Quick Start Guide for TMW .NET Protocol Components

This will provide users with previous experience with communication protocols, and/or the TMW architecture, a quick step-by-step guide to integrating the TMW .NET Protocol Components on their platform. First time users will also get a feel for the steps required to implement a TMW .NET Protocol Components on their platform.

This will serve to highlight each step of the implementation process and provide a very high level description of what is required. If more information is required, please refer to the sample applications and the detailed sections found later in this manual. For the sake of brevity not every situation can be explained in detail. Again, if there are any questions as to whether a specific step is appropriate to your TMW .NET Protocol Components please refer to the detailed sections later in the manual.

Install the TMW .NET Protocol Components

The TMW .NET Protocol Components are delivered as a standard windows setup file. For information about the Source Code Library version see Chapter 3.

The TMW .NET Protocol Components files should be installed in the desired target directory (<*installdir*>). TMW recommends retaining the directory structure of the SCL as it is supplied. To install TMW .NET Protocol Components simply run the setup file and follow the prompts.

We have included Visual Studio 2008 integrated help for the TMW .Net Protocol Components. To install this make sure Visual Studio is shutdown and run the installation setup program 'Install VS 2008 Help' from the Start Menu\Program Files\Triangle MicroWorks\.NET Protocol Components\Help. After this is installed selecting a object exported by the .NET Protocol Components such as MDNPRequest or M14Request in Visual studio and select F1 to get detailed help on the requests available.

In addition to VS 2008 integrated help we have also included standard compiled html help. This help can be accessed from the Start Menu\Program Files\Triangle MicroWorks\.NET Protocol Components\Help menu. All of the classes and methods exported by the TMW .NET Components may be viewed from this help document.

We have also included some sample programs to get you started. The samples that contain a Graphical User Interface (GUI) can be accessed at: Start Menu\Program Files\Triangle MicroWorks\.NET Protocol Components\Samples. These files should be used as a starting point for writing your own applications. The source code for all of the samples is located at: <installdir>\Samples.

The default <installdir> would normally be C:\Program Files\Triangle MicroWorks\.NET Protocol Components.

Included Samples

DNP

DNPMasterGUI – a Windows Forms application that implements a simple DNP master .

DNPMasterGUI.VBNet – a Windows Forms Visual Basic application that implements a simple DNP master.

DNPSlaveGUI – a Windows Forms application that implements a simple DNP slave (outstation).

DNPSlaveGUI.VBNet – a Windows Forms Visual Basic application that implements a simple DNP slave (outstation).

DNPMasterModem – a Windows Forms application that implements a simple DNP master that opens a channel disabled and allows modem control before the DNP protocol starts communicating.

DNPMasterFileTransfer – a Windows Forms application that implements a simple DNP master that allows you to open/close/read and write files and read directories. It uses the built in database interface to the Windows File System while using database events for other data types.

DNPMasterDatasets – a Windows Forms application that implements a DNP master that provides an interface for data set configuration, reads and writes.

DNPSlaveDatasets – a Windows Forms application that implements a DNP slave (outstation) that provides an interface for data sets.

DNPDatabaseEvents – shows how to use the event based model for a DNP master and slave (outstation).

DNPDatabaseEvents.VBNet – shows how to use the event based model for a DNP master and slave (outstation) in Visual Basic.

DNPMasterDatabaseEvents – shows how to use the event based model for a DNP master .

DNPSlaveDatabaseEvents – shows how to use the event based model for a DNP slave (outstation).

I101

I101MasterGUI – a Windows Forms application that implements a simple IEC 60870-5-101 master

I101SlaveGUI – a Windows Forms application that implements a simple IEC 60870-5-101 slave

I101MasterDatabaseEvents – shows how to use the event based model for an IEC 60870-5-101 master.

I101SlaveDatabaseEvents – shows how to use the event based model for an IEC 60870-5-101 slave.

I102

I102SlaveGUI – a Windows Forms application that implements a simple IEC 60870-5-102 slave

I102MasterGUI – a Windows Forms application that implements a simple IEC 60870-5-102 master

I102DatabaseEvents – shows how to use the event based model for an IEC 60870-5-102 master and slave

I102MasterDatabaseEvents – shows how to use the event based model for an IEC 60870-5-102 master.

