PROGRAMMER'S MANUAL

LoggerNet and LNServer Software Development Kits 4.5

Revision: 9/17

Campbell Scientific, Inc. Software SDK End User License Agreement (EULA)

NOTICE OF AGREEMENT: Please carefully read this EULA. By installing or using this software, you are agreeing to comply with the terms and conditions herein. If you do not want to be bound by this EULA, you must promptly return the software, any copies, and accompanying documentation in its original packaging to Campbell Scientific or its representative.

This agreement applies equally to the LoggerNet SDK software and the LNServer SDK software. The LoggerNet SDK and LNServer SDK software are hereinafter referred to, synonymously, as the SDK. The term "developer" herein refers to anyone using the SDK.

By accepting this agreement you acknowledge and agree that Campbell Scientific may from time-to-time, and without notice, make changes to one or more components of the SDK, or make changes to one or more components of other software on which the SDK relies. In no instance will Campbell Scientific be responsible for any costs or liabilities incurred by you or other third parties as a result of these changes.

LICENSE FOR USE: Campbell Scientific grants you a non-exclusive, non-transferable, royalty-free license to use this software in accordance with the following:

1) The purchase of this software allows you to install and use a single instance of the software on one physical computer or one virtual machine only.
2) This software cannot be loaded on a network server for the purposes of distribution or for access to the software by multiple operators. If the software can be used from any computer other than the computer on which it is installed, you must license a copy of the software for each additional computer from which the software may be accessed.
3) If this copy of the software is an upgrade from a previous version, you must possess a valid license for the earlier version of software. You may continue to use the earlier copy of software only if the upgrade copy and earlier version are installed and used on the same computer. The earlier version of software may not be installed and used on a separate computer or transferred to another party.
4) This software package is licensed as a single product. Its component parts may not be separated for use on more than one computer.
5) You may make one (1) backup copy of this software onto media similar to the original distribution, to protect your investment in the software in case of damage or loss. This backup copy can be used only to replace an unusable copy of the original installation media.
6) You may not use Campbell Scientific’s name, trademarks, or service marks in connection with any program you develop with the SDK. You may not state or infer in any way that Campbell Scientific endorses any program you develop, unless prior written approval is received from Campbell Scientific.
7) If the software program you develop requires you, your customer, or a third party to use additional licensable software from Campbell Scientific that software must be purchased from Campbell Scientific or its representative under the terms of its separate EULA.
8) This license allows you to redistribute the ActiveX (dll) controls and the communication DLL with the software developed using the SDK. No other Campbell Scientific examples, documentation, or source code may be distributed with your application.
9) The SDK may not be used to develop and publicly sell or distribute any product that directly competes with Campbell Scientific’s datalogger support software.
10) This Agreement does not give Developer the right to sell or distribute any other Campbell Scientific, Inc. Software (e.g., PC200W, VisualWeather, LoggerNet or any of their components, files, documentation, etc.) as part of Developer's application. Distribution of any other Campbell Scientific, Inc. software requires a separate distribution agreement.

The ActiveX® controls provided with the SDK include the files: CsiBrokerMap.dll, CsiCoraScript.dll, CsiDatalogger.dll, CsiDataSource.dll, CsiLogMonitor.dll, CsiServerDirect.dll (LoggerNet SDK only) and CsiServer.dll (LNServer SDK only). In

addition, a limited communication server DLL, CORALIB3D.DLL (LoggerNet SDK only), or an unlimited communication server DLL, CORALIB3.DLL (LNServer SDK only), is included with the SDK.

RELATIONSHIP: Campbell Scientific, Inc. hereby grants a license to use the SDK Controls in accordance with the license statement above. No ownership in Campbell Scientific, Inc. patents, copyrights, trade secrets, trademarks, or trade names is transferred by this Agreement. Developer may use these SDK controls to create as many applications as desired and freely distribute those applications. Campbell Scientific, Inc. expects no royalties or any other compensation outside of the SDK purchase price. Developer is responsible for supporting applications created using the SDK Controls.

RESPONSIBILITIES OF DEVELOPER

The Developer agrees:

- To provide a competent programmer familiar with Campbell Scientific, Inc. datalogger programming and software to write the applications.
- Not to sell or distribute documentation on use of the SDK Controls.
- Not to sell or distribute the applications that are provided as examples in the SDK.
- To develop original works. Developers may copy and paste portions of the code into their own applications, but their applications are expected to be unique creations.
- Not to sell or distribute applications that compete directly with any application developed by Campbell Scientific, Inc. or its affiliates.
- To assure that each application developed with the SDK Controls clearly states the name of the person or entity that developed the application. This information should appear on the first window the user will see.

WARRANTY

There is no written or implied warranty provided with the SDK software other than as stated herein. Developer agrees to bear all warranty responsibility of any derivative products distributed by Developer.

TERMINATION

Any license violation or breach of Agreement will result in immediate termination of the developer's rights herein and the return of all SDK materials to Campbell Scientific, Inc.

MISCELLANEOUS

Notices required hereunder shall be in writing and shall be given by certified or registered mail, return receipt requested. Such notice shall be deemed given in the case of certified or registered mail on the date of receipt. This Agreement shall be governed and construed in accordance with the laws of the State of Utah, USA. Any dispute resulting from this Agreement will be settled in arbitration.

This Agreement sets forth the entire understanding of the parties and supersedes all prior agreements, arrangements and communications, whether oral or written pertaining to the subject matter hereof. This agreement shall not be modified or amended except by the mutual written agreement of the parties. The failure of either party to enforce any of the provisions of this Agreement shall not be construed as a waiver of such provisions or of the right of such party thereafter to enforce each and every provision contained herein. If any term, clause, or provision contained in this Agreement is declared or held invalid by a court of competent jurisdiction, such declaration or holding shall not affect the validity of any other term, clause, or provision herein contained. Neither the rights nor the obligations arising under this Agreement are assignable or transferable.

If within 30 days of receiving the SDK product developer does not agree to the terms of license, developer shall return all materials without retaining any copies of the product and shall remove any use of the SDK Controls in any applications developed or distributed by Developer. CSI shall refund 1/2 of the purchase price within 30 days of receipt of the materials. In the absence of such return, CSI shall consider developer in agreement with the herein stated license terms and conditions.

Limited Guarantee

The following warranties are in effect for ninety (90) days from the date of shipment of the original purchase. These warranties are not extended by the installation of upgrades or patches offered free of charge.

Campbell Scientific warrants that the installation media on which the software is recorded and the documentation provided with it are free from physical defects in materials and workmanship under normal use. The warranty does not cover any installation media that has been damaged, lost, or abused. You are urged to make a backup copy (as set forth above) to protect your investment. Damaged or lost media is the sole responsibility of the licensee and will not be replaced by Campbell Scientific.

Campbell Scientific warrants that the software itself will perform substantially in accordance with the specifications set forth in the instruction manual when properly installed and used in a manner consistent with the published recommendations, including recommended system requirements. Campbell Scientific does not warrant that the software will meet licensee's requirements for use, or that the software or documentation are error free, or that the operation of the software will be uninterrupted.

Campbell Scientific will either replace or correct any software that does not perform substantially according to the specifications set forth in the instruction manual with a corrected copy of the software or corrective code. In the case of significant error in the installation media or documentation, Campbell Scientific will correct errors without charge by providing new media, addenda, or substitute pages. If Campbell Scientific is unable to replace defective media or documentation, or if it is unable to provide corrected software or corrected documentation within a reasonable time, it will either replace the software with a functionally similar program or refund the purchase price paid for the software.

All warranties of merchantability and fitness for a particular purpose are disclaimed and excluded. Campbell Scientific shall not in any case be liable for special, incidental, consequential, indirect, or other similar damages even if Campbell Scientific has been advised of the possibility of such damages. Campbell Scientific is not responsible for any costs incurred as a result of lost profits or revenue, loss of use of the software, loss of data, cost of re-creating lost data, the cost of any substitute program, telecommunication access costs, claims by any party other than licensee, or for other similar costs.

This warranty does not cover any software that has been altered or changed in any way by anyone other than Campbell Scientific. Campbell Scientific is not responsible for problems caused by computer hardware, computer operating systems, or the use of Campbell Scientific's software with non-Campbell Scientific software.

Licensee's sole and exclusive remedy is set forth in this limited warranty. Campbell Scientific's aggregate liability arising from or relating to this agreement or the software or documentation (regardless of the form of action; e.g., contract, tort, computer malpractice, fraud and/or otherwise) is limited to the purchase price paid by the licensee.

PLEASE READ FIRST

About this manual

Please note that this manual was originally produced by Campbell Scientific Inc. primarily for the North American market. Some spellings, weights and measures may reflect this origin.

Some useful conversion factors:

Area: 1 in^2 (square inch) = 645 mm^2

Length: 1 in. (inch) = 25.4 mm
1 ft (foot) = 304.8 mm
1 yard = 0.914 m
1 mile = 1.609 km

Mass: 1 oz. (ounce) = 28.35 g
1 lb (pound weight) = 0.454 kg

Pressure: 1 psi (lb/in^2) = 68.95 mb

Volume: 1 UK pint = 568.3 ml
1 UK gallon = 4.546 litres
1 US gallon = 3.785 litres

In addition, while most of the information in the manual is correct for all countries, certain information is specific to the North American market and so may not be applicable to European users.

Differences include the U.S standard external power supply details where some information (for example the AC transformer input voltage) will not be applicable for British/European use. *Please note, however, that when a power supply adapter is ordered it will be suitable for use in your country.*

Reference to some radio transmitters, digital cell phones and aerials may also not be applicable according to your locality.

Some brackets, shields and enclosure options, including wiring, are not sold as standard items in the European market; in some cases alternatives are offered. Details of the alternatives will be covered in separate manuals.

Part numbers prefixed with a "#" symbol are special order parts for use with non-EU variants or for special installations. Please quote the full part number with the # when ordering.

Recycling information

At the end of this product's life it should not be put in commercial or domestic refuse but sent for recycling. Any batteries contained within the product or used during the products life should be removed from the product and also be sent to an appropriate recycling facility.

Campbell Scientific Ltd can advise on the recycling of the equipment and in some cases arrange collection and the correct disposal of it, although charges may apply for some items or territories.

For further advice or support, please contact Campbell Scientific Ltd, or your local agent.

Campbell Scientific Ltd, 80 Hathern Road, Shepshed, Loughborough, LE12 9GX, UK
Tel: +44 (0) 1509 601141 Fax: +44 (0) 1509 601091
Email: support@campbellsci.co.uk
www.campbellsci.co.uk

Table of Contents

PDF viewers: These page numbers refer to the printed version of this document. Use the PDF reader bookmarks tab for links to specific sections.

Figures

Table

VB.NET Examples

Section 1. LoggerNet SDK and LNServer SDK Overview

This document serves as a programmer's reference for two Campbell Scientific software products; the LoggerNet SDK and the LNServer SDK. The products differ only in the functionality of the LoggerNet Server DLL supplied with each, and the unique ActiveX® control required to start and stop the respective server. The remainder of the supplied components are common to each product and are identical in their function, operation and use. All components are documented herein.

Hereafter, except where noted, the term SDK is used in reference to both products synonymously.

1.1 Purpose of the SDK

The ActiveX® controls comprising the SDK encapsulate the proprietary messaging protocol used between the LoggerNet server and client applications. These controls provide a means for developing applications that incorporate the functionality of a LoggerNet server without the need to understand the intricacies of the messaging protocol. Not only is development time reduced, but applications are insulated from future changes to the messaging protocol.

It is important to understand that it is the LoggerNet server that communicates directly with and collects data from a network of Campbell Scientific dataloggers. The SDK provides the means with which an application is able to ascertain and define the structure of the network, manage the server's communications with the network, access the collected data, and monitor the server's operation.

1.2 Requirements

1.2.1 Required Campbell Scientific, Inc. Software

The SDK supports the development of either client or standalone applications. Client applications communicate with an independent LoggerNet server, existing either on the local host or on a remote PC, via a TCP/IP connection. Alternately, by incorporating and distributing the provided LoggerNet Server DLL, a fully autonomous application can be achieved.

LoggerNet server version 1.1 or higher is required for client applications.

1.2.2 Development Tools

The SDK's ActiveX® controls have been tested with the following development tools for Microsoft® Windows®:

TABLE 1-1. Supported Development Tools	
Development Tool	**Examples Available**
Visual C++® (MFC)*	Yes
C#.NET	Yes
VB.NET	Yes

*Information about supported Visual Studio versions is provided in the readme.txt file located in C:\Campbellsci\LoggerNetSDK or C:\Campbellsci\LoggerNetServerSDK.

1.3 Included Components

1.3.1 Files Included in the SDK

By default, the installation of the SDK will create an application working directory in the root of the *C:\Campbellsci* directory. The top folder in the working directory will be named *LoggerNetSDK* or *LoggerNetServerSDK*, depending on the product installed. This folder will contain three additional folders: a *Controls* folder containing the six ActiveX® controls and the LoggerNet Server DLL; an *Examples* folder containing the example Visual Studio® project files; and a *Manuals* folder containing documentation and reference manuals.

1.3.1.1 ActiveX® Controls (DLLs)

The six ActiveX® controls are implemented as *Dynamically Linked Libraries* (DLLs) and are registered on the development host by default during the installation of the SDK. Any controls used in the development of an application must also be registered on the application host. The latter can be accomplished as part of an installation program or done manually using the *RegSvr32.exe* utility installed with the Windows OS.

Each of the six controls provides specific functionality and, as demonstrated in the example projects, can work independently from the others. Depending on the application requirements, only a few or all of the controls may be needed.

1.3.1.2 LoggerNet Server DLL

Fundamentally, the LoggerNet Server DLL provides the core functions of datalogger communication, data collection and storage. Additionally, it functions as the ‘server’ component of a client-server architecture by exposing an API for client applications. It is this interface that is abstracted by the SDK.

The LoggerNet Server DLL does not need to be registered but must be placed in the application folder, in the PATH environmental variable, or in the Windows system directory.

Depending on the product installed, one of two versions of the LoggerNet Server DLL is included with the SDK. The versions differ in the types of telecommunication devices supported.

1.3.1.2.1 Coralib3d.dll

The Coralib3d server is installed with the LoggerNet SDK. This limited function LoggerNet server supports only direct communications with the datalogger via RS-232, USB, or TCP/IP connections. The *CsiServerDirect* control is used to start and stop this server.

1.3.1.2.2 Coralib3.dll

The full function Coralib3 server is installed with the LNServer SDK. The *CsiServer* control is used to start and stop this server.

1.3.1.3 Manuals

The SDK Beginner's Guide contains information comparing available Campbell Scientific SDK products. The *LoggerNet and LNServer SDK Programmer's Reference* (this document) contains detailed information regarding the use of the SDK. The *CoraScript Interpreter Reference* (cora_cmd.pdf) provides a command reference for use with the *CsiCoraScript* control. All manuals are in PDF format.

1.3.1.4 Example Projects

Example Microsoft Visual Studio projects are included with the SDK that demonstrate the implementation of each of the ActiveX® controls. The example applications are written in various development languages; C#, VB.NET and C++ (VisualStudio-MFC). Most of the example projects are Windows Forms applications, but a Console project that implements the *CsiCoraScript* control is included in the *C#* and *VB.NET* folders.

The examples are intended to demonstrate how the SDK controls can be used in a typical application. While the examples do exercise the core functionality of each control, not every attribute is utilized. This is most true with regard to the multiple interfaces of the *CsiDataSource* control. However, a competent developer should have no difficulty in extrapolating the examples into a highly functional, custom application.

1.4 Developing .NET Applications Using the SDK

The ActiveX® components of the SDK are built on the Component Object Model (COM) architecture and expose a COM interface. To enable COM interoperability, the .NET Framework utilizes a *Runtime Callable Wrapper* (RCW). The RCW infrastructure enables communications between the .NET application and the COM interface, and provides data type marshaling and event handling.

In Microsoft's Visual Studio, the SDK components can be imported into a .NET Windows Forms application via one of two methods: by simply dropping the ActiveX® control onto the form or by adding a reference to the component type library. Either method will create a RCW class that, once instantiated, can be accessed like any other .NET object. The latter method was employed in the development of the .NET examples included with the SDK and in the example code illustrated in this document.

NOTE The ActiveX® controls in the SDK must run in a 32-bit process on 64-bit machines. Compiler options should be set to target the x86 platform.

1.4.1 Adding an SDK Control to a .NET Project

Before a control can be added to or referenced in a project, the control must be added to the Visual Studio *Toolbox*. In VS2012 or later, this is accomplished by right-clicking in the *Toolbox* and selecting **Choose Items** from the shortcut menu. In the resulting *Choose Toolbox Items* dialogue box, select the **COM Components** tab and check the SDK control(s) required by the project as shown in FIGURE 1-1.

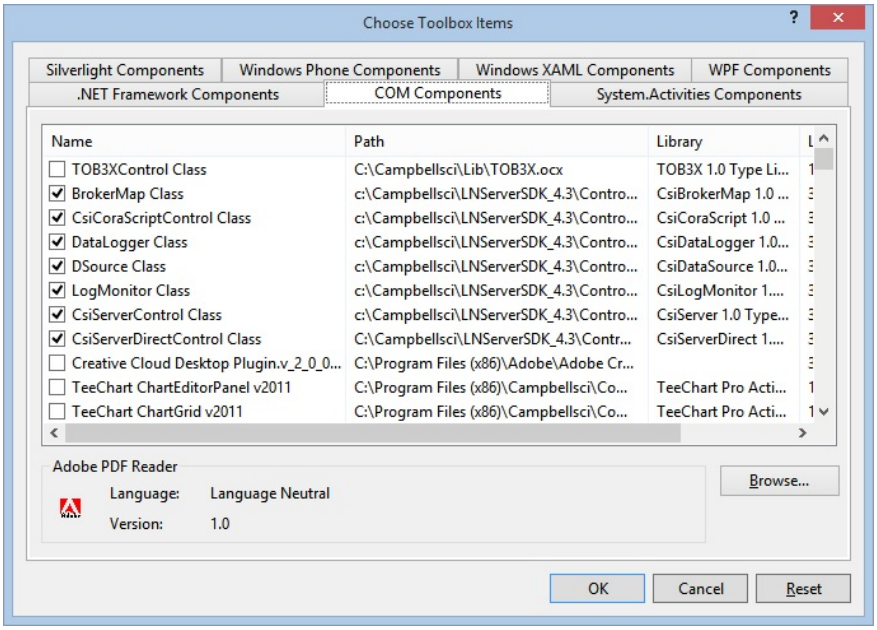

FIGURE 1-1. Adding an SDK Control to the Toolbox

1.4.2 Creating the *Runtime Callable Wrapper*

The RCW can be created by invoking the *tlbimp* utility from a command line or via Visual Studio.

To have Visual Studio create the RCW, a reference to the type library of the control must be added to the project. To invoke the *Reference Manager*, right-click the project in the *Solution Explorer* and select **Add Reference**. In the *Reference Manager* dialogue, select the **COM** tab and then **Type Libraries**. From the list of libraries, select the required control as shown in FIGURE 1-2.

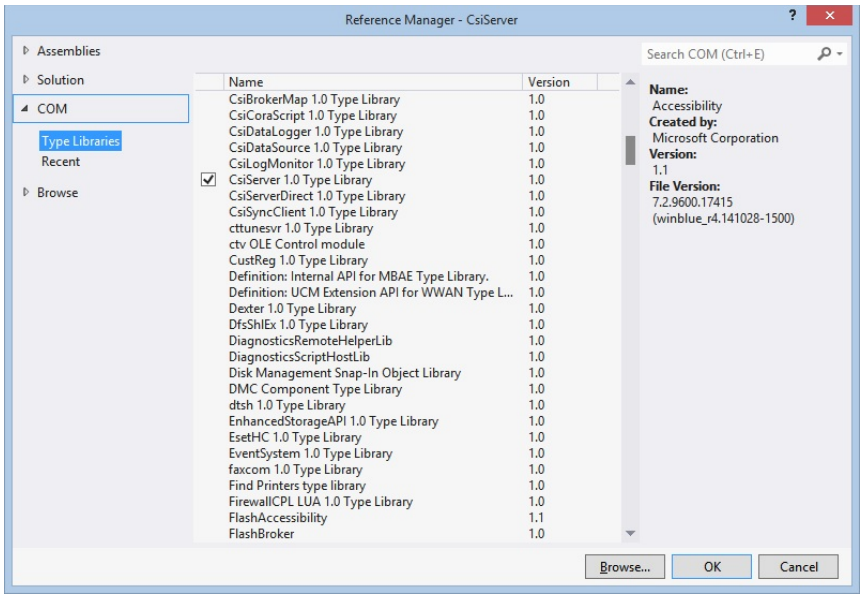

FIGURE 1-2. Adding a Reference to the SDK control's Type Library

For this example, clicking the **OK** button will create an RCW class named *Interop.CsiServerlib*. A *CsiServerLib* reference will be added to the *References* folder in the *Solution Explorer* and the file *Interop.CsiServerLib.dll* will be added to the *obj* folder.

NOTE By default, Visual Studio will embed the wrapper class within the main assembly of the project. However, for proper runtime functionality of the SDK controls, the RCW must be distributed with the application as a separate assembly.

To have Visual Studio create the RCW as a separate assembly, ensure that the **Embed Interop Types** property of the wrapper class is set to **False** before building the project. See FIGURE 1-3.

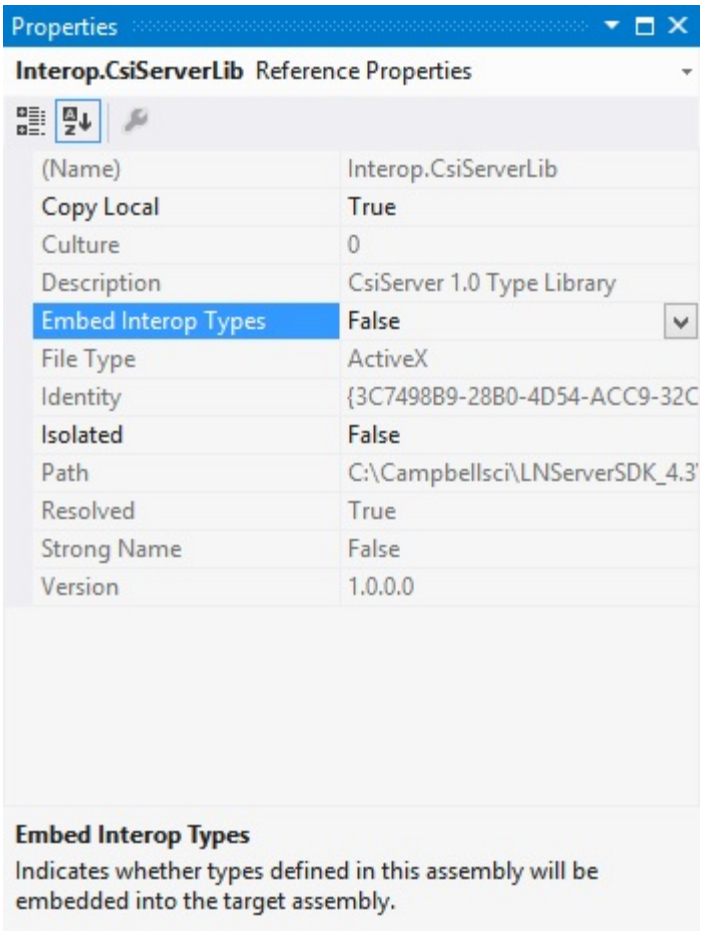

FIGURE 1-3. Wrapper Class Properties

Section 2. CsiServer and CsiServerDirect Controls

2.1 Purpose of the *CsiServer* and *CsiServerDirect* Controls

The *CsiCoraScript*, *CsiBrokerMap*, *CsiDatalogger*, *CsiDataSource* and *CsiLogMonitor* SDK controls must connect to and communicate with a LoggerNet server. Therefore, a LoggerNet server must be running on the network. In lieu of an existing LoggerNet server, the *CsiServer* and *CsiServerDirect* controls allow an application to start and stop the respective LoggerNet Server DLL included with the SDK (see Section 1.3.1.2, *LoggerNet Server DLL* *(p. 1-2)*).

Some consideration should be given to the type of software application required before beginning a project using the SDK. Campbell Scientific sells a complete *LoggerNet* software package that includes the LoggerNet server and many complex software clients. Many developers merely want to create a custom software interface that extends an existing *LoggerNet* installation. The included LoggerNet Server DLL will not be required for this type of application. However, if a standalone software solution is required that will replace or be used instead of Campbell Scientific's *LoggerNet* software package, the *CsiServer* or *CsiServerDirect* control will be required to activate the included LoggerNet Server DLL.

2.2 *CsiServer* and *CsiServerDirect* Interface

With the exception of their names and the respective LoggerNet Server DLL that each activates, the *CsiServer* and *CsiServerDirect* controls are identical. They each expose the following interface. See Section 14, *CsiServer and CsiServerDirect Control Reference* *(p. 14-1)*, for detailed descriptions of these properties, methods, and events.

2.2.1 Properties

- applicationWorkDir As String *(p. 14-1)*
- buildDate As String (read-only) *(p. 14-1)*
- logFileDir As String *(p. 14-2)*
- serverStarted As Boolean (read-only) *(p. 14-2)*
- serverVersion As String (read-only) *(p. 14-2)*
- serverWorkDir As String (Required) *(p. 14-3)*
- tcpPort As Integer *(p. 14-3)*
- tcpPortEx As Long *(p. 14-4)*

2.2.2 Methods

- startServer() *(p. 14-4)*
- stopServer() *(p. 14-5)*

2.2.3 Events

- onServerFailure (String reason) *(p. 14-5)*

Section 3. Developing an Application Using the CsiServer Control

3.1 Purpose

This section shows by example how to build a simple application using the SDK *CsiServer* control. Due to the functional similarities and identical interface, this section also serves as an example for developing an application using the *CsiServerDirect* control. The application's functions are:

1. Start the LoggerNet Server DLL (Coralib3.dll).
2. Display the functional status of the server.
3. Retrieve and display the version number of the LoggerNet Server DLL.
4. Stop the LoggerNet Server DLL.

3.2 Using the *CsiServer* Control

3.2.1 Getting Started with the *CsiServer* Control

This example assumes that:

- The *CsiServer* control has been correctly registered on the application host.
- The Coralib3.dll exists in the application folder, the PATH environmental variable, or the Windows® system directory.
- A Windows Forms application is to be developed using the Visual Studio® 2012 (or later) IDE, and the VB.NET programming language.

Complete the following steps first:

1. Start Visual Studio and create a new Visual Basic® Windows Forms Application targeting the .NET Framework 4.0.
2. Following the procedures outlined in Section 1.4, *Developing .NET Applications Using the SDK (p. 1-3)*, add the *CsiServer* control to the *Toolbox* and create the RCW class.
3. In the Solution Explorer, right-click the *Form1.vb* file and rename it *CsiServerForm.vb*.

3.2.2 *CsiServer* Control Application Example

Begin by modifying the blank form to create a Graphical User Interface (GUI) that supports the required functionality. The finished form should resemble the example shown in FIGURE 3-1.

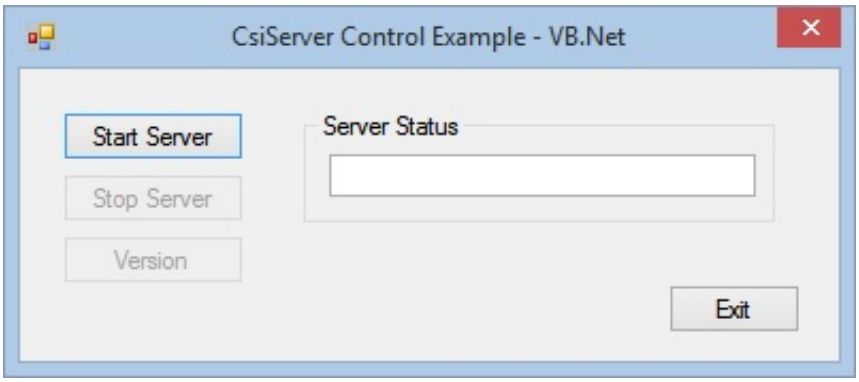

FIGURE 3-1. CsiServer Example

Once the interface has been designed, the necessary Visual Basic code can be added to attain the required functionality. The first order of business is to instantiate the RCW class (*CsiServerLib.CsiServerControl*). Add the following code to the *CsiServerForm* class.

```
Dim WithEvents csiServer As New CsiServerControl
```

Key elements of the application's file structure are defined by properties of the *CsiServer* control. The Application Working Directory (*applicationWorkDir*) is where the server will write data files. The server's configuration file (*CsiLgrNet.xml*) is maintained in the Server Working Directory (*serverWorkDir*). The server will write its log files to the Log File Directory (*logFileDir*).

It is **required** that the Server Working Directory be specified. If the Application or Log File directories are not specified, each will default to the location of the Server Working Directory. By convention, *serverWorkDir* is specified as *c:\campbellsci\LoggerNetServerSDK\sys\bin*, but may be pointed to a location of the developer's choosing.

The server's configuration file is the repository for the server settings as well as the settings for all other devices in the network map. Initially, the network map will be empty, and the network will need to be constructed using the *CsiCoraScript* control. Alternately, the *serverWorkDir* property can be pointed to an existing configuration file; presumably, from a previous or existing *LoggerNet* installation.

