

# **Technical Reference**



## **Tektronix Logic Analyzer Family TPI.NET Remote Client Design Guide Volume 2**

**071-1353-00**

This document applies to TLA System Software  
Version 4.3 and above.

**[www.tektronix.com](http://www.tektronix.com)**

Copyright © Tektronix, Inc. All rights reserved. Licensed software products are owned by Tektronix or its suppliers and are protected by United States copyright laws and international treaty provisions.

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable.

Tektronix products are covered by U.S. and foreign patents, issued and pending. Information in this publication supercedes that in all previously published material. Specifications and price change privileges reserved.

Tektronix, Inc., P.O. Box 500, Beaverton, OR 97077

TEKTRONIX and TEK are registered trademarks of Tektronix, Inc.

# TPI.NET Remote Client Design Guide

- 1 Introduction .....2
- 2 Implementing TPI.NET Remote Clients .....3
  - 2.1 Compiling Against the Shared DLL.....3
  - 2.2 Deployment.....3
  - 2.3 Connecting to the TLA Application .....4
    - 2.3.1 Connecting with a Configuration File.....4
  - 2.4 Accessing Interfaces.....5
  - 2.5 Receiving Events from the TLA Application .....5

# 1 Introduction

## TPI.NET Remote Clients Defined

TPI.NET Remote Clients are managed .NET applications that access functionality of the TLA application. Such an application runs as a separate process from the TLA application and may be run either on the same controller or on a PC connected via a network to the TLA. A remote client application has its own user interface separate from the TLA application. Access to the TLA is realized through remoting services built into .NET and through the shared DLL "TlaNetInterfaces.dll" that remote client applications compile against and use at runtime.

## Development Overview

TPI.NET remote clients are developed by creating managed applications that utilize services, classes, and interfaces defined by the TlaNetInterfaces.dll. Before a remote client can be fully implemented, the developer needs the following items:

- A .NET enabled compiler, Visual Studio .NET for example.
- A copy of TlaNetInterfaces.dll.
- Version 1.0 of the .NET Framework.

All remote client developers must use .NET-enabled compilers, which can create assemblies with code that executes under the Common Language Runtime. The .NET interface specifications for the TLA application have been written with the expectation that remote clients will be developed most often with Microsoft's Visual Basic, C#, or Managed C++.

In order to create a remote client application, the developer must have a copy of TlaNetInterfaces.dll, which is a DLL present in every V4.3 TLA software installation. It is an assembly that contains metadata for all TLA remote interfaces and TPI.NET types. The file holds the compiled definitions of all .NET types that are currently described in the document entitled "TPI.NET Reference Manual." Both the TLA application and all remote client applications are compiled using references to the metadata in this assembly. TlaNetInterfaces.dll serves a purpose similar to that of a header file. It holds the definitions of types that will be used by multiple projects, including projects that are compiled separately. The TlaNetInterfaces.dll may be copied from the folder "C:\Program Files\TLA 700\System" on the TLA system.

V4.3 TLA software is built against the version 1.0 of the .NET framework (SP2). Visual Studio 2002 creates applications for this version of the .NET framework and is highly recommended for building remote clients targeting the V4.3 TLA application. The TLA application ships with a redistributable install for the version 1.0 of the .NET framework and it may also be downloaded from the Microsoft web site.

Visual Studio 2003 creates applications for version 1.1 of the .NET framework and should not be used to build remote clients. At the time of release, V4.3 has not been tested against clients that use version 1.1 of the .NET framework.

## Code Samples

The same CD that contains this document also contains sample code for a small TPI.NET client application. This sample is presented in three separate languages supported by Microsoft Visual Studio .NET: Visual Basic .NET, C#, and C++ with managed extensions. Each sample has a solution file that can be opened and built to produce a working plug-in or client application.

The remote TPI.NET client application sample is in the following directories:  
TPI.NET and PlugIn Documentation\Remote TPI.NET Client Samples\CSharp Remote TPI.NET Client,

TPI.NET and PlugIn Documentation\Remote TPI.NET Client Samples\VB Remote TPI.NET Client,  
TPI.NET and PlugIn Documentation\Remote TPI.NET Client Samples\CPP Remote TPI.NET Client.

To use these sample projects, copy the directory contents of the desired sample from the CD into a directory on the computer that contains your development environment.

Please note that all the Visual Studio projects contain a pre-release version of the TlaNetInterfaces.dll assembly. If you experience any runtime problems related to TlaNetInterfaces.dll, you might get better results by compiling the samples, and any other TPI.NET clients, with the final release version of this assembly. The release version is always installed in the following directory of your TLA application software:

C:\Program Files\TLA 700\System\TlaNetInterfaces.dll

## 2 Implementing TPI.NET Remote Clients

This section describes how to implement TPI.NET remote clients so that the resulting application can be used with the TLA application. This section describes how to connect to the TLA application, special considerations regarding handling of asynchronous events from the TLA application, and limited information regarding how to navigate the hierarchy of TLA application objects available to the remote client. A more comprehensive specification of the object hierarchy is documented separately in "TPI.NET Reference Manual".