I102SlaveDatabaseEvents – shows how to use the event based model for an IEC 60870-5-102 slave.

I103
I103SlaveGUI – a Windows Forms application that implements a simple IEC 60870-5-103 slave.

I103MasterGUI – a Windows Forms application that implements a simple IEC 60870-5-103 master.

I103DatabaseEvents – shows how to use the event based model for an IEC 60870-5-103 master and slave.

I103MasterDatabaseEvents – shows how to use the event based model for an IEC 60870-5-103 master.

I103SlaveDatabaseEvents – shows how to use the event based model for an IEC 60870-5-103 slave.

I104

I104MasterGUI – a Windows Forms application that implements a simple IEC 60870-5-104 master

I1104SlaveGUI – a Windows Forms application that implements a simple IEC 60870-5-104 slave

I104RedundantMasterGUI – a Windows Forms application that implements a simple IEC 60870-5-104 master with redundancy enabled.

I104RedundantSlaveGUI – a Windows Forms application that implements a simple IEC 60870-5-104 slave with redundancy enabled.

I104MasterDatabaseEvents – shows how to use the event based model for an IEC 60870-5-104 master.

I104SlaveDatabaseEvents – shows how to use the event based model for an IEC 60870-5-104 slave.

Modbus

MMBWebService – a web service application that wraps a Modbus master as a web service.

SMBSimulator – a Windows Forms application that implements a simple Modbus slave that can be used with the MMBWebService sample.

MBDatabaseEvents – shows how to use the event based model for a Modbus master and slave.

MBMasterDatabaseEvents – shows how to use the event based model for a Modbus master.

MBSlaveDatabaseEvents – shows how to use the event based model for a Modbus slave

Create Interoperability Guide

Your documentation should include an Interoperability Document, as described in the IEC 60870-5 and DNP standards.

The TMW .NET Protocol Components includes an example Configuration/Interoperability Guide. This guide is named <*scl*> *CI Guide.rtf* and is located in the <*installdir* > directory. This document is a template with much of the information already supplied, indicating the features of the Source Code Library. You will need to edit the document to indicate the features your device will support and to describe methods of configuration.

Build the TMW .NET Protocol Components and Samples

The first step in implementing the TMW .NET Protocol Components should be to build the samples before making any modifications. Once installed, all TMW .NET Protocol Components sample code and/or binary files reside in the subdirectories under <*installdir*>.

For the source code version of TMW .NET Protocol Components all of the source files in each of these subdirectories must be compiled. This step confirms proper installation of the components as well as highlighting any compiler-specific issues. For information about the Source Code directory structure and other details refer to Chapter 3.

The project and source files for the .NET Component sample applications reside in <*installdir*>*\Samples\...*. To build a sample application, open the supplied Visual Studio project file (xxx.csproj) and build it with Visual Studio. This will create a solution file which may then be saved.

You should refer to these sample applications when completing the steps in the Quick Start Guide and building your own application.

Common TMW .NET Protocol Components Setup

Calls into the TMW .NET Protocol Components are limited to the following areas: licensing; application initialization; database initialization and managing channels, sessions, and (for IEC 60870-5 protocols) sectors.

Licensing, application, and database initialization is described in the following sections. Managing channels, sessions, and sectors is described in Section Chapter 0.

Licensing

When the TMW .NET Protocol Components are first installed on your machine they come with an initial Trial License. This license provides full functionality for a period of 21 days to evaluate the Protocol Components for purchase. During this trial period an application will only be allowed to run for 1 hour.

Triangle MicroWorks provides three TMW .NET Protocol Components Licensing options beyond the free trial license.

1. Single Use License – This is a license that is tied to a specific computer.
2. Redistributable License – This form gives the developer the right to re-distribute the TMW .NET Protocol Components with a single application to run on an unlimited number of computers.
3. Redistributable with Source Code License – This is similar to the Redistributable License except that Source code is provided to the developer.

Single Use License

Single use licensing supports installation and use on a single machine. The license is specific for the machine the software is installed on. In order to use single use licensing, the licensing tool is used to request an activation key. Once the activation key is received from Triangle MicroWorks Inc. it is then installed on the target machine using this same tool. The tool is accessible from the Start menu under the Licensing item.