The following code snippets illustrate the basic functionality of the *CsiServer* control. For more comprehensive code examples, refer to the VB.NET example project files supplied with the SDK.

The following code example illustrates starting the Server (*startSever()* method):

VB.NET Example 3-1. Starting the Server using the *startServer()* Method

```
Private Sub btnStartServer_Click(sender As Object, e As EventArgs) Handles btnStartServer.Click

  Try
    'First, check to see if the server is already started
    If csiServer.serverStarted Then
       txtStatus.Text = "Server Already Started"
    Else
        'Set required properties for the LoggerNet Server
        csiServer.serverWorkDir = "c:\campbellsci\LoggerNetServerSDK\sys\bin"
        'Start the LoggerNet Server
        csiServer.startServer()
        txtStatus.Text = "Server Started"

        'Update the form
        btnStartServer.Enabled = False
        btnStopServer.Enabled = True
        btnSvrVersion.Enabled = True
    End If

  Catch ex As Runtime.InteropServices.COMException
      'If the call to the control causes an error, a custom HRESULT will be returned.
      'This HRESULT will be captured in the InteropServices.COMException class
      'and cause the .Net Runtime to throw an exception.

      'We need to map the COMException.ErrorCode property to the values enumerated in
      'CSIServerLib.HRESULT_Errors and display the associated error.
       Dim com_ex As CsiServerLib.HRESULT_Errors = ex.ErrorCode
       MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, _
                       "A COM Exception was thrown")

  Catch ex As Exception
       MessageBox.Show(ex.Source & vbCrLf & ex.Message, "CSI Server Start: ERROR")
  End Try
End Sub
```

The following code example illustrates how to retrieve and display the server version (*serverVersion* property):

VB.NET Example 3-2. Retrieving and Displaying the Server Version using the *serverVersion* Property

```
Private Sub btnSvrVersion_Click(sender As Object, e As EventArgs) Handles btnSvrVersion.Click

  Try
      'Display the version of the server started by the CsiServer control
      MessageBox.Show("Server Version: " & csiServer.serverVersion, "CSI Server Version")

  Catch ex As Runtime.InteropServices.COMException
      'If the call to the control causes an error, a custom HRESULT will be returned.
      'This HRESULT will be captured in the InteropServices.COMException class
      'and cause the .Net Runtime to throw an exception.

      'We need to map the COMException.ErrorCode property to the values enumerated in
      'CSIServerLib.HRESULT_Errors and display the associated error.
      Dim com_ex As CsiServerLib.HRESULT_Errors = ex.ErrorCode
      MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, _
                      "A COM Exception was thrown")

  Catch ex As Exception
      MessageBox.Show(ex.Source & vbCrLf & ex.Message, "CSI Server Version Button: ERROR")
        End Try
End Sub
```

The following code example illustrates stopping the server (*stopServer* method):

VB.NET Example 3-3. Stopping the Server using the *stopServer()* Method

```
Private Sub btnStopServer_Click(sender As Object, e As EventArgs) Handles btnStopServer.Click

  Try
      'Stop the LoggerNet Server
    If csiServer.serverStarted Then
        csiServer.stopServer()
        txtStatus.Text = "Server Stopped"

        'Update the form
        btnStopServer.Enabled = False
        btnStartServer.Enabled = True
        btnSvrVersion.Enabled = False
    Else
        txtStatus.Text = "Server Already Stopped"
    End If

  Catch ex As Exception
      MessageBox.Show(ex.Source & vbCrLf & ex.Message, "CSI Server Stop: ERROR")
  End Try
End Sub
```

Add additional objects and functionality as necessary to meet the specific requirements of the application. Complete examples using the *CsiServer* and *CsiServerDirect* controls are included with the SDK installation.

Section 4. CsiCoraScript Control

4.1 Purpose of the *CsiCoraScript* Control

The *CsiCoraScript* control provides the ability to administer the LoggerNet server. There are many different settings and commands available with this control.

Specific LoggerNet server functions and tasks are set by passing *CoraScript* commands to the LoggerNet server. *CoraScript* commands execute LoggerNet server operations that include adding devices to the network map, data collection, listing table and datalogger information, and changing settings in the LoggerNet server and attached devices. *CoraScript* commands and their descriptions can be found in the *CoraScript Interpreter Reference* manual (cora_cmd.pdf) installed with the LoggerNet or LNServer SDK.

NOTE The *CsiCoraScript* control executes a single *CoraScript* command at a time. The following *CoraScript* commands are currently unsupported in the SDK: connect, disconnect, help, exit, bye, quit, and list-commands.

4.2 Connecting to the Server

There are two basic actions required for this control to connect to the LoggerNet server:

1. Set server properties:
   - serverName – The name or IP address of the LoggerNet server. The default value is localhost.
   - serverPort – The port on which the LoggerNet server is running. The default value is 6789.
   - serverLogonName (Optional) – If security has been enabled on the server, a valid logon name is required.
   - serverLogonPassword (Optional) – If security has been enabled on the server, a valid password that corresponds with a valid logon name is required.
2. Invoke the s*erverConnect()* method.

4.3 Using *CoraScript* Commands

CoraScript commands are used to setup and manipulate the LoggerNet server. A thorough knowledge of these powerful commands is recommended before attempting to make changes to settings or devices in the LoggerNet server. The following sections outline some basic commands that can be used to quickly set up devices and collect data from the network.

4.3.1 Setting up a Network

Some of the commands that can be used when initially setting up a datalogger network on the LoggerNet server include:

- add-device – used to add root ports, dataloggers, and telecommunication devices to the network map.
- set-device-setting – used to change settings of specific devices in the network map.
- delete-branch – used to remove a device and any children of a device from the network map.
- list-devices – shows the devices in the network map

The following example shows the basic *CoraScript* commands used to set up a CR10X connected directly to the LoggerNet server via RS-232:

```
add-device com-port COM1 as-child "";
add-device cr10x CR10X as-child "COM1";
```

The following example shows basic *CoraScript* commands used to set up a CR6 datalogger named MyCr6 connected to the LoggerNet server via Ethernet:

```
add-device tcp-com-port IPPort as-child "";
set-device-setting IPPort 15 192.168.25.04:6785;
add-device pakbus-port PakBusPort as-child IPPort;
add-device cr6 MyCr6 as-child PakBusPort;
```

4.3.2 Real-Time Data Display

Some developers want to display data values as quickly as they change in the datalogger. Each time a datalogger program executes, new values are written as input locations. Collecting these input locations provides a snapshot of the most recent values contained in the datalogger. The *CsiDataSource* control can be used to set up an advisor that will watch the *LoggerNet* data cache and display new or existing data values that are collected. *CoraScript* commands are used to set up the collect areas of *LoggerNet* and to enable scheduled collection of specific datalogger tables to automate the collection process.

Please note that although the commands below will enable collection of input locations from a datalogger, using input locations for real-time comparison of values can be problematic. When input locations are collected, the collection is merely a snapshot of the current values that exist in each location. If, for example, the datalogger program has not completely executed, some of the values collected may be new while other values may have not changed from the previous program execution. Please keep this information in mind if input locations are used in real-time data display or calculations. If correlating values are necessary, a better approach writes values to final storage every program execution and collects those values as quickly as possible.

4.3.2.1 Table-Data Dataloggers

The LoggerNet server, by default, creates a collect area for the **Public** or **InLocs** (Input Locations) table of table-data dataloggers such as the CR6 or CR10X-TD. The basic *CoraScript* commands that are used to enable collection and establish scheduled collection are:

- `set-collect-area-setting` – used to enable a device for collection
- `set-device-setting` – used to activate scheduled collection for a device

If you have added a CR6 named MyCr6 to the datalogger network and you have a program running on that device, the following command will enable the **Public** table for collection by activating the collect-area-setting scheduleEnabled (id = 2):

```
set-collect-area-setting MyCr6 public 2 1;
```

Every time a manual poll or any other collection occurs, data will be collected for the **Public** table of the CR6. If a *CsiDataSource* advisor has been created, it will trigger and display the new values. If you want to automate the data collection process, set the device's scheduled collection interval through the device setting collectSched (id = 5):

```
set-device-setting MyCr6 5 {1 19900101 300000 120000 3 86400000};
```

With the above setting, the LoggerNet server will automatically collect all tables enabled for collection from the CR6 every 300000 milliseconds. Once this setting is in place, the activated DataSource advisor will display updates as they are automatically collected.

4.3.2.2 Mixed-Array Dataloggers

Although the *CsiDataSource* control can create a temporary data cache to watch all input locations, mixed-array dataloggers, like the CR7 and CR10X, require additional commands to create a permanent collect area for input locations. Input Locations (InLocs) contain values that are usually stored every time the program executes. However, the LoggerNet server does not create a permanent data cache by default containing data from InLocs for a mixed-array datalogger. If a permanent collect area for InLocs is desired or only specific InLocs are needed, the collect area must be created manually in the LoggerNet server. The following commands are used to set up a permanent InLocs collect area for a mixed-array datalogger:

- create-inlocs-area – create a collect area containing specified input locations
- set-collect-area-setting – used to enable a device for collection
- set-device-setting – used to activate scheduled collection for a device

The following example sets up collection for two input locations of a CR10X by identifying the station, declaring a name for the collect area, and listing the input locations to include:

```
create-inlocs-area CR10X InLocsArea {1 "inlocs1"} {2 {inlocs2}};
```

Collect area names must always be unique. Therefore, if an attempt is made to create a collect area with exactly the same name as a collect area that already exists, the LoggerNet server will automatically index the name of the collect area being created. For example, if collect area InLocsArea already exists and an attempt is made to create another collect area with the same name, the LoggerNet server will automatically name the new collect area InLocsArea1.

To activate a collect area for collection and to automate the collection process use the following commands:

```
set-collect-area-setting CR10X InLocsArea 2 1;
set-device-setting CR10X 5 {1 19900101 300000 120000 3 86400000};
```

With the above setting, the LoggerNet server will automatically collect all tables enabled for collection from the CR10X every 300000 milliseconds. Once this setting is in place, the activated *CsiDataSource* advisor will display new data values as they are collected.

4.4 *CsiCoraScript* Interface

See Section 15, *CsiCoraScript Control Reference* ***(p. 15-1)***, for descriptions of these properties, methods, and events.

4.4.1 Properties

- serverConnected As Boolean (read-only) ***(p. 15-1)***
- serverLogonName As String ***(p. 15-1)***
- serverLogonPassword As String ***(p. 15-2)***
- serverName As String ***(p. 15-2)***
- serverPort As Long ***(p. 15-3)***

4.4.2 Methods

- executeScript(String script, Long asyncID) As String ***(p. 15-3)***
- serverConnect() ***(p. 15-4)***
- serverDisconnect() ***(p. 15-4)***

4.4.3 Events

- onScriptComplete(Long asyncID, String result) ***(p. 15-5)***
- onServerConnectStarted() ***(p. 15-5)***
- onServerConnectFailure(server_failure_type server_failure) ***(p. 15-5)***

Section 5. Developing an Application Using the CsiCoraScript Control

5.1 Purpose

This section shows an example of how to build a simple application using the *CsiCoraScript* control. The application's functions are:

1. Connect to a running LoggerNet server.

2. Execute *CoraScript* commands to administer the LoggerNet server.

5.2 Using the *CsiCoraScript* Control

5.2.1 Getting Started with the *CsiCoraScript* Control

The *CsiCoraScript* SDK control (an ActiveX® object) is used to administer the datalogger network by passing *CoraScript* commands to the LoggerNet server.

This example assumes that:

- The *CsiCoraScript* control has been correctly registered on the application host.

- A Windows® Forms application is to be developed using the Visual Studio® 2012 (or later) IDE and the VB.NET programming language.

- A LoggerNet server is running and accessible on the network

Complete the following steps first:

1. Start Visual Studio and create a new Visual Basic® Windows Forms Application targeting the .NET Framework 4.0.

2. Following the procedures outlined in Section 1.4, *Developing .NET Applications Using the SDK (p. 1-3)*, add the *CsiCoraScript* control to the *Toolbox* and create the RCW class.

3. In the Solution Explorer, right-click the *Form1.vb* file and rename it *CoraScriptForm.vb*.

5.2.2 *CsiCoraScript* Control Application Example

Begin by modifying the blank form to create a Graphical User Interface (GUI) that supports the required functionality. The finished form should resemble the example shown in FIGURE 5-1.

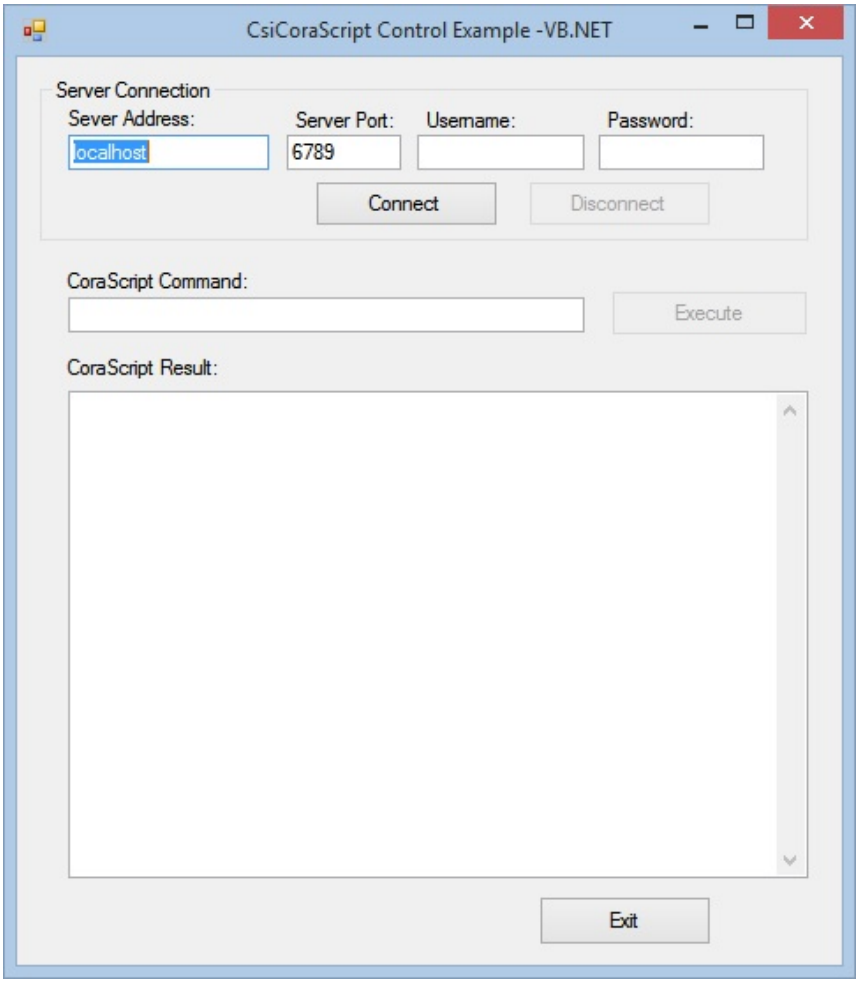

FIGURE 5-1. CsiCoraScript Example

Once the interface has been designed, the necessary Visual Basic code can be added to attain the required functionality. The first order of business is to instantiate the RCW class (*CsiCoraScriptLib.CsiCoraScriptControl*). Add the following code to the *CsiCoraScriptForm* class.

```
Dim WithEvents CsiCoraScript As New CsiCoraScriptControl
```

The following code snippets illustrate the basic functionality of the *CsiCoraScript* control. For more comprehensive code examples, refer to the VB.NET example project files supplied with the SDK.

The first task of the application is to establish a connection to LoggerNet server (*serverConnect()* method):

VB.NET Example 5-1. Establishing a Connection to a LoggerNet Server using the *serverConnect()* Method

```
Private Sub btnConnect_Click(sender As Object, e As EventArgs) Handles btnConnect.Click

  Try
    'Set connection properties
    CsiCoraScript.serverName = txtSvrAddress.Text
    CsiCoraScript.serverPort = Convert.ToInt32(txtSvrPort.Text)
    CsiCoraScript.serverLogonName = txtUsername.Text
    CsiCoraScript.serverLogonPassword = txtPassword.Text

    'Call serverConnect()
    'If a connection is made, the control will raise the onServerConnectStarted() event.
    'If a connection is not made, the control will raise the onServerConnectFailure event
    CsiCoraScript.serverConnect()

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .Net Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSICoraScriptLib.HRESULT_Errors and display the associated error.
    Dim com_ex As CsiCoraScriptLib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI CoraScript Connect Button : ERROR")
  End Try
End Sub
```

If the connection succeeds, the *onServerConnectStarted()* event gets triggered. The following code example illustrates how this event can be handled:

VB.NET Example 5-2. Handling the *onServerConnectStarted()* Event

```
Private Sub CsiCoraScript_onServerConnectStarted() Handles CsiCoraScript.onServerConnectStarted

  'This event is called when the CsiCoraScript control has successfully connected to the
server.
  Try
    'Update the form
    btnConnect.Enabled = False
    btnDisconnect.Enabled = True
    btnExecute.Enabled = True

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI CoraScript onServerConnectStarted Event : ERROR")
  End Try
End Sub
```

If the connection fails, the *onServerConnectFailure()* event gets called. This event will pass one of several enumerated *failure codes* to the application. The following code example illustrates how this event can be handled:

VB.NET Example 5-3. Handling the *onServerConnectFailure()* Event

```
Private Sub CsiCoraScript_onServerConnectFailure(ByVal server_failure As _
                                    CsiCoraScriptLib.server_failure_type) _
                                    Handles CsiCoraScript.onServerConnectFailure

  'This event is called when the attempt to connect to the server failed, or when the
  ‘established connection has been broken.
  Try
    'Show failure type
    MessageBox.Show("Could not connect to Server: " & server_failure.ToString(), _
                    "onServerConnect Failure Event")

    'Execute a Disconnect to insure a stable interface.
    btnDisconnect_Click(Me, New EventArgs())

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI CoraScript onServerConnectFailure Event : ERROR")
  End Try
End Sub
```

The following code example illustrates use of the *executeScript()* method for executing a *CoraScript* command:

VB.NET Example 5-4. Executing a CoraScript Command using the *executeScript()* Method

```
Private Sub btnExecute_Click(sender As Object, e As EventArgs) Handles btnExecute.Click
  Try

    'Send CoraScript command to the server
    'Sample CoraScript commands include:
    '   List-Devices;
    '   add-device com-port Comport before "";
    '   add-device pakbus-port PakBusPort as-child Comport;
    '   add-device cr1000 CR1000 as-child PakBusPort;

    'If the asyncID parameter of the executeScript() method is set to 0,
    'the command will execute synchronously (i.e., the program waits for the
    'execution of the CoraScript command to complete before continuing). The results
    'are returned to the caller.

    'If the asyncID parameter of the executeScript() method is other than 0,
    'the command will execute asynchronously (i.e., the program continues execution).
    'When the CoraScript command is complete, the onScriptComplete() event
    'will be raised and passed the results.

    'For this example we use synchronous execution.
    Me.Cursor = Cursors.WaitCursor
    txtCoraResult.Text = CsiCoraScript.executeScript(txtCoraScript.Text, 0)
    txtCoraResult.SelectionStart = txtCoraResult.Text.Length
    txtCoraResult.ScrollToCaret()
    Me.Cursor = Cursors.Default

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .Net Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSICoraScriptLib.HRESULT_Errors and display the associated error.
    Dim com_ex As CsiCoraScriptLib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message,
                    "CSI CoraScript Execute Button : ERROR")
  End Try
End Sub
```

Add additional objects and functionality as necessary to meet the specific requirements of the application. Complete examples using the *CsiCoraScript* control are included with the SDK installation.

Section 6. CsiBrokerMap Control

6.1 Purpose of the *CsiBrokerMap* Control

The *CsiBrokerMap* control gives developers access to the server's broker map, which is the list of brokers or dataloggers known by the LoggerNet server. This control also keeps track of all tables on each of the brokers including the table definitions or columns. This table information is derived from the collect areas that are known by the LoggerNet server after a datalogger program has been associated or table definitions have been retrieved. The user may also create collect areas manually in the LoggerNet server.

The application or user will use the information provided by the *CsiBrokerMap* control as input parameters for other controls in the SDK. For example, the *CsiCoraScript*, *CsiDatalogger*, and *CsiDataSource* controls will require specific information about brokers, tables, and columns existing in the network in order to function.

6.2 Connecting to the LoggerNet Server

There are two basic actions required to connect to the LoggerNet server:

1) Set server properties:

   - serverName – The name of the LoggerNet server or IP address. The default value is localhost.
   - serverPort – The port on which the LoggerNet server is running. The default value is 6789.
   - serverLogonName (Optional) – If security has been enabled on the server, a valid logon name is required.
   - serverLogonPassword (Optional) – If security has been enabled on the server, a valid password that corresponds with a valid logon name is required.

2) Invoke the *start()* method.

6.3 How Collections Work

The *CsiBrokerMap* uses the concept of *collections* in its implementation. Collections provide layers of objects and a standard way to access those objects. There are two basic ways to look at collections. The Visual Basic® (VB.Net) view describes how a VB.Net programmer would view a collection, which is simpler than for Visual C++®.

6.3.1 Visual Basic View of Collections

The *CsiBrokerMap* collections are simply three levels of grouped items. Brokers exist at the top-most level; within brokers are tables and within tables are columns. Each of these levels can be accessed with the dot operator in Visual Basic. The following example illustrates how to access all of the brokers in the *BrokerMap* collection:

6.3.1.1 Accessing Collections with For Each

```
For Each b in BrokerMap.Brokers
 Debug.Print b.name
Next
```

This simplistic code allows you to iterate through the *BrokerMap* collection simply without having to worry about indexes and going out of bounds. In the code above, it would be possible to access all of the tables in each broker by nesting a similar loop inside the existing one stating `For Each t in BrokerMap.Brokers(b).Tables`. By repeating similar code for the columns, the whole broker map could be displayed.

6.3.1.2 Accessing Collections with Indexes and Names

The brokers, tables, and columns can be accessed not only with the "`For Each`" loop, but also by index and name. Consider the following examples:

```
BrokerMap.Brokers("CR9000").Tables("minute").Columns("temp").size
For i = 0 to BrokerMap.Brokers.Count - 1
 Debug.Print BrokerMap.Brokers(i)
Next
```

The first line of code assumes that a datalogger named CR9000 with a table named minute exists in the broker map. The code also assumes a column named temp exists in the table named minute. These names could also be `String` variables instead of literal strings.

6.3.2 Visual C++ View of Collections

Visual C++ requires a little more work to capture the information provided by this control, but not much more than Visual Basic's iterative method using indexes. Please refer to the code in the Visual C++ examples included with the LoggerNet SDK installation.

6.4 *CsiBrokerMap* Interfaces

The following interfaces are included in the *CsiBrokerMap* control:

- Broker
- BrokerMap
- BrokerCollection
- Column
- ColumnCollection
- Table
- TableCollection

6.4.1 *BrokerMap* Interface

See Section 16.1, *BrokerMap Interface* ***(p. 16-1)***, for detailed descriptions of these properties, methods, and events.

6.4.1.1 Properties

- serverName As String ***(p. 16-1)***
- serverLogonName As String ***(p. 16-1)***
- serverLogonPassword As String ***(p. 16-2)***
- serverPort As Long ***(p. 16-3)***
- autoExpand As Boolean ***(p. 16-3)***
- serverConnected As Boolean ***(p. 16-4)***

6.4.1.2 Methods

- brokers() As Object ***(p. 16-4)***
- finish() ***(p.16-4)***
- start() ***(p. 16-5)***

6.4.1.3 Events

- onAllStarted() ***(p. 16-5)***
- onBrokerAdded(Object Broker) ***(p. 16-5)***
- onBrokerDeleted(Object Broker) ***(p. 16-6)***
- onFailure(BrokerMapFailureType failure_code) ***(p. 16-6)***
- onTableAdded(Object Broker, Object Table) ***(p. 16-7)***
- onTableDeleted(Object Broker, Object Table) ***(p. 16-7)***
- onTableChanged(Object Broker, Object Table) ***(p. 16-7)***
- onBrokerStarted(Object Broker) ***(p. 16-7)***

6.4.2 *BrokerCollection* Interface

See Section 16.2, *BrokerCollection Interface* ***(p. 16-7)***, for descriptions of these properties and methods.

6.4.2.1 Properties

- count As Long ***(p. 16-7)***

6.4.2.2 Methods

- Item(id) As Broker *(p. 16-8)*
- _NewEnum() (GetEnumerator() in .NET) *(p. 16-9)*

6.4.3 *Broker* Interface

See Section 16.3, *Broker Interface* *(p. 16-9)*, for descriptions of these properties and methods.

6.4.3.1 Properties

- id As Long *(p. 16-9)*
- name As String *(p. 16-9)*
- type As BrokerType *(p. 16-10)*
- datalogger_type As String *(p. 16-10)*
- allStarted As Boolean *(p. 16-11)*

6.4.3.2 Methods

- Tables() As Object *(p. 16-11)*
- start_expansion() *(p. 16-11)*

6.4.4 *TableCollection* Interface

See Section 16.4, *TableCollection Interface* *(p. 16-12)*, for descriptions of these properties and methods.

6.4.4.1 Properties

- Count As Long *(p. 16-12)*

6.4.4.2 Methods

- Item(id) As Table *(p. 16-12)*
- _NewEnum() (GetEnumerator() in .NET) *(p. 16-13)*

6.4.5 *Table* Interface

See Section 16.5, *Table Interface* *(p. 16-14)*, for descriptions of these properties and methods.

6.4.5.1 Properties

- interval As Long *(p. 16-14)*
- name As String *(p. 16-14)*
- originalSize As Long *(p. 16-14)*
- size As Long *(p. 16-15)*

6.4.5.2 Methods

- Columns() As Object *(p. 16-15)*
- start_expansion() *(p. 16-15)*

6.4.6 *ColumnCollection* Interface

See Section 16.6, *ColumnCollection Interface* ***(p. 16-16)***, for descriptions of these properties and methods.

6.4.6.1 Properties

- Count As Long ***(p. 16-16)***

6.4.6.2 Methods

- Item(id) As Column ***(p. 16-16)***
- _NewEnum() (GetEnumerator() in .NET) ***(p. 16-17)***

6.4.7 *Column* Interface

See Section 16.7, *Column Interface* ***(p. 16-18)***, for descriptions of these properties.

6.4.7.1 Properties

- description As String ***(p. 16-18)***
- name As String ***(p. 16-18)***
- process As String ***(p. 16-18)***
- type As CsiDataTypeCode ***(p. 16-19)***
- units As String ***(p. 16-20)***
- writable As Long ***(p. 16-21)***

Section 7. Developing an Application Using the CsiBrokerMap Control

7.1 Purpose

This section shows by example how to build a simple application using the *CsiBrokerMap* SDK control. The application's stated functions are:

1. Display names of all stations in the current network.

2. Upon selection of any single station, display tables associated with that station's currently running program.

3. Upon selection of any single table, display all fields (columns) included in that table.

The following section illustrates how to build an application that can perform these tasks using SDK controls and the LoggerNet server.

7.2 Using the *CsiBrokerMap* Control

7.2.1 Getting Started with the *CsiBrokerMap* Control

The *CsiBrokerMap* is an SDK control (an ActiveX® object) designed to display names of dataloggers in the current network. This control can also display names of all tables belonging to the selected datalogger and columns in the selected table. This information is derived from collect area information created when a program is associated with a datalogger or when table definitions are retrieved from the datalogger. Since the *CsiBrokerMap* control does not list devices if collect areas are not known, it may be necessary to use the *CoraScript* control to associate the program or to retrieve the table definitions.

This example assumes that:

- The *CsiBrokerMap* control has been correctly registered on the application host.

- A Windows® Forms application is to be developed using the Visual Studio® 2012 (or later) IDE and the VB.NET programming language.

- A LoggerNet server is currently running and accessible on the network.

- At least one station already exists in the LoggerNet server's network map.

- The datalogger program has been associated or table definitions have been retrieved.

Complete the following steps first:

1. Start Visual Studio and create a new Visual Basic® Windows Forms Application targeting the .NET Framework 4.0.

2. Following the procedures outlined in Section 1.4, *Developing .NET Applications Using the SDK* (p. 1-3), add the *CsiBrokerMap* control to the *Toolbox* and create the RCW class.

3. In the Solution Explorer, right-click the *Form1.vb* file and rename it *frmBrokerMap.vb*.

7.2.2 *CsiBrokerMap* Control Application Example

Begin by modifying the blank form to create a Graphical User Interface (GUI) that supports the required functionality. The finished form should resemble the example shown in FIGURE 7-1.