### 2.1 Compiling Against the Shared DLL

The remote client must be compiled against the "TlaNetInterfaces.dll" shared DLL. How this is accomplished depends upon the compiler and development environment being used.

Under Visual Studio 2002 C++, the path of the folder containing the shared DLL is specified through the Project Properties. Open "Configuration Properties", then "C/C++" and select the "General" category. Enter the folder name on the line labeled "Resolve #using References". Add the line **#using "TlaNetInterfaces.dll"** at the top of the .cpp source files of the remote client.

Under Visual Studio 2002 C#, a reference to the shared DLL is specified through the Solution Explorer. Select the project node from the solution tree control. Select "Add Reference..." from the Project menu and use the dialog to add a reference to the shared DLL.

The shared DLL contains type and interface definitions organized into three namespaces:

*Tektronix.LogicAnalyzer.Common*  
*Tektronix.LogicAnalyzer.TpiNet*  
*Tektronix.LogicAnalyzer.PlugIn*

Types and interfaces may be referred to with either fully qualified namespace names or by just their names when namespace scoping statements have been used.

### 2.2 Deployment

A remote client application may reside directly on the TLA controller or on a networked PC. In either case, version 1.0 of the .NET Framework must be installed on the host computer. The .NET Framework comes pre-installed on version 4.3 TLA controllers. The TLA application ships with a re-distributable install for the version 1.0 of the .NET framework and it may also be downloaded from the Microsoft web site.

A TPI.NET remote client has minimal installation requirements in order to be used with the TLA application. Generally, a remote client consists of the following files located in a common folder:

- The remote client .EXE executable file.
- Any .DLL files (if any) that are specific to the remote client itself.
- A copy of the TlaNetInterfaces.dll.
- A "remoteClient.exe.config" file. The filename should be changed to match the name of the remote application .EXE file and also include the ".config" extension. Configuration files are explained below.

No registry changes are necessary for a remote client application installation. The installation is accomplished simply by placing the appropriate files in a common folder as outlined above.

## 2.3 Connecting to the TLA Application

The TLA application exposes an ITlaSystem interface object to remote client applications. The ITlaSystem interface is the root object from which all other TLA application objects and interfaces are accessed. The TLA application must be running before a remote client can connect to it.

The TLA application publishes this interface with the help of the configuration file "C:\Program Files\TLA 700\tla700.exe.config".

The ITlaSystem object is a well-known singleton. It can be obtained by calling the framework method System.Activator.GetObject() with arguments specifying ITlaSystem type and a suitable url for the networked TLA controller. A simpler and more flexible means of obtaining the ITlaSystem object is by using a client configuration file and a utility class provided by the shared DLL.

### 2.3.1 Connecting with a Configuration File

A remote client configuration file resides in the same folder as the executable for the remote client. Using a configuration file is more flexible than hard coding a network address into the System.Activator.GetObject() call. It specifies the network address of the TLA with a line of the form

```
url="tcp://localhost:9000/TlaSystem"
```

The *localhost* component can be replaced by an IP address or symbolic IP name. This allows a remote client to target a different TLA system without needing to be recompiled.

An example remote client configuration file is located in

C:\Program Files\TLA 700\Samples\TPI.NET Samples\ITlaSystemRemoteClient.config

Copy this to the same folder as the remote client executable and rename it to match the filename of the remote client with a ".config" extension. The remote client named automate.exe would have a configuration file named automate.exe.config.

The shared DLL contains a Tektronix.LogicAnalyzer.TpiNet.RemoteTlaSystem utility class that uses this configuration file to call System.Activator.GetObject() for you and returns the ITlaSystem object. The RemoteTlaSystem.Connect() method takes the name of the configuration file and returns the ITlaSystem object.

C# example:

```
using Tektronix.LogicAnalyzer.TpiNet;  
public class MyTlaConnection
```

```

{
    public static ITlaSystem ConnectToTla()
    {
        return RemoteTlaSystem.Connect( "automate.exe.config" );
    }
}

```

An application developer might use the configuration file with a *localhost* setting to develop and debug the application on a PC. In this case, TLAVu could be installed on the PC and would be targeted by the application under development. The developer could also choose to target a real TLA controller during development by replacing *localhost* in the configuration file with the network address of the TLA controller.

## 2.4 Accessing Interfaces

The remote application communicates with the TLA application through the interfaces defined in the TlaNetInterfaces.dll. An interface is essentially a pure virtual class. It is a contract defining what methods, properties, and events a caller can expect and an implementer must provide.

