices

```
//============================================================
//
// Program Name:  SampleStartDevice.cs
//                      C# Example Application for dwclient.dll (Library Component)
//
// Description:   This CSharp application demonstrates the following deviceWISE
// functions:
//
//                      (1) Start a device
//                      (2) Read a single device value
//                      (3) Write a single device value
//                      (4) Stop a device
//
// Command Line Parameters:
//
//              The single parm on the command line is the value to write to the device.
//
//              For example,
//
//                   SampleStartDevice  1234
//
//              will read the current value, then write 1234, and then read the new value back.
//
// To build this sample application, execute the CSharp command line compiler as shown
//
//               csc  /t:exe  /r:dwclient.dll   SampleStartDevice.cs
//
// This Application and the associated dwclient library requires the
// Microsoft .NET Framework 2.0 or above.
// The libraries were built using the Microsoft .NET SDK 2.0 SP2 platform.
//
// Copyright(c) 2009.  ILS Technology, LLC.  All Rights Reserved.
//
//============================================================
// The sample program is provided to you on an "AS IS" basis, without warranty of any kind.
// ILS TECHNOLOGY HEREBY  EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER
EXPRESS
// OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
// MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
// Some jurisdictions do not allow for the exclusion or limitation of implied warranties,
// so the above limitations or exclusions may not apply to you.
// ILS TECHNOLOGY shall not be liable for any damages you suffer as a result of using,
// modifying or distributing the sample program or its derivatives.
//
//============================================================
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using DeviceWISE;

namespace sample1
```

```
{
   class Program
   {
      static void Main(string[ ] args)
      {
         int      rc                = 0;
         int      start_delay    = 5000;
         int      timeout           = 3000;
         int      data_count;
         int      data_length;
         string  data_value;

         string  var_name;
         string  device_name = "Local CPU 1";

         // pick up the data value to write from the command line
         data_value = String.Copy( args[0] );

         // create a local instance of the DW worker
         DwWorker worker = new DwWorker();

         // connect to the module, logon, and initialize the worker
         worker.Connect("secure://192.168.2.176");
         worker.Login("admin", "admin");
         worker.InitializeWorker();



         // *** start the device
         rc = worker.Device.Start( device_name, timeout );
         if ( rc == 0 )
            Console.WriteLine("Device=" + device_name +  " started, rc = " + rc);
         else
            Console.WriteLine("Device start failed, rc = " + rc);

         // *** let the device start operation complete
         Thread.Sleep( start_delay );

         DwWorker.DwVariable.VariableReadEntry vre = null;

         // *** read the current value and print it
         data_count  = 1;
         data_length = -1;
         var_name = "D[10]";
          rc = worker.Variable.Read( out vre, device_name, var_name, data_count, data_length,
         DwDataType.INT2, timeout );

         if ( rc == 0 ) System.Console.WriteLine("Read Current Value = " +
         vre.toINT2().ToString());

         DwWorker.DwVariable.VariableWriteEntry vrw = null;

         // *** write the new scalar value from command line
         data_count  = 1;
         data_length = -1;
         var_name = "D[10]";
```

```
            rc = worker.Variable.Write( out vrw,  device_name, var_name, data_count, data_length,
            DwDataType.INT2, data_value, timeout );

            if ( rc == 0 )
                System.Console.WriteLine( "Wrote New Updated Value = " + data_value );

            // *** read the current value and print it
            data_count  = 1;
            data_length = -1;
            var_name = "D[10]";
            rc = worker.Variable.Read( out vre, device_name, var_name, data_count, data_length,
            DwDataType.INT2, timeout );

            if ( rc == 0 )
              System.Console.WriteLine("Read Current Value = " + vre.toINT2().ToString());

            rc = worker.Device.Stop( device_name, timeout );
            if ( rc == 0 )
                Console.WriteLine("Device=" + device_name +  " stopped, rc = " + rc);
            else
                Console.WriteLine("Device=" + device_name +  " stop failed, rc = " + rc);

            // disconnect from deviceWISE
            worker.Disconnect();
        }
    }
}
```

Sample Application: Variables

```
//=====================================================================
//
// Program Name:  SampleReadWriteScalar.cs
//                      C# Example Application for dwclient.dll (Library Component)
//
// Description:   This CSharp application demonstrates reading/writing scalar values
//                     for INT2, INT4, FLOAT4, STRING, and BOOL datatypes
//
// Command Line Parameters:
//
//              The five parms on the command line are the values to write to the device
//               in the following order:    INT2   INT4   FLOAT4   STRING(even length)    BOOL
//
//              For example,
//
//                   SampleReadWriteScalar 12  2468   3.14  ABCD   1
//
// To build this sample application, execute the CSharp command line compiler as shown
//
//               csc  /t:exe  /r:dwclient.dll   SampleReadWriteScalar.cs
//
// This Application and the associated dwclient library requires the
// Microsoft .NET Framework 2.0 or above.
// The libraries were built using the Microsoft .NET SDK 2.0 SP2 platform.
//
// Copyright(c) 2009.  ILS Technology, LLC.  All Rights Reserved.