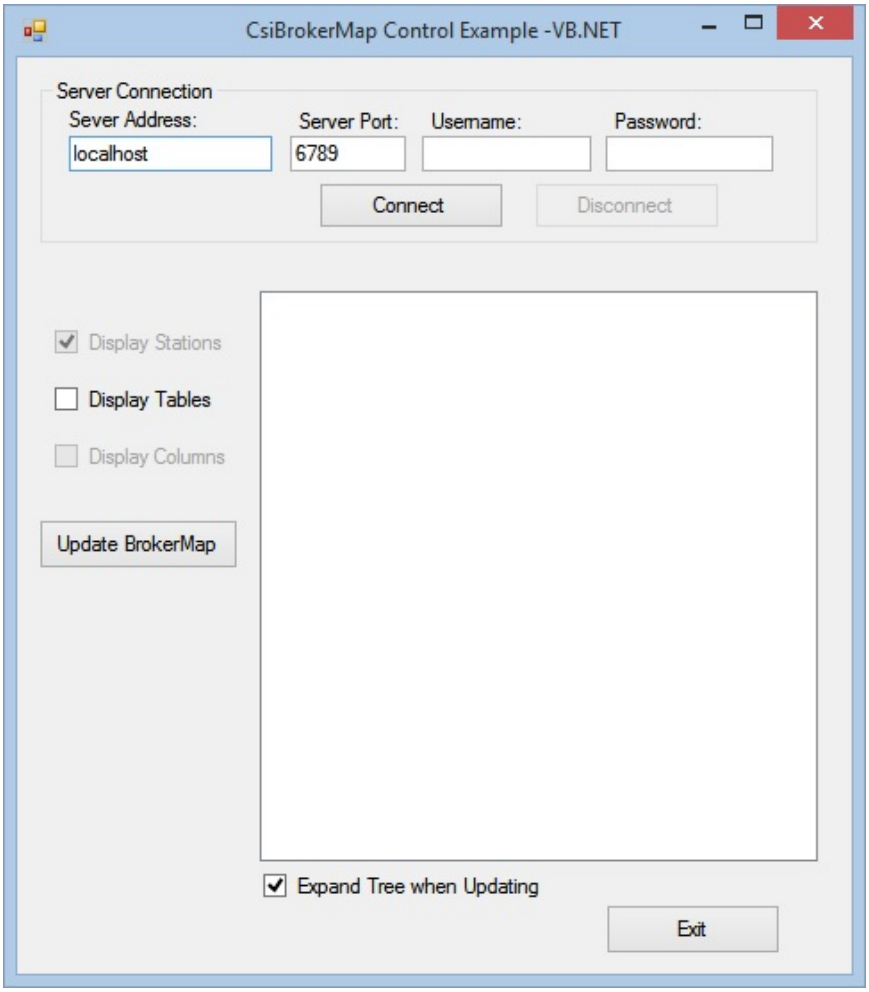

FIGURE 7-1. CsiBrokerMap Example

Once the interface has been designed, the necessary Visual Basic code can be added to attain the required functionality. The first order of business is to instantiate the RCW class (*CSIBROKERMAPLib.BrokerMap*). Add the following code to the *CsiBrokerMapForm* class.

```
Dim WithEvents CsiBrokerMap As New BrokerMap
```

The following code snippets illustrate the basic functionality of the *CsiBrokerMap* control. For more comprehensive code examples, refer to the VB.NET example project files supplied with the SDK.

The first task of the application is to establish a connection to LoggerNet server. The following code example illustrates using the *start()* method:

VB.NET Example 7-1. Establishing a Connection to the LoggerNet Server using the *start()* Method

```
Private Sub btnConnect_Click(sender As Object, e As EventArgs) Handles btnConnect.Click

  Try

    'Clear any current connection
    CsiBrokerMap.finish()

    'Set connection properties
    CsiBrokerMap.serverName = txtSvrAddress.Text
    CsiBrokerMap.serverPort = Convert.ToInt32(txtSvrPort.Text)
    CsiBrokerMap.serverLogonName = txtUsername.Text
    CsiBrokerMap.serverLogonPassword = txtPassword.Text

    'Disable refreshing of the tree until the onAllStarted event
    RefreshOn = False

    'Call start() to connect to the server and start the Broker Map query
    'If a connection is made, the control will raise the onAllStarted() event.
    'If a connection is not made, the control will raise the onFailure() event
    CsiBrokerMap.start()

    Me.Cursor = Cursors.WaitCursor

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .NET Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSIBROKERMAPLib.HRESULT_Errors and display the associated error.
    Dim com_ex As CSIBROKERMAPLib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI BrokerMap Connect Button : ERROR")
  End Try

End Sub
```

If the connection succeeds, the *onAllStarted()* event will be raised after all of the initial *onBrokerAdded()* and *onTableAdded()* events have been called and the broker map is known. The following code example illustrates how this event can be handled:

VB.NET Example 7-2. Handling the *onAllStarted()* Event

```
Private Sub CsiBrokerMap_onAllStarted() Handles CsiBrokerMap.onAllStarted

  'This event is called after all of the initial onBrokerAdded() and onTableAdded() events
  'have been called from the start() method and the Broker Map is known.

  Me.Cursor = Cursors.Default

  'Enable refreshing of tree on change events
  RefreshOn = True

  Try
     'Update the form
     btnConnect.Enabled = False
     btnDisconnect.Enabled = True
     btnUpdateBrokerMap.Enabled = True

     'Clear the tree view
     tvwDisplay.Nodes.Clear()

     'Create the Broker Map tree
     Me.RefreshTree()

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI BrokerMap onAllStarted Event: ERROR")
  End Try
End Sub
```

The *onAllStarted()* event handler enables and calls the Sub *RefreshTree()*. It is the *RefreshTree()* procedure that iterates through the collections to populate the TreeView object with the broker map. It is required that the *RefreshTree()* procedure be disabled prior to the handling of the *onAllStarted()* event to prevent the repetitive redrawing of the broker map due to the initial *onBrokerAdded()* and *onTableAdded()* events. Add the following line of code to the declarations section of the *CsiBrokerMapForm* class.

```
Dim RefreshOn As Boolean = False
```

The following code example illustrates the process of iterating through the Broker, Table and Column collections to populate the TreeView object with the broker map:

VB.NET Example 7-3. Populating the TreeView Object with the Broker Map

```
Private Sub RefreshTree()

  'Declare variables
  Dim myRootNode, myNode, myNode2 As TreeNode

  Try
    Me.Cursor = Cursors.WaitCursor

    'Clear the tree
    tvwDisplay.Nodes.Clear()

    'Setup the root node
    myRootNode = tvwDisplay.Nodes.Add("Root", "Broker Map")

    'Read the status of the checkboxes to determine the
    'names of the stations, tables and columns that will populate
    'the tree.

    'The Display Stations checkbox is always checked, so Stations will
    'always be displayed.

    'Iterate through every station/broker in the map.
    For Each b As CSIBROKERMAPLib.Broker In CsiBrokerMap.Brokers

        'Add each Station by name as a child of the root
        myNode = myRootNode.Nodes.Add(b.name, b.name)

        'If Display Tables is checked, add all the tables
        'for this current station as child nodes
        If chkDisplayTables.Checked Then

            For Each t As CSIBROKERMAPLib.Table In b.Tables

                'Add each table by name as child of the station
                myNode2 = myNode.Nodes.Add(t.name)

                'If Display Columns is checked, add all the columns
                'for this current Table as child nodes
                If chkDisplayColumns.Checked Then
                    For Each c As CSIBROKERMAPLib.Column In t.Columns

                        'Add each Column by name as child of the Table
                        myNode2.Nodes.Add(c.name)
                    Next 'c As CSIBROKERMAPLib.Column

                End If
            Next 't AS CSLIBROKERMAPLib.Table
        End If
    Next 'b AS CSIBROKERMAPLib.Broker

    'If Expand Tree when Updating is checked, expand the tree and scroll to the top
    If chkExpandTree.Checked Then
        tvwDisplay.ExpandAll()

    End If

    Me.Cursor = Cursors.Default

  Catch ex As Exception
    Me.Cursor = Cursors.Default
    MessageBox.Show("An error has occurred while populating the tree: " & ex.Message)
  End Try
End Sub
```

To ensure that the application is responsive to dynamic changes in the server's broker map, the *onBrokerAdded*, *onBrokerDeleted*, *onTableAdded*, *onTableDeleted*, and *onTableChanged* events should be handled to refresh the *TreeView* object. The following example code illustrates how the *onBrokerAdded()* event can be handled:

VB.NET Example 7-4. Handling the *onBrokerAdd()* Event

```
Private Sub CsiBrokerMap_onBrokerAdded(Broker As Object) Handles CsiBrokerMap.onBrokerAdded

  'This event is called as new brokers are added to the broker map.
  Try
    'A new station has been added, we should refresh the tree
    If RefreshOn Then
        Me.RefreshTree()
    End If

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI BrokerMap onBrokerAdded Event: ERROR")
  End Try
End Sub
```

Add additional functionality, error handling, and objects as necessary to meet the specific requirements of your application. Complete examples using the *CsiBrokerMap* control are included in the LoggerNet SDK installation.

Section 8. CsiDatalogger

8.1 Purpose of the *CsiDatalogger* Control

The *CsiDatalogger* control allows the developer to manage datalogger functions through the LoggerNet server. The basic managerial functions of this control include: sending a program to the datalogger, retrieving a program from the datalogger, checking the clock on the datalogger as well as setting it to the current time, setting variable values, and performing manual polls of the datalogger. Another important function creates an active connection between the server and the datalogger, to eliminate connection and disconnection overhead on slower connections.

8.2 Connecting to the Server

There are two basic actions required for this control to connect to the LoggerNet server:

1. Set server properties:
   - serverName – The name or IP address of the LoggerNet server . The default value is localhost.
   - serverPort – The port on which the LoggerNet server is running. The default value is 6789.
   - serverLogonName (Optional) – If security has been enabled on the server, a valid logon name is required.
   - serverLogonPassword (Optional) – If security has been enabled on the server, a valid password that corresponds with a valid logon name is required.
2. Invoke the *serverConnect()* method.

8.3 Datalogger Interface

8.3.1 Properties

8.3.2 Methods

8.3.3 Events

Section 9. Developing an Application Using the CsiDatalogger Control

9.1 Purpose

This section shows by example how to build a simple application using the *CsiDatalogger* SDK control. The application's stated functions are:

1. Connect to the LoggerNet server.
2. Establish an active connection with the datalogger.
3. Check and display time at the datalogger.
4. Retrieve data stored in the datalogger.
5. Send/Receive datalogger programs.

The following section illustrates how to build an application that can perform these tasks using the *CsiDatalogger* control and the LoggerNet server.

9.2 Using the *CsiDatalogger* Control

9.2.1 Getting Started with the *CsiDatalogger* Control

CsiDatalogger SDK control (an ActiveX® object) operates through the LoggerNet server to provide an application with the ability to interact with connected dataloggers.

This example assumes that:

- The *CsiDatalogger* control has been correctly registered on the application host.
- A Windows® Forms application is to be developed using the Visual Studio® 2012 (or later) IDE and the VB.NET programming language.
- A LoggerNet server is currently running and accessible on the network.
- At least one station already exists in the LoggerNet server's network map.

Complete the following steps first:

1. Start Visual Studio and create a new Visual Basic® Windows Forms Application targeting the .NET Framework 4.0.
2. Following the procedures outlined in Section 1.4, *Developing .NET Applications Using the SDK (p. 1-3)*, add the *CsiDatalogger* control to the *Toolbox* and create the RCW class.
3. In the Solution Explorer, right-click the *Form1.vb* file and rename it *frmDatalogger.vb*.

9.2.2 *CsiDatalogger* Control Application Example

Begin by modifying the blank form to create a Graphical User Interface (GUI) that supports the required functionality. The finished form should resemble the example shown in FIGURE 9-1.

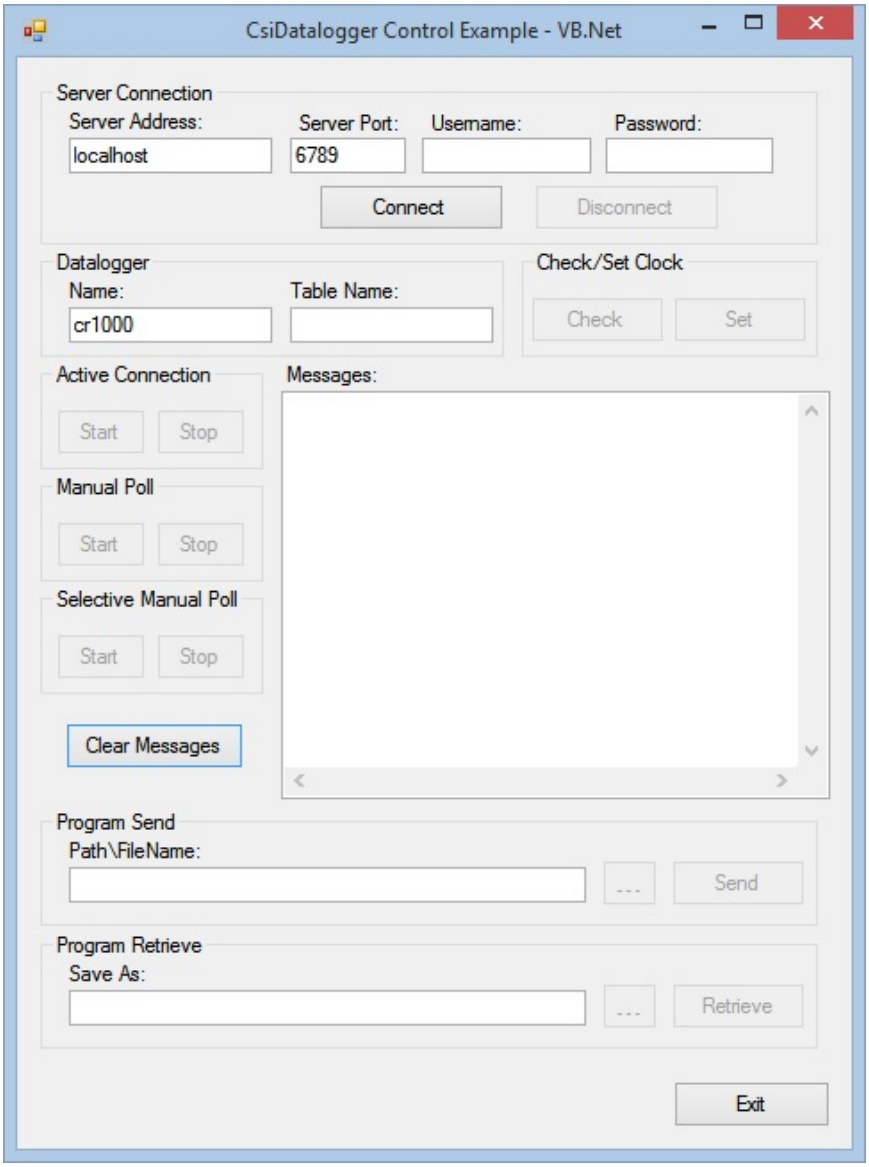

FIGURE 9-1. CsiDatalogger Example

Once the interface has been designed, the necessary Visual Basic code can be added to attain the required functionality. The first order of business is to instantiate the RCW class (*CSIDATALOGGERLib.Datalogger*). Add the following code to the *frmDatalogger* class:

```
Dim WithEvents CsiDatalogger As New DataLogger
```

Additionally, System.IO and System.Text namespace need to be imported. In the space above the *frmDatalogger* class declaration, add the following two lines of code:

```
Imports System.IO
Imports System.Text
```

The following code snippets illustrate the basic functionality of the *CsiDatalogger* control. For more comprehensive code examples, refer to the VB.NET example project files supplied with the SDK.

The first task of the application is to establish a connection to LoggerNet server. The following code example illustrates using the *serverConnect()* method:

VB.NET Example 9-1. Establishing a Connection to a LoggerNet Server using the *serverConnect()* Method

```
Private Sub btnConnect_Click(sender As Object, e As EventArgs) Handles btnConnect.Click

  Try
     'Set connection properties before connecting.
     CsiDatalogger.serverName = txtSvrAddress.Text
     CsiDatalogger.serverPort = Convert.ToInt32(txtSvrPort.Text)
     CsiDatalogger.serverLogonName = txtUsername.Text
     CsiDatalogger.serverLogonPassword = txtPassword.Text

     'Call serverConnect to Connect to the Server.
     'If a connection is made, the control will raise the onServerConnectStarted event.
     'If the connection fails, the onServerConnectFailure event will be raised.
     CsiDatalogger.serverConnect()

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .NET Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSIDATALOGGERLib.HRESULT_Errors and display the associated error.
    Dim com_ex As CSIDATALOGGERLib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger Connect Button: Error")
  End Try
End Sub
```

Many of the procedures in this example application require the displaying of text messages. The Sub *WriteMessages()* is used to facilitate this. The following code example illustrates the *WriteMessages()* procedure:

VB.NET Example 9-2. Displaying Text Messages using the *WriteMessage()* Sub

```
Private Sub WriteMessages(ByVal msg As String)

  Try
    'Add new message to textbox
    txtMessages.Text += vbCrLf & msg

    'Scroll down so the last entry is visible
    txtMessages.SelectionStart = txtMessages.Text.Length
    txtMessages.ScrollToCaret()

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger WriteMessages: ERROR")
  End Try
End Sub
```

If the *serverConnect()* method succeeds in establishing a connection with the server, the *onServerConnectStarted()* event is triggered. The following code example illustrates how the event can be handled:

VB.NET Example 9-3. Handling the *onServerConnectStarted()* Event

```
Private Sub CsiDatalogger_onServerConnectStarted() Handles CsiDatalogger.onServerConnectStarted

  'This event gets called once a connection has been established with the Server.
  Try
    'Write server connection message.
    WriteMessages(vbCrLf & "onServerConnectStarted()")
    WriteMessages("Connected to LoggerNet Server: " & CsiDatalogger.serverName)

    'Update the form.
    btnConnect.Enabled = False
    btnDisconnect.Enabled = True
    btnManPollStart.Enabled = True
    btnSelManPollStart.Enabled = True
    btnClockCheck.Enabled = True
    btnClockSet.Enabled = True
    btnLgrConStart.Enabled = True
    btnSendBrowse.Enabled = True
    btnPgmSend.Enabled = True
    btnPgmRetrieve.Enabled = True
    btnRetrieveBrowse.Enabled = True

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger onServerConnectStarted: Error")
  End Try
End Sub
```

If the *serverConnect()* method fails, the *onServerConnectFailure()* event is called and passes a failure code to the application. The following code example illustrates how the event can be handled to display the error code:

VB.NET Example 9-4. Handling the *onServerConnectFailure()* Event

```
Private Sub CsiDatalogger_onServerConnectFailure(ByVal failure_code As
CSIDATALOGGERLib.server_failure_type) Handles CsiDatalogger.onServerConnectFailure

  Try
    'Display the failure code.
    MessageBox.Show("The connection failed: " & failure_code.ToString, _
                    "onServerConnectFailure Event")

    Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger onServerConnectFailure Event: Error")
  End Try
End Sub
```

The default behaviour of the LoggerNet server is to establish a connection with a datalogger, execute the requisite transaction, and close the connection when completed. However, there may be scenarios where it would be more expedient to have the connection persist through multiple transactions rather than having to reestablish the connection for each. To facilitate this scenario, the *loggerConnectStart()* method can be called to establish a *persistent* connection with a datalogger. This is sometimes referred to as an *Active* connection and will persist until the *loggerConnectCancel()* method is called to close the connection.

It should be noted that the *CsiDatalogger* control can support only a single connection at a time. If one attempts to establish a second connection while a persistent connection is active, the `E_CSI_BUSY` error will be returned. If the application requires concurrent connections, multiple instances of the *CsiDatalogger* control will be required.

The following code example illustrates the use of the *loggerConnectStart()* method:

VB.NET Example 9-5. Using the *loggerConnectStart()* Method

```
Private Sub btnLgrConStart_Click(sender As Object, e As EventArgs) Handles btnLgrConStart.Click

  'This method will cause the server to establish a persistent connection to the specified
  'datalogger.

  Try
    'First, check to see if a datalogger name has been entered.
    If txtLoggerName.Text = "" Then
        MessageBox.Show("Must enter a Datalogger Name!")
        Exit Sub
    Else
        'Specify the datalogger with which to establish a persistent connection.
        CsiDatalogger.loggerName = txtLoggerName.Text
    End If

    'Make a high priority connection and write the message.
    CsiDatalogger.loggerConnectStart(CSIDATALOGGERLib.logger_priority_type.lp_priority_normal)
            WriteMessages(vbCrLf & "Attempting to establish an Active connection with: " _
            & CsiDatalogger.loggerName)

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .NET Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSIDATALOGGERLib.HRESULT_Errors and display the associated error.
    Dim com_ex As CSIDATALOGGERLib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger Active Connection Start Button: Error")
  End Try
End Sub
```

If the *loggerConnectStart()* method is successful, *onLoggerConnectStarted* event will be called. Otherwise the *onLoggerConnectFailure* event will be called and pass a failure code to the application. The *loggerConnectCancel()* method cancels the active connection and returns the server to the default behaviour.

The following code example illustrates the *loggerConnectCancel()* method:

VB.NET Example 9-6. Using the *loggerConnectCancel()* Method

```
Private Sub btnLgrConStop_Click(sender As Object, e As EventArgs) Handles btnLgrConStop.Click

  'This method cancels the current persistent connection.

  Try
    'Call the CsiDatalogger.loggerConnectCancel method to cancel the connection.
    CsiDatalogger.loggerConnectCancel()
    'Write the disconnect message.
    WriteMessages(vbCrLf & CsiDatalogger.loggerName & ": Active connection stopped.")

    'Update form
    btnLgrConStart.Enabled = True
    btnLgrConStop.Enabled = False
    txtLoggerName.Enabled = True

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger Active Connection Stop Button: Error")
  End Try
End Sub
```

The *clockCheckStart()* method will check the clock on the datalogger while the *clockSetStart()* method sets the clock on the datalogger to the time of the LoggerNet server. Both of these methods call the *onClockComplete()* event that returns a success indicator, a response code and, if successful, the current time of the datalogger clock.

The following code examples illustrate the *clockSetStart()* method followed by the *onClockComplete* event handler:

VB.NET Example 9-7. Using the *clockSetStart()* Method

```
Private Sub btnClockSet_Click(sender As Object, e As EventArgs) Handles btnClockSet.Click

  Try
    'If an active connection has been established, only the connected datalogger
    'can be accessed.
    If CsiDatalogger.loggerConnected Then
      'Start the clock set on the specified datalogger and write the message.
      'The onClockComplete event will be raised when the clock check is complete.
      CsiDatalogger.clockSetStart()
      WriteMessages(vbCrLf & CsiDatalogger.loggerName & ": Clock Set Started")
    Else
      'If no active connection is in effect, other dataloggers in the network map
      'can be accessed.
      'First check to see if a datalogger name has been entered.
      If txtLoggerName.Text = "" Then
          MessageBox.Show("Must enter a Datalogger Name!")
          Exit Sub
      Else
          'Specify the datalogger with which to establish a connection.
          CsiDatalogger.loggerName = txtLoggerName.Text
      End If

      'Start the clock set and write the message.
      'The onClockComplete event will be raised when the clock set is complete.
      CsiDatalogger.clockSetStart()
      WriteMessages(vbCrLf & CsiDatalogger.loggerName & ": Clock Set Started")
    End If

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .NET Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSIDATALOGGERLib.HRESULT_Errors and display the associated error.
    Dim com_ex As CSIDATALOGGERLib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger Clock Set Button: Error")
  End Try
End Sub
```

VB.NET Example 9-8. Handling the *onClockComplete()* Event

```
Private Sub CsiDatalogger_onClockComplete(ByVal successful As Boolean, ByVal response_code As
CSIDATALOGGERLib.clock_outcome_type, ByVal current_date As Date) _
        Handles CsiDatalogger.onClockComplete

  'This event is called after a clock check or a clock set method has completed.
  Try
    'Write a message in accordance with the results
    If successful Then
      'Write the dataloggers time
      WriteMessages("Current Datalogger Clock: " & current_date.ToString)
    Else
      'If the action was not successful, write the response_code
      WriteMessages("The Clock Check/Set failed: " & response_code.ToString)
    End If
  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger onClockComplete event: Error")
  End Try
End Sub
```

The *manualPollStart()* method initiates a data collection for all tables in the specified datalogger that are enabled for scheduled collection (collect area setting: *scheduleEnabled* = *True*). The *selectiveManualPollStart()* method initiates a data collection for only the specified table; regardless of the table's *scheduleEnabled* setting. Both of these methods have a corresponding cancel method (*manualPollCancel()* and *selectiveManualPollCancel()*, respectively) and a corresponding completion event (*onManualPollComplete()* and *onSelectiveManualPollComplete*, respectively) which returns the appropriate response code if the poll succeeded, failed, or was cancelled.

The following code example illustrates the *manualPollStart()* method:

VB.NET Example 9-9. Using the *manualPollStart()* Method

```
Private Sub btnManPollStart_Click(sender As Object, e As EventArgs) Handles
btnManPollStart.Click

  'This method will request that the server poll the tables in the datalogger
  'that have been enabled for scheduled collection in accordance with the
  'collect mode settings.
  Try
    'If an active connection has been established, only the connected datalogger
    'can be accessed.
    If CsiDatalogger.loggerConnected Then
      'Start the Manual Polling on the connected datalogger and write the message.
      'The onManualPollComplete event will be raised when the polling is complete.
      CsiDatalogger.manualPollStart()
      WriteMessages(vbCrLf & CsiDatalogger.loggerName & ": Manual Poll Started")
    Else
      'If no active connection is in effect, other dataloggers in the network map
      'can be accessed.
      'First check to see if a datalogger name has been entered.
      If txtLoggerName.Text = "" Then
          MessageBox.Show("Must enter a Datalogger Name!")
          Exit Sub
      Else
          'Specify the datalogger with which to establish a connection.
          CsiDatalogger.loggerName = txtLoggerName.Text
      End If

      'Start the Manual Polling and write the message.
      'The onManualPollComplete event will be raised when the polling is complete.
      CsiDatalogger.manualPollStart()
      WriteMessages(vbCrLf & CsiDatalogger.loggerName & ": Manual Poll Started")
    End If

    'Update the form
    btnManPollStart.Enabled = False
    btnManPollStop.Enabled = True

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .NET Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSIDATALOGGERLib.HRESULT_Errors and display the associated error.
    Dim com_ex As CSIDATALOGGERLib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger Manual Poll Start Button: Error")
  End Try
End Sub
```

The following code example illustrates the *manualPollCancel()* method:

VB.NET Example 9-10. Using the *manualPollCancel()* Method

```
Private Sub btnManPollStop_Click(sender As Object, e As EventArgs) Handles btnManPollStop.Click

  'This method will attempt to cancel a manual polling process. If successful, the
  'onManualPollComplete event will return an mp_outcome_aborted response_code.
  Try
    'Only call this method if a manual poll is in process.
    If CsiDatalogger.manualPollBusy Then
      'Call the method and write the message.
      CsiDatalogger.manualPollCancel()
      WriteMessages("Attempting to cancel manual poll!")
    Else
      MessageBox.Show("There is no manual poll in process")
    End If

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger Manual Poll Stop Button: Error")
  End Try
End Sub
```

The following code example illustrates the handling of the *onManualPollComplete()* event:

VB.NET Example 9-11. Handling the *onManualPollComplete()* Event

```
Private Sub CsiDatalogger_onManualPollComplete(ByVal successful As Boolean, response_code As
CSIDATALOGGERLib.manual_poll_outcome_type) _
         Handles CsiDatalogger.onManualPollComplete

  'This event is called when a manual poll transaction completes.
  Try
    'Write a message in accordance with the results
    If successful Then
        'Write the dataloggers time
        WriteMessages("Manual Poll was Successful")
    Else
        'If the action was not successful, write the response_code
        WriteMessages("Manual Poll Failed: " & response_code.ToString)
    End If

    'Update the form
    btnManPollStart.Enabled = True
    btnManPollStop.Enabled = False

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger onManualPollCompete event: Error")
  End Try
End Sub
```

The *programReceiveStart()* method retrieves the currently running program from a datalogger and saves it with the name and location specified by the filename parameter. The following code example illustrates the use of the *programReceiveStart()* method:

VB.NET Example 9-12. Using the *programReceiveStart()* Method

```
Private Sub btnPgmRetrieve_Click(sender As Object, e As EventArgs) Handles btnPgmRetrieve.Click

  'This method retrieves the currently running program from the connected datalogger and
  'saves that file as the specified filename.
  'The onProgramReceiveProgress() event will be called periodically as the file is being
  'transferred.
  'The onProgramReceiveComplete() event returns the success or failure results of the completed
  'process.

  Try
    'If an active connection has been established, only the connected datalogger
    'can be accessed.
    If CsiDatalogger.loggerConnected Then
      'Check to see if a program retrieve file name has been entered.
      If txtPgmRetrieve.Text = "" Then
          MessageBox.Show("Must enter a save as file name!")
          Exit Sub
      Else
          'Call the programReceiveStart() method.
          CsiDatalogger.programReceiveStart(txtPgmRetrieve.Text)
      End If
    Else
      'If no active connection is in effect, other dataloggers in the network map
      'can be accessed.
      'First check to see if a datalogger name has been entered.
      If txtLoggerName.Text = "" Then
          MessageBox.Show("Must enter a Datalogger Name!")
          Exit Sub
      Else
          'Specify the datalogger with which to establish a connection.
          CsiDatalogger.loggerName = txtLoggerName.Text
      End If
      'Check to see if a program retrieve file name has been entered.
      If txtPgmRetrieve.Text = "" Then
          MessageBox.Show("Must enter a save as file name!")
          Exit Sub
      Else
          'Call the programReceiveStart() method.
          CsiDatalogger.programReceiveStart(txtPgmRetrieve.Text)
      End If
    End If

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .NET Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSIDATALOGGERLib.HRESULT_Errors and display the associated error.
    Dim com_ex As CSIDATALOGGERLib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger Program Retrieve Button: Error")
  End Try
End Sub
```

The *onProgramReceiveProgress()* event is triggered and provides information regarding the progress of the program retrieval. The following code example illustrates how the event can be handled:

VB.NET Example 9-13. Handling the *onProgramReceiveProgress()* Event

```
Private Sub CsiDatatlogger_onProgramReceiveProgress(ByVal received_bytes As Long) Handles
CsiDatalogger.onProgramReceiveProgress

  'This event periodically returns a notification of how many bytes have been received from the
  'datalogger during the retrieval of a program.

  Try
    'Write the progress notification to the textbox.
    WriteMessages(vbCrLf & "Program Receive Progress:")
    WriteMessages(received_bytes.ToString & " bytes received.")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger onProgramReceiveProgress event: Error")
  End Try
End Sub
```

The *onProgramReceiveComplete()* event also runs when the file retrieval process either completes or fails. The following code example illustrates how the event can be handled:

VB.NET Example 9-14. Handling the *onProgramReceiveComplete()* Event

```
Private Sub CsiDataLogger_onProgramReceiveComplete(ByVal successful As Boolean, _
                ByVal response_code As   CSIDATALOGGERLib.prog_receive_outcome_type) Handles _
                CsiDatalogger.onProgramReceiveComplete

  'This event returns the success or failure information after a programReceiveSend() method
  'has completed.

  Try
    'If the Program Retrieval was successful, write the message.
    If successful Then
        WriteMessages(vbCrLf & "The program was successfully retrieved.")
    Else
        'Write the failure code to the textbox.
        WriteMessages(vbCrLf & "The program retrieval failed: " & response_code.ToString)
          End If

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger onProgramReceiveComplete event: Error")
  End Try
End Sub
```

The *programSendStart()* method sends a program to the specified datalogger and calls the *onProgramSendProgress()* event, the *onProgramSent()* event, and the *onProgramSendComplete()* event respectively.