No licensing changes need to be made to the startup code in the example files or in your own application. The software will detect the single use license and functionality based on that license will be allowed.

Redistributable License

When using this form of TMW .NET Protocol Components product licensing, the customer will receive a unique license key from Triangle MicroWorks Inc. upon payment and execution of the license agreement. This key should be included in the product the customer is creating as follows:

During initialization of the application (usually in the constructor) register for the TMWApplication.ObtainLicenseEvent. Note that the event handler should be established prior to the TMWApplication constructor which is called by
'`new TMWApplicationBuilder()`'

```
TMWApplication.ObtainLicenseEvent += new
TMWApplication.ObtainLicenseEventDelegate(TMWApplication_ObtainLicenseE
vent);
```

And implement the following event handler:

```
TMW.SCL.NETLicense TMWApplication_ObtainLicenseEvent()
{
  NETLicense license = new NETLicense();
  license.setLicenseKey("KeyFromTMW");
  return license;
}
```

The text "`KeyFromTMW`" in the event handler should be replaced with the actual license key provided by Triangle MicroWorks Inc.

Source Code Version

No licensing changes need to be made to the startup code in the example files or in your own application. The source code version does not require a license to run.

Application Initialization

All protocols share a single application object: TMWApplication. This object is created by creating a TMWApplicationBuilder object and asking for the TMWApplication object using the getAppl() method. The TMWApplication object has an application initialization method, InitEventProcessor(), which must be called once to initialize the TMW .NET Protocol Components. It also has a property, EnableEventProcessor, that should be set to true to periodically check if data is available on any of the communication channels.

```
TMWApplicationBuilder applBuilder = new TMWApplicationBuilder();
TMWApplication pAppl = TMWApplicationBuilder.getAppl();
pAppl.InitEventProcessor();
pAppl.EnableEventProcessor = true;
```

The property TMWApplication.LoopPeriod is used to specify the rate in milliseconds at which checks for data on a channel are performed.

Database Initialization

By default, all of the TMW .NET Protocol Components use a built-in database to store protocol data. Should you need to override this behavior, set the TMWSimDataBase.UseSimDatabase property to false. This will enable the event based database model allowing use of a custom user defined database. Most slave components s support a UseSimControlDatabase property also. This allows a slave application to separate support for Output Controls from the other database functionality. Examples of this can be found in the xxxDatabaseEvents samples.

TMW .NET Protocol Components Target

The TMW .NET Protocol Components uses the WinIoTarg.dll to implement the target layer communications for TCP/IP, RS232, etc… No user implementation of a target layer is required.

Manage Channels, Sessions, and Sectors

Channels, sessions, and sectors (in some protocols) are managed by methods, properties, and events found in the TMWChannel, TMWSession, and TMWSector classes and their protocol specific derivations.

These classes provide methods and events for opening, modifying, and closing channels, sessions, or sectors. Each of these classes provides configuration properties that can be used to configure the object. The constructors of these objects initialize the objects to common default values. After creating an object, the target application should modify the configuration properties to any specific values required by your application prior to opening the Channel, Session or Sector. To see all of the properties available for a .NET Component class you should use the integrated or html help functionality described above. For example select DNPChannel and enter F1 or search for DNPChannel in the html help file accessible from the Start Menu.

Install the TMW .NET Protocol Components

Here is a typical example (open a DNP Master) of this initialization process:

```
public partial class MasterForm : Form
{
  private DNPChannel masterChan;
  private MDNPSession masterSesn;

 …

  public MasterForm()
  {
    ...
    masterChan = new DNPChannel(TMW_CHANNEL_OR_SESSION_TYPE.MASTER);

    masterChan.Type = WINIO_TYPE.TCP;
    masterChan.Name = ".NET DNP Master";  // name displayed in analyzer window
    masterChan.WinTCPipAddress = "127.0.0.1";
    masterChan.WinTCPipPort = 20000;
    masterChan.WinTCPmode = TCP_MODE.CLIENT;
    masterChan.OpenChannel();

    masterSesn = new MDNPSession(masterChan);
    masterSesn.AuthenticationEnabled = false;

    masterSesn.OpenSession();
    ...
```

After opening the session it should be possible to test communications with a remote device. Note that the TMWApplication initialization code is not shown in this code segment, but should be performed as shown above.