//
//=====================================================================
//
// The sample program is provided to you on an "AS IS" basis, without warranty of any kind.
// ILS TECHNOLOGY HEREBY  EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER
EXPRESS
// OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
// MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
// Some jurisdictions do not allow for the exclusion or limitation of implied warranties,
// so the above limitations or exclusions may not apply to you.
// ILS TECHNOLOGY shall not be liable for any damages you suffer as a result of using,
// modifying or distributing the sample program or its derivatives.
//
//=====================================================================
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using DeviceWISE;

namespace sample1
{
   class Program
   {
      static void Main(string[ ] args)
      {
         int      rc                  = 0;
```

```
        int      timeout          = 3000;
        int      data_count;
        int      data_length;
        int      save_data_length;
        string  data_value;
        string  var_name;
        string  device_name = "Local CPU 1";

        System.Console.WriteLine(" ");
         System.Console.WriteLine("Command line has data values written:  INT2   INT4
         FLOAT4   STRING(even length)    BOOL");
        System.Console.WriteLine(" ");

        // create a local instance of the DW worker
        DwWorker worker = new DwWorker();

        // connect to the module, logon, and initialize the worker
        worker.Connect("secure://192.168.2.176");
        worker.Login("admin", "admin");
        worker.InitializeWorker();

        DwWorker.DwVariable.VariableWriteEntry vwe = null;

//*****************************************************************************************
//  write scalar values for INT2, INT4, FLOAT4, STRING, and BOOL
//*****************************************************************************************

        // *** write value for INT2
        data_count  = 1;
        data_length = -1;
        var_name = "D[10]";
        data_value = String.Copy( args[0] );
         rc = worker.Variable.Write( out vwe,  device_name, var_name, data_count, data_length,
         DwDataType.INT2, data_value, timeout );
        if ( rc == 0 )
            System.Console.WriteLine("Wrote  " + var_name +  "  Value = " + data_value );

        // *** write value for INT4
        data_count  = 1;
        data_length = -1;
        var_name = "D[12]";
        data_value = String.Copy( args[1] );
         rc = worker.Variable.Write( out vwe,  device_name, var_name, data_count, data_length,
         DwDataType.INT4, data_value, timeout );
        if ( rc == 0 )
            System.Console.WriteLine("Wrote  " + var_name +  "  Value = " + data_value );

        // *** write value for FLOAT4
        data_count  = 1;
        data_length = -1;
        var_name = "D[14]";
        data_value = String.Copy( args[2] );
         rc = worker.Variable.Write( out vwe,  device_name, var_name, data_count, data_length,
         DwDataType.FLOAT4, data_value, timeout );
        if ( rc == 0 )
            System.Console.WriteLine("Wrote  " + var_name +  "  Value = " + data_value );
```

```
        // *** write value for STRING
        data_count  = 1;
        var_name = "D[16]";
        data_value = String.Copy( args[3] );
        data_length = data_value.Length;
        save_data_length = data_length;   // used for subsequent read operation
         rc = worker.Variable.Write( out vwe,  device_name, var_name, data_count, data_length,
         DwDataType.STRING, data_value, timeout );
        if ( rc == 0 )
            System.Console.WriteLine("Wrote  " + var_name +  "  Value = " + data_value );

        // *** write value for BOOL
        data_count  = 1;
        data_length = -1;
        var_name = "M[10]";
        data_value = String.Copy( args[4] );
         rc = worker.Variable.Write( out vwe,  device_name, var_name, data_count, data_length,
         DwDataType.BOOL, data_value, timeout );
        if ( rc == 0 )
            System.Console.WriteLine("Wrote  " + var_name +  "  Value = " + data_value );

        System.Console.WriteLine(" ");

//*****************************************************************************************
//  read values back for INT2, INT4, FLOAT4, STRING, and BOOL
//*****************************************************************************************

        DwWorker.DwVariable.VariableReadEntry vre1 = null;
        // ********************* read INT2
        data_count  = 1;
        data_length = -1;
        var_name = "D[10]";
         rc = worker.Variable.Read( out vre1, device_name, var_name, data_count, data_length,
         DwDataType.INT2, timeout );
         if ( rc == 0 )
            System.Console.WriteLine(" Read  " + var_name +  "  Value = " +
                  vre1.toINT2( ).ToString( ) );

        DwWorker.DwVariable.VariableReadEntry vre2 = null;
        // ********************* read INT4
        data_count  = 1;
        data_length = -1;
        var_name = "D[12]";
         rc = worker.Variable.Read( out vre2, device_name, var_name, data_count, data_length,
         DwDataType.INT4, timeout );
         if ( rc == 0 )
            System.Console.WriteLine(" Read  " + var_name +  "  Value = " +
            vre2.toINT4( ).ToString( ) );
```

```
        DwWorker.DwVariable.VariableReadEntry vre3 = null;
        // ********************* read FLOAT4
        data_count  = 1;
        data_length = -1;