The following code example illustrates the *programSendStart()* method:

VB.NET Example 9-15. Using the *programSendStart()* Method

```
Private Sub btnPgmSend_Click(sender As Object, e As EventArgs) Handles btnPgmSend.Click

  'This method starts the process of sending a program file to the specified datalogger.

  Try
    'Declare variable
    Dim filename As String

    'If an active connection has been established, only the connected datalogger
    'can be accessed.
    If CsiDatalogger.loggerConnected Then
      'Check to see if a program file has been entered.
      If txtPgmSend.Text = "" Then
          MessageBox.Show("Must enter a valid Program file name!")
          Exit Sub
      Else
        'Extract the program file name from the path.
        filename = ExtractFilename(txtPgmSend.Text)
        'If a valid program file name is returned, send the program to the datalogger.
        If Not String.IsNullOrEmpty(filename) Then
            'call the programSendStart method.
            CsiDatalogger.programSendStart(txtPgmSend.Text, filename)
        Else
            MessageBox.Show("Invalid Program file name!")
            Exit Sub
        End If
      End If
    Else
      'If no active connection is in effect, other dataloggers in the network map
      'can be accessed.
      'First check to see if a datalogger name has been entered.
      If txtLoggerName.Text = "" Then
        MessageBox.Show("Must enter a Datalogger Name!")
        Exit Sub
      Else
        'Specify the datalogger with which to establish a connection.
        CsiDatalogger.loggerName = txtLoggerName.Text
      End If
      'Check to see if a program file has been entered.
      If txtPgmSend.Text = "" Then
        MessageBox.Show("Must enter a valid Program file name!")
        Exit Sub
      Else
        'Extract the program file name from the path.
        filename = ExtractFilename(txtPgmSend.Text)
        'If a valid program file name is returned, send the program to the datalogger.
        If Not String.IsNullOrEmpty(filename) Then
          'call the programSendStart method.
          CsiDatalogger.programSendStart(txtPgmSend.Text, filename)
        Else
            MessageBox.Show("Invalid Program file name!")
            Exit Sub
        End If
      End If
    End If

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .NET Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSIDATALOGGERLib.HRESULT_Errors and display the associated error.
    Dim com_ex As CSIDATALOGGERLib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")
```

```
  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, "CSI Datalogger Program Send
Button: Error")
  End Try
End Sub
```

The above code calls on a utility function (*ExtractFileName()*) to extract the file name from a path string. The following is the code for that function:

VB.NET Example 9-16. The *ExtractFileName()* Function

```
Private Function ExtractFilename(filepath As String) As String
  ' If path ends with a "\", it's a path only so return String.Empty.
  If filepath.Trim().EndsWith("\") Then Return String.Empty

  ' Determine where last backslash is.
  Dim position As Integer = filepath.LastIndexOf("\")
  ' If there is no backslash, assume that this is a file name.
  If position = -1 Then
    ' Determine whether file exists in the current directory.
    If File.Exists(Environment.CurrentDirectory + Path.DirectorySeparatorChar + filepath) Then
      Return filepath
    Else
      Return String.Empty
    End If
  Else
    ' Determine whether file exists using filepath.
    If File.Exists(filepath) Then
      ' Return file name without file path.
      Return filepath.Substring(position + 1)
    Else
      Return String.Empty
    End If
  End If
End Function
```

The following code example illustrates how the *onSendProgramProgress()* event can be handled:

VB.NET Example 9-17. Handling the *onSendProgramProgress()* Event

```
Private Sub CsiDatalogger_onProgramSendProgress(ByVal sent_bytes As Long, _
                                                ByVal total_bytes As Long)
                                                Handles CsiDatalogger.onProgramSendProgress

  'This event periodically returns notification of how many sent_bytes out of a program's
  'total_bytes have been sent to the datalogger.

  Try
     'Write the progress message.
     WriteMessages(vbCrLf & "Program Send Progress:")
     WriteMessages("Sending " & sent_bytes.ToString & " bytes out of " _
                               & total_bytes.ToString & " total.")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, "CSI Datalogger
onProgramSendProgress event: Error")
  End Try
End Sub
```

The following code example illustrates how the *onSendProgramComplete()* event can be handled:

VB.NET Example 9-18. Handling the *onSendProgramComplete()* Event

```
Private Sub CsiDatalogger_onProgramSendComplete(ByVal successful As Boolean, _
                  ByVal response_code As CSIDATALOGGERLib.prog_send_outcome_type, _
                  ByVal compile_results As String) Handles CsiDatalogger.onProgramSendComplete

  'This event is called when the program send process has completed.
  Try
    'If the program was sent successfully, write the message and compile results.
    If successful Then
        WriteMessages(vbCrLf & "The program send succeeded." & vbCrLf & compile_results)
    Else
        WriteMessages(vbCrLf & "The program send failed: " & response_code.ToString)
    End If

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI Datalogger onProgramSendComplete event: Error")
  End Try
End Sub
```

Additional functionality, error handling, and objects should be added as necessary beyond the example interface and code listed above to meet the specific requirements of your application. Complete examples using the *CsiDatalogger* control are included in the LoggerNet SDK installation.

Section 10. CsiDataSource Control

10.1 Purpose of the *CsiDataSource* Control

The *CsiDataSource* control allows an application to monitor data collected through the LoggerNet server. These sessions that monitor data are known as advisors. Advisors display data collected in the LoggerNet server data cache. This control can have multiple advisor sessions with a single server connection.

This control requires that the LoggerNet server collect data for the same tables or final storage areas that are being monitored. If you start an advisor on a table that is not being collected by the LoggerNet server, you will not receive any *onAdviseRecord* events. An exception to this rule occurs if you are monitoring input locations on a mixed-array datalogger. When you create an advisor for an input location on a mixed-array datalogger, a temporary data cache is created. Then, with the advisor ready, enabling scheduled collection with the datalogger will return records to the advisor.

The *CsiBrokerMap* control is often used in conjunction with this control to display what tables and columns can be monitored. Additionally, the *CsiDatalogger* control can also be used to issue a manual data poll and collect records from the datalogger.

10.2 Connecting to the Server

There are two basic actions required to connect to the LoggerNet server:

1. Set server properties:
    - serverName – The name or IP address of the LoggerNet server. The default value is localhost.
    - serverPort – The port on which the LoggerNet server is running. The default value is 6789.
    - logonName (Optional) – If security has been enabled on the server, a valid logon name is required.
    - logonPassword (Optional) – If security has been enabled on the server, the correct password for a valid logon name is required.
2. Invoke the *connect()* method.

10.3 *CsiDataSource* Interfaces

The following interfaces are used in the *CsiDataSource* control:

- DSource – the controlling interface
- Advisor – created through the DSource interface to monitor certain data columns on a specified station and table.
- Record – received in the event onAdviseRecord. A record is a collection of values that contain data.
- Value – contains the name and value of a single column.

10.3.1 *DSource* Interface

See Section 18.1, *DSource Interface* ***(p. 18-1)***, for descriptions of these properties, methods, and events.

10.3.1.1 Properties

- logonName As String ***(p. 18-1)***
- logonPassword As String ***(p. 18-1)***
- serverName As String ***(p. 18-2)***
- serverPort As Long ***(p. 18-2)***
- state As data_source_state ***(p. 18-3)***
- sendRecordBlocks as Boolean ***(p. 18-3)***

10.3.1.2 Methods

- connect() ***(p. 18-4)***
- createAdvisor() As Object ***(p. 18-4)***
- disconnect() ***(p. 18-5)***

10.3.1.3 Events

- onAdviseReady(Object myAdvisor) ***(p. 18-5)***
- onAdviseRecord(Object myAdvisor, Object myRecord) ***(p. 18-6)***
- onAdvisorFailure(csiAdvisorFailureCode failure, Advisor myAdvisor) ***(p. 18-6)***
- onControlFailure(csidsFailureCode failure_code) ***(p. 18-8)***
- onControlReady() ***(p. 18-8)***
- onVariableSetComplete(Long tran_id, Object myAdvisor, Boolean successful, variable_outcome_code response_code) ***(p. 18-9)***
- onAdviseRecords(Object myAdvisor, object record_collection) ***(p. 18-10)***

10.3.2 *Advisor* Interface

See Section 18.2, *Advisor Interface* ***(p. 18-11)***, for descriptions of these properties, methods, and events.

10.3.2.1 Properties

- advisorName As String *(p. 18-11)*
- orderOption As csidsOrderOptionType *(p. 18-11)*
- startDate As Date *(p. 18-12)*
- startFileMarkNo As Long *(p. 18-13)*
- startIntervalSeconds As Long *(p. 18-14)*
- startOption As csidsStartOptionType *(p. 18-14)*
- startRecordNo As Long *(p. 18-15)*
- startRecordNoString As String *(p. 18-16)*
- state As advisor_state *(p. 18-17)*
- stationName As String *(p. 18-17)*
- tableName As String *(p. 18-18)*
- startDateNanoSeconds As Long *(p. 18-18)*
- maxRecordsPerBlock As Long *(p. 18-18)*

10.3.2.2 Methods

- columns() As Object *(p. 18-19)*
- start() *(p. 18-19)*
- stop() *(p. 18-20)*
- variableSetCancel(Long tran_id) *(p. 18-20)*
- variableSetStart(String column_name, String value) as Long *(p. 18-20)*

10.3.3 *DataColumnCollection* Interface

See Section 18.3, *DataColumnCollection Interface* *(p. 18-21)*, for descriptions of these properties and methods.

10.3.3.1 Properties

- count As Long *(p. 18-21)*

10.3.3.2 Methods

- add(String column_name) *(p. 18-22)*
- addAll() *(p. 18-22)*
- find(String column_name) As Boolean *(p. 18-22)*
- Item(id) As DataColumn *(p. 18-23)*
- remove(String columnName) *(p. 18-23)*
- removeAll() *(p. 18-24)*
- _NewEnum() (GetEnumerator() in .NET) *(p. 18-24)*

10.3.4 *DataColumn* Interface

See Section 18.4, *DataColumn Interface* *(p. 18-24)*, for descriptions of these properties.

10.3.4.1 Properties

- name As String *(p. 18-24)*

10.3.5 Record

See Section 18.5, *Record Interface* ***(p. 18-25)***, for descriptions of these properties, methods, and events.

10.3.5.1 Properties

- fileMarkNo As Long ***(p. 18-25)***
- nanoSeconds as Long ***(p. 18-25)***
- recordNo As Long ***(p. 18-25)***
- timeStamp As Date ***(p. 18-26)***
- valuesCount As Long ***(p. 18-26)***

10.3.5.2 Methods

- Item(id) As Value ***(p. 18-26)***
- _NewEnum() (GetEnumerator() in .NET) ***(p. 18-27)***

10.3.6 *RecordCollection*

10.3.6.1 Properties

- Count As Long ***(p. 18-28)***

10.3.6.2 Methods

- Item(Value id, Record ppIRecord) ***(p. 18-28)***
- _NewEnum() (GetEnumerator() in .NET) ***(p. 18-29)***

10.3.7 *Value* Interface

See Section 18.7, *Value Interface* ***(p. 18-29)***, for descriptions of these properties, methods, and events.

10.3.7.1 Properties

- columnName As String ***(p. 18-29)***
- value As Variant ***(p. 18-29)***

Section 11. Developing an Application Using the CsiDataSource Control

11.1 Purpose

The *CsiDataSource* control primarily monitors data residing in the LoggerNet server data cache. The LoggerNet server data cache is a location where the server stores collected datalogger records. The control can also be used to see measurements performed in real-time; for example, values being recorded for input locations in mixed-array dataloggers. The *CsiBrokerMap* control often accompanies this control to display the names of tables and columns in each table so they can be selected for data monitoring. However, the example illustrated in this section requires that the user enter a station and table that are known to exist on the LoggerNet server and all columns will be monitored within that table. The application we develop will:

1. Connect to a LoggerNet server.
2. Allow the user to enter a known station and table.
3. Monitor data in all columns of the table.

The following section illustrates how to build an application that can perform these tasks using the *CsiDataSource* control and the LoggerNet server.

11.2 Using the *CsiDataSource* Control

11.2.1 Getting Started with the *CsiDataSource* Control

CsiDataSource is an SDK control (an ActiveX® object) designed to monitor data collected from the dataloggers in the LoggerNet network. This example assumes that:

- The *CsiDataSource* control has been correctly registered on the application host.
- A Windows® Forms application is to be developed using the Visual Studio® 2012 (or later) IDE and the VB.NET programming language.
- A LoggerNet server is currently running and accessible on the network.
- At least one station already exists in the LoggerNet server's network map.

Complete the following steps first:

1. Start Visual Studio and create a new Visual Basic® Windows Forms Application targeting the .NET Framework 4.0.
2. Following the procedures outlined in 1.4, *Developing .NET Applications Using the SDK (p. 1-3)*, add the *CsiDataSource* control to the *Toolbox* and create the RCW class.
3. In the Solution Explorer, right-click the *Form1.vb* file and rename it *frmDSource.vb*.

11.2.2 *CsiDataSource* Control Application Example

Begin by modifying the blank form to create a Graphical User Interface (GUI) that supports the required functionality. The finished form should resemble the example shown in FIGURE 11-1.

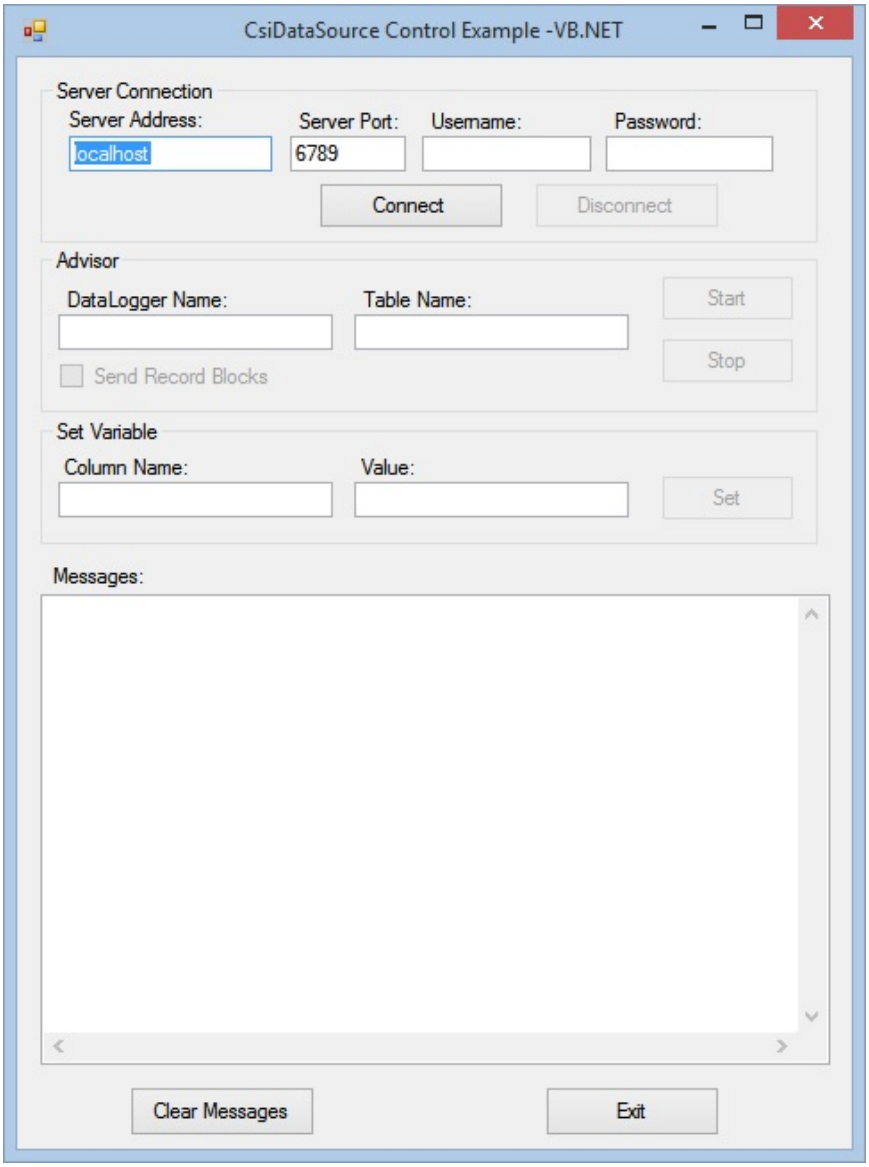

FIGURE 11-1. CsiDataSource Example

Once the interface has been designed, the necessary Visual Basic code can be added to attain the required functionality. The first order of business is to instantiate the RCW class (*CSIDATASOURCELib.DSource*). Also, an object of the type advisor must be declared. Add the following code to the *frmDSource* class:

```
Dim WithEvents CSIDataSource As New DSource

Private CurrentAdvisor As CSIDATASOURCELib.Advisor
```

Additionally, the System.Text namespace needs to be imported. In the space above the *frmDSource* class declaration, add the following code:

```
Imports System.Text
```

The following code snippets illustrate the basic functionality of the *CsiDataSource* control. For more comprehensive code examples, refer to the VB.NET example project files supplied with the SDK.

Many of the procedures in this example application require the displaying of data records and text messages. The Sub *WriteMessage()* is used to facilitate this. The following code example illustrates the *WriteMessage()* procedure:

VB.NET Example 11-1. The *WriteMessage* Procedure

```
Private Sub WriteMessage(ByVal Msg As String)

  'Get the number of lines displayed in the textbox.
  Dim lineCount = txtMessages.Lines.Length
  'Define a buffer for building the string to display.
  Dim buff As New StringBuilder

  Try
    'We want to limit the number of lines contained in the textbox.
    'The limit chosen is a compromise between displaying as much information
    'as practicable without degrading the performance of the textbox.
    If lineCount < 100 Then
      'Add the current contents of the textbox to the buffer
      'and append a new line.
      buff.Append(txtMessages.Text)
      buff.Append(Environment.NewLine)
    End If

    'Add the new message to the buffer.
    buff.Append(Msg)
    'Write the string buffer to the textbox, overwriting the existing text.
    txtMessages.Text = buff.ToString

    'Scroll down so that the last entry is visible.
    txtMessages.SelectionStart = txtMessages.Text.Length
    txtMessages.ScrollToCaret()

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI DataSource WriteMessage: ERROR")
  End Try
End Sub
```

The first task of the application is to establish a connection to the LoggerNet server. The following code example illustrates using the *connect()* method:

VB.NET Example 11-2. Establishing a Connection to the LoggerNet Server using the *connect()* Method

```
Private Sub btnConnect_Click(sender As Object, e As EventArgs) Handles btnConnect.Click

  Try
    'Set connection properties before connecting.
    CSIDataSource.serverName = txtSvrAddress.Text
    CSIDataSource.serverPort = Convert.ToInt32(txtSvrPort.Text)
    CSIDataSource.logonName = txtUsername.Text
    CSIDataSource.logonPassword = txtPassword.Text

    'Call connect()
    'If a connection is made, the control will raise the onControlReady() event.
    'If a connection is not made, the onControlFailure() event will raised.
    CSIDataSource.connect()

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .NET Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSIDATASOURCELib.HRESULT_Errors and display the associated error.
    Dim com_ex As CSIDATASOURCELib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI DataSource Connect Button : ERROR")
  End Try
End Sub
```

Once connected to the LoggerNet server, the advisor can be created by entering a known datalogger and table in the text fields and calling the *createAdvisor()* method. An application may use the *CsiBrokerMap* to display all stations and allow the user to select a specific datalogger, table, and column. However, in this example, the user merely enters the name of a datalogger and table known to exist in the LoggerNet server network map. An example of the code used to start an advisor that will monitor data in all columns of a specific datalogger and table can be found in the following example code:

VB.NET Example 11-3. Starting an Advisor to Monitor Data in All Columns of a Specific Datalogger and Table

```
Private Sub btnStartAdvisor_Click(sender As Object, e As EventArgs) Handles
btnStartAdvisor.Click

  Try
    'If there is a current connection, create a new advisor and add all columns
    If CSIDataSource.state = data_source_state.dataSourceConnected Then
      CurrentAdvisor = CSIDataSource.createAdvisor
      CurrentAdvisor.advisorName = "newAdvisor"
      CurrentAdvisor.stationName = txtDataLoggerName.Text
      CurrentAdvisor.tableName = txtTableName.Text

      'Select the order in which the Server will send records to the advisor
      CurrentAdvisor.orderOption = csidsOrderOptionType.csidsOrderLoggedWithoutHoles
      'Specify how the first record to send is selected
      CurrentAdvisor.startOption = csidsStartOptionType.csidsStartAtRecordId

      'Check the status of the Send Record Blocks checkbox
      If cbxSndRecBlks.CheckState = CheckState.Checked Then
         CSIDataSource.sendRecordBlocks = True
         'Set the maximum number of records per block
         CurrentAdvisor.maxRecordsPerBlock = 1024L
      Else
        CSIDataSource.sendRecordBlocks = False
      End If

      'Add all of the table columns to the advisor
      Dim myCols As CSIDATASOURCELib.IDataColumnCollection
      myCols = CurrentAdvisor.Columns
      myCols.addAll()

      'Start the advisor. If started, the onAdviseReady event and either
      'the onAdviseRecord or onAdviseRecords event will be raised.
      'If the advisor fails to start, the onAdvisorFailure event will be called.
      CurrentAdvisor.start()
    End If

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .NET Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSIDATASOURCELib.HRESULT_Errors and display the associated error.
    Dim com_ex As CSIDATASOURCELib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI DataSource StartAdvisor Button: ERROR")
  End Try
End Sub
```

After starting the advisor, the *onAdviseReady()* event will run and begin watching the specified table and columns for new data. Depending on the value of the *CsiDataSource.sendRecordBlocks* property, new data records are sent from the LoggerNet server cache individually via the *onAdviseRecord* event or in blocks via the *onAdviseRecords* event. For an example of handling the latter, see the example project code supplied with the SDK. The following code example illustrates how records are received via the *onAdviseRecord* event:

VB.NET Example 11-4. Receiving Records via the *onAdviseRecord* Event

```
Private Sub CSIDataSource_onAdviseRecord(ByVal myAdvisor As Object, myRecord As Object) Handles
CSIDataSource.onAdviseRecord

  'The DataSource onAdviseRecord event indicates that a new data record has been received by
  'an Advisor. The Advisor and associated Record are returned.
  Try
    'Declare variables
    Dim rec As CSIDATASOURCELib.IRecord
    rec = myRecord

    'Update form
    btnStartAdvisor.Enabled = False
    btnStopAdvisor.Enabled = True
    cbxSndRecBlks.Enabled = True
    btnSetVar.Enabled = True

    'Write the Advisor name and Record information.
    WriteMessage(vbCrLf & "onAdviseRecord()")
    WriteMessage("Advisor Name: " & CurrentAdvisor.advisorName)
    WriteMessage("FileMarkNo: " & rec.fileMarkNo.ToString())
    WriteMessage("RecordNo: " & rec.recordNo.ToString())
    WriteMessage("TimeStamp: " & rec.timeStamp.ToString())

    'Write record column values
    For Each Val As CSIDATASOURCELib.value In rec
        WriteMessage(Val.columnName & " : " & Val.value.ToString())
    Next Val

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI DataSource onAdviseRecord: ERROR")
  End Try
End Sub
```

The advisor will continue displaying new records as they are received until the *stop()* method is called to stop the advisor. The following code illustrates the use of this method:

VB.NET Example 11-5. Stopping an Advisor Using the *stop()* Method

```
Private Sub btnStopAdvisor_Click(sender As Object, e As EventArgs) Handles btnStopAdvisor.Click
  Try
    'Stop the Advisor
    CurrentAdvisor.stop()

    'Update form
    WriteMessage(vbCrLf & "Advisor Stopped.")
    btnStopAdvisor.Enabled = False
    btnStartAdvisor.Enabled = True
    btnSetVar.Enabled = False

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSI DataSource StopAdvisor Button: ERROR")
  End Try
End Sub
```

Additional functionality, error handling, and objects should be added as necessary beyond the example interface and code listed above to meet the specific requirements of your application. Complete examples using the *CsiDataSource* control are included in the LoggerNet SDK installation.

Section 12. CsiLogMonitor Control

12.1 Purpose of the *CsiLogMonitor* Control

The *CsiLogMonitor* control provides access to log message from the LoggerNet server. The log messages stream to this control as a text string. Use this control to display log messages or to monitor events as they occur on the server and call other operations or programs based on these LoggerNet server events.

The types of log files that can be retrieved from the LoggerNet server with the *CsiLogMonitor* control include the transaction log and the communication log. The transaction log messages use the following basic format:

"StationName", "MessageTypeCode", "Message"

The developer can create a program using the *CsiLogMonitor* control to filter each message by station name and watch for message types and messages that indicate a specific event. By parsing the transaction log text string and looking for the triggering messages listed below, the declared station event can be monitored.

The communication log messages use the following basic format:

"StationName", "Severity", "Message"

The severity types in the communication log are "S" for a status message, "W" for a warning message, and "F" for a failure message. Status messages are general communication messages, warning messages declare a possible problem and communication retries, and failure messages appear when all retries have been exhausted and communication will no longer be attempted by the LoggerNet server for a specific transaction.

12.2 *CsiLogMonitor* Interface

See Section 19, *CsiLogMonitor Control Reference* ***(p. 19-1)***, for detailed descriptions of these properties, methods, and events.

12.2.1 Properties

- commLogMonitorBusy As Boolean ***(p. 19-1)***
- commLogRecordsBack As Long ***(p. 19-1)***
- serverConnected As Boolean ***(p. 19-2)***
- serverLogonName As String ***(p. 19-2)***
- serverLogonPassword As String ***(p. 19-2)***
- serverName As String ***(p. 19-3)***
- serverPort As Long ***(p. 19-3)***
- tranLogMonitorBusy As Boolean ***(p. 19-4)***
- tranLogRecordsBack As Long ***(p. 19-4)***

12.2.2 Methods

- commLogMonitorStart() ***(p. 19-5)***
- commLogMonitorStop() ***(p. 19-5)***
- serverConnect() ***(p. 19-6)***
- serverDisconnect() ***(p. 19-6)***
- tranLogMonitorStart() ***(p. 19-6)***
- tranLogMonitorStop() ***(p. 19-7)***

12.2.3 Events

- onCommLogFailure(log_monitor_failure_type failure_code) ***(p. 19-7)***
- onCommLogRecord(Date timestamp, String comm_log_record) ***(p. 19-8)***
- onServerConnectFailure(server_failure_type failure_code) ***(p. 19-8)***
- onServerConnectStarted() ***(p. 19-9)***
- onTranLogFailure(log_monitor_failure_type failure_code) ***(p. 19-9)***
- onTranLogRecord(Date timestamp, String tran_log_record) ***(p. 19-10)***

Section 13. Developing an Application Using the CsiLogMonitor Control

13.1 Purpose

This section shows an example of how to build a simple application using the *CsiLogMonitor* control. The application's functions are:

1. Connect to a running LoggerNet server.

2. Monitor the LoggerNet server transaction and communication logs.

13.2 Using the *CsiLogMonitor* Control

13.2.1 Getting Started with the *CsiLogMonitor* Control

The *CsiLogMonitor* SDK control (an ActiveX® object) connects to the LoggerNet server and monitors transaction and communication logs.

