Keysight DigitalTestApps

KEYSIGHT

PROGRAMMING GUIDE

Notices

Revision

Version 7.20.0

Edition

September 1, 2023

Available in electronic format only

Published by:
Keysight Technologies, Inc.
1900 Garden of the Gods Road
Colorado Springs, CO 80907 USA

Warranty

Technology License

U.S. Government Rights

Safety Notices

CAUTION

A **CAUTION** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a **CAUTION** notice until the indicated conditions are fully understood and met.

WARNING

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

In This Book

This book is your guide to programming Keysight DigitalTestApps.

- Chapters 1-4 describe how to use an existing remote client:
  - Chapter 1, “Keysight DigitalTestApps Remote Interface,” starting on page 9.
  - Chapter 2, “Using Sample Remote Clients,” starting on page 17.
  - Chapter 3, “Remote Programming Languages and Sample Code,” starting on page 41.
  - Chapter 4, “Recommended Remote Programming Practices,” starting on page 85.
- Chapter 5, “Developing a Remote Client,” starting on page 91, describes how to create new remote clients.
- Chapter 6, “Troubleshooting,” starting on page 141, lists common problems and resolutions.

How to use this book

Start by reading the *Keysight DigitalTestApps Programming Getting Started* guide; then, read all the chapters in this guide in order.

Contents

In This Book / 3

1 Keysight DigitalTestApps Remote Interface

Remote Interface Documentation / 10

Keysight KS8400A Test Automation for PathWave (TAP) Interface / 13

.NET Framework Interface / 14

Keysight.DigitalTestApps.Framework.Remote / 14
Keysight.DigitalTestApps.Framework.Remote.Advanced / 14
Keysight.DigitalTestApps.Framework.Remote.Exceptions / 15

LabVIEW Interface / 16

2 Using Sample Remote Clients

ARSL Command Line Utility / 18

Simple GUI Remote Client / 20

Simple Message Handling Remote Client / 22

Message Handling Remote Client / 24

LabVIEW Simple Remote Client / 27

LabVIEW Demo Remote Client / 31

Connecting to the Automated Test Application / 33
Selecting/Running Tests / 34
Set/Get Configuration / 35
Obtaining Results / 36
Deleting Results / 37
Saving/Loading Projects / 37
ARSL / 38
Compliance Limit Set / 38

Application Sequencer / 39

3 Remote Programming Languages and Sample Code

The Keysight KS8400A Test Automation for PathWave (TAP) / 42

The Automated Test Remote Scripting Language (ARSL) / 44
- Example Code / 45
- Common Tasks / 46
- Example Program / 49
- Advanced Topic: Switch Matrix / 50

Microsoft .NET Framework / 52
- Example Code / 52
- Common Tasks / 53
- Example Programs / 56
- Advanced Topic: Event Handling / 58
- Advanced Topic: Switch Matrix / 58
- Advanced Topic: Measurement Server / 60
- Advanced Topic: Parallel Testing / 66

Python / 67
- Using the Python Visa Package / 67
- Using the Python for .NET Package / 67
- Example Code / 68
- Common Tasks / 70
- Example Programs / 75
- Advanced Topic: Event Handling / 78
- Advanced Topic: Switch Matrix / 82

4 Recommended Remote Programming Practices

Determine required tasks by first using the graphical user interface / 86

Verify the remote commands one at a time / 87

Check the log / 88

5 Developing a Remote Client

Plugin for Keysight KS8400A Test Automation for PathWave / 92

Simple Remote Client / 94
- File description / 96
- Source code description / 96

ARSL Command Line Utility Implementation / 98

Simple GUI Remote Client Implementation / 101

Simple Message Handling Remote Client Implementation / 105

Message Handling Remote Client Implementation / 108

LabVIEW Simple Remote Client / 116

LabVIEW Demo Client Construction / 128
Creating a Remote Client / 128
Registering an Event / 129
TestList Event / 129
SimpleMessage Event / 130
DataInput Event / 130
Loading a Project / 131
Discarding Unsaved Changes / 131
Saving a Project / 132
Setting a Config / 133
Overwriting an Existing Project / 133
Getting a Config / 133
Connection Prompt Action / 134
Signal Check Fail Action / 134
Running Tests / 134
Increasing Number of Trials / 135
Enabling Test Plan Feature / 136
Skipping Completed Permutations / 136
Execute ARSL / 136
Connect / 137
Appending/Replacing Test Results / 137
Getting Results / 137
Delete Results / 138
Refreshing the Test List / 139
Updating Your Automated Test Application / 139

6 Troubleshooting

Error Messages and Resolution / 142
How to Check Which Port an Application is Using / 145
How to Configure a Remote Interface User Port / 147
If your PC Has Two Network Interface Cards / 148
How To Test Two-way .NET Remoting Between Your PC and Your Scope / 149

Index

1 Keysight DigitalTestApps Remote Interface

Remote Interface Documentation / 10
Keysight KS8400A Test Automation for PathWave (TAP) Interface / 13
.NET Framework Interface / 14
LabVIEW Interface / 16

This chapter describes the fundamentals of the Keysight DigitalTestApps Remote Interface. If you haven't done this already, please read the *Getting Started* guide for important information needed to use this *Programming Guide* more effectively.

Whether you're using an existing remote client or developing your own, you need the following information to understand how to send instructions to the automated test application.

There are two basic types of commands: properties and methods.

- Properties represent data values in the automated test application. They can be read (query value) and in some cases written to (set value).
- Methods are functions which the automated test application can execute. They may require input parameters and/or provide return values.

In addition, some property actions and method calls use custom types defined by the automated test application. Custom types are classes containing the required data.

KEYSIGHT
TECHNOLOGIES

Remote Interface Documentation

Besides the Guide you are currently reading, there are other important documents which describe aspects of the Keysight DigitalTestApps Remote Interface:

In the Toolkit

The following document is located in the base folder of the Automated Test Application Remote Development Toolkit:

- Keysight_DigitalTestApps_Programming_Getting_Started.pdf

The following documents are located in the two "Documents" subfolders of the Automated Test Application Remote Development Toolkit:

- Keysight DigitalTestApps Remote Interface for TAP.chm.
- Keysight DigitalTestApps Remote Interface for .NET.chm.
- Keysight DigitalTestApps Remote Interface for LabVIEW.chm.

NOTE **Please see the readme.txt files located in the "Agilent-Keysight Transition" and "Keysight Apps Only" subfolders for guidance on which location to use.**

The .chm files describe each of the elements of the remote interface, including properties, methods, types, etc.

On the Server

The following application-specific document is located in the "help" folder of the automated test application's installation directory. For automated test apps that run directly on an oscilloscope, this folder may be found in either of these locations:

```
c:\Program Files\Agilent _or_ Keysight\(platform)\Apps\(application
           name)\help\(application name)_Remote_Prog_Ref.chm (or .pdf)
           where "(platform)" is FlexDCA, Infiniium, InfiniiVision, or
           M8070A
- Or -
(Infiniium real-time on WinXP only) c:\scope\apps\(application
           name)\help\(application name)_Remote_Prog_Ref.chm (or .pdf)
```

For automated test applications that run on a separate PC, this folder may be found in:

```
c:\Program Files\Agilent _or_ Keysight\(platform)\Apps\(application
           name)\help\(application name)_Remote_Prog_Ref.chm (or .pdf)
           where "(platform)" is FlexDCA, Infiniium, InfiniiVision, or
           M8070A
- Or -
(Infiniium real-time only)
           c:\Program Files\Keysight\Scope\Apps\(application
           name)\help\(application name)_Remote_Prog_Ref.chm (or .pdf)
```

This help file (.chm or .pdf) describes application-specific configuration variable names and values, test names and IDs, and external instrument names. These are used as parameters or returned as results by various elements of the Keysight

DigitalTestApps Remote Interface. The TAP.chm, .Net.chm, and LabVIEW.chm help files (described above) refer you to this application-specific document in their descriptions of remote interface elements that require application-specific values. You may view the remote interface help file(s) through the application's **Help > Remote Interface** menu (applications supporting remote interface version 2.00 or later, only).

NOTE

The TAP remote interface is a wrapper around the .NET Framework remote interface. The help topics in TAP.chm contain links to their associated .NET.chm topics, which often provide additional information about the feature. When using TAP.chm, keep in mind that the elements designed for TAP use are located in namespace Keysight.DigitalTestApps.TAP.Plugin.TestSteps, while those found in the Keysight.DigitalTestApps.Framework namespaces are for .NET Framework cross-reference use.

NOTE

The LabVIEW remote interface is a wrapper around the .NET Framework remote interface. The help topics in the LabVIEW.chm contain links to their associated .NET.chm topics, which often provide additional information about the feature. When using the LabVIEW.chm, keep in mind that the elements designed for LabVIEW use are located in namespace Keysight.DigitalTestApps.RemoteTestClient, while those found in the Keysight.DigitalTestApps.Framework namespaces are for .NET Framework cross-reference use.

In the User Interface

You may also obtain remote interface help directly from the application's user interface (applications supporting remote interface version 1.20 or later, only). Simply enable the "show remote interface hints" user preference (menu: **View->Preferences** :: **Remote** tab). If there is a remote command that can be used to accomplish an element's function, it can be found this way:

User Interface Element	Hint Location
Toolbar	Click a toolbar button.
Set Up tab **Run Tests** tab **Results** tab Various run-time dialogs	Right click a control and select **Remote Interface Hint**.
Select Tests tab	Select a test (by clicking on its name) and check the description pane at the bottom of the tab.
Configure tab	Select a configuration item and check the description pane on the right side of the tab.

TIP

To find out what version of the remote interface an application supports, check its **Help > About** screen (if it does not appear there, the version is older than 1.20).

Keysight KS8400A Test Automation for PathWave (TAP) Interface

The elements of the TAP API are contained in a plugin named DigitalTest Apps API. This plugin contains steps that wrap access to the properties and methods of the .NET Framework interface.

.NET Framework Interface

The elements of the .NET Framework interface are contained in a library named Keysight.DigitalTestApps.Framework.Remote.dll and are divided into three namespaces:

- "Keysight.DigitalTestApps.Framework.Remote" on page 14
- "Keysight.DigitalTestApps.Framework.Remote.Advanced" on page 14
- "Keysight.DigitalTestApps.Framework.Remote.Exceptions" on page 15

Keysight.DigitalTestApps.Framework.Remote

This namespace contains the core remote interface features. A good starting place to look at is the RemoteAteUtilities class, which contains the methods used to connect to the automated test application, and the IRemoteAte interface, where most of the commands you will use are found.

The features in this namespace involve the remote client initiating some action and the automated test application responding, making use of the "remote client -> automated test app path" of the remote interface.

For example, to execute a remote interface method:

1. The client computer initiates the action (the method call).
2. The .NET Remoting layer transports the request to the server (the application running on oscilloscope or PC).
3. The application executes the method and generates the return value.
4. The .NET Remoting layer transports the return value to the client.

Keysight.DigitalTestApps.Framework.Remote.Advanced

This namespace contains advanced programming features. The most important member is the AteEventSink, where most of the advanced commands you will use are found.

Some of the features in this namespace involve the automated test application initiating some action and the remote client responding, making use of the "automated test app -> remote client callback path" of the remote interface.

For example, one of the advanced features involves the ability to force dialogs that would normally display on the oscilloscope to display on the client instead. In this case:

1. The server (the application running on oscilloscope or PC) initiates the action (generates a message).
2. The .NET Remoting layer transports the message to the client computer.
3. The client displays the message and generates a response ("OK", "Cancel", etc.)

4 The .NET Remoting layer transports the response to the server.

Keysight.DigitalTestApps.Framework.Remote.Exceptions

This namespace contains remote interface-specific exception types that may be thrown during a remote operation.

LabVIEW Interface

The elements of the LabVIEW interface are contained in a library named Keysight.DigitalTestApps.RemoteTestClient.dll. This library contains methods that wrap access to the properties and methods of the .NET interface.

2 Using Sample Remote Clients

ARSL Command Line Utility / 18
Simple GUI Remote Client / 20
Message Handling Remote Client / 24
Simple Message Handling Remote Client / 22
LabVIEW Simple Remote Client / 27
LabVIEW Demo Remote Client / 31
Application Sequencer / 39

This chapter describes a set of ready-to-run sample remote clients that you can use to explore the Keysight DigitalTestApps Remote Interface.

KEYSIGHT TECHNOLOGIES

ARSL Command Line Utility

The easiest way to try out the remote interface is by using a simple command-line remote client. This client lets you use plain text to execute the commands found in the .NET Framework remote interface, which is described in Chapter 1, “Keysight DigitalTestApps Remote Interface,” starting on page 9.

To use this utility:

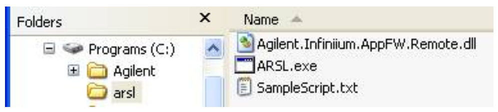
1 Using Windows Explorer, browse to the toolkit installation directory.

2 In the desired "Tools\C#" subdirectory, locate the "ARSL Command Line Utility" folder. Copy its contents to a new folder on your computer.

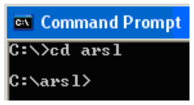
NOTE **Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.**

3 Copy the application file Keysight.DigitalTestApps.Framework.Remote.dll (or Agilent.Infiniium.AppFW.Remote.dll, see "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide) to this new folder as well.

4 The folder on your computer should now look like this:



5 Now open a Windows command prompt and browse to the folder you created:



```
C:\>cd arsl
C:\arsl>
```

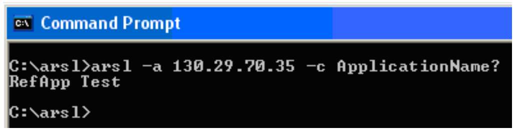
6 Ensure your configuration meets the requirements listed in "On the Server" in the *Keysight DigitalTestApps Programming Getting Started* guide, including launching the automated test application on the server.

7 Get the IP address of the target machine running the automated test application (oscilloscope or PC). On Windows-based machines, this may be obtained by typing "ipconfig" in a Command Prompt window.

8 Enter this in your Command Prompt window and press <enter>:

```
arsl -a 123.45.67.89 -c ApplicationName?
```

(substitute your actual IP address for 123.45.67.89)



Command Prompt

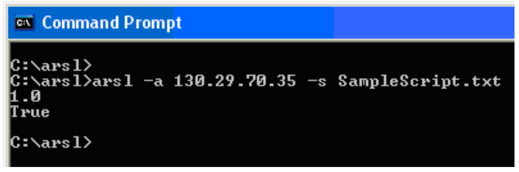
```
C:\arsl>arsl -a 130.29.70.35 -c ApplicationName?
RefApp Test

C:\arsl>
```

9 Observe the automated test application's response. In the example above, an application named Reference Application responded with "RefApp Test".

You may also use this utility to execute scripts. An example, "SampleScript.txt" is included with the executable. Execute the script by entering this:

```
arsl -a 123.45.67.89 -s SampleScript.txt
```

(substitute your actual IP address for 123.45.67.89)



Command Prompt

```
C:\arsl>
C:\arsl>arsl -a 130.29.70.35 -s SampleScript.txt
1.0
True

C:\arsl>
```

The command "ApplicationName?" and the contents of the file "SampleScript.txt" are examples of Keysight Automated Test Engine Remote Scripting Language (ARSL) commands. ARSL is described in more detail in "The Automated Test Remote Scripting Language (ARSL)" on page 44.

Note: To see all available options, enter this:

```
arsl -h
```

Simple GUI Remote Client

To see how a graphical user interface-based approach can work, try this client.

To use the simple GUI remote client:

1. Using Windows Explorer, browse to the toolkit installation directory.
2. In the desired "Tools\C#" subdirectory, locate the folder named "Simple GUI Remote Client". Copy its contents to a new folder on your computer.

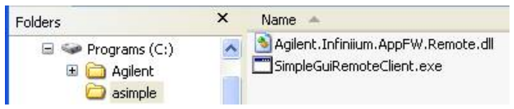
NOTE

Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.

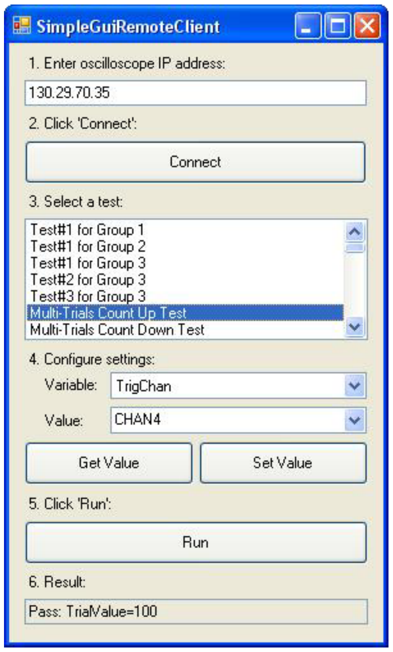
---

3. Copy the file Keysight.DigitalTestApps.Framework.Remote.dll (or Agilent.Infiniium.AppFW.Remote.dll, see "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide) to this new folder as well.

   The folder on your computer should now look like this:



4. Ensure your configuration meets the minimum requirements listed in "On the Server" in the *Keysight DigitalTestApps Programming Getting Started* guide.
5. Get the IP address of the target machine running the automated test application (oscilloscope or PC). On Windows-based machines, this may be obtained by typing "ipconfig" in a Command Prompt window.
6. Now execute the file SimpleGuiRemoteClient.exe. When the main dialog displays, follow the steps to run a test.



1. Enter oscilloscope IP address:

130.29.70.35

2. Click 'Connect':

3. Select a test:

Test#1 for Group 1
Test#1 for Group 2
Test#1 for Group 3
Test#2 for Group 3
Test#3 for Group 3
Multi-Trials Count Up Test
Multi-Trials Count Down Test

4. Configure settings:

Variable: TrigChan

Value: CHAN4

5. Click 'Run':

6. Result:

Pass: TrialValue=100

Simple Message Handling Remote Client

Now take a look at a client that demonstrates basic message handling capabilities of the remote interface.