The TMW .NET Protocol Components Database

The TMW .NET Protocol Components database is the mechanism for dealing with the processing of the protocol specific values, flags, timestamps, qualities, etc. in your application.

The interface between the TMW .NET Protocol Components and the application database is a set of events defined in TMWSimDatabase and its protocol specific derivations or a set of predefined TMWSimPoints and its protocol specific derivations.

There are 2 ways to use the database classes.

1. Use the built in simple database.
2. Use the .NET event interface in the database.

When the TMW .NET Protocol Components is shipped, the simple “simulated” database is enabled. In this mode the protocol data (i.e. values, flags, timestamps, qualities, etc.) are managed by the built in database and provided through the TMWSimDatabase and TMWSimPoint class hierarchy. After opening, each protocol specific Session or Sector will have a database associated with it. This database may be accessed through the Session or Sectors SimDatabase property. The SimDatabase has an UpdateDBEvent event associated with it that provides an event to notify the application of TMWSimPoint changes.

Here is an example using the built in database:

```
private void UpdateDBEvent(TMWSimPoint simPoint)
{
  if (InvokeRequired)
    Invoke(new UpdatePointDelegate(UpdateDBEvent), new object[] { simPoint });
  else
  {
    switch (simPoint.PointType)
    {
      case 1:
        // Binary Input
        updateBinaryInput(simPoint);
        break;
      case 10:
        // Binary Output (CROB)
        updateBinaryOutput(simPoint);
        break;
      case 20:
        // Binary Counters
        updateBinCntr(simPoint);
        break;
      case 30:
        // Analog Inputs
        updateAnalogInput(simPoint);
        break;
      case 40:
        // Analog Output
        updateAnalogOutput(simPoint);
        break;
      default:
        protocolBuffer.Insert("Unknown point type in database update routine");
        break;
    }
   }
}

...

db = (MDNPSimDatabase)masterSesn.SimDatabase;
// Register to receive notification of database changes
db.UpdateDBEvent += new TMWSimDataBase.UpdateDBEventDelegate(UpdateDBEvent);
```

The 2nd database model disables the built in database and utilizes .NET events on the TMWSimDatabase and its derivations to notify the application of various protocol events.   This mechanism provides a tighter integration to the protocol and leaves the management of protocol data to the application developer. This model is more appropriate when developing an application that provides its own database.

```
private static void OpenSlave()
{
  slaveChan = new DNPChannel(TMW_CHANNEL_OR_SESSION_TYPE.SLAVE);

  slaveChan.Type = WINIO_TYPE.TCP;
  slaveChan.Name = ".NET DNP Slave";  /* name displayed in analyzer window */
  slaveChan.WinTCPipAddress = "127.0.0.1";
  slaveChan.WinTCPipPort = 20000;
  slaveChan.WinTCPmode = TCP_MODE.SERVER;
  slaveChan.OpenChannel();

  slaveSesn = new SDNPSession(slaveChan);
  slaveSesn.UnsolAllowed = true;

  slaveSesn.OpenSession();

  sdb = (SDNPDatabase)slaveSesn.SimDatabase;

  SDNPDatabase.UseSimDatabase = false;
  SDNPDatabase.UseSimControlDatabase = false;


  // binary inputs
  sdb.BinInQuantityEvent += new
SDNPDatabase.BinInQuantityDelegate(SlaveBinInQuantityEvent);
  sdb.BinInGetPointEvent += new
SDNPDatabase.BinInGetPointDelegate(SlaveBinInGetPointEvent);
  SDNPDatabase.BinInReadEvent += new SDNPDatabase.BinInReadDelegate(SlaveBinInReadEvent);

  // analog inputs
  sdb.AnlgInQuantityEvent += new
SDNPDatabase.AnlgInQuantityDelegate(SlaveAnlgInQuantityEvent);
  sdb.AnlgInGetPointEvent += new
SDNPDatabase.AnlgInGetPointDelegate(SlaveAnlgInGetPointEvent);
  SDNPDatabase.AnlgInReadEvent += new
SDNPDatabase.AnlgInReadDelegate(SlaveAnlgInReadEvent);

// binary outputs
  sdb.BinOutQuantityEvent += new
SDNPDatabase.BinOutQuantityDelegate(SlaveBinOutQuantityEvent);
  sdb.BinOutGetPointEvent += new
SDNPDatabase.BinOutGetPointDelegate(SlaveBinOutGetPointEvent);
  SDNPDatabase.BinOutReadEvent += new
SDNPDatabase.BinOutReadDelegate(SlaveBinOutReadEvent);
  SDNPDatabase.BinOutSelectEvent += new
SDNPDatabase.BinOutSelectDelegate(SlaveBinOutSelectEvent);
  SDNPDatabase.BinOutOperateEvent += new
SDNPDatabase.BinOutOperateDelegate(SlaveBinOutOperateEvent);

  // analog outputs
  sdb.AnlgOutQuantityEvent += new
SDNPDatabase.AnlgOutQuantityDelegate(SlaveAnlgOutQuantityEvent);
  sdb.AnlgOutGetPointEvent += new
SDNPDatabase.AnlgOutGetPointDelegate(SlaveAnlgOutGetPointEvent);
  SDNPDatabase.AnlgOutReadEvent += new
SDNPDatabase.AnlgOutReadDelegate(SlaveAnlgOutReadEvent);
  SDNPDatabase.AnlgOutSelectEvent += new
SDNPDatabase.AnlgOutSelectDelegate(SlaveAnlgOutSelectEvent);
  SDNPDatabase.AnlgOutOperateEvent += new
SDNPDatabase.AnlgOutOperateDelegate(SlaveAnlgOutOperateEvent);


}
```