This example assumes that:

- The *CsiLogMonitor* control has been correctly registered on the application host.

- A Windows® Forms application is to be developed using the Visual Studio® 2012 (or later) IDE and the VB.NET programming language.

- A LoggerNet server is currently running and accessible on the network.

- At least one station already exists in the LoggerNet server's network map.

Complete the following steps first:

1. Start Visual Studio and create a new Visual Basic® Windows Forms Application targeting the .NET Framework 4.0.

2. Following the procedures outlined in Section 1.4, *Developing .NET Applications Using the SDK (p. 1-3)*, add the *CsiLogMonitor* control to the *Toolbox* and create the RCW class.

3. In the Solution Explorer, right-click the *Form1.vb* file and rename it *frmLogMonitor.vb*.

13.2.2 *CsiLogMonitor* Control Application Example

Begin by modifying the blank form to create a Graphical User Interface (GUI) that supports the required functionality. The finished form should resemble the example shown in FIGURE 13-1.

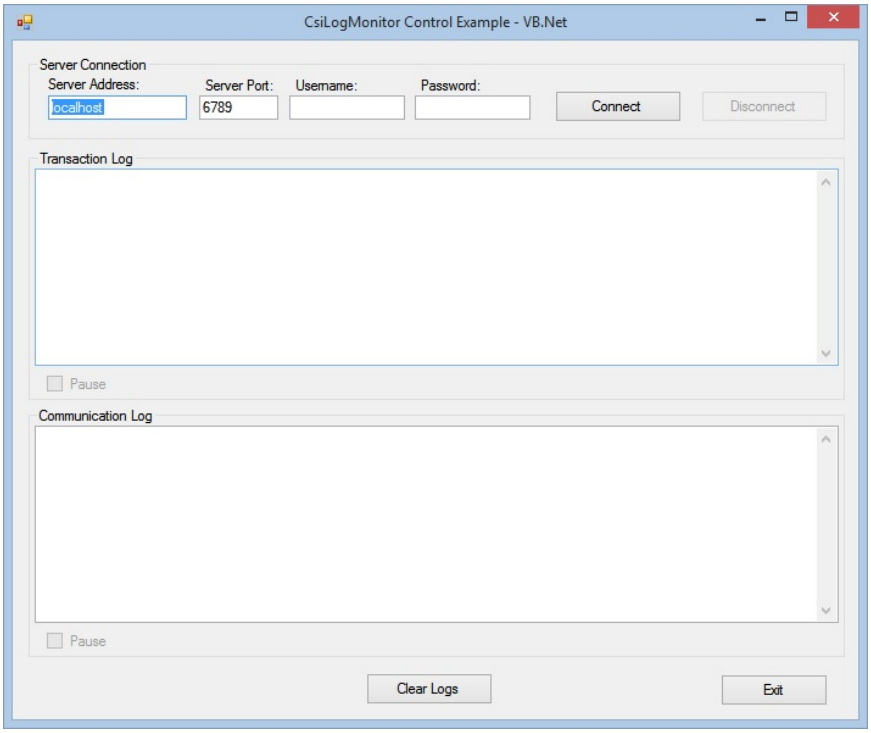

FIGURE 13-1. CsiLogMonitor Example

Once the interface has been designed, the necessary Visual Basic code can be added to attain the required functionality. The first order of business is to instantiate the RCW class (*CSILogMonitorLib.LogMonitor*). Add the following code to the *frmLogMonitor* class.

```
Dim WithEvents CsiLogMonitor As New LogMonitor
```

The following code snippets illustrate the basic functionality of the *CsiLogMonitor* control. For more comprehensive code examples, refer to the VB.NET example project files supplied with the SDK.

The first task of the application is to establish a connection to the LoggerNet server. The following code example illustrates using the *serverConnect()* method:

VB.NET Example 13-1. Establishing a Connection to the LoggerNet Server using the *serverConnect()* Method

```
Private Sub btnConnect_Click(sender As Object, e As EventArgs) Handles btnConnect.Click

  Try
    'Set connection properties before connecting.
    CsiLogMonitor.serverName = txtSvrAddress.Text
    CsiLogMonitor.serverPort = Convert.ToInt32(txtSvrPort.Text)
    CsiLogMonitor.serverLogonName = txtUsername.Text
    CsiLogMonitor.serverLogonPassword = txtPassword.Text

    'Call serverConnect().
    'If a connection is made, the control will raise the onServerConnectStarted() event.
    'If a connection is not made, the onServerConnectFailure() event will be raised.
    CsiLogMonitor.serverConnect()

  Catch ex As Runtime.InteropServices.COMException
     'If the call to the control causes an error, a custom HRESULT will be returned.
     'This HRESULOT will be captured in the InteropServices.COMException class
     'and cause the .Net Runtime to throw an exception.

     'We need to map the COMException.ErrorCode property to the values enumerated in
     'CSILogMonitorLib.HRESULT_Errors and display the associated error.
     Dim com_ex As CSILogMonitorLib.HRESULT_Errors = ex.ErrorCode
     MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CSILogMonitor Connect Button: Error")
  End Try
End Sub
```

If the connection succeeds, the *onServerConnectStarted()* event gets triggered. Otherwise, the *onServerConnectFailure()* event gets called if the connection fails.

In order to start monitoring the transaction log, the *tranLogMonitorStart()* method must be called. To monitor communication log messages, call the *commLogMonitorStart()* method.

The LoggerNet server maintains a buffer of historical log messages. By default, the last 100 log file messages will be retrieved when log monitoring first starts. To change the number of historical log messages that are retrieved, set the *commLogRecordsBack* and *tranLogRecordsBack* properties before starting log monitoring.

The following code example illustrates how the *onServerConnectStarted()* event can be handled to initialize properties and to start the collection of both types of log messages follows:

VB.NET Example 13-2. Handling the *onServerConnectStarted()* Event

```
Private Sub CsiLogMonitor_onServerConnectStarted() Handles CsiLogMonitor.onServerConnectStarted

  'This event is called when a successful connection has been made to the LoggerNet server.
  Try

    'Set the number of initial log records to display.
    CsiLogMonitor.tranLogRecordsBack = 25
    CsiLogMonitor.commLogRecordsBack = 25

    'Start the Trans log monitoring.
    'This method starts monitoring of the transaction log entries on the server.
    'The control will raise the onTranLogRecord() event as log entries are retrieved or
    'the onTranLogFailure() event if the method fails.
    CsiLogMonitor.tranLogMonitorStart()

    'Start the Comms log monitoring.
    'This method starts monitoring of the transaction log entries on the server.
    'The control will raise the onCommLogRecord() event as log entries are retrieved or
    'the onCommLogFailure() event if the method fails.
    CsiLogMonitor.commLogMonitorStart()

    'Update the form
    btnConnect.Enabled = False
    btnDisconnect.Enabled = True
    chkPauseComm.Enabled = True
    chkPauseTrans.Enabled = True

  Catch ex As Runtime.InteropServices.COMException
    'If the call to the control causes an error, a custom HRESULT will be returned.
    'This HRESULT will be captured in the InteropServices.COMException class
    'and cause the .Net Runtime to throw an exception.

    'We need to map the COMException.ErrorCode property to the values enumerated in
    'CSILogMonitorLib.HRESULT_Errors and display the associated error.
    Dim com_ex As CSILogMonitorLib.HRESULT_Errors = ex.ErrorCode
    MessageBox.Show(ex.Source & ": " & vbCrLf & com_ex.ToString, "A COM Exception was thrown")

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CsiLogMonitor onServerConnectStarted event: Error")
  End Try
End Sub
```

Log messages will be passed as Strings to the *onCommLogRecord()* and *onTranLogRecord()* events respectively as they are generated by the LoggerNet server. A timestamp for when the log message is generated is also passed to these events.

The following code example illustrates how the *onCommLogRecord()* event can be handled to display the log messages:

VB.NET Example 13-3. Handling the *onCommLogRecord()* Event

```
Private Sub CsiLogMonitor_onCommLogRecord(ByVal timestamp As Date, _
                                          ByVal comm_log_record As String) _
                                          Handles CsiLogMonitor.onCommLogRecord

  'This event is called when a Communication log record is passed from the server.
  Try
    'Limit the number of lines contained in the Communication Log textbox.
    If txtCommsLog.Lines.Length > 100 Then
        txtCommsLog.Clear()
    End If

    'Add the record to the Communication log textbox.
    If chkPauseComm.Checked = False Then
        txtCommsLog.Text += vbCrLf & timestamp.ToString & ": " & comm_log_record
        txtCommsLog.SelectionStart = txtCommsLog.Text.Length
        txtCommsLog.ScrollToCaret()
    End If

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CsiLogMonitor onCommLogRecord event: Error")
  End Try
End Sub
```

The following code example illustrates how the *onTranLogRecord()* event can be handled to display the log messages:

VB.NET Example 13-4. Handling the *onTranLogRecord()* Event

```
Private Sub CsiLogMonitor_onTranLogRecord(ByVal timestamp As Date, _
                                          ByVal tran_log_record As String) _
                                          Handles CsiLogMonitor.onTranLogRecord

  'This event is called when a Transaction log record is passed from the server
  Try
    'Limit the number of lines contained in the Transaction Log textbox.
    If txtTransLog.Lines.Length > 100 Then
      txtTransLog.Clear()
    End If

    'Add the record to the Transaction Log textbox.
    If chkPauseTrans.Checked = False Then
      txtTransLog.Text += vbCrLf & timestamp.ToString & ": " & tran_log_record
      txtTransLog.SelectionStart = txtTransLog.Text.Length
      txtTransLog.ScrollToCaret()
    End If
  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, _
                    "CsiLogMonitor onTranLogRecord event: Error")
  End Try
End Sub
```

Stop monitoring logs with the *commLogMonitorStop()* and *tranLogMonitorStop()* events. The following code example illustrates how the *commLogMonitorStop()* method can be used in conjunction with a checkbox control to pause and restart monitoring of the communication logs:

VB.NET Example 13-5. Using the *commLogMonitorStop()* Method to Pause and Restart Monitoring of Communication Logs

```
Private Sub chkPauseComm_CheckStateChanged(sender As Object, e As EventArgs) _
                                          Handles chkPauseComm.CheckStateChanged

  'We will use this checkbox to start and stop the Communication Log monitoring.
  Try
     If chkPauseComm.Checked = True Then
         CsiLogMonitor.commLogMonitorStop()
     Else
         'If connected to the server, start monitoring
         If CsiLogMonitor.serverConnected Then
             CsiLogMonitor.commLogMonitorStart()
         End If
     End If

  Catch ex As Exception
    MessageBox.Show(ex.Source & ": " & vbCrLf & ex.Message, "CsiLogMonitor
chkPauseComm_CheckStateChanged: Error")
  End Try
End Sub
```

Add additional functionality, error handling, and objects as necessary beyond the example interface and code listed above to meet the specific requirements of your application. Complete examples using the *CsiLogMonitor* control are included in the LoggerNet SDK installation.

Section 14. CsiServer and CsiServerDirect Control Reference

14.1 *CsiServer* and *CsiServerDirect* Interface

14.1.1 Properties

Server.applicationWorkDir

Name

```
Server.applicationWorkDir As String
```

Description

This property gives the location where the LoggerNet server data files are stored and must be set before starting *LoggerNet*. If this property needs to be changed after the LoggerNet server has been started, call *stopServer()*, set the new location, and then call *startServer()*.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The property cannot be set while a connection to the LoggerNet server is present

Server.buildDate

Name

```
Server.buildDate As String
```

Description

This read-only property displays the build date of the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_FAIL	Error: Unexpected error
E_CSI_NOT_STARTED	Error: The LoggerNet server is not started

Server.logFileDir

Name

```
Server.logFileDir As String
```

Description

This property specifies the location where the LoggerNet server writes log files and must be set before starting the LoggerNet server. If this property needs to be changed after the LoggerNet server has been started, call *stopServer()*, set the new location, and then call *startServer()*. By default, the log file directory will be placed in the LoggerNet server working directory.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The property cannot be set while a connection to the LoggerNet server is present

Server.serverStarted

Name

```
Server.serverStarted As Boolean
```

Description

This read-only value displays the current state of a LoggerNet server that has been started by the server control. If the LoggerNet server is running, this value will be TRUE. Otherwise, this value will be FALSE.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Server.serverVersion

Name

```
Server.serverVersion As String
```

Description

This property is a read-only value that displays the version of the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_FAIL	Error: Unexpected error
E_CSI_NOT_STARTED	Error: The LoggerNet server is not started

Server.serverWorkDir

Name

```
Server.serverWorkDir As String (Required)
```

Description

This required property must be specified before starting the LoggerNet server and describes the location of the LoggerNet server configuration files. This property must be set before starting the LoggerNet server or the *startServer()* event will fail. If this location needs to be changed after the LoggerNet server has been started, call *stopServer()*, set the new location, and then call *startServer()*.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The property cannot be set while a connection to the LoggerNet server is present

Server.tcpPort

Name

```
Server.tcpPort As Integer
```

Description

This property sets the TCP port that the LoggerNet server will use when listening for client connections and must be set before starting the LoggerNet server. *LoggerNet* uses the TCP port 6789 by default. This property accepts 1 to 32767 as valid values.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Server.tcpPortEx

Name

```
Server.tcpPortEx As Long
```

Description

This property sets the TCP port that the LoggerNet server will use when listening for client connections and must be set before starting the LoggerNet server. *LoggerNet* uses TCP port 6789 by default. This property accepts the full range of valid TCP port numbers 1 to 65535.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

14.1.2 Methods

Server.startServer()

Name

```
Server.startServer()
```

Description

This method starts the LoggerNet server. The Coralib3.dll or the Coralib3d.dll must exist in the application folder, the PATH environmental variable, or the Windows® directory or this method will fail.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_ALREADY_STARTED	Error: This error is returned if the server control has already started the LoggerNet server
E_CSI_INVALIDARG	Error: No working directory set
E_CSI_FAIL	Error: Another LoggerNet server not started by the server control is already running or an unexpected error has occurred

Server.stopServer()

Name

```
Server.stopServer()
```

Description

This method will stop the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

14.1.3 Events

Server_onServerFailure()

Name

```
onServerFailure(String reason)
```

Description

This event gets triggered when the LoggerNet server started by the server control fails.

Section 15. CsiCoraScript Control Reference

15.1 *CoraScript* Interface

15.1.1 Properties

CoraScript.serverConnected

Name

```
CoraScript.serverConnected As Boolean (read-only)
```

Description

This Boolean property describes the state of the connection between the *CoraScript* control and the LoggerNet server. The property returns `TRUE` if the connection exists. Otherwise, the property returns `FALSE`.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return

CoraScript.serverLogonName

Name

```
CoraScript.serverLogonName As String
```

Description

Specifies the account name that should be used when connecting to the LoggerNet server. If security is enabled on the target LoggerNet server, this string must be one of the account names recognized by the LoggerNet server.

Valid Values

If security is enabled on the target LoggerNet server, this string must be one of the account names recognized by the LoggerNet server.

Default Value

The default value for this property is an empty string. This property will only affect the operation of the control if security is enabled on the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The property cannot be set while a connection to the LoggerNet server is present

CoraScript.serverLogonPassword

Name

```
CoraScript.serverLogonPassword As String
```

Description

This property specifies the password that should be used when connecting to the LoggerNet server. If security is enabled on the target LoggerNet server, this password string must be associated with the account described in the logonName property.

Valid Values

If security is enabled on the target LoggerNet server, this string must be the password associated with the account named by CoraScript.serverLogonName.

Default Value

The default value for this property is an empty string. This property will only affect the operation of the control if security is enabled on the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The property cannot be set while a connection to the LoggerNet server is present

CoraScript.serverName

Name

```
CoraScript.serverName As String
```

Description

Specifies the TCP/IP interface address for the computer hosting the LoggerNet server. This string must be formatted either as a qualified Internet machine domain name or as an Internet address string. An example of a valid machine domain name address is *www.campbellsci.com*. An example of a valid Internet address string is `63.255.173.183`.

The default value for this property is the string `localhost`.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_BUSY`	Error: Attempt to set serverName while connected to the LoggerNet server

CoraScript.serverPort

Name

```
CoraScript.serverPort As Long
```

Description

Specifies the TCP port number that the LoggerNet server is using on the hosting computer. The valid range for this property is port 1 to port 65535.

The default value for this property is port 6789, which is the default port number assigned for the LoggerNet server. The default value for this property will connect to a LoggerNet server port in most cases.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_INVALIDARG`	Error: The port value is out of range or invalid
`E_CSI_BUSY`	Error: Attempt to set serverPort while connected to the LoggerNet server

15.1.2 Methods

CoraScript.executeScript()

Name

```
CoraScript.executeScript(String script, Long asychID)
As String
```

Description

This method allows a single *CoraScript* command to be executed by the LoggerNet server. Pass the *CoraScript* command in as the first parameter and use the second parameter to determine whether the method performs asynchronously or synchronously. If you want this command to execute synchronously, pass in a zero (0) for the asyncID. If an asyncID other than zero (0) is specified, the *onScriptComplete()* event will be triggered with the result and the asyncID that was specified.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_FAIL	Error: Unexpected Error

CoraScript.serverConnect()

Name

```
CoraScript.serverConnect()
```

Description

This method attempts to connect to the LoggerNet server using the values in the previously set properties: serverName, serverPort, serverLogonName, and serverLogonPassword. This method triggers *onServerConnectStarted()* if the connection is successful, or *onServerConnectFailure()* if the connection fails.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_FAIL	Error: Unexpected error

CoraScript.serverDisconnect()

Name

```
CoraScript.serverDisconnect()
```

Description

This method will disconnect from the LoggerNet server and will set the `serverConnected` state to `FALSE`. This method should only be called when the value of `serverConnected`, is `TRUE`. Otherwise, this method will return `E_CSI_NOT_CONNECTED`.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_NOT_CONNECTED	Error: The server is not connected

15.1.3 Events

CoraScript_onScriptComplete()

Name

```
onScriptComplete(Long asyncID, String result)
```

Description

This event displays the results from the method *CoraScript.executeScript()*. However, this event is only activated when an asyncID other than "0" is passed to that method.

CoraScript_onServerConnectStarted()

Name

```
onServerConnectStarted()
```

Description

The control has connected to the LoggerNet server.

CoraScript_onServerConnectFailure()

Name

```
onServerConnectFailure(server_failure_type
server_failure)
```

Description

An error has occurred that caused the connection to the LoggerNet server to fail for this control.

Table of Possible Failure Codes

Enumeration Name	Value	Description
server_failure_unknown	0	Indicates that an error has occurred but its nature is unknown
server_failure_logon	1	Indicates that this control was unable to logon to the LoggerNet server because either the logonName or logonPassword property is incorrect
server_failure_session	2	Indicates that the communication session with the LoggerNet server failed resulting in a failed transaction
server_failure_unsupported	3	The version of the LoggerNet server does not support this transaction

Enumeration Name	Value	Description
server_failure_security	4	Indicates that the account specified by logonName does not have sufficient privileges to start this transaction with the LoggerNet server
server_failure_bad_host_or_port	5	Indicates that either the serverName or the serverPort property is incorrect

Section 16. CsiBrokerMap Control Reference

16.1 *BrokerMap* Interface

16.1.1 Properties

BrokerMap.serverName

Name

```
BrokerMap.serverName As String
```

Description

Specifies the TCP/IP interface address for the computer that is hosting the LoggerNet server. This string must be formatted either as a fully qualified Internet machine domain name or as an IP address string. An example of a valid machine domain name address is *www.campbellsci.com*. An example of a valid IP address string is `63.255.173.183`.

Default Value

The default value for this property is the string `localhost`.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_BUSY`	Error: Attempt to set serverName while connected to the server

BrokerMap.serverLogonName

Name

```
BrokerMap.serverLogonName As String
```

Description

Specifies the account name that should be used when connecting to the LoggerNet server. If security is enabled on the target LoggerNet server, this string must be one of the account names recognized by the LoggerNet server.

Valid Values

If security is enabled on the target LoggerNet server, this string must be an account name recognized by the LoggerNet server. These accounts can be set up using the *Security Manager* that is part of the *LoggerNet Admin* software suite or through the *CsiCoraScript* control.

Default Value

The default value for this property is an empty string.

Notes

This property is only used if security is enabled on the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Attempt to set serverLogonName while connected to the LoggerNet server

BrokerMap.serverLogonPassword

Name

```
BrokerMap.serverLogonPassword As String
```

Description

This property specifies the password that should be used when connecting to the LoggerNet server. If security is enabled on the target LoggerNet server, this password string must be associated with the account described in the logonName property.

Valid Values

If security is enabled on the target LoggerNet server, this string must be a valid password associated with the account described in the serverLogonName property.

Default Value

The default value for this property is an empty string.

Notes

This property is only used if security is enabled on the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Attempt to set serverLogonPassword while connected to the LoggerNet server

BrokerMap.serverPort

Name

```
BrokerMap.serverPort As Long
```

Description

Specifies the TCP port number that the LoggerNet server is using on the hosting computer. The valid range for this property is 1 to 65535.

Default Value

The default value for this property, assigned to the LoggerNet server, is 6789. In most cases, the default value for this property is acceptable.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Attempt to set serverPort while connected to the LoggerNet server
E_CSI_INVALIDARG	Error: The port value is invalid (out of range)

BrokerMap.autoExpand

Name

```
BrokerMap.autoExpand As Boolean
```

Description

This setting determines if the broker will automatically expand to include all brokers and tables or if the *Broker.start_expansion()* method must be called to list all the brokers and *Table.start_expansion()* method to list all tables for each broker. If the list of brokers and tables is extensive, it may be quicker to list the brokers and expand the tables for each broker separately. The default setting is TRUE, which means that all brokers and tables will be expanded automatically.

Default Value

The default value for this property is TRUE.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

BrokerMap.serverConnected

Name

```
BrokerMap.serverConnected As Boolean
```

Description

This property describes the state of the connection between the BrokerMap control and the LoggerNet server. If the connection is active, the property is TRUE. Otherwise, the property is FALSE.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal Return

16.1.2 Methods

BrokerMap.Brokers()

Name

```
BrokerMap.Brokers() As Object
```

Description

Use this method to iterate through the brokers and return a broker collection.

BrokerMap.finish()

Name

```
BrokerMap.finish()
```

Description

This method tells the control to discontinue sending events or changes to the brokers, which holds the current broker map in a static format for your application. This method should only be called after the *start()* method has been invoked. Calling this method will cause the control to disconnect from the server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

BrokerMap.start()

Name

```
BrokerMap.start()
```

Description

This method starts the broker map query to get the brokers, tables, and columns. Immediately following the invocation of this method, the events *onBrokerAdded()* and *onTableAdded()* will follow to describe the brokers and tables currently in the broker map.

If there is already a connection to the server, this method will return the error `E_CSI_ALREADY_CONNECTED`. If an error occurs while trying to connect, this method will return the error `E_CSI_BAD_HOST_OR_PORT`.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_ALREADY_CONNECTED`	Error: Already connected to the LoggerNet server
`E_CSI_BAD_HOST_OR_PORT`	Error: Cannot connect. Property serverName or serverPort possibly wrong

16.1.3 Events

BrokerMap_onAllStarted()

Name

```
onAllStarted()
```

Description

This event is a result of invoking the *start()* method. This event gets called after all of the initial *onBrokerAdded()* and *onTableAdded()* events have been called from the *start()* method and the broker map is known.

BrokerMap_onBrokerAdded()

Name

```
onBrokerAdded(Object Broker)
```

Description

This event gets called as new brokers are added to the broker map. Information about the new broker can be accessed with the broker object returned with this event.

BrokerMap_onBrokerDeleted()

Name

```
onBrokerDeleted(Object Broker)
```

Description

This event gets called as brokers are deleted from the broker map. Information about the broker deleted from the broker map can be accessed with the broker object returned with this event. After the broker object returned by this event goes out of scope, the referenced object in the control will be permanently deleted. The broker is kept alive for this event so that its properties can be referenced by the client application one last time.

BrokerMap_onFailure()

Name

```
onFailure(BrokerMapFailureType failure_code)
```

Description

When the BrokerMap control fails, an error from the following table will be returned with this event:

Table of Failure Codes

Name	Value	Description
failure_unknown	0	The cause of the failure could not be determined
failure_connection_failed	1	The connection has failed. Check the serverName and serverPort
failure_invalid_logon	2	The LoggerNet server has security enabled and the logon is invalid. Check serverLogonName and serverLogonPassword
failure_server_security	3	The LoggerNet server has security enabled and you do not have sufficient privileges to complete this transaction
failure_table_browser	4	There has been an error while getting table information

BrokerMap_onTableAdded()

Name

```
onTableAdded(Object Broker, Object Table)
```

Description

This event gets called when a new table is added to a broker in the broker map. Information about the table added to the broker in the broker map can be accessed with the table object and broker object returned by this event.

BrokerMap_onTableDeleted()

Name

```
onTableDeleted(Object Broker, Object Table)
```

Description

This event gets called when a table is deleted from a broker in the broker map. The table that was deleted will be returned as a broker object and a table object with this event.

BrokerMap_onTableChanged()

Name

```
onTableChanged(Object Broker, Object Table)
```

Description

This event executes when a table in a broker changes. Information about the broker and table that changed are returned with this event.

BrokerMap_onBrokerStarted()

Name

```
onBrokerStarted(Object Broker)
```

Description

An event that indicates a broker is in a started state. Information about the broker is returned with this event.

16.2 *BrokerCollection* Interface

16.2.1 Properties

BrokerCollection.Count

Name

```
BrokerCollection.Count As Long
```

Description

This property returns the number of brokers in the network map.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

16.2.2 Methods

BrokerCollection.Item()

Name

```
BrokerCollection.Item(id) As Broker
```

Description

A broker can be referenced by an integer, a long, or by the name of the broker (a string). If the number is less than zero or is greater than the number of brokers minus one, the COM error E_CSI_ARRAY_OUT_OF_BOUNDS will be returned. If the broker cannot be found by name, the COM error E_CSI_NOT_FOUND will be returned.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_ARRAY_OUT_OF_BOUNDS	Error: Array out of bounds
E_CSI_NOT_FOUND	Error: Couldn't find the broker by name in the broker map
E_CSI_FAIL	Error: Wrong variant type passed to this method or unexpected error

Visual Basic®

Return Type
Broker

Example
Referencing the broker by a number value

```
Dim iterator As Long
For iterator = 0 to BrokerMap.Broker.Count - 1
 Debug.Print
 BrokerMap.Brokers(iterator).ID
Next iterator
```

Referencing the broker by name:

```
Dim brokerName as String
Dim myid as long
brokerName = "cr10x"
myid = BrokerMap.Brokers(brokerName).id
```

BrokerCollection._NewEnum()

Name

```
BrokerCollection._NewEnum()
```

Description

Return the next broker in the broker map sequence.

Important

This method is only intended for use with the Visual Basic programming language. Visual Basic programmers do not need to access this method directly but can use it indirectly with the **For Each** loop. This method is included in the documentation to explain why the method exists, but, again, it is not accessed directly.

Visual Basic

Example

```
Dim b As Broker
For Each b in BrokerMap.Brokers
  Debug.print b.name
Next
```

16.3 *Broker* Interface

16.3.1 Properties

Broker.ID

Name

```
Broker.id As Long
```

Description

This is a read-only property describing the unique ID of each broker.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Broker.name

Name

```
Broker.name As String
```

Description

This read-only property returns the name of a broker.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Broker.type

Name

```
Broker.type As BrokerType
```

Description

This read-only property returns the type of the broker.

Possible Values

Table of Broker Type Enumeration

Name	Value	Description
broker_active	1	The data broker associated with the current configuration of a device object
broker_backup	2	A data broker associated with a previous configuration of a device object
broker_client	3	A data broker created at the request of a client
broker_statistics	4	A data broker created by the LoggerNet server to report operating statistics

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Broker.datalogger_type

Name

```
Broker.datalogger_type As String
```

Description

The read-only device type of the broker.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Broker.allStarted

Name

```
Broker.allStarted As Boolean
```

Description

Set to TRUE when all the tables for the broker have been reported.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

16.3.2 Methods

Broker.Tables()

Name

```
Broker.Tables() As Object
```

Description

This method returns a reference to a TableCollection, which can be used to iterate through the tables in a broker.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Broker.start_expansion()

Name

```
Broker.start_expansion()
```

Description

If the BrokerMap autoExpand property has been set to FALSE, use this method to access the list of tables for a broker.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal Return

16.4 *TableCollection* Interface

16.4.1 Properties

TableCollection.Count

Name

```
TableCollection.Count As Long
```

Description

This property returns the number of tables in a TableCollection.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

16.4.2 Methods

TableCollection.Item()

Name

```
TableCollection.Item(id) As Table
```

Description

Returns the requested table if it exists. A table can be referenced by a number (like an index) or by a string (the name of the table). If the number is less than zero or is greater than the number of tables, the error E_CSI_ARRAY_OUT_OF_BOUNDS will be returned. If the table cannot be found by name, the error E_CSI_NOT_FOUND will be returned.