To use the simple message handling remote client:

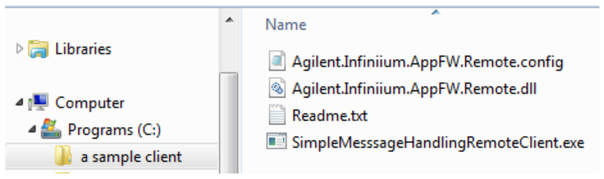
1 Using Windows Explorer, browse to the toolkit installation directory.

2 In the desired "Tools\C#" subdirectory, locate the folder named "Simple Message Handling Remote Client". Copy its contents to a new folder on your computer.

NOTE **Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.**

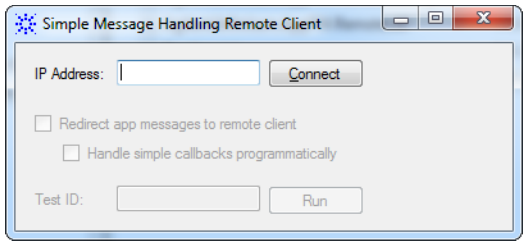
3 Copy the files Keysight.DigitalTestApps.Framework.Remote.dll and Keysight.DigitalTestApps.Framework.Remote.config (or Agilent.Infiniium.AppFW.Remote.dll and Agilent.Infiniium.AppFW.Remote.config, see "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide) to this new folder as well.

The folder on your computer should now look like this:


Libraries
Computer
Programs (C:)
a sample client
Name
Agilent.Infiniium.AppFW.Remote.config
Agilent.Infiniium.AppFW.Remote.dll
Readme.txt
SimpleMesssageHandlingRemoteClient.exe

4 Ensure your configuration meets the minimum requirements listed in "On the Server" in the *Keysight DigitalTestApps Programming Getting Started* guide.

5 Now execute the file SimpleMessageHandlingRemoteClient.exe:


Simple Message Handling Remote Client
IP Address:
Connect
Redirect app messages to remote client
Handle simple callbacks programmatically
Test ID:
Run

6 When the main dialog displays, enter the IP address of the target machine running the automated test application (oscilloscope or PC). On Windows-based machines, this may be obtained by typing "ipconfig" in a Command Prompt window.

7 Click "Connect".
8 Enter a test ID. See Chapter 1, "Remote Interface Documentation" on page 10, for more information on getting your application's test IDs.
9 Click "Run".

The rest of the dialog presents options for using the remote interface to manage messages that the remote application displays:

- **Redirect app messages to remote client**

  When checked, the automated test application will not display messages on the oscilloscope. Instead, messages will be displayed on the client PC using a default display algorithm provided by the Keysight.DigitalTestApps.Framework.Remote DLL.

- **Handle simple callbacks programmatically**

  When checked, the default algorithm used in the "Redirect" option above will be replaced by a custom algorithm defined in the Simple Message Handling Remote Client program for "simple" messages (those not requiring value entry).

NOTE

If you close and relaunch the automated test application, it will initialize with default settings, such as Redirect=false. In this case, you should close and relaunch the simple message handling client to reinitialize it as well.

Message Handling Remote Client

Now take a look at a client that demonstrates advanced message handling capabilities of the remote interface.

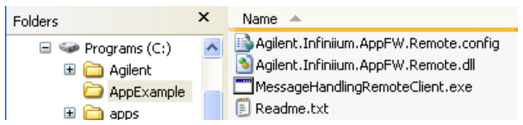
To use the message handling remote client:

1. Using Windows Explorer, browse to the toolkit installation directory.
2. In the desired "Tools\C#" subdirectory, locate the folder named "Message Handling Remote Client". Copy its contents to a new folder on your computer.

NOTE **Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.**

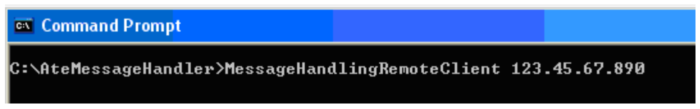
3. Copy the files Keysight.DigitalTestApps.Framework.Remote.dll and Keysight.DigitalTestApps.Framework.Remote.config (or Agilent.Infiniium.AppFW.Remote.dll and Agilent.Infiniium.AppFW.Remote.config, see "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide) to this new folder as well.

   The folder on your computer should now look like this:



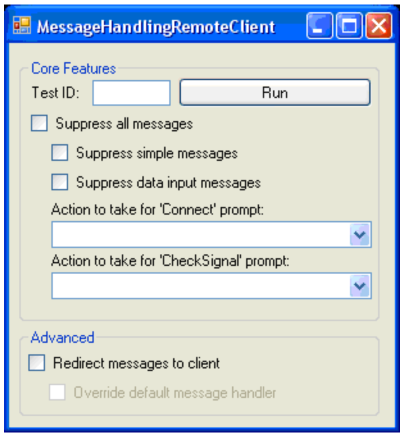
Agilent.Infiniium.AppFW.Remote.config
Agilent.Infiniium.AppFW.Remote.dll
MessageHandlingRemoteClient.exe
Readme.txt

4. Ensure your configuration meets the minimum requirements listed in "On the Server" in the *Keysight DigitalTestApps Programming Getting Started* guide.
5. Get the IP address of the target machine running the automated test application (oscilloscope or PC). On Windows-based machines, this may be obtained by typing "ipconfig" in a Command Prompt window.
6. Now execute the file MessageHandlingRemoteClient.exe, passing in the IP address of the remote application as the first parameter:



```
C:\AteMessageHandler>MessageHandlingRemoteClient 123.45.67.890
```

7. When the main dialog displays, enter a test ID. See Chapter 1, "Remote Interface Documentation" on page 10, for more information on getting your application's test IDs.


MessageHandlingRemoteClient
Core Features
Test ID:
Run
Suppress all messages
Suppress simple messages
Suppress data input messages
Action to take for 'Connect' prompt:
Action to take for 'CheckSignal' prompt:
Advanced
Redirect messages to client
Override default message handler

8 Click "Run".

The rest of the dialog presents options for using the remote interface to manage messages that the remote application displays:

The following options use core remote interface features:

- **Suppress simple messages**

  When checked, the remote application will not display application-specific "simple messages". These are defined as messages that use only the standard Microsoft button responses, for example, OK/Cancel, Yes/No, etc., and do not require the user to enter any values. When the application needs to display a simple message, it will instead automatically respond to it with a predetermined default button selection.

- **Suppress data input messages**

  When checked, the remote application will not display application-specific "data input" messages. These are defined as messages that require the user to enter a single alphanumeric value. When the application needs to display a data input message, it will instead attempt to automatically respond to it with a predetermined value. In cases where the automated test application has no automatic response, the test will abort.

- **Action to take for "Connect" prompt:**

  This combo box presents three options that affect what happens when the remote application needs to display the standard "Change Physical Connection/Setup" dialog:

  - **Abort** — When this option is selected, the remote application will throw an exception.
  - **AutoRespond** — When this option is selected, the remote application suppresses the dialog and assumes you have made the connection.
  - **Display** — When this option is selected, the remote application will display the dialog.

- **Action to take for "CheckSignal" prompt:**

  This combo box presents three options that affect what happens when the remote application needs to display the standard "Check Signal" dialog:

  - **Abort** — When this option is selected, the remote application will throw an exception.
  - **AutoRespond** — When this option is selected, the remote application suppresses the dialog and assumes the signal is acceptable.
  - **Display** — When this option is selected, the remote application will display the dialog.

The following options use advanced remote interface features:

- **Redirect messages to client**

  When checked, the remote application will not display any of the above message types on the oscilloscope. Instead, the message will be displayed on the client PC using a default display algorithm provided by the Keysight.DigitalTestApps.Framework.Remote DLL.

- **Override default message handler**

  When checked, the default algorithm used in the "Redirect" option above will be replaced by a custom algorithm defined in the Message Handling Remote Client program.

The Core Feature options override the Advanced options. Thus, the application will not redirect simple messages to the client if they are being suppressed.

NOTE

If you close and relaunch the automated test application, it will initialize with default settings, such as suppress=false. In this case, you should close and relaunch the message handling client to reinitialize it as well.

LabVIEW Simple Remote Client

Now take a look at a simple remote client that demonstrates the use of the LabVIEW programming environment to control an automated test application. See "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide for a list of minimum requirements.

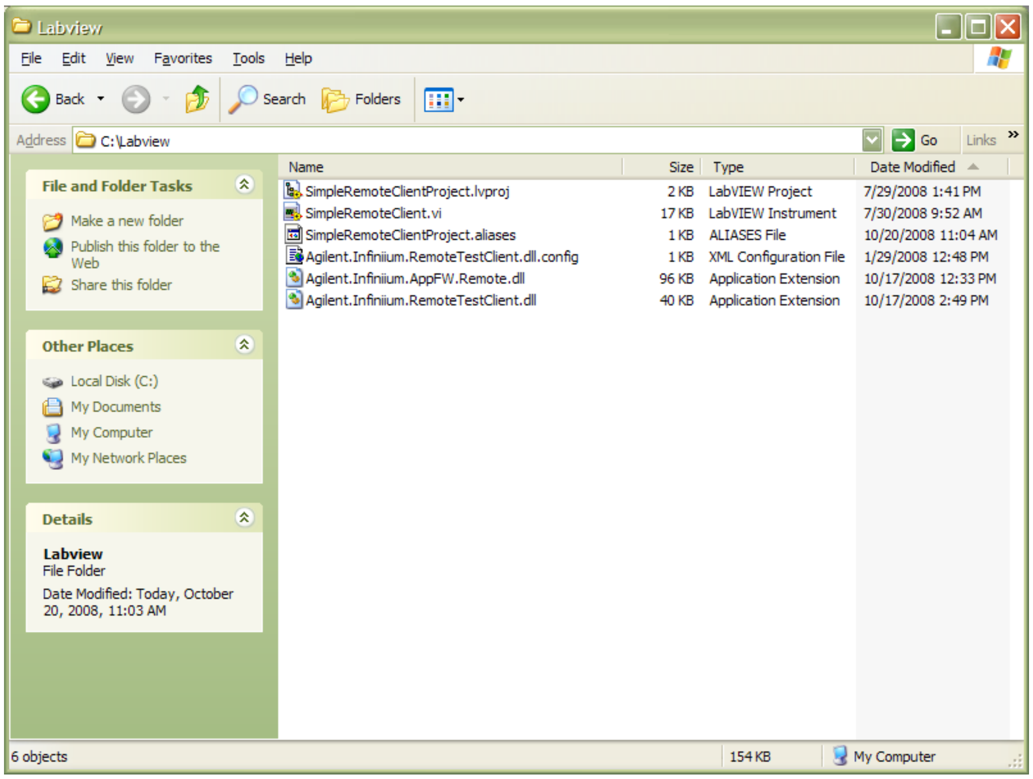
1 Using Windows Explorer, browse to the toolkit installation directory that corresponds to the remote interface version supported by the automated test application you wish to control.

2 In the "Agilent-Keysight Transition\Source" subdirectory, locate the "LabView\Simple Client" folder. Copy its contents to "C:\LabView" on your computer.

3 In the "Agilent-Keysight Transition\Tools" subdirectory, locate the "LabView Remote Adapter" folder. Copy its contents to "C:\LabView".

4 Copy the application file Agilent.Infiniium.AppFW.Remote.dll (see "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide) to "C:\LabView".

5 The folder on your computer should now look like this:



Labview
File Folder
Date Modified: Today, October 20, 2008, 11:03 AM

Name	Size	Type	Date Modified
SimpleRemoteClientProject.lvproj	2 KB	LabVIEW Project	7/29/2008 1:41 PM
SimpleRemoteClient.vi	17 KB	LabVIEW Instrument	7/30/2008 9:52 AM
SimpleRemoteClientProject.aliases	1 KB	ALIASES File	10/20/2008 11:04 AM
Agilent.Infiniium.RemoteTestClient.dll.config	1 KB	XML Configuration File	1/29/2008 12:48 PM
Agilent.Infiniium.AppFW.Remote.dll	96 KB	Application Extension	10/17/2008 12:33 PM
Agilent.Infiniium.RemoteTestClient.dll	40 KB	Application Extension	10/17/2008 2:49 PM

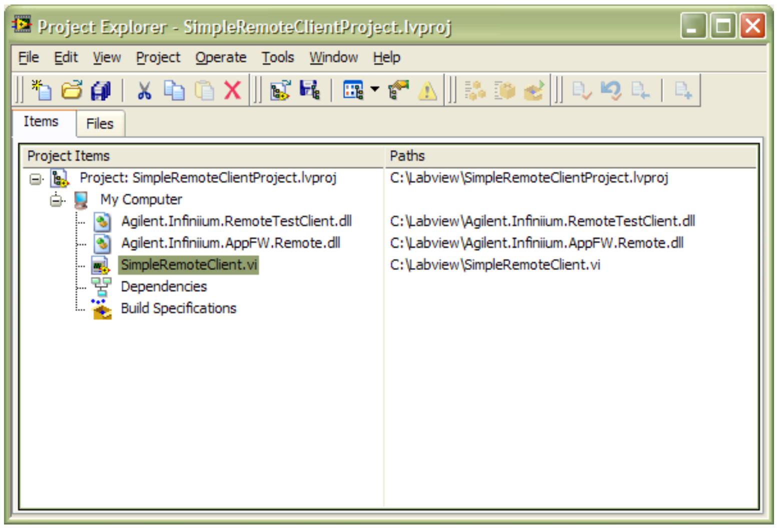
6 objects 154 KB My Computer

6 Ensure your configuration meets the minimum requirements listed in "On the Server" in the *Keysight DigitalTestApps Programming Getting Started* guide.

7 Get the IP address of the target machine running the automated test application (oscilloscope or PC). On Windows-based machines, this may be obtained by typing "ipconfig" in a Command Prompt window.

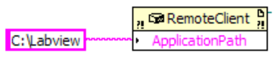
8 Open the file "SimpleRemoteClientProject.lvproj".

9 In the LabView Project Explorer, open the file "SimpleRemoteClient.vi".

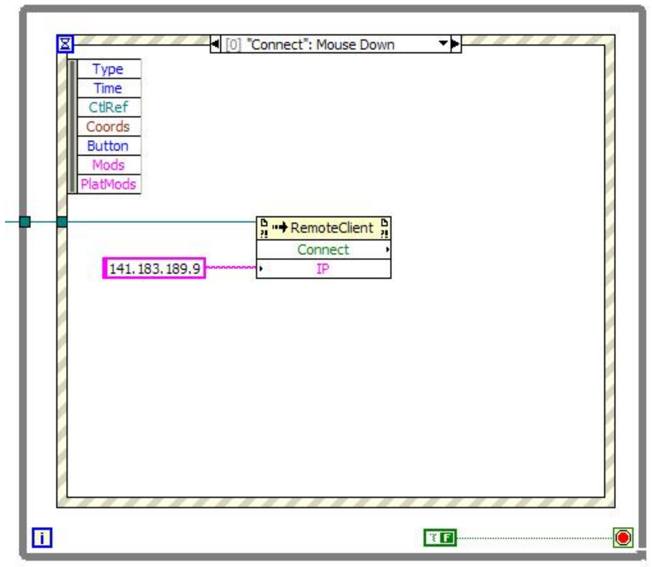

Project Explorer - SimpleRemoteClientProject.lvproj
File Edit View Project Operate Tools Window Help
Items Files

Project Items	Paths
Project: SimpleRemoteClientProject.lvproj	C:\Labview\SimpleRemoteClientProject.lvproj
My Computer	
Agilent.Infiniium.RemoteTestClient.dll	C:\Labview\Agilent.Infiniium.RemoteTestClient.dll
Agilent.Infiniium.AppFW.Remote.dll	C:\Labview\Agilent.Infiniium.AppFW.Remote.dll
SimpleRemoteClient.vi	C:\Labview\SimpleRemoteClient.vi
Dependencies	
Build Specifications	

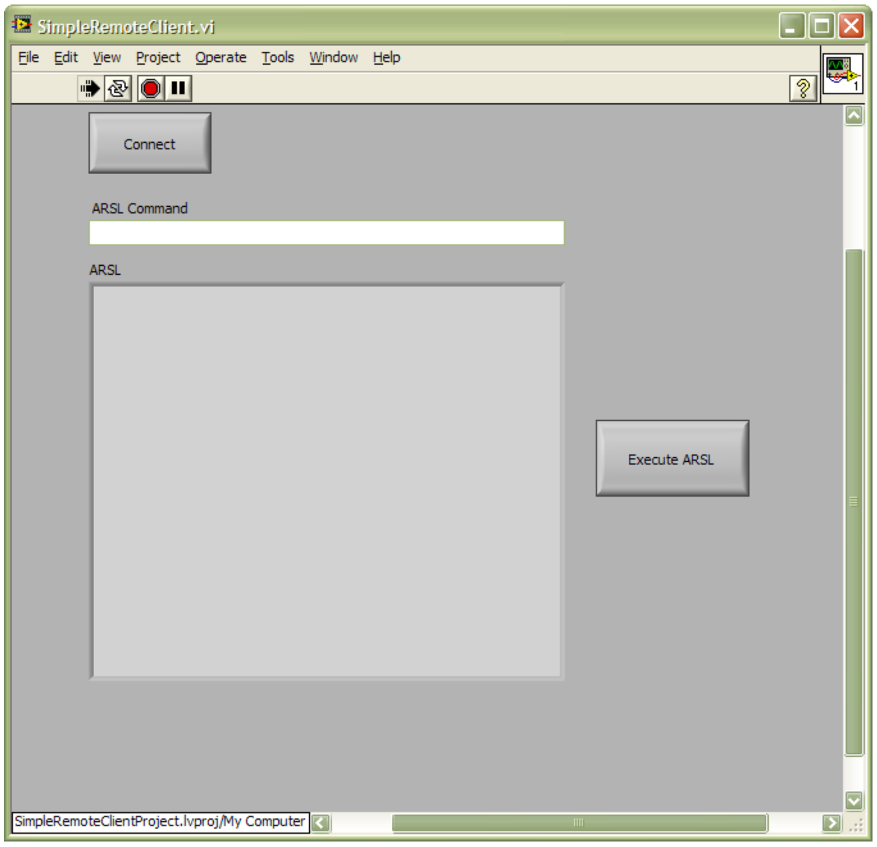
10 Before launching the example program, the path for the working directory needs to be changed. Change the path to point to the directory where both the Agilent.Infiniium.AppFW.Remote.dll and Agilent.Infiniium.RemoteTestClient.dll are located.