At run time, the remote application obtains a handle to an instance of an object implementing the ITlaSystem interface. The ITlaSystem object is obtained through a call to RemoteTlaSystem.Connect( filename ) as described in the section above.

The ITlaSystem interface provides some values directly, such as the application software version string through the ITlaSystem.SWVersion property. The ITlaSystem interface also provides handles to other interface objects. These in turn may provide values or services directly or they may provide handles to other interfaces. The interface objects are accessed as a hierarchy with ITlaSystem as the root.

The complete list of interfaces is documented separately in " TPI.NET Reference Manual".

## 2.5 Receiving Events from the TLA Application

A remote application can register to receive notification of events from the TLA application. Events include acquisition started, acquisition complete, data sources added or deleted, module saved, module loaded, and many more. Different interfaces in TlaNetInterfaces.dll define different events.

Support for events is a standard feature of the .NET programming languages and compilers. An event is a member of a class or interface in the same sense that methods and properties are members. The ITlaRunControl interface has a RunCompleted event that is expressed in C# as:

```
event System.EventHandler RunCompleted;
```

An event member has a name (RunCompleted) and a delegate type (System.EventHandler). The delegate type defines the signature of any handler method that can be called in response to the event. More information on event syntax and delegates is available in the Visual Studio documentation.

### Multi-threading

When one application callback is invoked in response to an event from another application, the .NET framework invokes this callback from a thread of a thread pool on the receiving process. The callback does not execute in the main thread of the receiving application. If a remote application utilizes events from the TLA application, it becomes a multi-threaded application.

A remote client utilizing events from the TLA application must be designed as a multi-threaded application. This includes making its data access thread-safe and preventing deadlock situations.

### Asynchronous and Non-Blocking

In order to prevent deadlocks associated with maintaining thread-safe data, the TLA application sends remote event notifications on a worker thread and does not block waiting for remote callbacks to complete. From the remote client application perspective, events may be received well after the event of interest actually occurred. The callback implementation should not assume that the TLA application is still in the same state as when the event originally occurred.

### Remoter Event Shims

Under the .NET Framework, the server (TLA application) needs access to the assembly (remote EXE file) containing the event handler. This presents a problem when sending events to remote clients.

To solve this problem, a number of small “shim” classes are defined in the TlaNetInterfaces.dll. These shim classes are available to both the TLA application and the remote client through the shared DLL. The shim classes are used as a bridge between the TLA event and the remote client. A remote client that attempts to register for events directly rather than with shims will experience a run time exception.

The remote client creates a new instance of a shim and registers the shim’s OnRemoteEvent method on the TLA event. The remote client registers its actual callback on the shim’s RemoteEventOccurred event. When the TLA event fires, the shim’s OnRemoteEvent method is called and it fires the RemoteEventOccurred event to call the remote clients actual callback.

A C# example of using the EventRemoter shim for the ITlaRunControl.RunCompleted event:

```
using Tektronix.LogicAnalyzer.Common;
using Tektronix.LogicAnalyzer.TpiNet;
public class TestRunCompleted
{
    EventRemoter shim;

    public void RegisterRunCompleted( ITlaRunControl runControl )
    {
        shim = new EventRemoter();
        // Connect TLA event to shim
        runControl.RunCompleted +=
            new System.EventHandler(shim.OnRemoteEvent);
        // Connect shim event to client handler
        shim.RemoteEventOccurred +=
            new System.EventHandler(this.HandleRunCompleted);
    }
    public void UnregisterRunCompleted( ITlaRunControl runControl )
    {
        // Disconnect TLA event from shim
        runControl.RunCompleted -=
            new System.EventHandler(shim.OnRemoteEvent);
    }
}
```



```

        // Disconnect shim event from client handler
        shim.RemoteEventOccurred -=
            new System.EventHandler(this.HandleRunCompleted);
        shim = null;
    }
    void HandleRunCompleted( System.Object sender, System.EventArgs e )
    {
        System.Console.WriteLine("RunCompleted event received.");
    }
}

```

The TlaNetInterfaces.dll provides the following remote event shims in the Tektronix.LogicAnalyzer.TpiNet namespace:

Delegate	Shim Class
System.EventHandler	EventRemoter
CollectionChangedHandler	CollectionChangedRemoter
RepetitiveStopHandler	RepetitiveStopRemoter

The TlaNetInterfaces.dll provides the following remote event shims in the Tektronix.LogicAnalyzer.Common namespace:

Delegate	Shim Class
ObjectEventHandler	ObjectEventRemoter
SearchCompletedHandler	SearchCompletedRemoter
LockedItemChangedHandler	LockedItemChangedRemoter
RunStateChangedHandler	RunStateChangedRemoter