Prototypes

TableCollection.Item(Number) – Array index
TableCollection.Item(String) – Table name

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_ARRAY_OUT_OF_BOUNDS	Error: Array subscript out of bounds
E_CSI_NOT_FOUND	Error: Table not found by name in the broker map
E_CSI_FAIL	Error: Wrong variant type passed or unexpected error

Visual Basic

Return Type
Table

Example
By number:

```
long iterator
For iterator = 0 to BrokerMap.Broker("cr9000").Tables.Count - 1
 Debug.Print BrokerMap.Brokers("cr9000").Tables.ID
Next iterator
```

By string:

```
Dim tableName as String
Dim myid as long
tableName = "cr10x"
myid = BrokerMap.Broker("cr9000").Tables(tableName).id
```

TableCollection._NewEnum()

Name

```
TableCollection._NewEnum()
```

Description

Return the next Table in the sequence.

Important

This method is only intended for use with Visual Basic. Visual Basic programmers do not need to access this method directly. They use it indirectly by using the collections with the **For Each** loop. This method is included in the documentation to explain why the method exists, but, again, it is not accessed directly.

16.5 *Table* Interface

16.5.1 Properties

Table.interval

Name

```
Table.interval As Long
```

Description

The time interval between records. If the table is event-driven, a value of zero will be used.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Table.name

Name

```
Table.name As String
```

Description

This read-only property returns the name of the table.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Table.originalSize

Name

```
Table.originalSize As Long
```

Description

This property returns the number of records that can be stored in the original datalogger table.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Table.size

Name

```
Table.size As Long
```

Description

This property returns the number of records that can be stored in this table.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

16.5.2 Methods

Table.Columns()

Name

```
Table.Columns() As Object
```

Description

This method is used as a reference for a ColumnCollection to get the columns of a table.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Visual Basic

Example

```
Dim cc As ColumnCollection
Set cc = BrokerMap.Brokers("cr9000").Tables("public").Columns
```

Table.start_expansion

Name

```
Table.start_expansion()
```

Description

If the BrokerMap autoExpand property has been set to FALSE, use this method to access the list of columns for a table within a broker.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal Return

16.6 *ColumnCollection* Interface

16.6.1 Properties

ColumnCollection.Count

Name

```
ColumnCollection.Count As Long
```

Description

This property returns the number of columns in the ColumnCollection.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

16.6.2 Methods

ColumnCollection.Item()

Name

```
ColumnCollection.Item(id) As Column
```

Description

This method returns the reference id for a column. If the number is less than zero or is greater than the number of columns, the error E_CSI_ARRAY_OUT_OF_BOUNDS will be returned. If the column cannot be found by name, the error E_CSI_NOT_FOUND will be returned.

Prototypes

ColumnCollection.Item(Number) – Array index
ColumnCollection.Item(String) – Table name

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_ARRAY_OUT_OF_BOUNDS	Error: Array out of bounds
E_CSI_NOT_FOUND	Error: Column not found in broker map by name
E_CSI_FAIL	Error: Wrong variant type passed or unexpected error

Visual Basic

Return Type
Column

Examples

```
(1)
Dim myColumn as Column
BrokerMap.Brokers("cr9000").Tables("public").Columns.Item(0)

(2)
Dim myColumn as Column
BrokerMap.Brokers("cr9000").Tables("public").Columns(0)

(3)
Dim myColumn as Column
BrokerMap.Brokers("cr9000").Tables("public").Columns.Item("speed")

(4)
Dim myColumn as Column
BrokerMap.Brokers("cr9000").Tables("public").Columns("speed")

Examples (1) and (2) are equivalent, as well as examples
(3) and (4).  The default method for collection interfaces
is Item().
```

ColumnCollection._NewEnum()

Name

```
ColumnCollection._NewEnum()
```

Description

Return the next Column in the sequence.

Important This method is only intended for use with Visual Basic. Visual Basic programmers do not need to access this method directly. They use it indirectly by using the collections with the **For Each** loop. This method is included in the documentation to explain why the method exists, but, again, there is no need to access this method directly.

16.7 *Column* Interface

16.7.1 Properties

Column.description

Name

```
Column.description As String
```

Description

This read-only property returns a description of the column.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Column.name

Name

```
Column.name As String
```

Description

This read-only property returns the name of the column.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Column.process

Name

```
Column.process As String
```

Description

A read-only property that identifies the processing performed on the data. For data coming from table-data and mixed-array dataloggers, this value will be an empty string.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Column.type

Name

```
Column.type As CsiDataTypeCode
```

Description

This read-only property identifies the type of data for the column. Following are the possible values for this enumerated property:

Table of Data Type Enumeration

Name	Value	Description
dt_CsiUInt1	1	1 byte unsigned int
dt_CsiUInt2	2	2 byte unsigned int
dt_CsiUInt4	3	4 byte unsigned int
dt_CsiInt1	4	1 byte signed int
dt_CsiInt2	5	2 byte signed int
dt_CsiInt4	6	4 byte signed int
dt_CsiInt8	32	8 byte signed int
dt_CsiFs2	7	2 byte final storage (also known as FP2)
dt_CsiFs3	15	3 byte final storage (also known as FP3)
dt_CsiFs4	26	4 byte final storage
dt_CsiFsf	27	allows storage of either CsiFs2 or CsiFs4 Requires 4 bytes
dt_CsiFp4	8	4 byte CSI float
dt_CsiIeee4	9	4 byte IEEE float
dt_CsiIeee8	18	8 byte IEEE float
dt_CsiBool	10	1 byte Boolean (0 or 1)
dt_CsiBool8	17	1 byte bit field
dt_CsiSec	12	4 byte sec since 1 Jan 1990
dt_CsiUSec	13	6 byte 10s of microseconds since 1 Jan 1990
dt_CsiNSec	14	4 byte sec since 1 Jan 1990 + 4 byte nanoseconds
dt_CsiAscii	11	fixed-length string
dt_CsiAsciiZ	16	null-terminated variable-length string
dt_CsiInt4Lsf	20	4 byte signed int (LSB first)
dt_CsiUInt2Lsf	21	2 byte signed int (LSB first)

Name	Value	Description
dt_CsiUInt4Lsf	22	4 byte signed int (LSB first)
dt_CsiNSecLsf	23	same as nanoseconds with the components in LSB
dt_CsiIeee4Lsf	24	4 byte IEEE float (LSB first)
dt_CsiIeee8Lsf	25	8 byte IEEE float (LSB first)
dt_CsiInt8Lsf	33	8 byte signed int (LSB first)
dt_CsiBool2	30	2 byte Boolean (non-zero = true)
dt_CsiBool4	31	4 byte Boolean (non-zero = true)
dt_CsiInt2Lsf	19	2 byte signed int (LSB first)
dt_CsiLgrDate	29	8 bytes of nanoseconds since 1990
dt_CsiLgrDateLsf	28	8 bytes of nanoseconds since 1990 (LSB first)

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Column.units

Name

```
Column.units As String
```

Description

This read-only property identifies the data engineering units.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Column.writable

Name

```
Column.writable As Long
```

Description

This property is read-only and describes whether or not this column can be changed or set by using the *variableSet()* method as described in the *CsiDatalogger* control.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Section 17. CsiDatalogger Control Reference

17.1 *CsiDatalogger* Interface

17.1.1 Properties

Datalogger.clockBusy

Name

```
Datalogger.clockBusy As Boolean
```

Description

This property describes the state of the control concerning clock transactions. If a clock check or a clock set is currently executing, clockBusy returns `TRUE`, and any attempt to execute another clock check or clock set will return an error.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return

Datalogger.loggerConnected

Name

```
Datalogger.loggerConnected As Boolean
```

Description

This Boolean property describes the state of the LoggerNet server connection management invoked from *loggerConnectStart()*. This property only describes the state of connection management not the state of the physical connection to the datalogger. To monitor the physical line state, start an advisor with the DataSource control and monitor the statistics table for that device. For information on devices statistics tables, look in the appendix of this document.

If connection management is active, a persistent connection between the server and the datalogger is present or in process. This type of connection can be very useful if you must make requests to the datalogger on a frequent basis because you avoid reconnection overhead for each request. To turn off active connection management, see *loggerConnectCancel* *(p. 17-8)*.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return

Datalogger.loggerName

Name

```
Datalogger.loggerName As String
```

Description

Specifies the datalogger or station name that will be accessed.

Valid Values

This property must match one of the actual datalogger device names in the LoggerNet server network map.

Default Value

The default value for this property is an empty string.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The property cannot be set while a connection to the datalogger is present

Datalogger.manualPollBusy

Name

```
Datalogger.manualPollBusy As Boolean
```

Description

This Boolean property describes the state of the control concerning a manual poll. If a manual poll is currently executing then manualPollBusy will return TRUE, and any attempt to execute another manual poll will return an error.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger.programReceiveBusy

Name

```
Datalogger.programReceiveBusy As Boolean
```

Description

This read-only, Boolean property describes the state of the LoggerNet server in relation to the method *programReceiveStart()*. If the LoggerNet server is

currently retrieving a program from the datalogger, this property will return TRUE.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger.programSendBusy

Name

```
Datalogger.programSendBusy As Boolean
```

Description

This Boolean property describes the state of the LoggerNet server in relation to the method *programSendStart()*. If the LoggerNet server is currently sending a program to the datalogger, this property will return TRUE.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger.selectiveManualPollBusy

Name

```
Datalogger.selectiveManualPollBusy As Boolean
```

Description

This Boolean property describes the state of the control concerning a selective manual poll. If a selective manual poll is currently in process, this property will return TRUE.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger.serverConnected

Name

```
Datalogger.serverConnected As Boolean
```

Description

This Boolean property describes the state of the connection between the client application and the LoggerNet server. If the connection is successful, the property is returned as `TRUE`. Otherwise, the property is returned as `FALSE`.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return

Datalogger.serverLogonName

Name

```
Datalogger.serverLogonName As String
```

Description

Specifies the account name that should be used when connecting to the LoggerNet server. If security is enabled on the target LoggerNet server, this string must be one of the account names recognized by the LoggerNet server.

Valid Values

If security is enabled on the target LoggerNet server, this property must be one of the account names recognized by the LoggerNet server. These accounts can be set up using the *LoggerNet Security Administration Client* that is part of the *LoggerNet* software suite or the *CsiCoraScript* control that is part of the SDK.

Default Value

The default value for this property is an empty string. This property is only used if security is enabled on the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_BUSY`	Error: The property cannot be set while a connection to the LoggerNet server is present

Datalogger.serverLogonPassword

Name

```
Datalogger.serverLogonPassword As String
```

Description

This property specifies the password that should be used when connecting to the LoggerNet server. If security is enabled on the target LoggerNet server, this password string must be associated with the account described in the logonName property.

Valid Values

If security is enabled on the target LoggerNet server, this property must be the password associated with the account described by serverLogonName.

Default Value

The default value for this property is an empty string. This property is only used if security is enabled on the LoggerNet server.

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_BUSY`	Error: The property cannot be set while a connection to the LoggerNet server is present

Datalogger.serverName

Name

```
Datalogger.serverName As String
```

Description

This property specifies the TCP/IP interface address for the computer hosting the LoggerNet server. This string must be formatted either as a fully qualified Internet machine domain name or as an IP address string. An example of a valid machine domain name address is *www.campbellsci.com*. An example of a valid IP address string is `207.201.118.35`. The default value for this property is the string, `localhost`.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_BUSY`	Error: Attempt to set serverName while connected to the LoggerNet server

Datalogger.serverPort

Name

```
Datalogger.serverPort As Long
```

Description

Specifies the TCP port number that the LoggerNet server is using on the hosting computer. The valid range for this property is 1 to 65535.

Default Value

The default value for this property, assigned to the LoggerNet server during install, is 6789. In most cases, the default value for this property is acceptable.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_INVALIDARG	Error: The port value is invalid
E_CSI_BUSY	Error: Attempt to set serverPort while connected to the LoggerNet server

17.1.2 Methods

Datalogger.clockCancel()

Name

```
Datalogger.clockCancel()
```

Description

This method should be called to cancel either a *clockCheckStart()* or a *clockSetStart()*. If the clock set or clock check was successfully cancelled, the event *onClockComplete()* will return a cancellation code. If the *clockCancel()* was called too late in the process, the event *onClockComplete()* will return either a success or failure code instead. This method should only be called when the *clockCheckStart()* method or the *clockSetStart()* method is in process.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger.clockCheckStart()

Name

```
Datalogger.clockCheckStart()
```

Description

This method should be called to check the date and time on a specified datalogger. This method should only be called when the value of serverConnected is TRUE. If not, this method will return E_CSI_NOT_CONNECTED. Upon completion, this method will fire the event onClockComplete.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Clock communication is busy servicing a request
E_CSI_FAIL	Error: Unexpected error
E_CSI_NOT_CONNECTED	Error: Not connected to the LoggerNet server

Datalogger.clockSetStart()

Name

```
Datalogger.clockSetStart()
```

Description

This method should be called to set the date and time on the specified datalogger to the date and time of the LoggerNet server. This method should only be called when the value of serverConnected is TRUE. If not, this method will return E_CSI_NOT_CONNECTED. Upon completion, this method calls the event onClockComplete.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Clock communication is busy servicing a request
E_CSI_FAIL	Error: Unexpected error
E_CSI_NOT_CONNECTED	Error: Not connected to the LoggerNet server

Datalogger.loggerConnectCancel()

Name

```
Datalogger.loggerConnectCancel()
```

Description

This method cancels an active connection between the LoggerNet server and the specified datalogger. When a persistent connection is cancelled, the LoggerNet server returns to the default behaviour of connecting to the datalogger for each transaction and disconnecting from the datalogger after each transaction finishes.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger.loggerConnectStart()

Name

```
Datalogger.loggerConnectStart(logger_priority_type
priority)
```

Parameters

The following values indicate the priority of maintaining the connection when other devices might need the resources:

Table of 'Priority' Values

priority_high = 0
priority_normal = 1
priority_low = 2

Description

This method will open a persistent connection to the specified datalogger. Keeping the connection open will allow the LoggerNet server to handle multiple transactions without disconnecting. The default behaviour of the server is to shut down a link unless there is a reason (a client sponsored transaction, a setting such as PakBus® port always open or hangup delay, or an internal transaction (scheduled poll) pending). Keeping the connection open is very helpful if it takes a considerable amount of time for the server to connect to a datalogger, such as on a dialup connection. In most cases, a persistent connection is not required.

This method should only be called when the value of serverConnected, is TRUE. If not, this method will return E_CSI_NOT_CONNECTED. This method triggers onLoggerConnectStarted or onLoggerConnectFailure, depending on its success or failure.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: A persistent communications link has already been started with this datalogger
E_CSI_FAIL	Error: Unexpected error
E_CSI_NOT_CONNECTED	Error: Not connected to the LoggerNet server

Datalogger.manualPollCancel()

Name

```
Datalogger.manualPollCancel()
```

Description

This method should be called to cancel a *manualPollStart()* command. If the manual poll was successfully cancelled, the event *onManualPollComplete()* will return a cancellation code. If the *manualPollCancel()* was called too late in the manual poll process, the event *onManualPollComplete()* will return either a success or failure code instead. This method should only be called when a manual poll is in process.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger.manualPollStart()

Name

```
Datalogger.manualPollStart()
```

Description

This method will initiate a collection from all areas that are marked to be polled with scheduled collection. This method should only be called when the value of serverConnected is TRUE. If not, this method will return E_CSI_NOT_CONNECTED. Upon completion, this method calls the event *onManualPollComplete()*.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Manual poll communication is busy servicing a request
E_CSI_FAIL	Error: Unexpected error
E_CSI_NOT_CONNECTED	Error: Not connected to the LoggerNet server

Datalogger.programReceiveCancel()

Name

```
Datalogger.programReceiveCancel()
```

Description

This method attempts to cancel the *programReceiveStart()* command. Mixed-array dataloggers will not recognize this request and will continue to transfer their program even though the datalogger control is no longer receiving it.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger.programReceiveStart()

Name

```
Datalogger.programReceiveStart(String fileName)
```

Description

This method retrieves the current program from the connected datalogger and saves that file as the specified filename. This event triggers *onProgramReceiveProgress()* and *onProgramReceiveComplete()* during the *programReceive()* and after the *programReceive()* respectively.

This method should only be called when the value of serverConnected is TRUE. If not, this method will return E_CSI_NOT_CONNECTED.

Parameters

FileName: This location is the full path and name where the file will be saved.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: A previous call to programReceiveStart() has not completed
E_CSI_FAIL	Error: Unexpected error
E_CSI_NOT_CONNECTED	Error: Not connected to the LoggerNet server

Datalogger.programSendCancel()

Name

```
Datalogger.programSendCancel()
```

Description

This method attempts to cancel the *programSendStart()* method. The program send process can be cancelled if it has not already begun. Otherwise, the method will be ignored.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger.programSendStart()

Name

```
Datalogger.programSendStart(String file_name, String
program_name)
```

Description

This method starts to transfer a file designated by file_name to the specified datalogger. It also calls the events: *onProgramSendProgress()*, *onProgramSent()*, and *onProgramSendComplete()*. This method should only be called when the value of serverConnected is TRUE. Otherwise, this method will return E_CSI_NOT_CONNECTED.

Parameters

file_name: The full path on the local machine designating the location of the program that will be sent.

program_name: Designates the name of the program that will be sent to the specified datalogger. The file will be placed on the "CPU" device by default; there are currently no other options.

The file name should have no path specification, but should merely be the name of the file. If this setting is specified as an empty string, the name will be derived from the file_name property. The server may truncate the file name on Crx000 dataloggers in order to make it fit the file system on those devices.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: A previous call to programSendStart() has not completed
E_CSI_FAIL	Error: Unexpected error
E_CSI_NOT_CONNECTED	Error: Not connected to the LoggerNet server

Datalogger.selectiveManualPollCancel

Name

```
Datalogger.selectiveManualPollCancel()
```

Description

This method should be called to cancel a *selectiveManualPollStart()* command. If the selective manual poll was successfully cancelled, the event *onSelectiveManualPollComplete()* will return a selective manual poll aborted code. If the *selectiveManualPollCancel()* was called too late in the manual poll process, the event *onSelectiveManualPollComplete()* will return either a success or failure code instead. This method should only be called when a selective manual poll is in process.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger.selectiveManualPollStart

Name

```
Datalogger.selectiveManualPollStart(collect_area As
String)
```

Description

Use this method to poll a specific table in a datalogger. Upon completion, this method calls the event *onSelectiveManualPollComplete()*.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: A previous call has not completed
E_CSI_FAIL	Error: Unexpected error
E_CSI_NOT_CONNECTED	Error: Not connected to the LoggerNet server

Datalogger.serverConnect()

Name

```
Datalogger.serverConnect()
```

Description

This method attempts to connect to the LoggerNet server using the previously set properties: serverName, serverPort, serverLogonName, and serverLogonPassword. This method triggers onServerConnectStarted or onServerConnectFailure depending on its success or failure.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BAD_HOST_OR_PORT	Error: Server name or port is invalid or unreachable
E_CSI_ALREADY_CONNECTED	Error: Already connected to the LoggerNet server

Datalogger.serverDisconnect()

Name

```
Datalogger.serverDisconnect()
```

Description

This method will disconnect from the LoggerNet server. This method will set serverConnected to FALSE and should only be called when the value of serverConnected is TRUE.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

17.1.3 Events

Datalogger_onClockComplete()

Name

```
onClockComplete(Boolean successful,
clock_outcome_type response_code, Date current_date)
```

Parameters

successful: Describes whether a clock set or clock check succeeded.

response_code: The following list describes the possible response codes from a clock transaction:

Table of Response Code Values

Enumeration Name	Value	Description
co_outcome_unknown	0	Indicates that an error has occurred but its nature is unknown
co_outcome_success_clock_checked	1	Indicates that the clock was successfully checked on the specified datalogger (see loggerName)
co_outcome_success_clock_set	2	Indicates that the clock was successfully set on the specified datalogger (see loggerName)
co_outcome_session_failed	3	Indicates that the communication session with the LoggerNet server failed resulting in the clock check/set transaction failing
co_outcome_invalid_logon	4	Indicates that this control was unable to logon to the LoggerNet server because either the serverLogonName or serverLogonPassword property is incorrect
co_outcome_server_security_blocked	5	Indicates that the account specified by serverLogonName does not have sufficient privileges assigned to start the transaction with the LoggerNet server
co_outcome_communication_failed	6	Indicates that there was a communication failure between the LoggerNet server and the datalogger. If this happens, retry the transaction.
co_outcome_communication_disabled	7	Indicates that LoggerNet has not been set up to communicate with this datalogger. You will need to enable communications before you will be able to successfully communicate with the datalogger.

Enumeration Name	Value	Description
co_outcome_logger_security_blocked	8	Indicates that security has been enabled on the LoggerNet server and that the account specified by serverLogonName does not have sufficient privileges to communicate with the datalogger
co_outcome_invalid_device_name	9	Indicates that the device named by loggerName was not found in the broker map
co_outcome_unsupported	10	Indicates that the device loggerName does not support this transaction
co_outcome_cancelled	11	Indicates that a previous clock check or set command was cancelled successfully
co_outcome_device_busy	12	Indicates the datalogger is busy with another transaction

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return

Datalogger_onLoggerConnectFailure()

Name

```
onLoggerConnectFailure(logger_failure_type fail_code)
```

Description

This event indicates there was an error when making an persistent connection with the specified datalogger.

Parameter

Table of Fail Code Values

Enumeration Name	Value	Description
lf_failure_unknown	0	Indicates that an error has occurred but its nature is unknown
lf_failure_unexpected	1	Indicates than an unexpected error has occurred

Enumeration Name	Value	Description
lf_failure_connection_failed	2	Indicates that the connection failed. This can happen if a connection has been successfully established but then lost or an invalid serverName or serverHostPort property value was specified. This type of failure can also occur if the IP stack on the server host or on the host for this application is not configured correctly.
lf_failure_invalid_logon	3	Indicates that this control was unable to logon to the LoggerNet server because either the serverLogonName or serverLogonPassword property is incorrect
lf_failure_server_security_blocked	4	Indicates that security has been enabled on the server and that the serverLogonName does not have sufficient privileges or serverLogonPassword is incorrect
lf_failure_device_name_invalid	5	Indicates that the device loggerName was not found in the network map
lf_failure_server_terminated_transaction	6	Indicates that the server has terminated the transaction
lf_failure_device_does_not_support	7	Indicates that the device loggerName does not support this transaction
lf_failure_path_does_not_support	8	This transaction is not supported for this network path. The name of the blocking device will be supplied as the next parameter.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return

Datalogger_onLoggerConnectStarted()

Name

```
onLoggerConnectStarted()
```

Description

This event gets called when a connection to the datalogger has been established and is a result of invoking the method *loggerConnectStart()*.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger_onManualPollComplete()

Name

```
onManualPollComplete(Boolean successful,
manual_poll_outcome_type response_code)
```

Description

A response from the LoggerNet server upon the completion of a manual poll.

Parameters

successful: Describes whether the manual poll was successful.

response_code: The following list describes the possible response codes from a manual poll transaction.

Table of Response Code Values

Enumeration Name	Value	Description
mp_outcome_unknown	0	Indicates that an error has occurred but its nature is unknown
mp_outcome_success	1	Indicates that the manual poll was successful on the specified datalogger
mp_outcome_invalid_logon	2	Indicates that this control was unable to logon to the LoggerNet server because either the serverLogonName or serverLogonPassword property is incorrect
mp_outcome_server_session_failed	3	Indicates that the communication session with the server failed resulting in the manual poll transaction failing
mp_outcome_invalid_device_name	4	Indicates that the datalogger device loggerName was not found in the broker map
mp_outcome_unsupported	5	Indicates that the device does not support the manual poll transaction
mp_outcome_server_security_blocked	6	Indicates that the account specified by serverLogonName does not have sufficient privileges assigned to start the transaction with the LoggerNet server
mp_outcome_logger_security_blocked	7	Indicates that security is set on the datalogger blocking this transaction

Enumeration Name	Value	Description
mp_outcome_comm_failure	8	Indicates that there was a communication failure between the LoggerNet server and the datalogger. If this happens, retry the transaction.
mp_outcome_communication_disabled	9	Indicates that LoggerNet has been set up not to communicate with this datalogger. Enable communications before attempting communication with the datalogger.
mp_outcome_table_defs_invalid	10	Indicates that the table definitions in the LoggerNet server do not match those in the datalogger
mp_outcome_aborted	11	Indicates that a previous manual poll command was cancelled successfully
mp_outcome_logger_locked	12	Indicates that the datalogger is locked
mp_outcome_file_io_failed	13	Indicates that the LoggerNet server could not write to the data cache
mp_outcome_no_table_defs	14	Indicates that table definitions have not been downloaded by the LoggerNet server

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger_onProgramCompiled()

Name

`onProgramCompiled()`

Description

This event returns notification when the program has compiled successfully on the datalogger and table definitions are being retrieved.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return.

Datalogger_onProgramReceiveComplete()

Name

```
onProgramReceiveComplete(Boolean successful,
prog_receive_outcome_type response_code)
```

Description

This event gets called when the method *programReceiveStart()* has completed.

Parameters

successful: Describes if the program was retrieved successfully.

response_code: The following table describes the possible response codes.

Table of Possible Response Codes

Enumeration Name	Value	Description
pr_success	0	Indicates that the program was received successfully
pr_failure_unknown	1	Indicates that an unknown failure has occurred
pr_failure_no_cached_file	2	Indicates that the datalogger does not have a file to receive
pr_failure_logger_communication_error	3	Indicates that the connection failed. This can happen if a connection has been successfully established but then lost or because an invalid serverName or serverPort property value was specified. This type of failure can also occur if the IP stack on the server host or on the host for this application is not configured correctly.
pr_failure_disabled_communication	4	Indicates that LoggerNet has not been set up to communicate with this datalogger
pr_failure_logger_security	5	Indicates that the LoggerNet server cannot communicate with the datalogger because the datalogger security code is incorrect
pr_failure_invalid_server_logon	6	Indicates that the serverLogonName or the serverLogonPassword is incorrect
pr_failure_server_connection_failure	7	Indicates that the control could not connect to the server
pr_failure_invalid_device_name	8	Indicates that the device set in the property loggerName could not be found in the network map
pr_failure_cannot_open_file	9	Indicates that the file could not be opened for writing. You may not have permissions to write in that directory or the file may be in use.

Enumeration Name	Value	Description
pr_failure_server_security	10	Indicates that security has been enabled on the LoggerNet server and that you do not have sufficient privileges to connect
pr_failure_not_supported	11	Indicates that this transaction is not supported
pr_aborted_by_client	12	Indicates that this transaction was cancelled

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger_onProgramReceiveProgress()

Name

```
onProgramReceiveProgress(Long Received_bytes)
```

Description

This event periodically returns notification of how many bytes have been received from the datalogger during the retrieval of a program. This event gets called after the *programReceiveStart()* method has been called.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger_onProgramSendComplete()

Name

```
onProgramSendComplete(Boolean successful,
prog_send_outcome_type response_code, String
compile_result)
```

Description

This event gets called when the program sending process has finished.

Parameters

successful: Describes if the programSendStart was successful.

response_code: Found in the table of possible response codes.

compile_result: Result string from the datalogger.

Table of Possible Response Codes

Enumeration Name	Value	Description
ps_outcome_unknown	0	Indicates that an error has occurred but its nature is unknown
ps_outcome_success	1	Indicates that the program was sent successfully
ps_outcome_in_progress	2	Indicates that another program file send transaction is already in progress
ps_outcome_invalid_program_name	3	Indicates that the program specified to send is invalid or non-existent
ps_outcome_server_resource_error	4	Indicates that the LoggerNet server has encountered a resource error
ps_outcome_communication_failed	5	Indicates that the connection failed. This can happen if a connection has been successfully established but then lost or because an invalid serverName or serverPort property value was specified. This type of failure can also occur if the IP stack on the server host or on the host for this application is not configured correctly.
ps_outcome_communication_disabled	6	Indicates that LoggerNet has not been set up to communicate with this datalogger
ps_outcome_logger_compile_error	7	Indicates that the datalogger was unable to compile the program. The program should be reviewed for errors and resent to the datalogger.
ps_outcome_logger_security_failed	8	Indicates that the LoggerNet server cannot communicate with the datalogger because the datalogger security code is incorrect
ps_outcome_invalid_logon	9	Indicates that the property serverLogonName or serverLogonPassword is invalid
ps_outcome_session_failed	10	Indicates that the communication session with the server failed causing the program send transaction to fail
ps_outcome_invalid_device_name	11	Indicates that the device named by loggerName was not found in the network map
ps_outcome_cannot_open_file	12	Indicates that the program to send could not be opened to read
ps_outcome_server_security_failed	13	Indicates that the LoggerNet server has security enabled and that the serverLogonName or serverLogonPassword is incorrect
ps_outcome_logger_buffer_full	14	Indicates that the datalogger's storage buffer is full

Enumeration Name	Value	Description
ps_outcome_network_locked	15	Indicates that the network is locked by another transaction
ps_outcome_aborted_by_client	16	Indicates that this transaction has been cancelled
ps_outcome_table_defs_failed	17	Indicates that the table definitions were not obtained from the datalogger

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger_onProgramSendProgress()

Name

```
onProgramSendProgress(Long sent_bytes, Long
total_bytes)
```

Description

This event periodically returns notification of how many `sent_bytes` out of a program's `total_bytes` have been sent to the datalogger. This event could be helpful in a progress bar and gets called periodically after invoking the *programSendStart()* method.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger_onProgramSent()

Name

```
onProgramSent()
```

Description

This event returns notification when the program has been sent but gets called before the program has been compiled on the datalogger and table definitions have been retrieved.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger_onSelectiveManualPollComplete()

Name

```
onSelectiveManualPollComplete(Boolean successful,
selective_manual_poll_outcome_type response_code)
```

Description

The response from the LoggerNet server when the selective manual poll completes.