11 Change the IP to point to the scope where the automated test application is running.



12 Launch the program. You should see the following screen:


SimpleRemoteClient.vi
File Edit View Project Operate Tools Window Help
Connect
ARSL Command
ARSL
Execute ARSL
SimpleRemoteClientProject.lvproj/My Computer

13 Click Connect.

14. Enter the ARSL Command string.
15. Click Execute ARSL.

    Observe the indicator for the status/return value of the ARSL Command executed.

LabVIEW Demo Remote Client

Now take a look at a more advanced client that demonstrates the use of the LabVIEW programming environment to control an automated test application. See "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide for a list of minimum requirements.

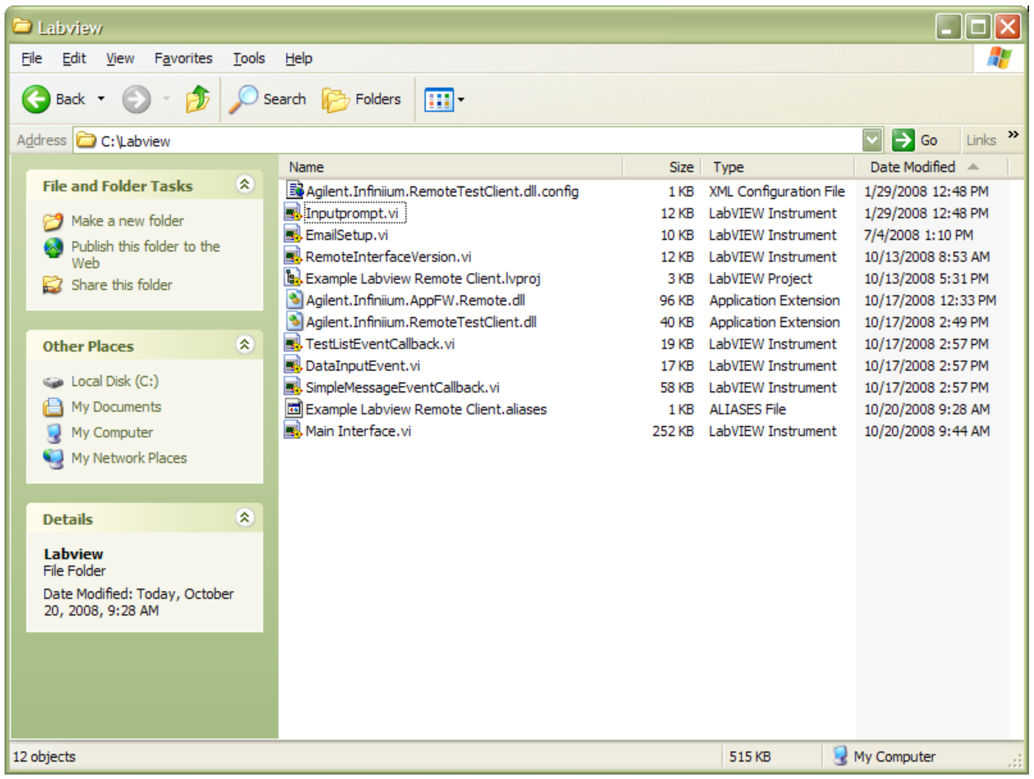
1 Using Windows Explorer, browse to the toolkit installation directory that corresponds to the remote interface version supported by the automated test application you wish to control.

2 In the "Agilent-Keysight Transition\Source" subdirectory, locate the "LabView\Demo Client" folder. Copy its contents to "c:\LabView" on your computer.

3 In the "Agilent-Keysight Transition\Tools" subdirectory, locate the "LabView Remote Adapter" folder. Copy its contents to "c:\LabView".

4 Copy the application file Agilent.Infiniium.AppFW.Remote.dll (see "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide) to "c:\LabView".

5 The folder on your computer should now look like this:



Labview
File Folder
Date Modified: Today, October 20, 2008, 9:28 AM

Name	Size	Type	Date Modified
Agilent.Infiniium.RemoteTestClient.dll.config	1 KB	XML Configuration File	1/29/2008 12:48 PM
Inputprompt.vi	12 KB	LabVIEW Instrument	1/29/2008 12:48 PM
EmailSetup.vi	10 KB	LabVIEW Instrument	7/4/2008 1:10 PM
RemoteInterfaceVersion.vi	12 KB	LabVIEW Instrument	10/13/2008 8:53 AM
Example Labview Remote Client.lvproj	3 KB	LabVIEW Project	10/13/2008 5:31 PM
Agilent.Infiniium.AppFW.Remote.dll	96 KB	Application Extension	10/17/2008 12:33 PM
Agilent.Infiniium.RemoteTestClient.dll	40 KB	Application Extension	10/17/2008 2:49 PM
TestListEventCallback.vi	19 KB	LabVIEW Instrument	10/17/2008 2:57 PM
DataInputEvent.vi	17 KB	LabVIEW Instrument	10/17/2008 2:57 PM
SimpleMessageEventCallback.vi	58 KB	LabVIEW Instrument	10/17/2008 2:57 PM
Example Labview Remote Client.aliases	1 KB	ALIASES File	10/20/2008 9:28 AM
Main Interface.vi	252 KB	LabVIEW Instrument	10/20/2008 9:44 AM

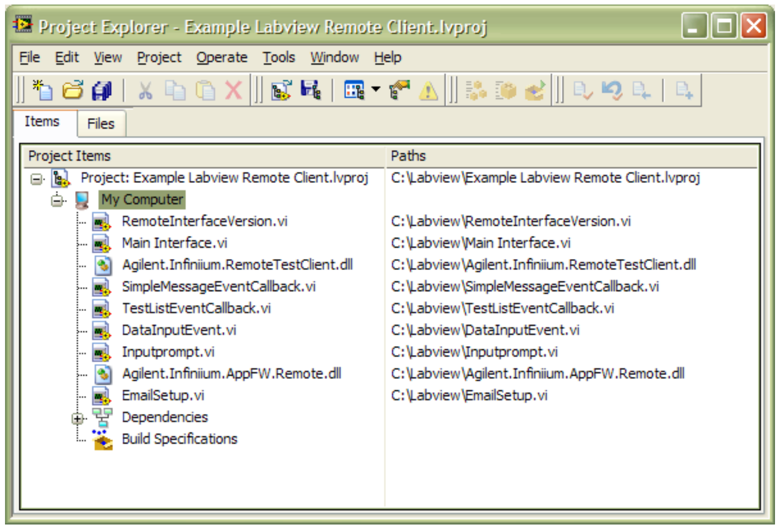
12 objects 515 KB My Computer

6 Ensure your configuration meets the minimum requirements listed in "On the Server" in the *Keysight DigitalTestApps Programming Getting Started* guide.

7 Get the IP address of your oscilloscope. This may be obtained by displaying the "GPIB Setup" dialog via the "Utilities" menu on the oscilloscope.

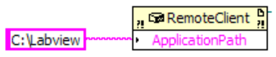
8 Open the file "Example Labview Remote Client.lvproj".

9 In the LabView Project Explorer, open the file "Main Interface.vi".

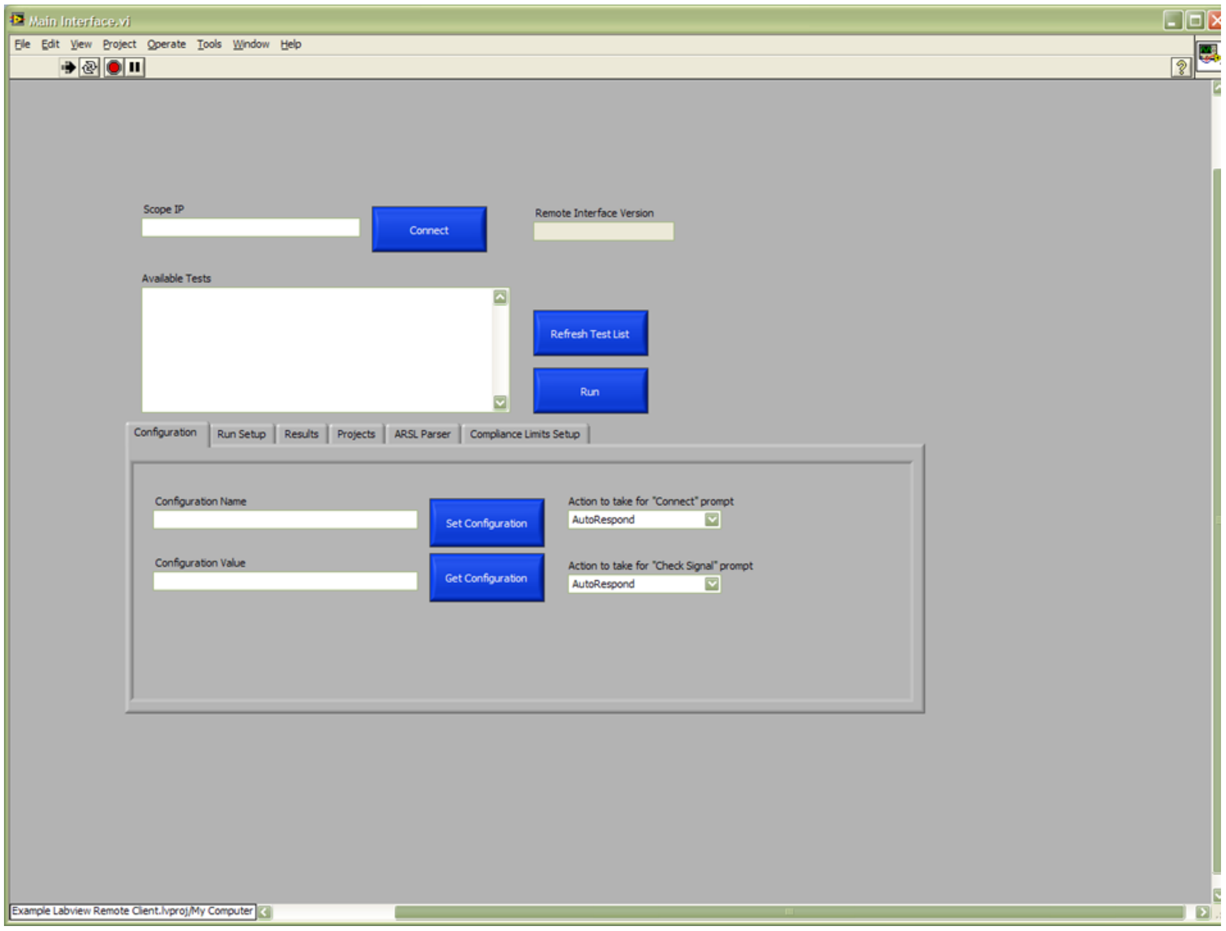

Project Explorer - Example Labview Remote Client.lvproj
File Edit View Project Operate Tools Window Help
Items Files

Project Items	Paths
Project: Example Labview Remote Client.lvproj	C:\Labview\Example Labview Remote Client.lvproj
My Computer	
RemoteInterfaceVersion.vi	C:\Labview\RemoteInterfaceVersion.vi
Main Interface.vi	C:\Labview\Main Interface.vi
Agilent.Infiniium.RemoteTestClient.dll	C:\Labview\Agilent.Infiniium.RemoteTestClient.dll
SimpleMessageEventCallback.vi	C:\Labview\SimpleMessageEventCallback.vi
TestListEventCallback.vi	C:\Labview\TestListEventCallback.vi
DataInputEvent.vi	C:\Labview\DataInputEvent.vi
Inputprompt.vi	C:\Labview\Inputprompt.vi
Agilent.Infiniium.AppFW.Remote.dll	C:\Labview\Agilent.Infiniium.AppFW.Remote.dll
EmailSetup.vi	C:\Labview\EmailSetup.vi
Dependencies	
Build Specifications	

10 Before launching the example program, the path for the working directory needs to be changed. Change the path to point to the directory where both the Agilent.Infiniium.AppFW.Remote.dll and Agilent.Infiniium.RemoteTestClient.dll are located.



11 Launch the program. You should see the following screen:

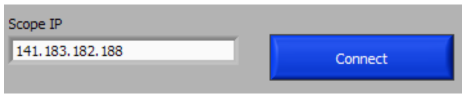


Action to take for "Connect" prompt: AutoRespond

Action to take for "Check Signal" prompt: AutoRespond

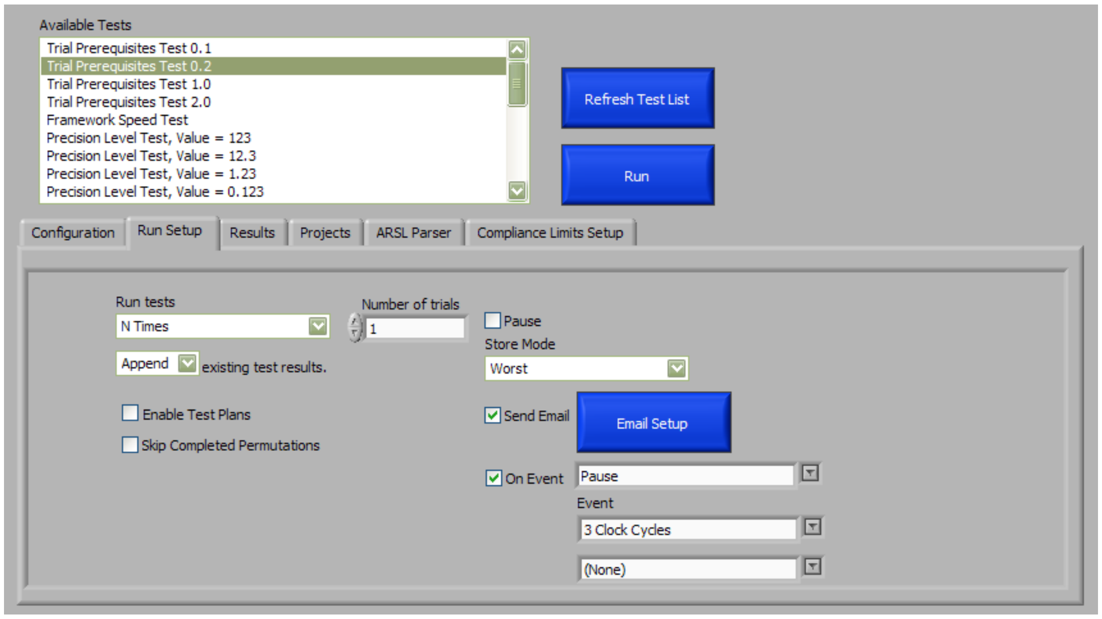
Connecting to the Automated Test Application

Enter the IP address of the oscilloscope where the Automated Test Application is running; then, click **Connect**.



Scope IP: 141.183.182.188

Selecting/Running Tests


Available Tests
- Trial Prerequisites Test 0.1
- Trial Prerequisites Test 0.2
- Trial Prerequisites Test 1.0
- Trial Prerequisites Test 2.0
- Framework Speed Test
- Precision Level Test, Value = 123
- Precision Level Test, Value = 12.3
- Precision Level Test, Value = 1.23
- Precision Level Test, Value = 0.123

Run tests: N Times
Number of trials: 1
Append existing test results.
Pause
Store Mode: Worst
Send Email
On Event: Pause
Event: 3 Clock Cycles
(None)

Select the tests to run from the list box; then, click **Run** to start the tests. If you select multiple tests, they will run one after another. The list of available tests depends on the current configuration setting.

The number of trials to run for the selected tests can be changed via the spinner control.

Click **Refresh Test List** to update the choice of tests available.

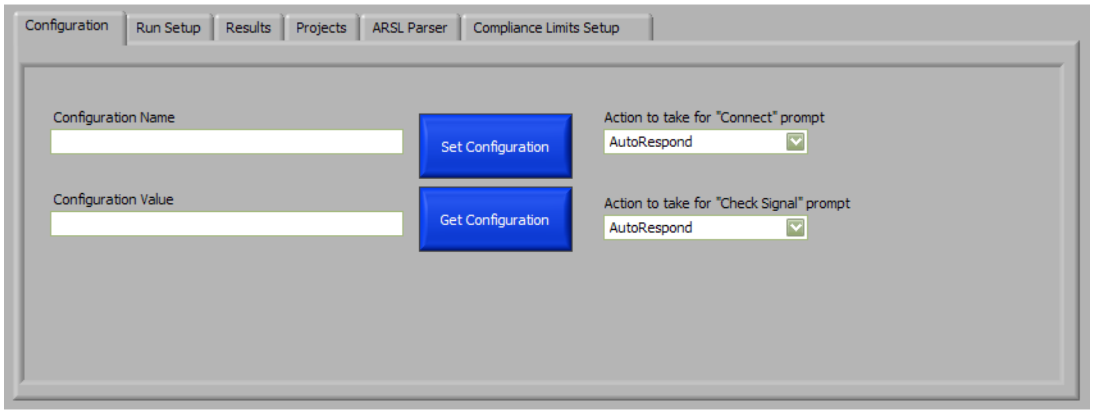
Test Plans When a remote application supports the Test Plan feature, check **Enable Test Plans** to run the tests with it. Check **Skip Completed Permutations** to skip all completed permutations of the Test Plan feature.

Results Select the preferred **Store Mode** for keeping track of the test results such as worst trials, best trials, last N trials, and events. Existing test results may be **Replaced** or have new results **Appended** to them. Refer to "Keysight DigitalTestApps Remote Interface for .NET.chm" for more details on the different store modes available.

Events In order for the application to react to a particular event of interest, check **On Event** and select the action to be taken when that event occurs. Select the **Event** of interest, there can be up to 2 events at a time. The choice of the 1st event would affect what is available as the 2nd choice. Note that certain events are only applicable to specific automated test applications.

E-mail Check **Send Email** to enable email sending when tests have completed execution or aborted.