        var_name = "D[14]";
         rc = worker.Variable.Read( out vre3, device_name, var_name, data_count, data_length,
         DwDataType.FLOAT4, timeout );
         if ( rc == 0 )
            System.Console.WriteLine(" Read  " + var_name +  "  Value = " +
            vre3.toFLOAT4( ).ToString( ) );


        DwWorker.DwVariable.VariableReadEntry vre4 = null;
        // ********************* read STRING
        data_count  = 1;
        data_length = save_data_length;
         var_name = "D[16]";
         rc = worker.Variable.Read( out vre4, device_name, var_name, data_count, data_length,
         DwDataType.STRING, timeout );
         if ( rc == 0 )
            System.Console.WriteLine(" Read  " + var_name +  "  Value = " +
            vre4.toSTRING( ).ToString( ) );


        DwWorker.DwVariable.VariableReadEntry vre5 = null;
        // ********************* read BOOL
        data_count  = 1;
        data_length = -1;
        var_name = "M[10]";
         rc = worker.Variable.Read( out vre5, device_name, var_name, data_count, data_length,
         DwDataType.BOOL, timeout );
         if ( rc == 0 )
            System.Console.WriteLine(" Read  " + var_name +  "  Value = " +
            vre5.toBOOL( ).ToString( ) );

        System.Console.WriteLine(" ");

        // disconnect from deviceWISE
        worker.Disconnect();
      }
   }
}
```

Sample Application: Reading/Writing Lists of Variables

```
//=========================================================================
//
// Program Name:  SampleList.cs
//                      C# Example Application for dwclient.dll (Library Component)
//
// Description:   This CSharp application demonstrates writing/reading lists of variables
//
// To build this sample application, execute the CSharp command line compiler as shown
//
//              csc  /t:exe  /r:dwclient.dll   SampleList.cs
//
// This Application and the associated dwclient library requires the
// Microsoft .NET Framework 2.0 or above.
// The libraries were built using the Microsoft .NET SDK 2.0 SP2 platform.
//
// Copyright(c) 2009.  ILS Technology, LLC.  All Rights Reserved.
//
//=========================================================================
//
// The sample program is provided to you on an "AS IS" basis, without warranty of any kind.
// ILS TECHNOLOGY HEREBY  EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER
EXPRESS
// OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
// MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
// Some jurisdictions do not allow for the exclusion or limitation of implied warranties,
// so the above limitations or exclusions may not apply to you.
// ILS TECHNOLOGY shall not be liable for any damages you suffer as a result of using,
// modifying or distributing the sample program or its derivatives.
//
//=========================================================================
using System;
using System.Collections.Generic;
using System.Text;
using DeviceWISE;

namespace test {
   class Program {
      static void Main(string[] args) {
         DwWorker worker = new DwWorker();
         worker.Connect("secure://192.168.2.176");
         worker.Login("admin", "admin");
         worker.InitializeWorker();

         // *** create a new list of variables to write
         List<DeviceWISE.DwWorker.DwVariable.VariableWrite> writeVariables =
          new List<DwWorker.DwVariable.VariableWrite>();
```

```
            // *** add 3 variables to the list to write
            writeVariables.Add(new DwWorker.DwVariable.VariableWrite("D[10]",   1, -1,
DwDataType.INT2, "10"));
            writeVariables.Add(new DwWorker.DwVariable.VariableWrite("D[20]",   3, -1,
DwDataType.INT2, "4,5,6"));
            writeVariables.Add(new DwWorker.DwVariable.VariableWrite("D[30]",   1,  4,
DwDataType.STRING, "abcd"));

           // *** write the list of variables to the device
           List<DeviceWISE.DwWorker.DwVariable.VariableWriteEntry> writeResults;
           worker.Variable.Write(out writeResults, "Local CPU 1", writeVariables, -1);

           foreach ( DeviceWISE.DwWorker.DwVariable.VariableWriteEntry variable in writeResults )
               {
               Console.Out.WriteLine("Wrote to " + variable.Name + " with status " + variable.Status);
               }


            // *** create a new list of variables to read
             List<DeviceWISE.DwWorker.DwVariable.VariableRead> readVariables = new
             List<DwWorker.DwVariable.VariableRead>();

             readVariables.Add(new DwWorker.DwVariable.VariableRead("D[10]",  1, -1,
             DwDataType.INT2));
             readVariables.Add(new DwWorker.DwVariable.VariableRead("D[20]",  3, -1,
             DwDataType.INT2));
             readVariables.Add(new DwWorker.DwVariable.VariableRead("D[30]",  1,  4,
             DwDataType.STRING));

            // *** read the list of variables from the device
            List<DeviceWISE.DwWorker.DwVariable.VariableReadEntry> readResults;
            worker.Variable.Read(out readResults, "Local CPU 1", readVariables, -1);

            Console.Out.WriteLine("Read " + readResults[0].Name + " as value " +
            readResults[0].toINT2());

            Console.Out.Write("Read " + readResults[1].Name + " with values: ");
            foreach ( int value in readResults[1].toINT2Array() )
               Console.Out.Write(value + " ");
            Console.Out.WriteLine();

            Console.Out.WriteLine("Read " + readResults[2].Name + " as value " +
            readResults[2].toSTRING());

            worker.Disconnect();
        }
    }
}
```