Parameters

successful: Describes if the polling was successfully.

response_code: The following table describes the possible response codes.

Table of Response Code Values

Enumeration Name	Value	Description
smp_outcome_unknown	0	Indicates that an unknown error as occurred
smp_outcome_success	1	Indicates that the selective manual poll was successful
smp_outcome_invalid_logon	2	Indicates that this control was unable to logon to the LoggerNet server because either the serverLogonName or serverLogonPassword property is incorrect
smp_outcome_server_session_failed	3	Indicates that the communication session with the server failed causing the selective manual poll transaction to fail
smp_outcome_invalid_device_name	4	Indicates that the datalogger device loggerName was not found in the broker map
smp_outcome_unsupported	5	Indicates that the device does not support the selective manual poll process
smp_outcome_server_security_blocked	6	Indicates that the account specified by serverLogonName does not have sufficient privileges assigned to start the transaction with the LoggerNet server
smp_outcome_logger_security_blocked	7	Indicates that security is set on the datalogger blocking this transaction

Enumeration Name	Value	Description
smp_outcome_comm_failure	8	Indicates that there was a communication failure between the LoggerNet server and the datalogger
smp_outcome_communication_disabled	9	Indicates that communication to this datalogger has been disabled in the LoggerNet server
smp_outcome_table_defs_invalid	10	Indicates that the table definitions in the LoggerNet server do not match those in the datalogger
smp_outcome_table_name_invalid	11	Indicates that the table specified was not found
smp_outcome_file_io_failure	12	Indicates that the LoggerNet server could not write to the data cache table
smp_outcome_logger_busy	13	Indicates that the datalogger is busy with another transaction
smp_outcome_aborted	14	Indicates that the selective manual poll was successfully cancelled

Datalogger_onServerConnectFailure()

Name

```
onServerConnectFailure(server_failure_type
failure_code)
```

Description

This event gets called if a connection cannot be established with the LoggerNet server using the *serverConnect()* method.

Parameters

failure_code: The following are possible values for failure_code.

Table of Possible Failure Codes

Enumeration Name	Value	Description
server_failure_unknown	0	Indicates that an unknown failure has occurred
server_failure_logon	1	Indicates that there was a failure connecting to the LoggerNet server because either serverLogonName or serverLogonPassword is incorrect
server_failure_session	2	Indicates that the communication session with the server failed resulting in the serverConnect transaction failing
server_failure_unsupported	3	Indicates that the datalogger defined in the property loggerName could not support this transaction

Enumeration Name	Value	Description
server_failure_security	4	Indicates that the server has security enabled and that the serverLogonName or the serverLogonPassword properties did not have sufficient privileges to perform this method
server_failure_bad_host_or_port	5	Indicates that either the serverName or the serverPort property is incorrect

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Datalogger_onServerConnectStarted()

Name

```
onServerConnectStarted()
```

Description

This event gets called once a connection has been established with the LoggerNet server using the *serverConnect()* method.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Section 18. CsiDataSource Control Reference

18.1 *DSource* Interface

18.1.1 Properties

DSource.logonName

Name

```
DSource.logonName As String
```

Description

Specifies the account name that should be used when connecting to the LoggerNet server. If security is enabled on the target LoggerNet server, this string must be one of the account names recognized by the LoggerNet server.

Default Value

The default value for this property is an empty string. This property is only used if security is enabled on the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal Return
E_CSI_BUSY	Error: Attempt to set serverLogonPassword while connected to the LoggerNet server

DSource.logonPassword

Name

```
DSource.logonPassword As String
```

Description

This property specifies the password that should be used when connecting to the LoggerNet server. If security is enabled on the target LoggerNet server, this password string must be associated with the account described in the logonName property.

Default Value

The default value for this property is an empty string. This property is only used if security is enabled on the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Attempt to set serverLogonPassword while connected to the LoggerNet server

DSource.serverName

Name

```
DSource.serverName As String
```

Description

This property specifies the TCP/IP interface address for the computer hosting the LoggerNet server. This string must be formatted either as a fully qualified Internet machine domain name or as an IP address string. An example of a valid machine domain name address is *www.campbellsci.com*. An example of a valid IP address string is `207.201.118.35`.

Default Value

The default value for this property is the string, `localhost`.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Attempt to set serverName while connected to the LoggerNet server

DSource.serverPort

Name

```
DSource.serverPort As Long
```

Description

This property specifies the TCP port number that the LoggerNet server is using on the hosting computer. The valid range for this property is 1 to 65535.

Default Value

The default value for this property, assigned to the LoggerNet server during install, is 6789. In most cases, the default value for this property is acceptable.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Cannot write to this property because there is a connection to the LoggerNet server
E_CSI_INVALIDARG	Error: Value out of range

DSource.state

Name

```
DSource.state As data_source_state
```

Description

This property describes the state of the control in regards to a connection with the LoggerNet server. The following are the possible values of this property:

Table of Possible Values

Enumeration Name	Value	Description
dataSourceDisconnected	1	The control is currently disconnected and its read/write properties are accessible
dataSourceConnecting	2	The connect method has been invoked and the control is attempting to connect to the LoggerNet server. Properties are read-only at this time.
dataSourceConnected	3	The connect method has been successfully invoked and the control has a connection to the server. It is appropriate at this time to create advisors and start them.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

DSource.sendRecordBlocks

Name

```
DSource.sendRecordBlocks As Boolean
```

Description

When set to TRUE, records will be sent back from *LoggerNet* to an advisor in blocks rather than one at a time. This is a more efficient method of receiving records if a large number of records are being collected.

Default Value

This property is set to FALSE by default.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

18.1.2 Methods

DSource.connect()

Name

```
DSource.connect()
```

Description

This method allows you to connect to the LoggerNet server. When you invoke this method, the control will attempt to connect to the specified LoggerNet server. If it succeeds, you will receive the event onControlReady. If you are already connected, you will receive the COM error E_CSI_ALREADY_CONNECTED. If the serverName and/or serverPort properties cannot be resolved or are incorrect, you will receive the error code E_CSI_BAD_HOST_OR_PORT.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_ALREADY_CONNECTED	Error: Already connected to the server
E_CSI_BAD_HOST_OR_PORT	Error: Server hostname or port is incorrect

DSource.createAdvisor()

Name

```
DSource.createAdvisor()As Object
```

Description

This method creates a new advisor object. Keep a reference to the advisor so it will not go out of scope. If you create and start an advisor but don't get any data, you are probably letting the advisor go out of scope. When handling multiple advisors, use a collection or list. The property advisorName is provided for convenience when using a collection to hold names of the advisors you create.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_NOT_CONNECTED	Error: The control is not connected to the LoggerNet server and therefore cannot create any advisors. Connect to the LoggerNet server first
E_FAIL	Error: An unexpected error has occurred

Visual Basic®

Example

```
Dim myAdvisor As new advisor
Set myAdvisor = DSource.createAdvisor
```

DSource.disconnect()

Name

```
DSource.disconnect()
```

Description

This method attempts to disconnect from the current LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

18.1.3 Events

DSource_onAdviseReady()

Name

```
onAdviseReady(Object myAdvisor)
```

Description

This event returns notification that an advisor has been started and will send *onAdviseRecord()* events when records are collected by the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

DSource_onAdviseRecord()

Name

```
onAdviseRecord(Object myAdvisor, Object myRecord)
```

Description

This event returns notification of newly acquired data from an advisor. If records are not being acquired, the advisor will not display them. Please make sure the tables specified in the advisor are enabled for collection through the use of *CoraScript* commands (set-collect-area-setting setting ID 2). Once the tables are enabled for collection, use the datalogger control to manually collect records or use the *CoraScript* control to enable scheduled collection.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

DSource_onAdvisorFailure()

Name

```
onAdvisorFailure(csiAdvisorFailureCode failure,
Advisor myAdvisor)
```

Description

Indicates there was a failure with the advisor specified in myAdvisor.

Parameters

Table of Possible Response Codes

Enumeration Name	Value	Description
csiAdvisorFailureUnknown	0	Indicates that an error has occurred but its nature is unknown
csiAdvisorFailureConnectionFailed	1	Indicates that the connection failed. This can happen if a connection has been successfully established but then lost or because an invalid serverName or serverPort property value was specified. This type of failure can also occur if the IP stack on the server host or on the host for this application is not configured correctly.
csiAdvisorFailureInvalidLogon	2	Indicates that this control was unable to logon to the LoggerNet server because either the logonName or logonPassword property is incorrect

Enumeration Name	Value	Description
csiAdvisorFailureInvalidStationName	3	Indicates that the datalogger device named by stationName is not found in the server's network map at the time the advisor is started. Changes made to the station name after the advisor is started are triggered with code value 9 – csiAdvisorFailureStationShutDown (see below).
csiAdvisorFailureInvalidTableName	4	Indicates that the table specified by tableName does not exist for the specified station at the time the advisor is started. A table name change that occurs after the advisor is activated will trigger code value 8 – csiAdvisorFailureTableDeleted (see below).
csiAdvisorFailureServerSecurity	5	Indicates that the account specified by logonName does not have sufficient privileges assigned to start the data advise transaction with the LoggerNet server
csiAdvisorFailureInvalidStartOption	6	Indicates that the startOption is either invalid or not supported by the LoggerNet server
csiAdvisorFailureInvalidOrderOption	7	Indicates that the orderOption is either invalid or not supported by the LoggerNet server
csiAdvisorFailureTableDeleted	8	Indicates that the table has been deleted (or renamed) while the data advise transaction is in progress. This can happen if table definitions are refreshed on the device or if a new program file is sent to the datalogger.
csiAdvisorFailureStationShutDown	9	Indicates that the station that owns the table has been shut down while the data advise transaction is in progress. This can happen if the device is deleted, renamed, or if the LoggerNet server is shut down.
csiAdvisorFailureUnsupported	10	The version of the LoggerNet server doesn't support this transaction
csiAdvisorFailureInvalidColumnName	11	Indicates that the column name is invalid
csiAdvisorFailureInvalidArrayAddress	12	Indicates that the array address is invalid

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

DSource_onControlFailure()

Name

```
onControlFailure(csidsFailureCode failure_code)
```

Description

This event is triggered when an error has occurred that affects the control as a whole.

Table of Possible Failure Codes

Enumeration Name	Value	Description
csidsFailureUnknown	0	Indicates that an error has occurred but its nature is unknown
csidsFailureLogon	1	Indicates that this control was unable to logon to the LoggerNet server because either the logonName or logonPassword property is incorrect
csidsFailureSession	2	Indicates that the communication session with the server failed resulting in failed transactions
csidsFailureUnsupported	3	The version of the LoggerNet server doesn't support this transaction
csidsFailureSecurity	4	Indicates that the account specified by logonName does not have sufficient privileges to start the transaction with the LoggerNet server

NOTE Other codes besides those shown above are included in the enumeration of the DataSource control's interface, but they are never triggered.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

DSource_onControlReady()

Name

```
onControlReady()
```

Description

This event is triggered when a connection to the server has been established and is a result of invoking the *connect()* method. Once this event has been called, advisors can be created and started.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

DSource_onVariableSetComplete()

Name

```
onVariableSetComplete(Long tran_id, Object myAdvisor,
Boolean successful, variable_outcome_code
response_code)
```

Description

This event gets called when the method *variableSetStart()* has completed.

Parameters

tran_id: The transaction ID used to track this event.

myAdvisor: References the advisor that started the variable set transaction.

successful: Indicates whether the transaction succeeded.

response_code: Values from the following table.

Table of Possible Response Code Outcomes

Enumeration Name	Value	Description
vo_outcome_unknown	0	Indicates that the outcome could not be determined
vo_outcome_succeeded	1	Indicates that the setting of the variable was set successfully
vo_outcome_connection_failed	2	Indicates that the control could not connect to the LoggerNet server
vo_outcome_invalid_logon	3	Indicates that the logonName or logonPassword was incorrect
vo_outcome_server_security_blocked	4	Indicates that security has been enabled on the LoggerNet server and that you do not have sufficient privileges to connect
vo_outcome_column_read_only	5	Indicates that the column sent is read-only
vo_outcome_invalid_table_name	6	Indicates that the table name was not found on the datalogger
vo_outcome_invalid_column_name	7	Indicates that the column name was not found on the datalogger

Enumeration Name	Value	Description
vo_outcome_invalid_subscript	8	Indicates that the index of the variable was invalid. For array values, subscripts start at "1".
vo_outcome_invalid_data_type	9	Indicates that the type of the data sent for this variable does not match the variable type
vo_outcome_communication_failed	10	Indicates that communication has failed during this transaction
vo_outcome_communication_disabled	11	Indicates that *LoggerNet* has not been set up to communicate with this datalogger
vo_outcome_logger_security_blocked	12	Indicates that the datalogger's security has been enabled and you do not have sufficient privileges to set a variable
vo_outcome_unmatched_logger_table_definition	13	Indicates that the LoggerNet server's table definitions are not the same as the datalogger's table definitions
vo_outcome_invalid_device_name	14	Indicates that the device named by stationName could not be found in the network map
vo_outcome_aborted_by_user	15	Indicates that a VariableSetCancel command successfully prevented the variable change from occurring

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

DSource_onAdviseRecords()

Name

```
onAdviseRecords(Object myAdvisor, Object
record_collection)
```

Description

This event notification returns a block of records delivered by *LoggerNet* to an active advisor. The sendRecordBlocks property must be set to TRUE and the table specified in the advisor must be enabled for collection for this event to work.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

18.2 *Advisor* Interface

18.2.1 Properties

Advisor.advisorName

Name

```
Advisor.advisorName As String
```

Description

A user-defined field used to distinguish between advisors.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Advisor.orderOption

Name

```
Advisor.orderOption As csidsOrderOptionType
```

Description

This property specifies the order in which the LoggerNet server will send records to the advisor. This property must use one of the following values:

Table of Possible csidsOrderOptionType Values

Enumeration Name	Value	Description
csidsOrderCollected	1	The records will be sent in the same order that the LoggerNet server collects them. This option can send the records out of sequence particularly with Campbell Scientific table-data dataloggers but all collected records will be sent.

Enumeration Name	Value	Description
csidsOrderLoggedWithHoles	2	The records will be sent in the order they were logged in the datalogger. This order is determined by the record number (which is assigned by the datalogger) and the file mark number (which is assigned by the server) to create a unique key for each record. If a record has not yet been collected but the LoggerNet server judges (by datalogger table size) that the record can still be collected, no record will be sent until the missing record (hole) has either been collected or the LoggerNet server decides that the record can no longer be collected.
csidsOrderLoggedWithoutHoles	3	The records will be sent in the order that they were logged by the datalogger. This option is similar to the csidsOrderLoggedWithHoles only uncollected records (holes) will be skipped.
csidsOrderRealTime	4	The records will be sent in the order they were logged in the datalogger but if more than one record is collected at a time, all other records except for the most recent of the collection will be ignored.

Default Value

The default value for this property is csidsOrderRealTime (4).

Notes

This property can be read at any time but can only be set when the state of the property is advisorStopped.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_BUSY`	Error: The advisor is started and already accessing the LoggerNet server data

Advisor.startDate

Name

```
Advisor.startDate As Date
```

Description

This property specifies the timestamp for the earliest record to be selected when the value of the startOption property is csidsStartAtTimeStamp.

Notes

This property can be read at any time but can only be set when the state of the property is advisorStopped.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The advisor is started and already accessing the LoggerNet server data

Advisor.startFileMarkNo

Name

```
Advisor.startFileMarkNo As Long
```

Description

In conjunction with startRecordNo, this property specifies the first record to be sent when the value of startOption is equal to csidsStartAtRecordId. The file mark number is an internal tag used by *LoggerNet* that is applied to each record. The file mark number is assigned to each record by the LoggerNet server and used in combination with the record ID to create a unique key for each record. If the value of this property is specified as 0xffffffff (–1 if treated as a signed number), the server will start in the current file mark and ignore any previous file marks.

Valid Values

Any integer from 0 to 2147483647 inclusive is a valid value.

Default Value

The default value for this property is 0.

Notes

This property can be read at any time but can only be set when the state of the property is advisorStopped.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Cannot write to property. Advisor is running. Disconnect first with stop.

Advisor.startIntervalSeconds

Name

```
Advisor.startIntervalSeconds As Long
```

Description

This property specifies the number of seconds back from the newest record in the table to collect when the value of startOption is set to csidsStartRelativeToNewest.

Valid Values

A valid value must either be zero or a positive integer.

Default Value

The default value for this property is 0 (meaning select the newest record).

Notes

This property can be read at any time but can only be set when the state of the property is advisorStopped.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The advisor is started and already accessing the LoggerNet server data

Advisor.startOption

Name

```
Advisor.startOption As csidsStartOptionType
```

Description

This property specifies how to select the first record when retrieving collected data from the LoggerNet server data cache.

Valid Values

This property must take on one of the following values:

Table of Possible csidsStartOptionType Values

Enumeration Name	Value	Description
csidsStartAtRecordId	1	The first record will be the record identified by startFileMarkNo and startRecordNo. If no such record exists in the table, the record that is closest and newer than the specified record will be selected.

Enumeration Name	Value	Description
csidsStartAtTimeStamp	2	The first record that has a timestamp equal to the timestamp specified by the startDate will be selected. If no such record exists in the table, the record that has the closest timestamp that is newer than the one specified will be selected.
csidsStartAtNewest	3	The newest record (determined by the combination of record number and file mark number) will be selected.
csidsStartAfterNewest	4	The next new record to be logged in the table will be the first record sent.
csidsStartRelativeToNewest	5	The first record selected will be the one that has a timestamp closest to the timestamp of the newest record less the value of startIntervalSeconds.
csidsStartAtRecordOffset	6	The first record selected will be a specified number of records back from the newest in the data cache.

Default Value

The default value for this property is csidsStartAtNewest (3).

Notes

This property can be read at any time but can only be set when the state of the property is advisorStopped.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_BUSY`	Error: The advisor is started and already accessing the LoggerNet server data

Advisor.startRecordNo

Name

```
Advisor.startRecordNo As Long
```

Description

This property, in conjunction with the property startFileMarkNo, specifies the first record to be sent when the value of startOption is equal to csidsStartAtRecordId. Any value can be assigned to this property.

Default Value

The default value for this property is 0.

Notes

This property can be read at any time but can only be set when the state of the property is advisorStopped. Internally the control and the LoggerNet server treat this property as an unsigned 32-bit integer. Visual Basic and other container environments, however, do not have the capability of formatting and properly manipulating unsigned integers. Developers in these environments should consider using the startRecordNoString property instead.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Cannot write to property. Advisor has already started. Stop the advisor first.

Advisor.startRecordNoString

Name

```
Advisor.startRecordNoString As String
```

Description

This property, in conjunction with startFileMarkNo, is used to specify the first record to be sent when the value of startOption is equal to csidsStartAtRecordId. This string should be formatted as an unsigned integer with a range of 0 to 4294967295.

Default Value

The default value for this property is 0.

Notes

This property can be read at any time but can only be set when the state of the property is advisorStopped.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The advisor is started and already accessing the LoggerNet server data

Advisor.state

Name

```
Advisor.state As advisor_state
```

Description

This property returns the current state of the advisor. The following table describes the states that might be returned:

Table of Possible Advisor State Values

Enumeration Name	Value	Description
advisorStopped	1	The advisor is stopped and its properties can be modified. This is the default state when an advisor is created.
advisorStarting	2	The control is starting but is not yet in a state to listen for data. No properties can be set at this point. The control is in a state where none of its properties can be set.
advisorStarted	3	The advisor is waiting for data from the server and will notify the client through onAdviseRecord when new data arrives.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Advisor.stationName

Name

```
Advisor.stationName As String
```

Description

This property describes the name of the station that will be monitored for data. Whenever this property is set, the DataColumns in the DataColumnCollection for this advisor are removed in order to avoid having invalid columns in the collection for a station and a table.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The advisor is started and already accessing the LoggerNet server data

Advisor.tableName

Name

```
Advisor.tableName As String
```

Description

This property describes the name of the table in the LoggerNet server being monitored by the advisor. Whenever this property is set, the DataColumns in the DataColumnCollection for this advisor are removed in order to avoid having invalid columns in the collection for a station and a table.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The advisor is started and already accessing the LoggerNet server data

Advisor.startDateNanoSeconds

Name

```
Advisor.startDateNanoSeconds As Long
```

Description

This property specifies the sub-second resolution to associate with the start date.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The advisor is started and already accessing the LoggerNet server data

Advisor.maxRecordsPerBlock

Name

```
Advisor.maxRecordsPerBlock As Long
```

Description

This property sets the maximum number of records that will be included in a block of records received from *LoggerNet* if the sendRecordBlocks property is set to TRUE. The default value is 100.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The advisor is started and already accessing the LoggerNet server data

18.2.2 Methods

Advisor.columns

Name

```
Advisor.columns() As Object
```

Description

This method returns a reference to the DataColumnCollection for this advisor, which can be used to iterate through the DataColumns.

Visual Basic

Return Value
DataColumnCollection

Example
```
Dim dcc As DataColumnCollection
dcc = myAdvisor.Columns
```

Advisor.start()

Name

```
Advisor.start()
```

Description

This method starts the advisor to monitor data for a specified station, table, and column. This is an asynchronous event that calls *onAdvisorRecord()*. If the advisor fails, the *onAdvisorFailure()* event will get called.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_ALREADY_CONNECTED	Error: This advisor has already been started
E_FAIL	Error: An unexpected error has occurred

Advisor.stop()

Name

```
Advisor.stop()
```

Description

This method will stop the advisor from monitoring the *LoggerNet* data cache for transactions. When an advisor is stopped, its properties can be modified.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Advisor.variableSetCancel()

Name

```
Advisor.variableSetCancel(Long tran_id)
```

Description

This method attempts to cancel a *variableSetStart()* transaction. The event *onVariableSetComplete()* will notify you if the cancellation was successful. This method should only be called when the state of advisorStarted is TRUE.

Parameter

tran_id: The unique transaction ID given by *variableSetStart()*.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Advisor.variableSetStart()

Name

```
Advisor.variableSetStart(String column_name, String
value) As Long
```

Description

This method sets a variable in the specified datalogger. The event *onVariableSetComplete()* will be called upon the completion of *variableSetStart()*. This method should only be called when the state dataSourceConnected is TRUE and an advisor has been started. If not, this method will return E_CSI_NOT_CONNECTED.

Parameters

columnName: The name of the column that is being changed. If this is an array value, then use the *CRBasic Editor* syntax for arrays. Parentheses are used with element subscripts separated by commas.

```
myArray(3) or,

myArray(2,4,1)
```

If the column is not an array value, the brackets for the index are not needed.

value: The value of the variable as a String.

Return Value

The transaction ID associated with this command can be used to cancel a specific variable set command with *variableSetCancel()* or to keep track of the variables displayed in a form that were set successfully.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_FAIL	Error: Unexpected error
E_CSI_NOT_CONNECTED	Error: Not connected to the LoggerNet server or no current advisor started

18.3 *DataColumnCollection* Interface

18.3.1 Properties

DataColumnCollection.count

Name

```
DataColumnCollection.count As Long
```

Description

This property returns the number of DataColumns in the collection.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

18.3.2 Methods

DataColumnCollection.add()

Name

```
DataColumnCollection.add(String column_name)
```

Description

This method adds a column name to the collection of DataColumns. By adding a column name to this collection, you tell the advisor to retrieve values in the record for that column. The column name added must be valid for the station and table specified in the advisor. If no column names are added to this collection, data records will only contain file mark numbers, record numbers, and timestamps.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_FAIL	Error: The column name is not a valid column for this station and table

DataColumnCollection.addAll()

Name

```
DataColumnCollection.addAll()
```

Description

This method adds all of the columns for the defined station and table to the DataColumnCollection. If any previous columns existed in the collection for this advisor, they will be cleared out before the new DataColumns are added.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

DataColumnCollection.find()

Name

```
DataColumnCollection.find(String column_name) As
Boolean
```

Description

This property returns whether the specified column exists in the DataColumnCollection.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

DataColumnCollection.Item()

Name

```
DataColumnCollection.Item(id) As DataColumn
```

Description

A DataColumn can be referenced by a numeric type such as an integer or a long. If the number is less than zero or is greater than the number of brokers –1, then the COM error E_CSI_ARRAY_OUT_OF_BOUNDS will be returned.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_INVALIDARG	Error: An incorrect variant type was passed. Expecting a numerical value
E_CSI_ARRAY_OUT_OF_BOUNDS	Error: The numerical index was out of the bounds of the array. Please specify a value from zero (0) to Count - 1
E_CSI_FAIL	Error: An unexpected error has occurred

DataColumnCollection.remove()

Name

```
DataColumnCollection.remove(String columnName)
```

Description

This method removes the specified column from the DataColumnCollection. If the column does not exist in the collection, an error will be returned.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_INVALIDARG	Error: Cannot remove. The column specified does not exist in the collection

DataColumnCollection.removeAll()

Name

```
DataColumnCollection.removeAll()
```

Description

This method removes all of the DataColumns that are presently a part of the DataColumnCollection. This method does not return an error if the collection is already empty.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

DataColumnCollection._NewEnum()

Name

```
DataColumnCollection._NewEnum()
```

Description

Returns the next data column in the sequence.

Important

This method is only intended for use with the Visual Basic programming language. Visual Basic programmers do not need to access this method directly. They use it indirectly by using the collections with the **For Each** loop. This method is included in the documentation to explain why the method exists, but, again, there is no need to access this method directly.

18.4 *DataColumn* Interface

18.4.1 Properties

DataColumn.name

Name

```
DataColumn.name As String
```

Description

This read-only property gives the name of the DataColumn added to the DataColumnCollection.

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

18.5 *Record* Interface

18.5.1 Properties

Record.fileMarkNo

Name

```
Record.fileMarkNo As Long
```

Description

This read-only property returns the file mark number associated with the current record. The file mark number is assigned to each record by the LoggerNet server and used in combination with the record ID to create a unique key for each record. This property can take on any value from 0 to 2147483647.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Record.nanoSeconds

Name

```
Record.nanoSeconds As Long
```

Description

This read-only property returns the sub-second resolutions of the timestamp associated with the current record. This property can take on any value from 0 to 2147483647.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Record.recordNo

Name

```
Record.recordNo As Long
```

Description

This read-only property returns the record number associated with the current record. This property can take on any value from 0 to 2147483647.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Record.timeStamp

Name

```
Record.timeStamp As Date
```

Description

This read-only property returns the timestamp associated with the current record.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Record.valuesCount

Name

```
Record.valuesCount As Long
```

Description

This read-only property returns the number of values in this record.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

18.5.2 Methods

Record.Item()

Name

```
Record.Item(id) As Value
```

Description

This method returns a reference to a value found by the specified ID. A broker can be referenced by an integer (a Long) or by the name of the broker (a String). If the number is less than zero or is greater than the number of brokers, the COM error E_CSI_ARRAY_OUT_OF_BOUNDS will be returned. If the

broker cannot be found by name, the COM error E_CSI_NOT_FOUND will be returned.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_ARRAY_OUT_OF_BOUNDS	Error: Array out of bounds
E_CSI_NOT_FOUND	Error: Couldn't find the broker by name in the broker map
E_CSI_FAIL	Error: Wrong variant type passed to this method or unexpected error

Visual Basic

Return Type
Value

Example
Number value (like an array):

```
Long iterator
For iterator = 0 to myRecord.Count - 1
 ... = myRecord(iterator).value
Next iterator
```

Referencing the Broker by name:

```
DIM valueName as String
valueName = "battTemp"
DIM value as long
value = myRecord("battTemp").value
       OR
value = myRecord(valueName).value
```

Record._NewEnum()

Name

```
Record._NewEnum()
```

Description

Returns the next value in the record.

Important

This method is only intended for use with the Visual Basic programming language. Visual Basic programmers do not need to access this method directly. They use it indirectly by using the collections with the **For Each** loop. This method is included in the documentation to explain why the method exists, but, again, there is no need to access this method directly.

Visual Basic

Example

```
Dim v As value
For Each v in myRecord
   ... = v.value
Next
```

18.6 *RecordCollection*

18.6.1 Properties

RecordCollection.Count

Name

```
RecordCollection.Count As Long
```

Description

The number of values in the collection.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal Return

18.6.2 Methods

RecordCollection.Item()

Name

```
RecordCollection.Item(Value id, Record ppIRecord)
```

Description

This method is used to iterate through the values by the specified index ID.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_ARRAY_OUT_OF_BOUNDS	Error: Array out of bounds
E_CSI_FAIL	Error: An unexpected error has occurred

RecordCollection._NewEnum()

Name

```
RecordCollection._NewEnum()
```

Description

Returns the next record.