Set/Get Configuration



Configuration Name

Configuration Value

Action to take for "Connect" prompt: AutoRespond

Action to take for "Check Signal" prompt: AutoRespond

To set an automated test application configuration value, enter the "Configuration Name" and "Configuration Value" in the appropriate text boxes and click **Set Configuration**.

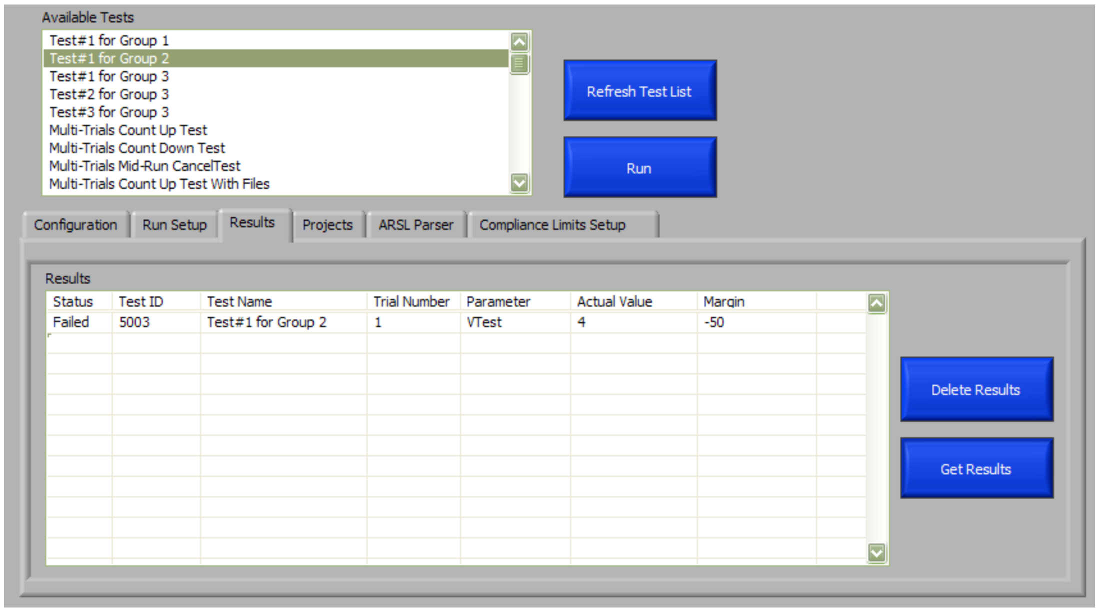
To get a configuration value from the automated test application, enter the Configuration Name in to the appropriate text box and click **Get Configuration**. The value for that variable will be displayed in the "Configuration Value" text box.

Refer to the individual automated test applications and framework guide for the available configurations.

Select the preferred action to be taken whenever the application requests for a change in the physical set up. Refer to "Keysight DigitalTestApps Remote Interface for .NET.chm" for more details on the individual actions.

Select the preferred action to be taken whenever the application checks the signal before execution of a test. Refer to "Keysight DigitalTestApps Remote Interface for .NET.chm" for more details on the individual actions.

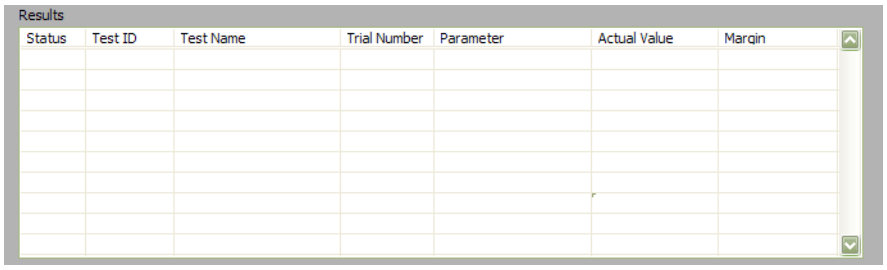
Obtaining Results



Results

Status	Test ID	Test Name	Trial Number	Parameter	Actual Value	Margin
Failed	5003	Test#1 for Group 2	1	VTest	4	-50

Select the tests from the list box, and click **Get Test Results** to view the results. The results will be displayed in the following table:



Results

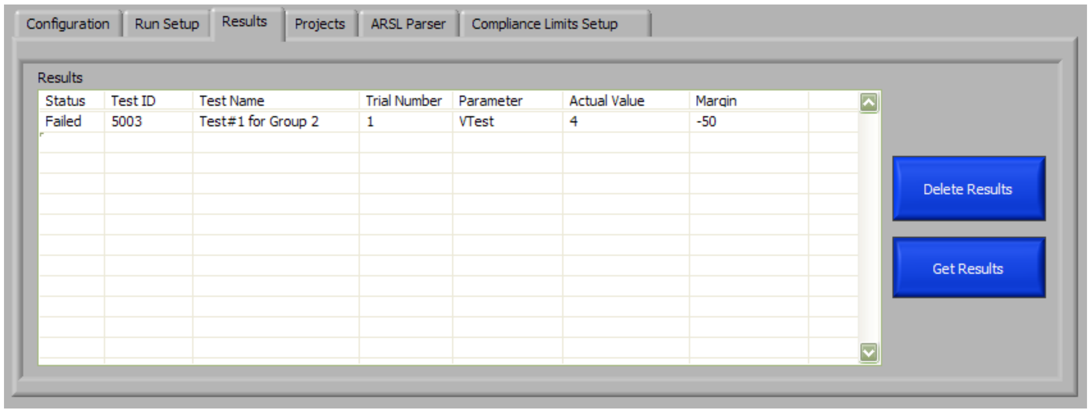
Status	Test ID	Test Name	Trial Number	Parameter	Actual Value	Margin

The following attributes are displayed for each selected test:

1. Status – Pass / Fail.
2. Test Name.
3. Test ID.
4. Trial Number.
5. Parameter that was measured.
6. Value of the measurement.
7. Passing margin.

If a test with no result is selected, the "Parameter Name" column displays "No Results".

Deleting Results

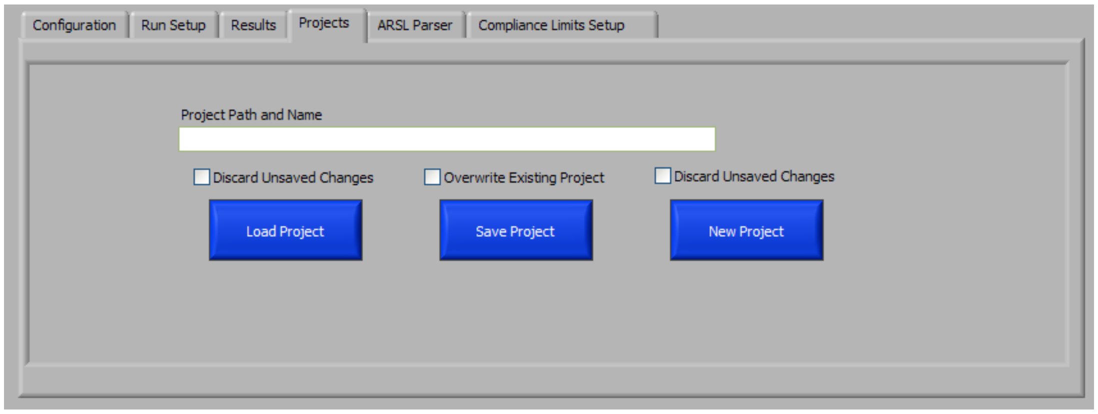


Results

Status	Test ID	Test Name	Trial Number	Parameter	Actual Value	Margin
Failed	5003	Test#1 for Group 2	1	VTest	4	-50

To delete all the results for all tests, click **Delete Results**.

Saving/Loading Projects

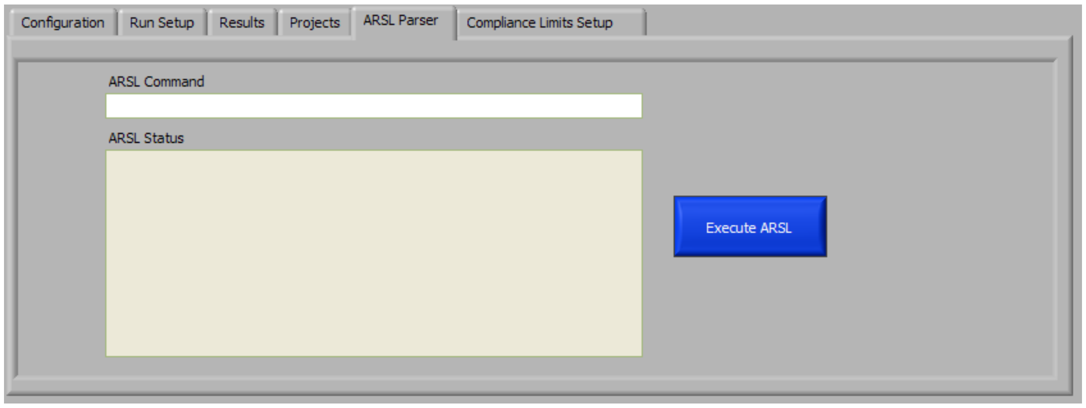


To save a project, enter the full path of the location on the oscilloscope, for example, C:\TestData\USB\GenericHubDevice; then, click **Save Project**. The project is called "GenericHubDevice" and saved at "C:\TestData\USB\GenericHubDevice". Note that this saves the project on the oscilloscope. Checking **Overwrite Existing Project** causes a project already saved with the same name to be overwritten.

To load the project saved in the example above, the path would be "C:\TestData\USB\GenericHubDevice\GenericHubDevice". Click **Load Project** to load a previously saved project. Checking **Discard Unsaved Changes** loads the selected project, ignoring all previously unsaved changes for the current project.

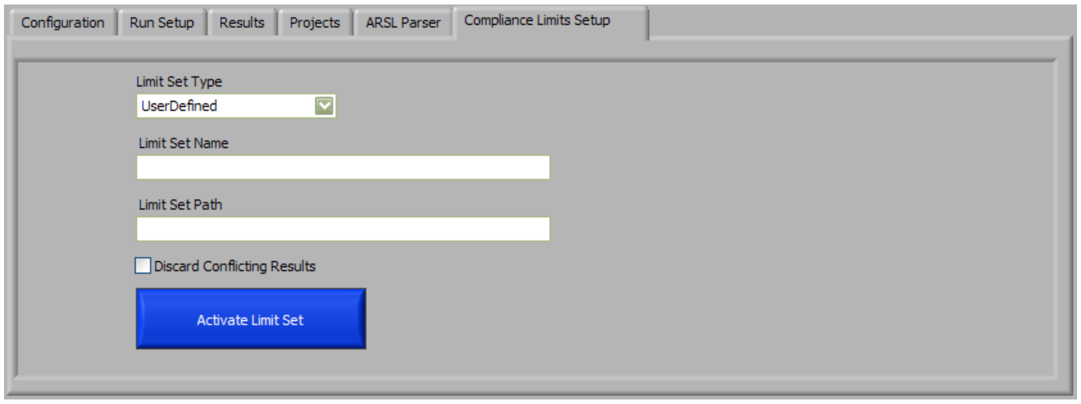
To create a new project, click **New Project**. Checking **Discard Unsaved Changes** creates a new project ignoring all the previously unsaved changes for the current project.

ARSL



Enter the ARSL command and click **Execute ARSL**. The results are displayed in the ARSL Status box.

Compliance Limit Set



Limit Set Type: UserDefined

Select the **Limit Set Type** of choice and provide the **Limit Set Name** and **Limit Set Path** (the path setting is only applicable for UserDefined limits). Check **Discard Conflicting Results** to remove all conflicting results when a limits set is activated.

Application Sequencer

The Application Sequencer lets you construct test plans consisting of one or more saved projects created by one or more Infiniium automated test applications. For example, if you are performing USB Type-C testing, the Application Sequencer will let you execute USB, Display Port, and Thunderbolt tests and view results all from one console.

See this folder for more information:

```
C:\ProgramData\Keysight\DigitalTestApps\Remote Toolkit\Version x
   \Tools\DigitalTest Application Sequencer\
```

3 Remote Programming Languages and Sample Code

The Keysight KS8400A Test Automation for PathWave (TAP) / 42
The Automated Test Remote Scripting Language (ARSL) / 44
Microsoft .NET Framework / 52
Python / 67

This chapter describes some of the languages you may use to program an automated test application, and provides code snippets to demonstrate language use.

The Keysight KS8400A Test Automation for PathWave (TAP)

The Keysight KS8400A Test Automation for PathWave (TAP) version 9 provides powerful, flexible and extensible test sequence and test plan creation with additional capabilities that optimize your test software development and overall performance. The N5452A Remote Toolkit includes a "DigitalTest Apps API" plugin for TAP that enables you to include control of automated test apps in your TAP test plans using a set of predefined steps enabling connections, configuration, test execution and results export.

To add this plugin to your TAP installation, use TAP to import the associated TapPackage:

1. In the TAP user interface, open **Menu: Tools > PackageManager > Settings** and add this N5452A toolkit path to the Package Repositories List:

   ```
   C:\ProgramData\Keysight\DigitalTestApps\Remote Toolkit\Version x\
   Tools\TAP\
   ```

   and click **OK**.
2. In the TAP PackageManager window, click the **Packages** tab, click on the "DigitalTest Apps API" entry in the list and click **Install**. Close the Package Manager.
3. In the TAP user interface, open **Menu: View > Panels > Steps**. The steps for controlling an automated test app will be located in the 'DigitalTest Apps API' tree.

See the help file, Keysight DigitalTestApps Remote Interface for TAP.chm, for a description of these steps.

The DigitalTest Apps API plugin includes two sample TAP test plans. After you install the plugin, the test plans will be located in:

```
C:\Program Files\Keysight\Test Automation\Packages\DigitalTest Apps API\
*.TapPlan
```

Open these samples using the TAP user interface; then configure the steps' settings.

Both of these sample test plans are designed to control a sample automated test application generated by the Keysight D9010UDAA User Defined Application generator (license required). Installers for the sample UDAs may be found in the remote toolkit installation:

```
C:\ProgramData\Keysight\DigitalTestApps\Remote Toolkit\Version x\Tools\
TAP\ SetupUda_Application_TAP Xxx Demo_v#.exe
```

Execute these installers either on the PC where TAP is running or on an oscilloscope. If using an Infiniium oscilloscope, restart the measurement software to install the app into its menu system. Now you are ready to execute the sample

TAP test plans and watch them launch and control a UDA. If you add an SQLite listener, then after running tests, you can view the test results & settings the test plans published using a sample templates installed by the plugin here:

```
C:\Program Files\Keysight\Test Automation\Packages\DigitalTest Apps API\
*.TapReport
```

The Automated Test Remote Scripting Language (ARSL)

Sometimes it is more convenient to access the automated test application remote interface using a textual syntax rather than the standard property/method call syntax. This is possible by using a command language called the Automated Test Engine (ATE) Remote Scripting Language, or "ARSL". The ARSL Command Line Utility (described in "ARSL Command Line Utility" on page 18) takes advantage of this capability to present an easy-to-use textual interface to an automated test application.

The ARSL interface is implemented by using a method provided by the standard .NET Framework remote interface: "ExecuteArsl()". This method takes one parameter: the textual remote interface command.

Here are some C# examples:

```
// Get ATE remote interface object
IRemoteAte remoteAte = RemoteAteUtilities.GetRemoteAte(ipAddress);

// Perform the property query "ApplicationName"
string appName = remoteAte.ExecuteArsl("ApplicationName?");

// Perform the method call "SetConfig(variable,value)"
remoteAte.ExecuteArsl("SetConfig Speed Fast");
```

The argument in the ExecuteArsl call is the ARSL command. Remote interface members may be translated into ARSL using the following conversion rule map:

Action	.NET Framework Syntax	ARSL Syntax
Property query	var = PropertyName;	PropertyName?
Property set	PropertyName = val;	PropertyName val
Method call	MethodName(arg1,arg2);	MethodName arg1 arg2

In ARSL, property set and method call arguments are separated by spaces. If the argument contains reserved characters (spaces, single quotes, or double quotes), you must surround the value accordingly:

Argument	Legal ARSL formats
Ab	Ab 'Ab' "Ab"
A b	'A b' "A b"

Argument	Legal ARSL formats
A'b	"A'b"
A"b	'A"b'

You can also use ARSL to set properties whose types are custom types and to call methods that take custom type parameters:

```
PropertyName "property1=value;property2=value"
MethodName "property1=value;property2=value"
```

This technique results in the following:

1. An object of the type required by the property or method is created
2. The object is modified by setting the its specified properties to the specified values
3. The modified object is then used to set the property or is passed to the method.

The following examples should make this more clear:

```
.NET Framework Syntax
------------------
SomeOptions options = new SomeOptions();
options.Option1 = "a";
options.Option2 = "b";
remoteAte.Foo(options);
```

```
ARSL Syntax
-------------------
Foo Option1=a;Option2=b
```

The ARSL parser will automatically detect that Foo requires a parameter of type SomeOptions and will then construct one. It will then set the two specified properties on the SomeOptions object and finally pass this object as a parameter to the Foo method.

Note that, in this syntax, multiple property sets are separated by a semicolon. In the example above, the option set statements did not contain any spaces, so the pair did not need to be enclosed in quotes.

In the examples below, the option set statements contain a space and thus need to be enclosed in quotes:

```
Foo 'Option3=A b'
Foo "Option3=A b"
Foo "Option1=A b;Option2=c"
```

Example Code

The following ARSL code snippets demonstrate property actions and method invocations:

```
# Property actions
# ----------------

# Read the remote interface version
RemoteInterfaceVersion?

# Prevent these dialogs from displaying during run
SuppressSimpleMessages true

# Method invocations
# ------------------

# Set one of the application's user-configurable options
SetConfig speed fast

# Save the current configuration
SaveProject myProjectName
```

In addition, some property actions and method calls use custom types defined by the automated test application. Custom types are classes containing the required data. The following code snippets are examples of this:

```
# Custom property types
# ---------------------

# Query
ExistingResultsAction?

# Set
ConnectionPromptAction AutoRespond

# Custom types used in methods
# ----------------------------
SaveProjectCustom "Name=MyProject;BaseDirectory=d:\
MyProjects;OverwriteExisting=true"

GetResults
```

Common Tasks

Here are common tasks you may want to perform in your programs. When there is more than one way to accomplish the task, alternatives are listed with a usage hint. For more information on each of the commands mentioned below, see the help file, "Keysight DigitalTestApps Remote Interface for .NET", located in the same directory as this programming guide.