The UseSimDatabase and UseSimControlDatabase properties on the TMWSimDatabase switches between these two modes. Note that in the above example they are set to false.

Note also that the event handlers are registered with the database so that your application can process the requests through the protocol.

Issue Commands (Master Sessions)

For Master Sessions, commands are issued by calling methods on the TMWRequest class and its derivations. The request can have an event specified that will notify the application of the request status (i.e. completion, failure, etc…).

When the request is completed, the TMW .NET Protocol Components will call the event. The event arguments include the request and the received response.

Information on protocol-specific commands is given in the protocol-specific integrated or html help described above.

Here is an example of issuing a request (Master DNP Binary output command):

```
private void BinOutOn_Click(object sender, EventArgs e)
{
  // Determine which point this request is for
  ushort pointNumber = Convert.ToUInt16((sender as Control).Tag);

  // Build and send the request
  CROBInfo crobData = new CROBInfo(pointNumber, CROB_CTRL.LOFF, 1, 0, 0);
  CROBInfo[] crobArray = { crobData };

  MDNPSimBinOut point = db.LookupBinOut(pointNumber);
  MDNPRequest request = new MDNPRequest(masterSesn);
  request.BinaryCommand(MDNPRequest.DNP_FUNCTION_CODE.SELECT, true, true, 100,
(byte)MDNPRequest.DNP_QUALIFIER.Q_8BIT_INDEX, crobArray);
}
```

Add Support for Change Events (Slave Sessions)

For Slave sessions, the TMW .NET Protocol Components can be configured to periodically scan for change events by setting the appropriate property on the Session or Sector. Alternately, events can be generated directly by calling the appropriate AddEvent method on the TMWSimPoint object derivation. To have more control over the event data or if the simple built-in database is not being used methods on the session or sector can also be used to add events, for example SDNPSession::BinInAddEvent(..) or S14Sector::MspAddEvent(..);

Enabling scanning for change events is a configuration option that is specified when opening the session or sector.

Examples of this can be found in the slave sample applications.

Test your Implementation

To test your implementation, you will need a system or application to act as the 'opposite' end of the communication link. For example, if you are implementing a Slave device, you will need a corresponding Master.

Triangle MicroWorks provides a flexible, scriptable Test Harness that can be used for this purpose. The Communication Protocol Test Harness is a Windows application that acts as a simple Master or Slave device and can be programmed with an automated test sequence through a scripting capability.

The protocol analyzer output from the Test Harness allows you to verify the proper operation of the device you are testing. It also allows you to generate a reference protocol analyzer output file that can be used for comparison to later tests to verify only intended modifications have occurred.

For more information please see the Triangle MicroWorks, Inc. web site at *'http://www.TriangleMicroWorks'*.