Important

This method is not accessed directly. It is used indirectly with the use of a **For Each** loop.

18.7 Value Interface

18.7.1 Properties

Value.columnName

Name

```
Value.columnName As String
```

Description

This property returns the name of the column.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Value.value

Name

```
Value.value As Variant
```

Description

This property returns the actual data value.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

Section 19. CsiLogMonitor Control Reference

19.1 *LogMonitor* Interface

19.1.1 Properties

LogMonitor.commLogMonitorBusy

Name

```
LogMonitor.commLogMonitorBusy As Boolean
```

Description

This Boolean property describes the state of the LogMonitor control's monitoring of communication logs on the LoggerNet server. The property returns `TRUE` if the communication logs are being actively monitored. Otherwise, the property returns `FALSE`.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return

LogMonitor.commLogRecordsBack

Name

```
LogMonitor.commLogRecordsBack As Long
```

Description

The LoggerNet server maintains a communication log history buffer that can be accessed using this property. When the *commLogMonitorStart()* method is called, by default 100 historical log files will be retrieved from the LoggerNet server. If a different number of historical log entries are desired, set this property to the exact number of entries to initially retrieve from the LoggerNet server. This number must be one or greater.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_INVALIDARG`	Error: The number must be one or greater
`E_CSI_BUSY`	Error: Attempting to set this property while the logs are being actively monitored

LogMonitor.serverConnected

Name

```
LogMonitor.serverConnected As Boolean
```

Description

This Boolean property describes the state of the connection between the LogMonitor control and the LoggerNet server. The property returns `TRUE` if the connection exists. Otherwise, the property returns `FALSE`.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return

LogMonitor.serverLogonName

Name

```
LogMonitor.serverLogonName As String
```

Description

This property specifies the account name that should be used when connecting to the LoggerNet server. If security is enabled on the target LoggerNet server, this string must be one of the account names recognized by the LoggerNet server.

Default Value

The default value for this property is an empty string. This property will only affect the operation of the control if security is enabled on the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_BUSY`	Error: The property cannot be set while a connection to the LoggerNet server is present

LogMonitor.serverLogonPassword

Name

```
LogMonitor.serverLogonPassword As String
```

Description

This property specifies the password that should be used when connecting to the LoggerNet server. If security is enabled on the target LoggerNet server,

this string must be the password associated with the account named by LogMonitor.serverLogonName.

Default Value

The default value for this property is an empty string. This property will only affect the operation of the control if security is enabled on the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: The property cannot be set while a connection to the LoggerNet server is present

LogMonitor.serverName

Name

```
LogMonitor.serverName As String
```

Description

This property specifies the TCP/IP interface address for the computer hosting the LoggerNet server. This string must be formatted either as a qualified Internet machine domain name or as an Internet address string. An example of a valid machine domain name address is *www.campbellsci.com*. An example of a valid Internet address string is `63.255.173.183`.

Default Value

The default value for this property is the string `localhost`.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_BUSY	Error: Attempt to set serverName while connected to the LoggerNet server

LogMonitor.serverPort

Name

```
LogMonitor.serverPort As Long
```

Description

This property specifies the TCP port number that the LoggerNet server is using on the hosting computer. The valid range for this property is port 1 to port 65535.

Default Value

The default value for this property is port 6789, which is the default port number assigned for the LoggerNet server. Therefore, the default value for this property will connect to a LoggerNet server port in most cases.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_INVALIDARG	Error: The port value is out of range or invalid
E_CSI_BUSY	Error: Attempt to set serverPort while connected to the LoggerNet server

LogMonitor.tranLogMonitorBusy

Name

```
LogMonitor.tranLogMonitorBusy As Boolean
```

Description

This Boolean property describes the state of the LogMonitor control accessing transaction logs on the LoggerNet server. The property returns TRUE if the communication logs are being accessed. Otherwise, the property returns FALSE.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

LogMonitor.tranLogRecordsBack

Name

```
LogMonitor.tranLogRecordsBack As Long
```

Description

The LoggerNet server maintains a transaction log history buffer that can be accessed using this property. When the *tranLogMonitorStart()* method is called, by default 100 historical log files will be retrieved from the LoggerNet server. If a different number of historical log entries are desired, set this property to the exact number of entries to initially retrieve from the LoggerNet server. This number must be one or greater.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_INVALIDARG	Error: The number must be one or greater
E_CSI_BUSY	Error: Attempting to set this property while the logs are being actively monitored

19.1.2 Methods

LogMonitor.commLogMonitorStart()

Name

```
LogMonitor.commLogMonitorStart()
```

Description

This method starts monitoring the communication log entries on the LoggerNet server. This method triggers *onCommLogRecord()* as log entries are retrieved or *onCommLogFailure()* if the method fails.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_FAIL	Error: Unexpected error
E_CSI_NOT_CONNECTED	Error: Not Connected to the LoggerNet server
E_CSI_BUSY	Error: Log monitoring is already active

LogMonitor.commLogMonitorStop()

Name

```
LogMonitor.commLogMonitorStop()
```

Description

This method will stop active monitoring of the communication logs on the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

LogMonitor.serverConnect()

Name

```
LogMonitor.serverConnect()
```

Description

This method attempts to connect to the LoggerNet server using the values in the previously set properties: serverName, serverPort, serverLogonName, and serverLogonPassword. This method triggers *onServerConnectStarted()* if the connection is successful, or *onServerConnectFailure()* if the connection fails.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return
`E_CSI_FAIL`	Error: Unexpected error

LogMonitor.serverDisconnect()

Name

```
LogMonitor.serverDisconnect()
```

Description

This method will disconnect from the LoggerNet server and will set the `serverConnected` state to FALSE.

COM Return Values

Table of Possible Values

Code	Meaning
`S_OK`	Success: Normal return

LogMonitor.tranLogMonitorStart()

Name

```
LogMonitor.tranLogMonitorStart()
```

Description

This method starts monitoring the transaction log entries on the LoggerNet server. This method triggers *onTranLogRecord()* as log entries are retrieved or *onTranLogFailure()* if the method fails.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return
E_CSI_FAIL	Error: Unexpected error

LogMonitor.tranLogMonitorStop()

Name

```
LogMonitor.tranLogMonitorStop()
```

Description

This method will stop active monitoring of the transaction logs on the LoggerNet server.

COM Return Values

Table of Possible Values

Code	Meaning
S_OK	Success: Normal return

19.1.3 Events

LogMonitor_onCommLogFailure()

Name

```
onCommLogFailure(log_monitor_failure_type
failure_code)
```

Description

This event indicates an error has occurred while trying to retrieve communication log entries from the LoggerNet server. This failure codes are in the following table:

Table of Possible Failure Codes

Enumeration Name	Value	Description
lm_failure_unknown	0	Indicates that an error has occurred but its nature is unknown
lm_failure_session_failure	1	Indicates that communication with the LoggerNet server failed resulting in a failed session
lm_failure_invalid_logon	2	Indicates that this control was unable to logon to the LoggerNet server because either the logonName or logonPassword property is incorrect

Enumeration Name	Value	Description
lm_failure_server_security_blocked	3	Indicates that the account specified by logonName does not have sufficient privileges to start this transaction with the LoggerNet server
lm_failure_unsupported_transaction	4	This version of the LoggerNet server does not support this transaction
lm_invalid_log_id	5	The log ID is not valid. Note: this ID is only used internally by the LogMonitor control
lm_failure_server_cancelled	6	The LoggerNet server is shutting down the connection

LogMonitor_onCommLogRecord()

Name

```
onCommLogRecord(Date timestamp, String
comm_log_record)
```

Description

When actively monitoring the communication log, this event is triggered when a new log record is passed from the LoggerNet server. The communication log entry is a string that contains the station name, message type, and message. Possible message types include “S” for status, “W” for warning, and “F” for failure.

LogMonitor_onServerConnectFailure()

Name

```
onServerConnectFailure(server_failure_type
failure_code)
```

Description

This event indicates there was an error with the connection to the LoggerNet server. This event triggers when an error has occurred that affects the control as a whole.

Table of Possible Failure Codes

Enumeration Name	Value	Description
server_failure_unknown	0	Indicates that an error has occurred but its nature is unknown
server_failure_logon	1	Indicates that this control was unable to logon to the LoggerNet server because either the logonName or logonPassword property is incorrect

Enumeration Name	Value	Description
server_failure_session	2	Indicates that the communication session with the LoggerNet server failed resulting in a failed transaction
server_failure_unsupported	3	The version of the LoggerNet server does not support this transaction
server_failure_security	4	Indicates that the account specified by logonName does not have sufficient privileges to start this transaction with the LoggerNet server
server_failure_bad_host_or_port	5	Indicates that either the serverName or the serverPort property is incorrect

LogMonitor_onServerConnectStarted()

Name

```
onServerConnectStarted()
```

Description

This event triggers when the LogMonitor control has connected to the LoggerNet server.

LogMonitor_onTranLogFailure()

Name

```
onTranLogFailure(log_monitor_failure_type
failure_code)
```

Description

This event indicates an error has occurred while trying to retrieve transaction log entries from the LoggerNet server. This event triggers when an error has occurred that affects the method that monitors the transaction logs on the LoggerNet server.

Table of Possible Failure Codes

Enumeration Name	Value	Description
lm_failure_unknown	0	Indicates that an error has occurred but its nature is unknown
lm_failure_session_failure	1	Indicates that communication with the LoggerNet server failed resulting in a failed session

Enumeration Name	Value	Description
lm_failure_invalid_logon	2	Indicates that this control was unable to logon to the LoggerNet server because either the logonName or logonPassword property is incorrect
lm_failure_server_security_blocked	3	Indicates that the account specified by logonName does not have sufficient privileges to start this transaction with the LoggerNet server
lm_failure_unsupported_transaction	4	This version of the LoggerNet server does not support this transaction
lm_invalid_log_id	5	The log ID is not valid. Note: this ID is only used internally by the LogMonitor control
lm_failure_server_cancelled	6	The LoggerNet server is shutting down the connection

LogMonitor_onTranLogRecord()

Name

```
onTranLogRecord(Date timestamp, String
tran_log_record)
```

Description

When actively monitoring the transaction log, this event is triggered when a new log record is passed from the LoggerNet server. The transaction log entry is a string that contains the station name, message number, and message.

Appendix A. Server and Device Operational Statistics Tables

The LoggerNet server and devices in the network map maintain statistics that help to describe their operation. These statistics are made available to the clients in a collection of tables associated with a special data broker of type "__Statistics__". The LoggerNet server guarantees that there is only one data broker of this type available.

Each device in the network map is represented by two tables in the Statistics data broker. The names of the tables are the result of appending the strings "_hist" and "_std" to the device name. The network controller also maintains statistics regarding the operation of the server in general. The statistics are available in the "__LgrNet___controller__" table.

A.1 Device History Statistics

The name of a history table for a device is the result of appending the string "_hist" to the device name. This table consists of three columns and has a row size of seventy-two. A new record of the table is generated every ten minutes. This allows the table to describe the operation of the datalogger over the last 24 hours if the LoggerNet server version is 1.3.6.8 or greater. If the LoggerNet server version is less than 1.3.6.8, only the last 12 hours will be stored. The counters for this table are set to zero at the beginning of each ten-minute interval. The columns of the table are as follows:

A.1.1 Attempts

Column Name: "Attempts"
Column Definition Description: "Attempts"
Type: uint4
Description: Records the total number of communication attempts the device made during the ten-minute interval. This counter is incremented by one for every entry that appears in the communication status log and is associated with the device.

A.1.2 Failures

Column Name: "Failures"
Column Definition Description: "Failures"
Type: uint4
Description: Records the total number of communication failures that the device experienced during the ten-minute interval. This counter is incremented by one for every "F" record that appears in the communication status log and is associated with the device.

A.1.3 Retries

Column Name: "Retries"
Column Definition Description: "Retries"
Type: uint4
Description: Records the total number of retries that the device experienced during the ten-minute interval. This counter is incremented by one for every

"W" record that appears in the communication status log and is associated with the device.

A.2 Device Standard Statistics

The name of the standard statistics table associated with a device is the result of appending the string "_std" to the device name. The number of columns in the table is variable depending on the device type although there are statistics that are common to all device types.

A.2.1 Communication Enabled

Column Name: "Communication Enabled"
Column Definition Description: "Comm Enabled"
Type: Boolean
Applies To: All Device Types
Description: Relays whether communication is enabled for this device.

A.2.2 Average Error Rate

Column Name: "Avg Error Rate"
Column Definition Description: "Avg Err %"
Type: Float
Applies To: All Device Types
Description: A running average of the number of "W" or "F" messages that are logged in the communication status log for the device versus the total number of messages logged.

A.2.3 Total Retries

Column Name: "Total Retries"
Column Definition Description: "Total Retries"
Type: uint4
Applies To: All Device Types
Description: A running total of the number of communication retry events that have been logged since the device was started or the statistic was last reset.

A.2.4 Total Failures

Column Name: "Total Failures"
Column Definition Description: "Total Failures"
Type: uint4
Applies To: All Device Types
Description: A running total of the number of communication failure events that have been logged since the device was stared or the statistic was last reset.

A.2.5 Total Attempts

Column Name: "Total Attempts"
Column Definition Description: "Total Attempts"
Type: uint4
Applies To: All Device Types
Description: A running total of the number of communication attempts that have been made for the device since the device was started or the statistic was last reset.

A.2.6 Communication Status

Column Name: “Communication Status”
Column Definition Description: “Comm Status”
Type: Byte Enumeration
Applies To: All Device Types
Description: Describes the current communication state of the device. The following values are defined:

1. Normal (last communication succeeded).
2. Marginal (last communication required at least one retry).
3. Critical (last communication failed).
4. Unknown (No communication attempt occurred during the interval).

A.2.7 Last Clock Check

Column Name: “Last Clock Check”
Column Definition Description: “Last Clk Chk”
Type: timestamp
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, CR3000, and RF95T.
Description: Relays the server time when the clock was last checked.

A.2.8 Last Clock Set

Column Name: “Last Clock Set”
Column Definition Description: “Last Clk Set”
Type: timestamp
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, CR3000, and RF95T.
Description: Relays the server time when the clock was last set.

A.2.9 Last Clock Difference

Column Name: “Last Clock Diff”
Column Definition Description: “Last Clk Diff”
Type: int8
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, CR3000, and RF95T.
Description: Relays the difference between the server clock and the datalogger clock at the last time the clock was checked or set.

A.2.10 Collection Enabled

Column Name: “Collection Enabled”
Column Definition Description: “Coll Enabled”
Type: Boolean

Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, and CR3000.
Description: Set to true to indicate that the scheduled collection is enabled for the datalogger.

A.2.11 Last Data Collection

Column Name: "Last Data Collection"
Column Definition Description: "Last Data Coll"
Type: timestamp
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, and CR3000.
Description: The server time when the last data collection took place for the datalogger. This statistic will be updated after a manual poll or scheduled data collection succeeds or partially succeeds (brings in some data from some areas but not all data from all selected areas).

A.2.12 Next Data Collection

Column Name: "Next Data Collection"
Column Definition Description: "Next Data Coll"
Type: timestamp
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, and CR3000.
Description: The server time when the next polling event will take place for the datalogger with the currently active schedule.

A.2.13 Last Collect Attempt

Column Name: "Last_Collect_Attempt"
Column Definition Description: "Last Coll Attempt"
Type: timestamp
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, and CR3000.
Description: Describes the last time data collection (manual poll or scheduled collection) was started for this device.

A.2.14 Collection State

Column Name: "Collection State"
Column Definition Description: "Coll State"
Type: Enumeration
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, and CR3000.
Description: The current state of scheduled collection for the datalogger. The following values are defined:

1. Normal – The normal collection schedule is active.
2. Primary – The primary retry schedule is active.
3. Secondary – The secondary retry schedule is active.
4. Schedule Off – The collection schedule is disabled.
5. Comm Disabled – Communication for this device, one of its parents, or for the entire network is disabled.
6. Invalid Table Defs – Collection for this station is disabled until the table definitions are refreshed.
7. Network Paused – Automated operations are paused for the network.
8. Unreachable – The device cannot be reached through the network.

A.2.15 Values in Last Collection

Column Name: "Vals in Last Collect"
Column Definition Description: "Vals Last Coll"
Type: uint4
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, and CR3000.
Description: The number of scalar values that have been collected from the datalogger since the last poll began.

A.2.16 Values to Collect

Column Name: "Values to Collect"
Column Definition Description: "Vals to Coll"
Type: uint4
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, and CR3000.
Description: The number of scalar values expected in the current or last poll.

A.2.17 Values in Holes

Column Name: "Values in Holes"
Column Definition Description: "Holes"
Type: uint4
Applies To: CR10T, CR10X-TD, CR510T, CR23X-TD, CR1000X series, CR6 series, CR300 series, CR800, CR1000, CR3000, and CRVW.
Description: The number of values in holes that need to be collected from the datalogger.

A.2.18 Values in Uncollectable Holes

Column Name: "Values in Uncollectable Holes"
Column Definition Description: "Uncoll Holes"
Type: uint4

Applies To: CR10T, CR0X-TD, CR510-TD, CR23X-TD, CR1000X series, CR6 series, CR300 series, CR800, CR1000, CR3000, and CRVW.
Description: The total number of values that have been in uncollectable holes since the device was started or the statistic was reset.

A.2.19 Line State

Column Name: "Line State"
Column Definition Description: "Line State"
Type: Enumeration
Applies To: All Devices
Description: The current line state for this device. The following values are defined:

1. Not Applicable – In its current configuration, this device will not communicate directly with the server. This value will appear in association with BMP1 dataloggers connected to the server through an RF95T.

2. Off-Line – The server has no communication resources open for this device.

3. On-Line – The server has communication resources open for this device.

4. Transparent – This device has been dialed to reach a child device.

5. Undialing – The child devices have gone off-line and this device is cleaning up the link so that it can go to an off-line state.

6. Comm-Disabled – Communications are disabled for either this device, its parent, or for the whole network.

7. Unreachable – This device cannot be reached through the network.

8. Pending – The device has requested the link from its parent but that request is still pending.

9. Targeted – The device has requested the link from its parent and its parents are being dialed to open the link.

10. Waiting – The device is a TCP comm port waiting for an incoming connection for call-back.

A.2.20 Polling Active

Column Name: "Polling_Active"
Column Definition Description: "Polling Active"
Type: Boolean
Applies To: All datalogger devices
Description: Reflects whether there is presently a polling operation that is active for the device. A value of true indicates that some sort of polling is taking place.

A.2.21 FS1 to Collect

Column Name: "FS1_Values_to_Collect"
Column Definition Description: "FS1 to Collect"
Type: uint4
Applies To: 21X, CR7, CR10, CR10X, CR500, CR510, CR23X
Description: Reflects the total number of final storage values that need to be collected from final storage area one of a mixed-array datalogger if collect is active for that area. If collection is not active for that area, this statistic reflects the last count that should have been collected.

A.2.22 FS1 Collected

Column Name: "FS1_Values_Collected"
Column Definition Description: "FS1 Collected"
Type: uint4
Applies To: 21X, CR7, CR10, CR10X, CR500, CR510, CR23X
Description: Reflects the total number of final storage values that have been collected from a mixed-array datalogger's final storage area one.

A.2.23 FS2 to Collect

Column Name: "FS2_Values_to_Collect"
Column Definition Description: "FS2 to Collect"
Type: uint4
Applies To: CR10, CR10X, CR510, CR23X
Description: Reflects the total number of final storage values that need to be collected from final storage area two of a mixed-array datalogger if collect is active for that area. If collection is not active for that area, this statistic reflects the last count that should have been collected.

A.2.24 FS2 Collected

Column Name: "FS2_Values_Collected"
Column Definition Description: "FS2 Collected"
Type: uint4
Applies To: CR10, CR10X, CR510, CR23X
Description: Reflects the total number of final storage values that have been collected from a mixed-array datalogger's final storage area two.

A.2.25 Logger Ver

Column Name: "Logger_Interface_Version"
Column Definition Description: "Logger Ver"
Type: uint4
Applies To: 21X, CR7, CR10, CR10X, CR500, CR510, CR23X
Description: Relays the datalogger interface version as given in the datalogger's response to the "A" command.

A.2.26 Watchdog Err

Column Name: "Watchdog_Timer_Reset Count"
Column Definition Description: "Watchdog Err"
Type: uint4
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series,

CR300 series, CR800, CR1000, and CR3000.
Description: Relays the datalogger watchdog error count as given in the mixed-array datalogger's response to the "A" command.

A.2.27 Prog Overrun

Column Name: "Program_Table_Overruns_Count"
Column Definition Description: "Prog Overrun"
Type: uint4
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR800, CR1000, CR3000, CR300 series, CR9000, CR5000
Description: Relays the number of datalogger program overruns that have occurred since the last reset as given in the mixed-array datalogger's response to the "A" command.

A.2.28 Mem Code

Column Name: "Memory_Size_Code"
Column Definition Description: "Mem Code"
Type: uint4
Applies To: 21X, CR7, CR10, CR10X, CR500, CR510, CR23X
Description: Relays the memory size code as given by the mixed-array datalogger's response to the "A" command.

A.2.29 Collect Retries

Column Name: "Collect_Retries"
Column Definition Description: "Coll Retries"
Type: uint4
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, and CR3000.
Description: Reports the number of collection retries that the datalogger device has had since the first collection error occurred. This statistic is reset to zero when the logger returns to a normal collection state.

A.2.30 Low Voltage Stopped Count

Column Name: "Low_Volt_Stopped"
Column Definition Description: "Low Volt Stopped"
Type: uint4
Applies To: CR10X, CR500, CR510, CR23X, CR1000X series, CR1000, CR800, CR3000, CR6 series, and CRVW.
Description: Reports the number of times that a mixed-array datalogger has shut itself down because its supply voltage has been too low. This information is read from the "A" command.

A.2.31 Low Five Volts Error Count

Column Name: "Low_5v"
Column Definition Description: "Low 5v"
Type: uint4
Applies To: CR23X, CR1000X series, CR1000, CR800, CR3000

Description: Reports the number of times the datalogger's +5 volt supply has been reported below five volts.

A.2.32 Lithium Battery Voltage

Column Name: "Lith_Batt_Volt"
Column Definition Description: "Lith Batt Volt"
Type: Float
Applies To: CR10X, CR500, CR510, CR23X, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR800, CR1000, CR3000, CRS451, CRVW, CR9000, CR5000.
Description: Reports the lithium battery voltage on mixed-array dataloggers. This value is extracted from the results of the "A" command.

The CR300 series has a unique field "LithiumBattery" of type String. If the internal battery supplied sufficient power to maintain the clock while external power was absent, the field will display "OK, ON POWER UP." If the internal battery is missing or failed to supply enough power while external power was absent, the field will display "FAIL, ON POWER UP." The LithiumBattery field is only updated on power up, that is, when external power is first applied.

A.2.33 Table Definitions State

Column Name: "TableDefsState"
Column Definition Description: "Table Defs State"
Type: Enumeration
Applies To: CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23XPB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, and CR3000.
Description: Relays the current state of cached table definitions for a table-based datalogger. The following values are defined:

1. None – No table definitions have been received from the datalogger.

2. Current – The LoggerNet server's table definitions are believed to be current for the datalogger.

3. Suspect – A collection attempt has returned an invalid table definitions code. The LoggerNet server needs to verify the table definitions for the datalogger.

4. Getting – Indicates that the LoggerNet server is currently trying to get the table definitions from the datalogger.

5. Invalid – The table definitions are known to be invalid and the need to be refreshed before collection can continue.

A.2.34 Link Time Remaining

Column Name: "Link Time Remaining"
Column Definition Description: "Link Time Remaining"
Type: uint4
Applies To: 21X, CR7X, CR10, CR10X, CR500, CR510, CR23X, CR10T, CR10X-TD, CR510T, CR23X-TD, CR9000, CR5000, CR10X-PB, CR510-PB, CR23X-PB, CR205, CR210, CR215, CR1000X series, CR6 series, CR300 series, CR800, CR1000, CR3000.

Description: Relates the number of milli-seconds remaining for the device to remain on-line based upon its value of maxTimeOnLine for that device as well as for its parent device. If there is no limitation, the value will be the largest possible integer (0xFFFFFFFF). This statistic will be re-calculated every 10 seconds.

A.2.35 RFTD Blacklisted

Column Name: "RFTD Blacklisted"
Column Definition Description: "RFTD Blacklisted"
Type: bool
Applies To: CR10X-TD, CR510-TD, CR23X-TD, RF95-PB
Description: Specifies a value of true if the station is on an RF-TD network and the server believes that the station has been blacklisted by the RF base. A station will be placed on the base's blacklist if it fails to respond to a communication attempt from the server. The base will remove the station from that list when it has responded to a time-division polling attempt by the base.

A.3 Server Statistics

The statistics relating to the host machine for the LoggerNet server or to the operation of the LoggerNet sever as a whole can be found in the table name "__LgrNet____controller__". These statistics are updated every ten seconds. There is only one row defined for the table. The statistics available in this table are as follows:

A.3.1 Disc Space Available

Column Name: "DiscSpaceAvail"
Column Definition Description: "Disc Space Avail"
Type: int8
Description: Relays how many bytes are free on the volume where the server's working directory resides.

A.3.2 Available Virtual Memory

Column Name: "AvailVirtMem"
Column Definition Description: "Avail Virt Mem"
Type: uint4
Description: Relays the amount of virtual memory that is available to the server process.

A.3.3 Used Virtual Memory

Column Name: "UsedVirtMem"
Column Definition Description: "Used Virt Mem"
Type: uint4
Description: Relays the amount of virtual memory that is being used by the server process. This value is derived from the AvailVirtMem by subtracting the value of that statistic from the maximum win32 memory size.

A.3.4 Restart Count

Column Name: "RestartCount"
Column Definition Description: "Restart Count"
Type: uint4

Description: Relates the number of times that the server has restarted automatically after aborting due to an unexpected exception. This is the equivalent of the datalogger watchdog count.

A.3.5 Up Time

Column Name: “UpTime”
Column Definition Description: “Up Time”
Type: int8
Description: Relates the number of milliseconds that the server has been operational.

A.3.6 Last Backup Time

Column Name: “lastBackupTime”
Column Definition Description: “lastBackupTime”
Type: timestamp
Description: Specifies the last time that a backup (automated or otherwise) took place.

A.3.7 Next Auto Backup

Column Name: “nextAutoBackup”
Column Definition Description: “nextAutoBackup”
Type: timestamp
Description: Specifies the next time that an automated backup will take place. If automated backups are not enabled, this statistic will have a value of 1 January 1990.

Campbell Scientific Companies

Campbell Scientific, Inc.
815 West 1800 North
Logan, Utah 84321
UNITED STATES
www.campbellsci.com • info@campbellsci.com

Campbell Scientific Canada Corp.
14532 – 131 Avenue NW
Edmonton AB T5L 4X4
CANADA
www.campbellsci.ca • dataloggers@campbellsci.ca

Campbell Scientific Africa Pty. Ltd.
PO Box 2450
Somerset West 7129
SOUTH AFRICA
www.campbellsci.co.za • cleroux@csafrica.co.za

Campbell Scientific Centro Caribe S.A.
300 N Cementerio, Edificio Breller
Santo Domingo, Heredia 40305
COSTA RICA
www.campbellsci.cc • info@campbellsci.cc

Campbell Scientific Southeast Asia Co., Ltd.
877/22 Nirvana@Work, Rama 9 Road
Suan Luang Subdistrict, Suan Luang District
Bangkok 10250
THAILAND
www.campbellsci.asia • info@campbellsci.asia

Campbell Scientific Ltd.
Campbell Park
80 Hathern Road
Shepshed, Loughborough LE12 9GX
UNITED KINGDOM
www.campbellsci.co.uk • sales@campbellsci.co.uk

Campbell Scientific Australia Pty. Ltd.
PO Box 8108
Garbutt Post Shop QLD 4814
AUSTRALIA
www.campbellsci.com.au • info@campbellsci.com.au

Campbell Scientific Ltd.
3 Avenue de la Division Leclerc
92160 ANTONY
FRANCE
www.campbellsci.fr • info@campbellsci.fr

Campbell Scientific (Beijing) Co., Ltd.
8B16, Floor 8 Tower B, Hanwei Plaza
7 Guanghua Road
Chaoyang, Beijing 100004
P.R. CHINA
www.campbellsci.com • info@campbellsci.com.cn

Campbell Scientific Ltda.
Fahrenheitstraße 13
28359 Bremen
GERMANY
www.campbellsci.de • info@campbellsci.de

Campbell Scientific do Brasil Ltda.
Rua Apinagés, nbr. 2018 ─ Perdizes
CEP: 01258-00 ─ São Paulo ─ SP
BRASIL
www.campbellsci.com.br • vendas@campbellsci.com.br

Campbell Scientific Spain, S. L.
Avda. Pompeu Fabra 7-9, local 1
08024 Barcelona
SPAIN
www.campbellsci.es • info@campbellsci.es

Please visit www.campbellsci.com to obtain contact information for your local US or international representative.