NOTE

If a method demonstrated below does not enable you to customize the action as desired, check the command help file to see if there is a similarly named "Xxx**Custom**" method that provides more flexibility.

- "Launching the automated test app" on page 47
- "Suppressing message prompts" on page 47

- "Configure settings found on the application's 'Set Up' and 'Configure' tabs" on page 48
- "Select tests to be run" on page 48
- "Run Tests" on page 48
- "Get Results" on page 48
- "Save the project" on page 49
- "Start a new project" on page 49
- "Open a project" on page 49
- "Exiting the automated test app" on page 49

Launching the automated test app

- Manually. Use any time.

  Infiniium automated test apps: Use the oscilloscope menu: **Analyze > Automated Test Apps > (AppName)**

  FlexDCA automated test apps: Use the oscilloscope menu: **Apps > Automated Test Apps > (AppName)**

  Automated test apps: On your PC, launch the app using the Start menu or a shortcut.
- Using remote command. Send this SCPI command to Infiniium or FlexDCA:

  ```
  :SYSTEM:LAUNCH 'AppName'
  ```

  Where AppName is the name exactly as it appears in the menu system (see alternative above).

Suppressing message prompts

- Suppress "OK button" message prompts of type "Info", "Warning", or "Error". Use any time.

  ```
  SuppressSimpleMessages true
  ```
- Suppress "OK/Cancel button" messages that ask you to enter a value. Use only if the application has defined a default value for that prompt; otherwise test will be aborted.

  ```
  SuppressDataInputMessages true
  ```
- Suppress connection prompts. Use any time.

  ```
  ConnectionPromptAction AutoRespond
  ```
- Suppress "Signal Missing" prompts. Use any time.

  ```
  SignalCheckFailAction AutoRespond
  ```
- Suppress all of the above. See above for appropriate usage.

```
SuppressMessages true
```

Configure settings found on the application's 'Set Up' and 'Configure' tabs

```
SetConfig 'variableName1' 'value1'
```

Select tests to be run

- Select all of the tests currently shown on the application's 'Select Tests' tab. Use any time.

  ```
  SelectAllTests
  ```

- Select a subset of the available tests. Use any time.

  ```
  SelectedTests = 123,456,etc.
  ```

Run Tests

- With defaults. Use any time.

  ```
  Run
  ```

- With options. Use when you want to override the user preferences currently active in the target application.

  ```
  RunCustom 'StopRunIfTestAborts=true'
  ```

Get Results

- Quick overall result. Use any time.

  ```
  Passed?
  ```

- Get full details for all existing test results. Use any time.

  ```
  GetResults
  ```

- Get a subset of all available results. Use any time.

  ```
  GetResultsCustom 'TestIds=123,456;IncludeCsvData=true'
  ```

- CSV report. Use when you want CSV results in a separate file. File "results.csv" will be located inside the project.

  ```
  ExportResultsCsv
  SaveProject 'Project1'
  ```

- CSV text. Use when you want to see the CSV results on the remote client.

  ```
  GetResultsCustom 'IncludeCsvData=true'
  ```

- HTML Report.

  The .html file is located in the saved project.

Save the project

- Using default location property. Use when that location does not already have a project by the same name (although re-saves to the same location are allowed).

  ```
  SaveProject 'Project1'
  ```

- Using options. Use any time.

  ```
  SaveProjectCustom Name=Project1;BaseDirectory=c:\...\
  MyProjectDir;OverwriteExisting=true'
  ```

Start a new project

- Checking method. Use when you want to be warned when unsaved results exist.

  ```
  NewProject false
  ```

- Forced method. Use when you always want this command to complete (unsaved results will be discarded).

  ```
  NewProject true
  ```

Open a project

- Using default location. Use when you want to be warned when unsaved results exist.

  ```
  OpenProject 'Project1'
  ```

- Using options. Use any time.

  ```
  OpenProjectCustom 'FullPath=c:\...\MyProjectDir\
  Project1;DiscardUnsaved=true'
  ```

Exiting the automated test app

- Safer method. Use when you do not want to risk losing unsaved results (action will fail if unsaved results exist).

  ```
  Exit false true
  ```

- Forced method. Use when you always want this command to complete (unsaved results will be discarded).

  ```
  Exit true true
  ```

Example Program

```
SuppressMessages true
NewProject true
ExistingResultsAction Append
SelectedTests 123
Run
SaveProjectCustom 'Name=MyProjectName;OverwriteExisting=true'
```

NOTE

Replace 123 with actual test ID for your application.

See Chapter 1, "Remote Interface Documentation" on page 10, for more information on getting your application's test IDs.

Advanced Topic: Switch Matrix

The following ARSL code snippets demonstrate how to configure the Switch Matrix controller.

Using Automatic Mode

```
SwitchMatrixOn true
SwitchMatrixSelectModeAuto 'Keysight U3020A S26'
SwitchMatrixConnectToInstrument 1 'lan[123.456.7890]:inst0'
```

If additional drivers were created by the app during execution of `SwitchMatrixSelectModeAuto`, simply execute `SwitchMatrixConnectToInstrument` again for each driver ID and instrument address. Each driver must be connected to a separate switch matrix instrument.

NOTE

Replace 123.456.7890 with the full SICL address or VISA alias of your switch instrument.

Using Manual Mode

```
SwitchMatrixOn true
SwitchMatrixSelectModeManual
SwitchMatrixAddDriver 'Keysight U3020A S26'
SwitchMatrixConnectToInstrument 1 'lan[123.456.7890]:inst0'
```

To create an additional driver, simply add these lines:

```
SwitchMatrixAddDriver 'Keysight U3020A S26'
SwitchMatrixConnectToInstrument 2 'lan[123.456.7890]:inst0'
```

Each driver must be connected to a separate switch matrix instrument.

NOTE

Replace 123.456.7890 with the full SICL address or VISA alias of your switch instrument.

Now you need to manually define each signal path, each one starting from the test point on the Device Under Test, through the switch, and ending with the scope channel.

Here is an example of a single-ended signal going into scope channel 3:

```
SwitchMatrixDefineSignalPath 'CLK' 'Driver=1;Slot=1;Input=1' '3'
```

Here is an example of a differential signal going into a differential probe connected to scope channel 4:

```
SwitchMatrixDefineSignalPath 'Strobe+' 'Driver=1;Slot=1;Input=1' '4+'
SwitchMatrixDefineSignalPath 'Strobe-' 'Driver=1;Slot=2;Input=1' '4-'
```

Using Either Mode

All that remains is to assign a correction method to each signal path. This is optional but highly recommended.

Here is an example for one signal path that is using a single-ended probe:

```
SwitchMatrixSetPathPrecisionProbe 'CLK' '3' 'On=True;Mode=Probe;Calibrat
ion=MyCalName'
```

Here is an example for one signal path that is using a differential probe:

```
SwitchMatrixSetPathPrecisionProbe 'Strobe+' '4+' 'On=True;Mode=Probe;Cal
ibration=MyCalName'
```

NOTE **With differential probes, you do not separately assign correction to the negative half of the signal.**

Microsoft .NET Framework

When you need to access the full capabilities of the remote interface, use a .NET Framework programming language. This enables your programs to make use of conditional statements, looping, event subscriptions, return value analysis, and more.

Example Code

The following C# code snippets demonstrate property actions and method invocations:

```
// Property actions
// ----------------

// Read the remote interface version
string version = remoteApp.RemoteInterfaceVersion;

// Prevent these dialogs from displaying during run
remoteApp.SuppressSimpleMessages = true;

// Method invocations
// ------------------

// Set one of the applications user-configurable options
remoteApp.SetConfig("speed", "fast");

// Save the current configuration
string fullPath = remoteApp.SaveProject("myProject");

// Custom property types
// ---------------------

// Query
ExistingResultsAction action = remoteApp.ExistingResultsAction;

// Set
remoteApp.ConnectionPromptAction = CustomPromptAction.AutoRespond;

// Custom types used in methods.
SaveProjectOptions options = new SaveProjectOptions();
options.Name = "MyProject";
options.BaseDirectory = "d:\MyProjects";
options.OverwriteExisting = true;
string fullPath = remoteApp.SaveProjectCustom(options);

ResultContainer results = remoteApp.GetResults();
MeasurementResult worstResult = results.WorstResults[0];
```

Common Tasks

Here are common tasks you may want to perform in your programs. When there is more than one way to accomplish the task, alternatives are listed with a usage hint. For more information on each of the commands mentioned below, see the help file, "Keysight DigitalTestApps Remote Interface for .NET", located in the same directory as this programming guide.

NOTE

If a method demonstrated below does not enable you to customize the action as desired, check the command help file to see if there is a similarly named "Xxx Custom**" method that provides more flexibility.**

- "Launching the automated test app" on page 53
- "Suppressing message prompts" on page 54
- "Configure settings found on the application's 'Set Up' and 'Configure' tabs" on page 54
- "Select tests to be run" on page 54
- "Run Tests" on page 54
- "Get Results" on page 54
- "Save the project" on page 55
- "Start a new project" on page 55
- "Open a project" on page 56
- "Exiting the automated test app" on page 56

Launching the automated test app

- Manually. Use any time.

  Infiniium automated test apps: Use the oscilloscope menu: **Analyze > Automated Test Apps > (AppName)**

  FlexDCA automated test apps: Use the oscilloscope menu: **Apps > Automated Test Apps > (AppName)**

  Automated test apps: On your PC, launch the app using the Start menu or a shortcut.

- Using remote command. Send this SCPI command to Infiniium or FlexDCA:

  ```
  :SYSTEM:LAUNCH 'AppName'
  ```

  Where AppName is the name exactly as it appears in the menu system (see alternative above).

Suppressing message prompts

- Suppress "OK button" message prompts of type "Info", "Warning", or "Error". Use any time.

  ```
  remoteAte.SuppressSimpleMessages = true;
  ```

- Suppress "OK/Cancel button" messages that ask you to enter a value. Use only if the application has defined a default value for that prompt; otherwise test will be aborted.

  ```
  remoteAte.SuppressDataInputMessages = true;
  ```

- Suppress connection prompts. Use any time.

  ```
  remoteAte.ConnectionPromptAction = CustomPromptAction.AutoRespond;
  ```

- Suppress "Signal Missing" prompts. Use any time.

  ```
  remoteAte.SignalCheckFailAction = CustomPromptAction.AutoRespond;
  ```

- Suppress all of the above. See above for appropriate usage.

  ```
  remoteAte.SuppressMessages = true;
  ```

Configure settings found on the application's 'Set Up' and 'Configure' tabs

```
remoteAte.SetConfig("variableName1", "value1");
```

Select tests to be run

- Select all of the tests currently shown on the application's 'Select Tests' tab. Use any time.

  ```
  remoteAte.SelectAllTests();
  ```

- Select a subset of the available tests. Use any time.

  ```
  remoteAte.SelectedTests = new int[]{123,456,etc.};
  ```

Run Tests

- With defaults. Use any time.

  ```
  remoteAte.Run();
  ```

- With options. Use when you want to override the user preferences currently active in the target application.

  ```
  RunOptions options = new RunOptions();
  options.StopRunIfTestAborts = true;
  remoteAte.RunCustom(options);
  ```

Get Results

- Quick overall result. Use any time.

  ```
  bool passed = remoteAte.Passed;
  ```

- Get full details for all existing test results. Use any time.

  ```
  ResultContainer results = remoteAte.GetResults();
  ```

- Get a subset of all available results. Use any time.

  ```
  ResultOptions options = new ResultOptions();
  options.TestIds = new int[]{123,456};
  remoteAte.GetResultsCustom(options);
  ```

- CSV report. Use when you want CSV results in a separate file. File "results.csv" will be located inside the project.

  ```
  remoteAte.ExportResultsCsv();
  remoteAte.SaveProject("Project1");
  ```

- CSV text. Use when you want to see the CSV results on the remote client.

  ```
  ResultOptions options = new ResultOptions();
  options.TestIds = new int[]{123,456};
  options.IncludeCsvData = true;
  remoteAte.GetResultsCustom(options);
  ```

- HTML Report.

  The .html file is located in the saved project.

Save the project

- Using default location property. Use when that location does not already have a project by the same name (although re-saves to the same location are allowed). Otherwise, save will be aborted.

  ```
  remoteAte.SaveProject("Project1");
  ```

- Using options. Use any time.

  ```
  SaveProjectOptions options = new SaveProjectOptions();
  options.Name = "Project1";
  options.BaseDirectory = @"c:\...\MyProjectDir";
  options.OverwriteExisting = true;
  remoteAte.SaveProjectCustom (options);
  ```

Start a new project

- Checking method. Use when you want to be warned when unsaved results exist.

  ```
  remoteAte.NewProject(false);
  ```

- Forced method. Use when you always want this command to complete (unsaved results will be discarded).

  ```
  remoteAte.NewProject(true);
  ```

Open a project

- Using default location. Use when you want to be warned when unsaved results exist.

```
remoteAte.OpenProject("Project1");
```

- Using options. Use any time.

```
OpenProjectOptions options = new OpenProjectOptions();
options.FullPath = "c:\\...\\MyProjectDir\\Project1";
options.DiscardUnsaved = true;
remoteAte.OpenProjectCustom(options);
```

Exiting the automated test app

- Safer method. Use when you do not want to risk losing unsaved results (action will fail if unsaved results exist).

```
remoteAte.Exit(false, true);
```

- Forced method. Use when you always want this command to complete (unsaved results will be discarded).

```
remoteAte.Exit(true, true);
```

Example Programs

The following example:

1. Loads an existing project.
2. Executes multiple tests one at a time, giving you an opportunity to modify the device under test in between tests.
3. At the end of the run, saves the resulting project.
4. Exports the tests results to a .csv file for post processing.

```
using System;
using System.IO;
using Keysight.DigitalTestApps.Framework.Remote;

...
  IRemoteAte remoteAte =
      RemoteAteUtilities.GetRemoteAte("123.45.67.890");
  remoteAte.SuppressMessages = true;

  // TODO: Read/write share directory "c:\Automated Test Apps" on the
  // PC that will be running this automation program (directory name
  // is arbitrary).
  const string baseDirectory = "Automated Test Apps";

  // Tell the app to load the starting project from the PC this
  // automation program is running on.
  OpenProjectOptions openProjectOptions = new OpenProjectOptions();
  openProjectOptions.DiscardUnsaved = true;
  openProjectOptions.FullPath = "\\\\MyPCName\\" + baseDirectory +
      "\\DDR3\\Base Project\\Base Project.proj";
  remoteAte.OpenProjectCustom(openProjectOptions);

  // Run the first test
  remoteAte.SelectedTests = new int[] {123};
  remoteAte.Run();

  // Modify DUT/switch, etc. here

  // Run the second test
  // This unselects test 123.
  remoteAte.SelectedTests = new int[] {456};
  // This appends 456 results to the existing results for test 123.
  remoteAte.Run();

  // Tell the app to save the entire project to the PC this automation
  // program is running on.
  SaveProjectOptions saveProjectOptions = new SaveProjectOptions();
  saveProjectOptions.BaseDirectory = "\\\\MyPCName\\" + baseDirectory +
      "\\DDR3";
  saveProjectOptions.Name = "My DDR3 Device";
  saveProjectOptions.OverwriteExisting = true;
  remoteAte.SaveProjectCustom(saveProjectOptions);

  // Export CSV results.
  ResultOptions resultOptions = new ResultOptions();
  resultOptions.IncludeCsvData = true;
  ResultContainer resultContainer =
      remoteAte.GetResultsCustom(resultOptions);
  StreamWriter writer = null;

  try
  {
     // Because this file is being created by this automation program
     // which is running on the PC, use the local, not network, path.
     // For convenience sake, this example saves it to the directory
     // which contains the entire project; file name is arbitrary.
     writer = new StreamWriter("c:\\" + baseDirectory + "\\DDR3\\" +
         saveProjectOptions.Name + "\\AllResults.csv");
     writer.WriteLine(resultContainer.CsvResults);
  }
  finally
  {
    if (writer != null) writer.Close();
  }
...
```

NOTE Replace "123.45.67.890" with the actual IP address of the oscilloscope running your automated test application, and replace "123" with an actual test ID for the application.

See Chapter 1, "Remote Interface Documentation" on page 10, for more information on getting your application's test IDs.

Advanced Topic: Event Handling

The above examples demonstrate how to use the "forward facing" (remote client -> automated test application) direction of the remote interface. The remote interface also has a "callback" (automated test application -> remote client) direction, described in "Keysight.DigitalTestApps.Framework.Remote.Advanced" on page 14. A sample implementation using this capability for prompts can be found in: "Message Handling Remote Client Implementation" on page 108.

There are other events besides message prompts that a remote client can subscribe to. These are listed in the Keysight DigitalTestApps Remote Interface for .NET help file in the AteEventSink Class topic.

Here is an example in which the remote client wants to be notified every time a test is about to start executing:

```
// Enable forward remote interface (remote client -> automated test app)
mRemoteAte = RemoteAteUtilities.GetRemoteAte(ipAddress);

// Enable asynchronous callbacks (automated test app -> remote client)
// User must place the .config file in the same location as the remote c
lient's .exe
string path = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Loca
tion);
string configFullPath = Path.Combine(path, "Keysight.DigitalTestApps.Fra
mework.Remote.config");
RemotingConfiguration.Configure(configFullPath, false);
mAteEventSink = RemoteAteUtilities.CreateAteEventSink(mRemoteAte, this,
ipAddress);

// Subscribe to the TestStarting event
mAteEventSink.TestStartingEvent += TestStartingEventHandler;

// Define the event handler method
private void TestStartingEventHandler(object sender, TestStartingEventAr
gs e)
{
   int idOfTestStartingNow = e.TestID;
}
```

NOTE

If the event you subscribe to fires inside of a run, then when you call the Run() method do so in a worker thread to ensure your main thread is available to receive the callback message from the firing event.

Advanced Topic: Switch Matrix

The following C# code snippets demonstrate how to configure the Switch Matrix controller.

Using Automatic Mode

```
remoteAte.SwitchMatrixOn = true;
remoteAte.SwitchMatrixSelectModeAuto("Keysight U3020A S26");
remoteAte.SwitchMatrixConnectToInstrument(1, "lan[123.456.7890]:inst0");
```

If additional drivers were created by the app during execution of `SwitchMatrixSelectModeAuto`, simply execute `SwitchMatrixConnectToInstrument` again for each driver ID and instrument address. Each driver must be connected to a separate switch matrix instrument.

NOTE

Replace 123.456.7890 with the full SICL address or VISA alias of your switch instrument.

Using Manual Mode

```
remoteAte.SwitchMatrixOn = true;
remoteAte.SwitchMatrixSelectModeManual();
remoteAte.SwitchMatrixAddDriver("Keysight U3020A S26");
remoteAte.SwitchMatrixConnectToInstrument(1,"lan[123.456.7890]:inst0");
```

To create an additional driver, simply add these lines:

```
remoteAte.SwitchMatrixAddDriver("Keysight U3020A S26");
remoteAte.SwitchMatrixConnectToInstrument(2,"lan[123.456.7890]:inst0");
```

Each driver must be connected to a separate switch matrix instrument.

NOTE

Replace 123.456.7890 with the full SICL address or VISA alias of your switch instrument.

Now you need to manually define each signal path, each one starting from the test point on the Device Under Test, through the switch, and ending with the scope channel.

Here is an example of a single-ended signal going into scope channel 3:

```
SwitchInfo info = new SwitchInfo();
info.Driver = 1;
info.Slot = 1;
info.Input = 1;
remoteAte.SwitchMatrixDefineSignalPath("CLK",info,"3");
```

Here is an example of a differential signal going into a differential probe connected to scope channel 4:

```
SwitchInfo info = new SwitchInfo();
info.Driver = 1;
info.Slot = 1;
info.Input = 1;
remoteAte.SwitchMatrixDefineSignalPath("Strobe+",info,"4+");
info = new SwitchInfo();
```

```
info.Driver = 1;
info.Slot = 2;
info.Input = 1;
remoteAte.SwitchMatrixDefineSignalPath("Strobe-",info,"4-");
```

Using Either Mode

All that remains is to assign a correction method to each signal path. This is optional but highly recommended.

Here is an example for one signal path that is using a single-ended probe:

```
PrecisionProbeOptions options = new PrecisionProbeOptions();
options.On = true;
options.Mode = PrecisionProbeOptions.PrecisionProbeMode.Probe;
options.Calibration = "MyCalName";
remoteAte.SwitchMatrixSetPathPrecisionProbe("CLK", "3", options);
```

Here is an example for one signal path that is using a differential probe:

```
PrecisionProbeOptions options = new PrecisionProbeOptions();
options.On = true;
options.Mode = PrecisionProbeOptions.PrecisionProbeMode.Probe;
options.Calibration = "MyCalName";
remoteAte.SwitchMatrixSetPathPrecisionProbe("Strobe+","4+",options);
```

NOTE **With differential probes, you do not separately assign correction to the negative half of the signal.**

Advanced Topic: Measurement Server

The following C# sample programs demonstrate how to automate a compliance app operating in Measurement Server mode. Be sure to first install an instance of the Keysight KS8108 Resource Arbiter on a visible machine. Configure one Resource Arbiter node with "Acquire" capability, one or mode nodes with "Measure" capability, and (optionally) one node with "Report" capability.

If you choose to not declare a node to handle the Report task, the Manager App will automatically take on that responsibility. This configuration is known as "synchronous results access" and the Manager app is the only app your remote program will need to communicate with. In this configuration, when you execute the Run method, the Manager will not return control to your remote program until all tests have completed and all result are available. Therefore, you will be able to query for results immediately following the call to Run.

If you choose to dedicate a node to handle the Report task, you will be operating in a configuration known as "asynchronous results access", and your remote program will need to communicate with both the Manager and Reporter Apps. In this configuration, when you execute the Run method on the Manager, it will return control to your remote program as soon as it has sent the last task to the Worker Farm. At that time, it is likely that results will not yet be available for one or more tests. Therefore, you will need to subscribe to an asynchronous event on the Reporter to be notified when results are available so you can query that app for them.

Synchronous Results Example

```
private static IRemoteAte ManagerApp;
private static AteEventSink AteEventSinkManagerApp;

private static void Main(string[] args)
{
   // TODO: Launch Manager App.

   string managerIpAddress = args[0];

   try
   {
      // Establish outbound connection to Manager App.
      ManagerApp = RemoteAteUtilities.GetRemoteAteCustom(new GetRemoteAt
eOptions
      {
         IpAddress = managerIpAddress,
         TimeoutInSeconds = 30,
         CloseStartupMessageBoxes = true
      });

      // (Optional) Establish callback connection from Manager App if
      // you want to programmatically handle prompts.
      // For more information, see Advanced Topic: Event Handling
      // (above).
      string path = Path.GetDirectoryName(Assembly.GetExecutingAssembly(
).Location);
      string configFullPath = Path.Combine(path, "Keysight.DigitalTestAp
ps.Framework.Remote.config");
      RemotingConfiguration.Configure(configFullPath, false);
      AteEventSinkManagerApp = RemoteAteUtilities.CreateAteEventSink(Man
agerApp, null, managerIpAddress);
      AteEventSinkManagerApp.SimpleMessageEvent += MySimpleMessageEventH
andler;

      // Caution: Do not establish outbound or callback connections
      // to Measurer Apps.

      // Suppress prompts you don't want to programmatically handle.
      ManagerApp.ConnectionPromptAction = CustomPromptAction.AutoRespond
;

      // Tell Manager app how to locate Measurer machines in the
      // Worker Farm.
      ManagerApp.ConnectAppToResourceArbiter(@"http://123.45.67.890:5544
1");

      // Loop: For each testing scenario
         try
         {
            ManagerApp.NewProject(true);
            ManagerApp.SelectedTests = new[] {MyTestIds};
            ManagerApp.SetConfig("VariableName", "Value");
            ManagerApp.RunCustom(new RunOptions {RunId = "MyDeviceID"});
            // Note: If you don't set the Run ID, Manager will
            // automatically generate a unique one.

            // For each test, here is what happens during this run:
            // First, Manager calls back into your program with device
            // control or other messages (as needed, if registered).
            // Next, Manager captures waveforms and sends them to a
            // Measure Worker in the Farm.
            // Next, Measure Worker sends test results back to Manager.

            // When all test results for this run have been received,
            // this program continues to the next statement.
         }
         catch (Exception exc)
         {
            // Log or prompt error.
         }

         ResultContainer rc = ManagerApp.GetResults();
         string runId = rc.RunId;
         bool aborted = ManagerApp.MeasServerDidAbortOccur(runId);
         int[] abortedIds = ManagerApp.MeasServerGetAbortedTestIds(runId
);
         int[] abortedTestIds = ManagerApp.MeasServerGetAbortedTestIds(r
unId);
      // Next testing scenario
   }
   catch (Exception exc)
   {
      // Log or prompt error.
   }
   finally
   {
      // Disconnect callback connection.
      AteEventSinkManagerApp?.Dispose();
   }
}

private static void MySimpleMessageEventHandler(object sender, MessageEv
entArgs e)
{
   if (e.ID == "Message ID 1")
   {
      // Configure Device or take other appropriate actions ...
      e.Response = DialogResult.OK;
   }
   // else if (e.ID == "Message ID 2") ...
}
```

Asynchronous Results Example

```
private static IRemoteAte ManagerApp;
private static IRemoteAte ReporterApp;
private static AteEventSink AteEventSinkManagerApp;
private static AteEventSink AteEventSinkReportApp;
private static int ReporterCallbacksPending;

private static void Main(string[] args)
{
   // TODO: Launch Manager App and Reporter App.

   string managerIpAddress = args[0];
   string reporterIpAddress = args[1];

   try
   {
      // Establish outbound connection to Manager App.
      ManagerApp = RemoteAteUtilities.GetRemoteAteCustom(new GetRemoteAt
eOptions
      {
         IpAddress = managerIpAddress,
         TimeoutInSeconds = 30,
         CloseStartupMessageBoxes = true
      });

      // Establish outbound connection to Reporter App.
      ReporterApp = RemoteAteUtilities.GetRemoteAteCustom(new GetRemoteA
teOptions
      {
         IpAddress = reporterIpAddress,
         TimeoutInSeconds = 30,
         CloseStartupMessageBoxes = true
      });

      string path = Path.GetDirectoryName(Assembly.GetExecutingAssembly(
).Location);
      string configFullPath = Path.Combine(path, "Keysight.DigitalTestAp
ps.Framework.Remote.config");
      RemotingConfiguration.Configure(configFullPath, false);

      // (Optional) Establish callback connection from Manager App if
      // you want to programmatically handle prompts.
      // For more information, see Advanced Topic: Event Handling
      // (above).
      AteEventSinkManagerApp = RemoteAteUtilities.CreateAteEventSink(Man
agerApp, null, managerIpAddress);
      AteEventSinkManagerApp.SimpleMessageEvent += MySimpleMessageHandle
r;

      // Establish callback connection from Reporter App.
      AteEventSinkReportApp = RemoteAteUtilities.CreateAteEventSink(Repo
rterApp, null, reporterIpAddress);
      AteEventSinkReportApp.MeasServerResultsAvailableEvent += MyResults
AvailableEventHandler;

      // Caution: Do not establish outbound or callback connections
      // to Measurer Apps.

      // Suppress prompts you don't want to programmatically handle.
      ManagerApp.ConnectionPromptAction = CustomPromptAction.AutoRespond
```

```
;

      // Tell Manager app how to locate Measurer machines in the
      // Worker Farm.
      ManagerApp.ConnectAppToResourceArbiter(@"http://123.45.67.890:5544
1");

      // Initialize a flag for determining when all results for all
      // runs have been received.
      ReporterCallbacksPending = 0;

      // Loop: For each testing scenario
         try
         {
            ManagerApp.NewProject(true);
            ManagerApp.SelectedTests = new[] {MyTestIds};
            ManagerApp.SetConfig("VariableName", "Value");
            ++ReporterCallbacksPending;
            ManagerApp.RunCustom(new RunOptions {RunId = "MyDeviceID"});
            // Note: Choose a descriptive run ID; don't use the default
            // generated by the Manager App (see below for reason).
            // Caution: Do not use the same ID for multiple runs unless
            // all results from previous uses have been received.

            // For each test, here is what happens during this run:
            // First, Manager calls back into your program with device
            // control or other messages (as needed, if registered).
            // Next, Manager captures waveforms and sends them to a
            // Measure Worker in the Farm.
            // Then this program continues to the next statement.

            // Meanwhile, the asynchronous Worker tasks in the Farm
            // continue where Measurer apps make measurements and send
            // results to Reporter.
            // When all results are available for this run, the Report
            // app fires a MeasServerResultsAvailable callback into
            // this program.
            // We don't have to wait for those results and can proceed
            // to start the next testing scenario.
         }
         catch (Exception exc)
         {
            // Log or prompt error.
         }
      // Next testing scenario

      // We're finished running all testing scenarios.
      while (ReporterCallbacksPending > 0)
      {
         // Keep the Reporter callback connection alive until the last
         // results callback arrives.
         Thread.Sleep(1000);
      }
   }
   catch (Exception exc)
   {
      // Log or prompt error.
   }
   finally
   {
      // Disconnect callback connections.
      AteEventSinkManagerApp?.Dispose();
      AteEventSinkReportApp?.Dispose();
   }
}

private static void MySimpleMessageHandler(object sender, MessageEventAr
gs e)
{
   if (e.ID == "Message ID 1")
   {
      // Configure Device or take other appropriate actions ...
      e.Response = DialogResult.OK;
   }
   // else if (e.ID == "Message ID 2") ...
}

private static void MyResultsAvailableEventHandler(object sender, MeasSe
rverResultsAvailableEventArgs e)
{
   // This method gets called by the Reporter every time it receives
   // the last test result for a run.
   // Note: Run results may arrive in a different order than the runs
   // were executed by the Manager, depending on the performance of
   // the Measurer Farm.
   // For example, when a run with a complex measurement is followed
   // by a run with a simple measurement, the 2nd run's results may be
   // ready first.
   // Choosing a descriptive run ID will enable you to know which run
   // the results are for.
   try
   {
      if (e.ErrorOccurred)
      {
         MessageBox.Show(e.ErrorMessage);
      }
      else
      {
         ResultContainer resultContainer = ReporterApp.GetResults();
         string runId = resultContainer.RunId;
         bool abortOccurred = ReporterApp.MeasServerDidAbortOccur(runId)
;
         int[] abortedTestIds = ReporterApp.MeasServerGetAbortedTestIds(
runId);
         AbortedTestInfo[] abortedTestInfo = ReporterApp.MeasServerGetAb
ortedTestInfos(runId);
      }
   }
   finally
   {
      --ReporterCallbacksPending;
   }
}
```

Advanced Topic: Parallel Testing

To accelerate testing of multiple devices, you may control multiple automated test applications (running on different scopes) in parallel using the multi-threading capabilities available in .NET Framework. For example:

```
public void DoMultiScopeTesting()
{
   Thread t1 = _StartTestingOnScope("<IP address 1>");
   Thread t2 = _StartTestingOnScope("<IP address 2>");
   t1.Join();
   t2.Join();
}

private Thread _StartTestingOnScope(string ipAddress)
{
   var t = new Thread(() => _Execute(ipAddress));
   t.Start();
   return t;
}

private static void _Execute(string ipAddress)
{
   IRemoteAte remoteAte = RemoteAteUtilities.GetRemoteAte(ipAddress);
   ...
}
```

This program will let both automated test applications run without waiting for each other. The calls to _StartTestingOnScope are not blocking, so the program starts both apps running and then proceeds to the first Join statement, where it waits for the first app to complete. Then, the program moves on to the second Join where it waits for the second app to complete (it's okay if it finished first).

NOTE **This solution requires .NET Framework 3.5 or newer on the remote client. Multi-threading is also possible with older .NET Framework versions using less robust techniques.**

Python

You can combine the ease of text-based programming with the depth of the .NET Framework API by choosing an interpreted language with .NET Framework compatibility, such as CPython. By installing the Pythonnet package, you gain access to the power of multi-paradigm programming: You can choose to write traditional "function call" code utilizing the .NET Framework API or you can choose to write SCPI-like "text-based" code which takes advantage of the ARSL syntax. Both of these are demonstrated below.

Furthermore, using the Python Visa package, you can even launch the automated test app from the same script that controls it.

Using the Python Visa Package

The Python VISA package, or PyVISA, is free open source software available from: http://pyvisa.sourceforge.net/. When you install this package you will be able to connect to an Keysight oscilloscope or FlexDCA and launch an automated test application using code as simple as:

```
import visa
infiniium = visa.instrument("<VISA>")
   # Where <VISA> is the VISA address of the oscilloscope, for example,
   # visa.instrument("TCPIP0::123.45.67.890::inst0:INSTR")
infiniium.write(":SYSTEM:LAUNCH '<application name>'")
   # Where <application name> is the exact text that appears in the
   # scope or FlexDCA applications menu, for example,
   # infiniium.write(":SYSTEM:LAUNCH 'N5393C PCIExpress Test App'")
```

Using the Python for .NET Package

The Python for .NET package is free open source software available from: https://pypi.org/project/pythonnet/. When you install this package, you will be able to connect to and control a launched Keysight automated test application using code as simple as:

```
import clr
clr AddReference("Keysight.DigitalTestApps.Framework.Remote")
from Keysight.DigitalTestApps.Framework.Remote import *
remoteObj = RemoteAteUtilities.GetRemoteAte("<ip>")
   # Where <ip> is the oscilloscope's IP address, for example,
   # RemoteAteUtilities.GetRemoteAte("123.45.67.890")
remoteApp = IRemoteAte(remoteObj)
remoteApp.<method or property>
   # Where <method or property> is any method or property found in
   # the application's .NET Framework interface, described in Chapter 1.
   # For example,
   # remoteApp.Run()
```

To make this work, you need to copy a few files to your Python installation directory.

If you are using Python 2.7.x:

1 First copy these files from the Keysight N5452A Remote Toolkit "Tools" folder to c:\Python27\DLLs:

```
Keysight.DigitalTestApps.Framework.Remote.dll
Keysight.DigitalTestApps.Framework.Remote.config
```

2 Next, download the 32-or 64-bit *cp27*.whl file from https://pypi.org/project/pythonnet/#files and execute this in a command window to install it:

```
pip install downloaded_file.whl
```

This will install these two files into C:\Python27\DLLs:

```
clr.pyd
Python.Runtime.dll
```

If you are using Python 3.7.x:

1 First copy these files from the Keysight N5452A Remote Toolkit "Tools" folder to C:\Users\<your user name>\AppData\Local\Programs\Python\Python37\DLLs:

```
Keysight.DigitalTestApps.Framework.Remote.dll
Keysight.DigitalTestApps.Framework.Remote.config
```

2 Next, execute this in a command window:

```
pip install Pythonnet
```

This will download and execute the latest *cp37*.whl to copy these two files into C:\Users\<your user name>\AppData\Local\Programs\Python\Python37\Lib\site-packages:

```
clr.pyd
Python.Runtime.dll
```

NOTE **Be sure to keep the clr.pyd and Python.Runtime.dll files together in the same directory.**

Example Code

The following Python code snippets demonstrate property actions and method invocations:

```
# Property actions
# ----------------
# Read the remote interface version
version = remoteApp.RemoteInterfaceVersion
# Prevent these dialogs from displaying during run
remoteApp.SuppressSimpleMessages = True
```

```
# Method invocations
# ------------------
# Set one of the applications user-configurable options
remoteApp.SetConfig("speed", "fast")
# Save the current configuration
fullPath = remoteApp.SaveProject("myProject")

# Custom property types
# ---------------------
# Query
action = remoteApp.ExistingResultsAction
# Set
remoteApp.ConnectionPromptAction = CustomPromptAction.AutoRespond
# Custom types used in methods.
options = SaveProjectOptions()
options.Name = "MyProject"
options.BaseDirectory = "d:\\MyProjects"
options.OverwriteExisting = True
fullPath = remoteApp.SaveProjectCustom(options)
results = remoteApp.GetResults()
extremeResult = results.ExtremeResults[0];
```

The following Python code snippets demonstrate the equivalent actions using SCPI-like ARSL-formatted strings:

```
# Property actions
# ----------------
# Read the remote interface version
version = remoteApp.ExecuteArsl("RemoteInterfaceVersion?")
remoteApp.ExecuteArsl("SuppressSimpleMessages True")

# Method invocations
# ------------------
# Set one of the applications user-configurable options
remoteApp.ExecuteArsl("SetConfig 'speed' 'fast'")
# Save the current configuration
fullPath = remoteApp.ExecuteArsl("SaveProject 'myProject'")

# Custom property types
# ---------------------
# Query
action = remoteApp.ExecuteArsl("ExistingResultsAction?")
# Set
remoteApp.ExecuteArsl("ConnectionPromptAction AutoRespond")
# Custom types used in methods.
fullPath = remoteApp.ExecuteArsl("SaveProjectCustom " + \
           "'Name=MyProject;BaseDirectory=" + \
           "d:\\MyProjects;OverwriteExisting=True'")
results = remoteApp.ExecuteArsl("GetResults")
```

Common Tasks

Here are common tasks you may want to perform in your programs. When there is more than one way to accomplish the task, alternatives are listed with a usage hint. Both method call and ARSL-formatted text paradigms are presented. For more information on each of the commands mentioned below, see the help file, "Keysight DigitalTestApps Remote Interface for .NET", located in the same directory as this programming guide.

NOTE

If a method demonstrated below does not enable you to customize the action as desired, check the command help file to see if there is a similarly named "Xxx**Custom**" method that provides more flexibility.

- "Launching the automated test app" on page 70
- "Suppressing message prompts" on page 71
- "Configure settings found on the application's 'Set Up' and 'Configure' tabs" on page 71
- "Select tests to be run" on page 72
- "Run Tests" on page 72
- "Get Results" on page 72
- "Save the project" on page 74
- "Start a new project" on page 74
- "Open a project" on page 74
- "Exiting the automated test app" on page 75

Launching the automated test app

- Manually. Use any time.

  Infiniium automated test apps: Use the oscilloscope menu: **Analyze > Automated Test Apps > (AppName)**

  FlexDCA automated test apps: Use the oscilloscope menu: **Apps > Automated Test Apps > (AppName)**

  Automated test apps: On your PC, launch the app using the Start menu or a shortcut.

- Using remote command. Send this SCPI command to Infiniium or FlexDCA:

  ```
  :SYSTEM:LAUNCH 'AppName'
  ```

  Where AppName is the name exactly as it appears in the menu system (see alternative above).

Suppressing message prompts

- Suppress "OK button" message prompts of type "Info", "Warning", or "Error". Use any time.

  Method call:

  ```
  remoteApp.SuppressSimpleMessages = True
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("SuppressSimpleMessages True")
  ```

- Suppress "OK/Cancel button" messages that ask you to enter a value. Use only if the application has defined a default value for that prompt; otherwise test will be aborted.

  Method call:

  ```
  remoteApp.SuppressDataInputMessages = True
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("SuppressDataInputMessages True")
  ```

- Suppress connection prompts. Use any time.

  Method call:

  ```
  remoteApp.ConnectionPromptAction = CustomPromptAction.AutoRespond
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("ConnectionPromptAction AutoRespond")
  ```

- Suppress "Signal Missing" prompts. Use any time.

  Method call:

  ```
  remoteApp.SignalCheckFailAction = CustomPromptAction.AutoRespond
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("SignalCheckFailAction AutoRespond")
  ```

- Suppress all of the above. See above for appropriate usage.

  Method call:

  ```
  remoteApp.SuppressMessages = True
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("SuppressMessages True")
  ```

Configure settings found on the application's 'Set Up' and 'Configure' tabs

Method call:

```
remoteApp.SetConfig("variableName1", "value1")
```

Text:

```
remoteApp.ExecuteArsl("SetConfig 'variableName1' 'value1'")
```

Select tests to be run

- Select all of the tests currently shown on the application's 'Select Tests' tab. Use any time.

  Method call:

  ```
  remoteApp.SelectAllTests()
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("SelectAllTests")
  ```

- Select a subset of the available tests. Use any time.

  Method call:

  ```
  remoteApp.SelectedTests = [123,456,etc.]
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("SelectedTests  123,456,etc.")
  ```

Run Tests

- With defaults. Use any time.

  Method call:

  ```
  remoteApp.Run()
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("Run")
  ```

- With options. Use when you want to override the user preferences currently active in the target application.

  Method call:

  ```
  runOptions = RunOptions()
  runOptions.StopRunIfTestAborts = True
  remoteApp.RunCustom(runOptions)
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("RunCustom 'StopRunIfTestAborts=True'")
  ```

Get Results

- Quick overall result. Use any time.

  Method call:

  ```
  passed = remoteAte.Passed
  ```

Text:

```
remoteApp.ExecuteArsl("Passed?")
```

- Get full details for all existing test results. Use any time.

  Method call:

  ```
  results = remoteAte.GetResults()
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("GetResults")
  ```

- Get a subset of all available results. Use any time.

  Method call:

  ```
  resultOptions = ResultOptions()
  resultOptions.TestIds = [123,456]
  remoteApp.GetResultsCustom(resultOptions)
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("GetResultsCustom 'TestIds=123,456'")
  ```

- CSV report. Use when you want CSV results in a separate file. File "results.csv" will be located inside the project.

  Method call:

  ```
  remoteApp.ExportResultsCsv()
  remoteApp.SaveProject("Project1")
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("ExportResultsCsv")
  remoteApp.ExecuteArsl("SaveProject 'Project1'")
  ```

- CSV text. Use when you want to see the CSV results on the remote client.

  Method call:

  ```
  resultOptions = ResultOptions()
  resultOptions.TestIds = [123,456]
  resultOptions.IncludeCsvData = True
  remoteApp.GetResultsCustom(resultOptions)
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("GetResultsCustom 'TestIds=123,456;IncludeCsvDa
  ta=True'")
  ```

- HTML Report.

  The .html file is located in the saved project.

Save the project

- Using default location property. Use when that location does not already have a project by the same name (although re-saves to the same location are allowed). Otherwise, save will be aborted.

  Method call:

  ```
  remoteApp.SaveProject("Project1")
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("SaveProject 'Project1'")
  ```

- Using options. Use any time.

  Method call:

  ```
  saveProjectOptions = SaveProjectOptions()
  saveProjectOptions.Name = "Project1"
  saveProjectOptions.BaseDirectory = "c:\\...\\MyProjectDir"
  saveProjectOptions.OverwriteExisting = True
  remoteApp.SaveProjectCustom(saveProjectOptions)
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("SaveProjectCustom 'Name=Project1;BaseDirectory
  =c:\\...\\MyProjectDir;OverwriteExisting=True'")
  ```

Start a new project

- Checking method. Use when you want to be warned when unsaved results exist.

  Method call:

  ```
  remoteApp.NewProject(False)
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("NewProject False")
  ```

- Forced method. Use when you always want this command to complete (unsaved results will be discarded).

  Method call:

  ```
  remoteApp.NewProject(True)
  ```

  Text:

  ```
  remoteApp.ExecuteArsl("NewProject True")
  ```

Open a project

- Using default location. Use when you want to be warned when unsaved results exist.

Method call:

```
remoteApp.OpenProject("Project1")
```

Text:

```
remoteApp.ExecuteArsl("OpenProject 'Project1'")
```

- Using options. Use any time.

  Method call:

```
openProjectOptions = OpenProjectOptions()
openProjectOptions.FullPath = "c:\\...\\MyProjectDir\\Project1"
openProjectOptions.DiscardUnsaved = True
remoteApp.OpenProjectCustom(openProjectOptions)
```

  Text:

```
remoteApp.ExecuteArsl("OpenProjectCustom 'FullPath=c:\\...\\
MyProjectDir\\Project1;DiscardUnsaved=True'")
```

Exiting the automated test app

- Safer method. Use when you do not want to risk losing unsaved results (action will fail if unsaved results exist).

  Method call:

```
remoteApp.Exit(False, True)
```

  Text:

```
remoteApp.ExecuteArsl("Exit False True")
```

- Forced method. Use when you always want this command to complete (unsaved results will be discarded).

  Method call:

```
remoteApp.Exit(True, True)
```

  Text:

```
remoteApp.ExecuteArsl("Exit True True")
```

Example Programs

The following example:

1. Launches the DDR3 automated test application.
2. Loads an existing project.
3. Executes multiple tests one at a time, giving you an opportunity to modify the device under test in between tests.
4. At the end of the run, saves the resulting project.
5. Exports the tests results to a .csv file for post processing.

The example is presented in both method call and text paradigms.

Using Method Calls

```
"""Import the compiled Python Visa module"""
import visa

"""Connect to the scope"""
remoteScope = visa.instrument("enter scope's VISA address here")

"""Launch the DDR3 application"""
remoteScope.write(":SYSTEM:LAUNCH 'DDR3 Test'")


"""Import the compiled Pythonnet module"""
import clr

"""Import the Keysight automated test app remote library DLL"""
clr.AddReference("Keysight.DigitalTestApps.Framework.Remote")
from Keysight.DigitalTestApps.Framework.Remote import *

"""Connect to the automated test application running on the scope
This will wait for the application to be fully launched and ready
before proceeding"""
scopeIpAddress = "123.45.67.890"
remoteObj = RemoteAteUtilities.GetRemoteAte(scopeIpAddress)
remoteApp = IRemoteAte(remoteObj)


"""Prevent dialogs from displaying during the run"""
remoteApp.SuppressMessages = True

"""Ensure app settings are in a known state"""
remoteApp.NewProject(True)

"""Select the tests to be run
Use commas to separate multiple IDs"""
remoteApp.SelectedTests = [50000]

"""Set a variable found on the app's 'Configure' tab
Get variable name and valid options from app's remote help file
or from GUI hints"""
remoteApp.SetConfig("TrigChan","CHAN4")

"""Start the run. This is a blocking call...
program will wait until app is finished."""
remoteApp.Run()

"""Save the project"""
saveOptions = SaveProjectOptions()
saveOptions.BaseDirectory = "c:\\temp"
saveOptions.Name = "Demo"
saveOptions.OverwriteExisting = True
projectFullPath = remoteApp.SaveProjectCustom(saveOptions)
print projectFullPath

"""This will print the extreme results to the console window
in which this script is running."""
results = remoteApp.GetResults()
```

```
"""This will get all results in csv format"""
resultOptions = ResultOptions()
resultOptions.TestIds = [123]
resultOptions.IncludeCsvData = True
customResults = remoteApp.GetResultsCustom(resultOptions)
print customResults.CsvResults
```

Using ARSL-Formatted Text

```
"""Import the compiled Python Visa module"""
import visa

"""Connect to the scope"""
remoteScope = visa.instrument("enter scope's VISA address here")

"""Launch the DDR3 application"""
remoteScope.write(":SYSTEM:LAUNCH 'DDR3 Test'")


"""Import the compiled Pythonnet module"""
import clr

"""Import the Keysight automated test app remote library DLL"""
clr.AddReference("Keysight.DigitalTestApps.Framework.Remote")
from Keysight.DigitalTestApps.Framework.Remote import *

"""Connect to the automated test application running on the scope
This will wait for the application to be fully launched and ready
before proceeding"""
scopeIpAddress = "123.45.67.890"
remoteObj = RemoteAteUtilities.GetRemoteAte(scopeIpAddress)
remoteApp = IRemoteAte(remoteObj)


"""Prevent dialogs from displaying during the run"""
remoteApp.ExecuteArsl("SuppressMessages True")

"""Ensure app settings are in a known state"""
remoteApp.ExecuteArsl("NewProject True")

"""Select the tests to be run
Use commas to separate multiple IDs"""
remoteApp.ExecuteArsl("SelectedTests 123")

"""Set a variable found on the app's 'Configure' tab
Get variable name and valid options from app's remote help file
or from GUI hints"""
remoteApp.ExecuteArsl("SetConfig 'TrigChan' 'CHAN4'")

"""Start the run. This is a blocking call...
program will wait until app is finished."""
remoteApp.ExecuteArsl("Run")

"""Save the project"""
print remoteApp.ExecuteArsl("SaveProjectCustom 'BaseDirectory=" + \
      "c:\\temp;Name=Demo;OverwriteExisting=True'")

"""This will print the extreme results to the console window in which
```

```
this script is running."""
print remoteApp.ExecuteArsl("GetResults")

"""This will get all results in csv format"""
print remoteApp.ExecuteArsl("GetResultsCustom " + \
      "'TestIds=50000;IncludeCsvData=True'")
```

NOTE **Replace "123.45.67.890" with the actual IP address of the oscilloscope running your automated test application, and replace "123" with an actual test ID for the application.**

See Chapter 1, "Remote Interface Documentation" on page 10, for more information on getting your application's test IDs.

Advanced Topic: Event Handling

The above examples demonstrate how to use the "forward facing" (remote client -> automated test application) direction of the remote interface. The remote interface also has a "callback" (automated test application -> remote client) direction, described here "Keysight.DigitalTestApps.Framework.Remote.Advanced" on page 14. Sample implementations using this capability follow.

Message Handling Example

The Python script shown below accomplishes the following tasks:

- Connects to an automated test application.
- Establishes the callback pipe.
- Redirects messages (prompts) to the remote client machine.
- Programmatically responds to a message (without displaying it).

```
"""Import the compiled Pythonnet module"""
import clr

"""Import .NET Framework types"""
from System.Runtime.Remoting import *
from System.Windows.Forms import *

"""Import the Keysight automated test app remote library DLL"""
clr.AddReference("Keysight.DigitalTestApps.Framework.Remote")
from Keysight.DigitalTestApps.Framework.Remote import *

"""Define the event callback handler"""
def SimpleMessageEventHandler(source, args):
     print 'Received message: ' + args.Message

     """Check for 'Run Ended' message using stable message ID
        (available in Remote Interface Version 2.80 and later,
        find IDs in app's Remote_Prog_Ref.chm)"""
     if args.ID == "602F9866-F975-42b7-842C-D8447E5E3FCB":
           args.Response = DialogResult.OK
     else:
           print 'Message not handled'

     """For Remote Interface Version 2.70 and older, must parse
        args.Message property text to determine this is 'Run Ended'
        message"""
     """e.g.
     if args.Message.find("All selected tests completed") >= 0:
     """

"""Connect to the automated test application running on the scope
This will wait for the application to be fully launched and ready
before proceeding"""
scopeIpAddress = "123.45.67.890"
remoteObj = RemoteAteUtilities.GetRemoteAte(scopeIpAddress)
remoteApp = IRemoteAte(remoteObj)

"""Verify Connection"""
print remoteApp.ApplicationName

"""Establish callback path"""
configFileFullPath = "c:\Python27\DLLs\
Keysight.DigitalTestApps.Framework.Remote.config";
RemotingConfiguration.Configure(configFileFullPath, False);
eventSink = RemoteAteUtilities.CreateAteEventSink(remoteApp, None, scope
IpAddress)

"""Subscribe to message events"""
eventSink.RedirectMessagesToClient = True
eventSink.SimpleMessageEvent += SimpleMessageEventHandler

"""Run tests"""
try:
     remoteApp.SelectedTests = [123]
     remoteApp.Run()
except Exception, e:
     print e.Message

"""Unsubscribe from message events and clean up"""
eventSink.SimpleMessageEvent -= SimpleMessageEventHandler
eventSink.Dispose()
```

NOTE

Replace "c:\Python27\DLLs\" with the actual location of the configuration file, replace "123.45.67.890" with the actual IP address of the oscilloscope running your automated test application, and replace "123" with an actual test ID for the application.

CAUTION

Compliance apps generate messages having Unicode encoding. Therefore, if you print messages received from the automated test application you may encounter a UnicodeEncodeError exception if the messages contain certain characters, for example, single quote characters U+2018 and U+2019. Please see your Python documentation for instructions on how to handle Unicode text.

Tips:

1. Use of the remote interface's callback path requires a .NET Remoting configuration file, Keysight.DigitalTestApps.Framework.Remote.config, which contains settings for the callback pipe. This file may be found in the remote toolkit in the Tools subdirectory.
2. The example script only handles the prompt that appears at the end of a run, because of this line:

   ```
   if args.Message.find("All selected tests completed") >= 0:
   ```

   To handle other prompts, add another case to the function, for example:

   ```
   if args.Message.find("All selected tests completed") >= 0:
       args.Response = DialogResult.OK
   elif args.Message.find("Some other message") >= 0:
       args.Response = DialogResult.Yes
   else:
       print 'Message not handled'
   ```

NOTE

Replace "Yes" with the actual name of a button that appears on the prompt.

Basic Events Handling Example

There are other events besides message prompts that a remote client can subscribe to. These are listed in the Keysight DigitalTestApps Remote Interface for .NET help file in the AteEventSink Class topic.

Here is an example in which the remote client wants to be notified every time a test is about to start executing. The Python script shown below accomplishes the following tasks:

- Connects to an automated test application.
- Establishes the callback pipe.
- Subscribes to the TestStarted event to get notified whenever any test starts running.
- Starts a run.

```
"""Import the compiled Pythonnet module"""
import clr

"""Import .NET Framework types"""
from System.Runtime.Remoting import *

"""Import the Keysight automated test app remote library DLL"""
clr.AddReference("Keysight.DigitalTestApps.Framework.Remote")
from Keysight.DigitalTestApps.Framework.Remote import *

"""Define the event callback handler"""
def TestStartedEventHandler(source, args):
```

```
    print('Test ' + str(args.TestID) + ' Started')

"""Connect to the automated test application running on the scope
This will wait for the application to be fully launched and ready
before proceeding"""
scopeIpAddress = "123.45.67.890"
remoteObj = RemoteAteUtilities.GetRemoteAte(scopeIpAddress)
remoteApp = IRemoteAte(remoteObj)

"""Verify Connection"""
print( remoteApp.ApplicationName)

"""Establish callback path"""
configFileFullPath = r"c:\Python27\DLLs\"
RemotingConfiguration.Configure(configFileFullPath, False);
eventSink = RemoteAteUtilities.CreateAteEventSink(remoteApp, None, scope
IpAddress)

"""Subscribe to message events"""
eventSink.TestStartingEvent += TestStartedEventHandler

"""Run tests"""
try:
    remoteApp.SelectedTests = [123]
    remoteApp.Run()
except Exception as error:
    print( error.Message)

"""Unsubscribe from message events and clean up"""
eventSink.TestStartingEvent -= TestStartedEventHandler
eventSink.Dispose()
```

Reassigning Actions in a Dummy Thread to Python's Main Thread

Note that code located inside an automated test app event handler, such as the SimpleMessageEventHandler, is executing in a "dummy" thread, which are threads of control started outside the Python threading module that have limited functionality. Certain actions cannot succeed inside a dummy thread, such as making a DDE (Dynamic Data Exchange) call. To handle this limitation, you need to use a mechanism that can reassign such actions to Python's main thread.

The remote toolkit contains a class named CrossThreadFunctionInvoker that enables you to do this, along with two programs that demonstrate usage of the class. These are contained in the Source\Python subdirectory:

File	Description
CrossThreadFunctionInvoker.py	Contains the CrossThreadFunctionInvoker class and a simple program to demonstrate its usage
CrossThreadFunctionInvokerAppExample .py	Contains a sample automated test app automation program that uses the CrossThreadFunctionInvoker class

Advanced Topic: Switch Matrix

The following Python code snippets demonstrate how to configure the Switch Matrix controller.

Using Automatic Mode

```
remoteApp.SwitchMatrixOn = True
remoteApp.SwitchMatrixSelectModeAuto("Keysight U3020A S26")
remoteApp.SwitchMatrixConnectToInstrument(1, "lan[123.456.7890]:inst0")
```

If additional drivers were created by the app during execution of `SwitchMatrixSelectModeAuto`, simply execute `SwitchMatrixConnectToInstrument` again for each driver ID and instrument address. Each driver must be connected to a separate switch matrix instrument.

NOTE Replace 123.456.7890 with the full SICL address or VISA alias of your switch instrument

Using Manual Mode

```
remoteApp.SwitchMatrixOn = True
remoteApp.SwitchMatrixSelectModeManual()
remoteApp.SwitchMatrixAddDriver("Keysight U3020A S26")
remoteApp.SwitchMatrixConnectToInstrument(1,"lan[123.456.7890]:inst0")
```

To create an additional driver, simply add these lines:

```
remoteApp.SwitchMatrixAddDriver("Keysight U3020A S26");
remoteApp.SwitchMatrixConnectToInstrument(2,"lan[123.456.7890]:inst0")
```

Each driver must be connected to a separate switch matrix instrument.

NOTE Replace 123.456.7890 with the full SICL address or VISA alias of your switch instrument.

Now you need to manually define each signal path, each one starting from the test point on the Device Under Test, through the switch, and ending with the scope channel.

Here is an example of a single-ended signal going into scope channel 3:

```
info = SwitchInfo()
info.Driver = 1
info.Slot = 1
info.Input = 1
remoteApp.SwitchMatrixDefineSignalPath("CLK",info,"3")
```

Here is an example of a differential signal going into a differential probe connected to scope channel 4:

```
info = SwitchInfo()
info.Driver = 1
info.Slot = 1
info.Input = 1
remoteApp.SwitchMatrixDefineSignalPath("Strobe+",info,"4+")
info = SwitchInfo()
info.Driver = 1
info.Slot = 2
info.Input = 1
remoteApp.SwitchMatrixDefineSignalPath("Strobe-",info,"4-")
```

Using Either Mode

All that remains is to assign a correction method to each signal path. This is optional but highly recommended.

Here is an example for one signal path that is using a single-ended probe:

```
options = PrecisionProbeOptions()
options.On = True
options.Mode = PrecisionProbeOptions.PrecisionProbeMode.Probe
options.Calibration = "MyCalName"
remoteApp.SwitchMatrixSetPathPrecisionProbe("CLK", "3", options)
```

Here is an example for one signal path that is using a differential probe:

```
options = PrecisionProbeOptions()
options.On = True
options.Mode = PrecisionProbeOptions.PrecisionProbeMode.Probe
options.Calibration = "MyCalName"
remoteApp.SwitchMatrixSetPathPrecisionProbe("Strobe+","4+",options)
```

NOTE **With differential probes, you do not separately assign correction to the negative half of the signal.**

4 Recommended Remote Programming Practices

Determine required tasks by first using the graphical user interface / 86
Verify the remote commands one at a time / 87
Check the log / 88

Determine required tasks by first using the graphical user interface

Before you write remote programming code to perform a task, it is often best to begin by determining the steps required to execute the task via the user interface on the oscilloscope. This will often help you avoid problems caused by incorrect step sequencing.

For example, automated test applications often manage large list of tests by providing a test filter. This is usually presented as a radio button group on the "Set Up" tab that determines which tests are visible on the "Select Tests" tab. Suppose these were your options:

"Set Up" tab:

- Speed: o Low o High

Imagine that when "Low" is selected, a configuration variable named "Speed" is set to value "Low" and the "Select Tests" tab is populated with three low-speed tests:

"Select Tests" tab:

- All Tests
  - Low Speed Test #1
  - Low Speed Test #2
  - Low Speed Test #3

Similarly, when "High" is selected on the "Set Up" tab, variable "Speed" is set to "High" and a different set of tests are shown:

"Select Tests" tab:

- All Tests
  - High Speed Test #1
  - High Speed Test #2
  - High Speed Test #3

Using the application's user interface, it is impossible to cause an invalid situation because only the currently available tests are displayed for you to select. However, via the remote interface you can request any test by setting the SelectedTests property. If you ask for an unavailable test, such as asking for High Speed Test #1 when "Speed" = "Low", the automated test application will throw an exception across the remote interface that your client will need to handle.

By first exploring the task via the user interface, though, you can avoid this exception by realizing you must first call SetConfig("Speed", "High") before calling RunTests() for a high-speed test.

Verify the remote commands one at a time

After you've converted some of the manual steps into remote interface commands (see "Determine required tasks by first using the graphical user interface" on page 86), execute them one at a time, either in the debugger or by sending them using the ARSL Command Line Utility. In many cases, you will be able to verify their success by visually inspecting the application on the oscilloscope.

Check the log

The automated test application logs all incoming remote commands to a file. You can visually inspect this file to see how your remote commands executed in relation to other events that get logged.

The log file is named message_#.log and may be found in one of these locations:

- For automated test apps that run directly on an oscilloscope:
  - Remote Interface Versions 1.10 and earlier: C:\scope\apps\(application name)\Scratchpad\
  - Remote Interface Version 1.20 and later: C:\Documents and Settings\All Users\Application Data\Keysight\Infiniium\Apps\(application name)\Project\
  - Remote Interface Version 1.40 and later:
    - WinXP: C:\Documents and Settings\All Users\Application Data\Agilent\Infiniium *_or_* FlexDCA\Apps\(application name)\Log\
    - Win7: C:\ProgramData\Agilent *_or_* Keysight\Infiniium *_or_* FlexDCA\Apps\(application name)\Log\
- For automated test apps that run on a separate PC:
  - WinXP: C:\Documents and Settings\All Users\Application Data\Agilent\Infiniium *_or_* FlexDCA\Apps\(application name)\Project\

    - Or -

    C:\Documents and Settings\All Users\Application Data\Agilent\Scope\Apps\(application name)\Log
  - Win7: C:\ProgramData\Agilent *_or_* Keysight\(platform)\Apps\(application name)\log\

    where "(platform)" is FlexDCA, Infiniium, InfiniiVision, or M8070A

A standard .NET Framework command gets logged this way:

```
1/8/2008 11:20:39 AM RPIC 55 RemoteInterfaceVersion?
1/8/2008 11:20:39 AM RPIR 55 <1.0>
```

The fields contain the following information:

```
1/8/2008 11:20:39 AM RPIC 55 RemoteInterfaceVersion?
-------------------- ---- -- ----------------------
         1             2   3           4
```

1 Timestamp.

2 Remote command log code.

3 Remote command sequence number.

4 Remote command.

A unique sequence number identifies command/response pairs. The remote command log code types are:

RPIC	Remote Programming Interface Command.
RPIR	Remote Programming Interface Response.

Here's another example. An ARSL command looks like this:

```
1/8/2008 11:22:36 AM RPIC 72 ExecuteArsl RemoteInterfaceVersion?
1/8/2008 11:22:36 AM RPIC 73 RemoteInterfaceVersion?
1/8/2008 11:22:36 AM RPIR 73 <1.0>
1/8/2008 11:22:36 AM RPIR 72 <>
```

Here you see that the ARSL command is translated into its .NET Framework equivalent and then executed.

If the remote command results in user messages being generated, these will appear in the log as well. For example:

```
1/8/2008 11:25:22 AM RPIC 74 SelectedTests 50000
1/8/2008 11:25:22 AM RPIR 74 <>
1/8/2008 11:25:24 AM RPIC 75 Run
1/8/2008 11:25:35 AM TLCL Information All Tests Complete
Total Test Time: 0.1093743 sec
Multi-Trials Count Up Test: 0.1093743s
1/8/2008 11:25:35 AM RPIR 75 <>
```

In this example, a test named "Multi-Trials Count Up Test" is selected and run. The "all tests complete" message is logged as well. Here are the field descriptions for user message log entries:

```
1/8/2008 11:25:35 AM TLCL Information All Tests Complete
-------------------- ---- ------------------------------
          1           2                 3
```

1 Timestamp.

2 User Message log code.

3 User Message.

Messages are grouped into three categories:

Tell	Errors, warnings, and info messages that have only an OK button.
Ask	Messages include other response types such as "Retry/Cancel" or "Yes/No".
AskForData	Messages that require the user to enter a value.

Here is the complete list of user message log codes you may encounter during remote operation:

ALCL	Ask message displayed locally (on oscilloscope).
ARMT	Ask message redirected to remote client.
ASPR	Ask message suppressed by remote client.
DLCL	AskForData message displayed locally (on oscilloscope).
DRMT	AskForData message redirected to remote client.
DSPR	AskForData message suppressed by remote client.
TLCL	Tell message displayed locally (on oscilloscope).
TRMT	Tell messages redirected to remote client.
TSPR	Tell messages suppressed by remote client.

5 Developing a Remote Client

Plugin for Keysight KS8400A Test Automation for PathWave / 92
Simple Remote Client / 94
ARSL Command Line Utility Implementation / 98
Simple GUI Remote Client Implementation / 101
Simple Message Handling Remote Client Implementation / 105
Message Handling Remote Client Implementation / 108
LabVIEW Simple Remote Client / 116
LabVIEW Demo Client Construction / 128

This chapter shows you how to develop remote client executables using the Keysight KS8400A Test Automation for PathWave (TAP) version 9, .NET Framework language C#, and the graphical LabVIEW 8.5 programming environment. The Automated Test Application Remote Development Toolkit provides sample projects to enable you to experiment with the remote client development by starting with existing code. C# and LabVIEW implementations are provided.

Plugin for Keysight KS8400A Test Automation for PathWave

The DigitalTest Apps API plugin for TAP requires TAP version 9 or newer. While the plugin is complete enough to enable you to perform most tasks, the source code for the plugin is included in this toolkit enabling you to modify and extend it if desired:

```
C:\ProgramData\Keysight\DigitalTestApps\Remote Toolkit\Version x\Source\
TAP\Plugins\API\  (Visual Studio 2017 solution)
```

The primary classes of interest are found in the solution's TestSteps folder. Most of the classes in the folder correspond to a single TAP step, with the exception of a few base classes:

```
class TestStep (provided by TAP)
- class DigitalTestStep (provided by the plugin): Base class for all
  our plugin steps
  -- (most of remaining classes)
  -- class PathCorrectionStep: Base class for path correction-related
     plugin steps
      --- class InfiniiSimConfigureOneChannelStep
      --- class PrecisionProbeConfigureOneChannelStep
```

If you add new steps, inherit your class from the DigitalTestStep, which:

- Wraps the Run method with exception handling and logging
- Generates the settings summary suffix that appears next to the step names in the test plan list

Instead of modifying the API plugin, consider making all of your local changes in a separate plugin (see below for an example). You can add new steps there or override steps found in the API plugin. By keeping your local changes separated, you will be able to use updates to the API plugin in future N5452A Remote Toolkit releases without having to merge your changes back in.

If you want to programmatically handle and respond to prompts and events generated by the automated test app, then create a new plugin in which you define a new class that inherits from ManageMessageStep and overrides its "OnXxxEvent" methods. For example, "My DDR5 Plugin" containing "ManageDDR5MessagesStep". Then, when you install both plugins, use your new step in place of the DigitalTest Apps API ManageMessagesStep.

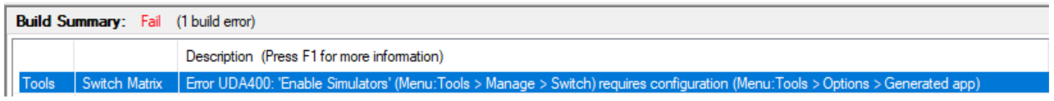
An example of how to do this may be found here:

```
C:\ProgramData\Keysight\DigitalTestApps\Remote Toolkit\Version x\Source\
TAP\Plugins\SampleEventHandler\  (Visual Studio 2017 solution)
```

The source code for the sample UDAs is also included:

```
C:\ProgramData\Keysight\DigitalTestApps\Remote Toolkit\Version x\Source\
TAP\UDA\DemoApps\ (UDA version 3.30 project files)
```

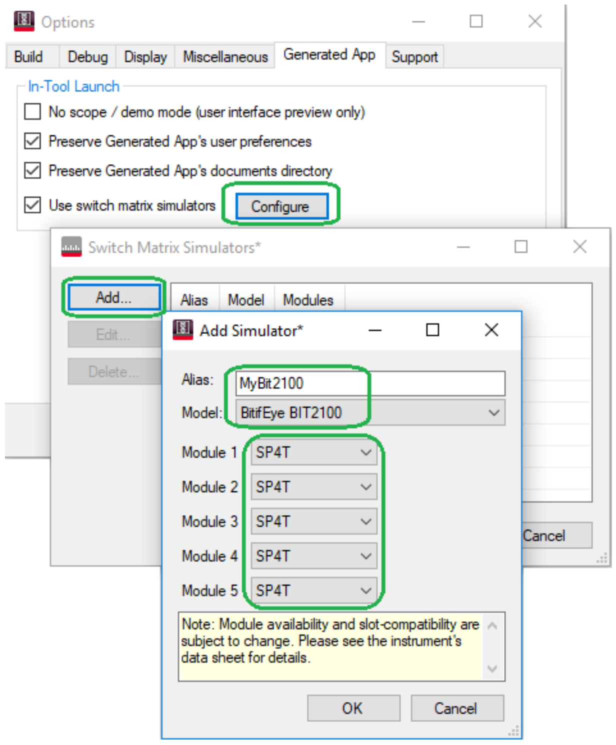
If you get this error when you build the project:



Build Summary: Fail (1 build error)

		Description (Press F1 for more information)
Tools	Switch Matrix	Error UDA400: 'Enable Simulators' (Menu:Tools > Manage > Switch) requires configuration (Menu:Tools > Options > Generated app)

then configure a switch matrix simulator in the UDA tool using Menu: Tools > Options:



Options

Build | Debug | Display | Miscellaneous | Generated App | Support

In-Tool Launch

- ☐ No scope / demo mode (user interface preview only)
- ☑ Preserve Generated App's user preferences
- ☑ Preserve Generated App's documents directory
- ☑ Use switch matrix simulators [Configure]

Switch Matrix Simulators*

Add... | Edit... | Delete... — Alias | Model | Modules — Cancel

Add Simulator*

Alias: MyBit2100

Model: BitifEye BIT2100

Module 1: SP4T

Module 2: SP4T

Module 3: SP4T

Module 4: SP4T

Module 5: SP4T

Note: Module availability and slot-compatibility are subject to change. Please see the instrument's data sheet for details.

OK | Cancel

Simple Remote Client

The easiest way to begin is to start with the "Simple Remote Client" demo program.

1 On the computer, browse to the location where the toolkit is installed. In the desired "Source\C#\" subdirectory, locate the "SimpleRemoteClient" folder. It contains a C# implementation of the client. Copy this folder to another location on your computer so you can modify the copy without losing the original files.

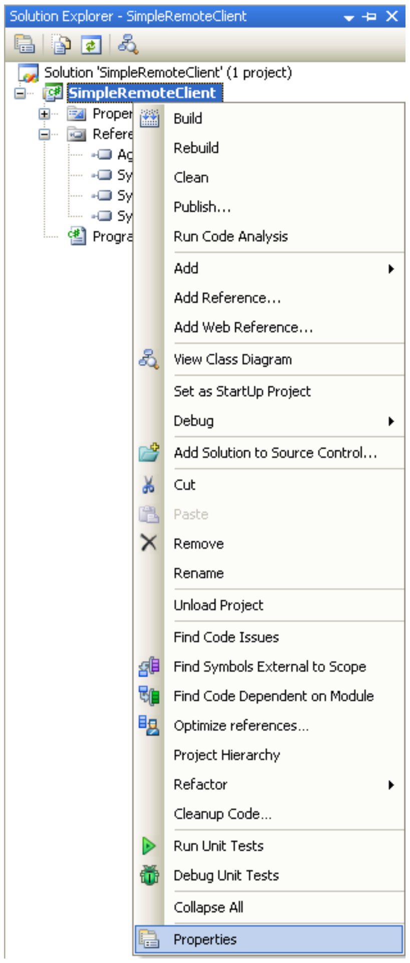
2 Copy the Keysight.DigitalTestApps.Framework.Remote.dll application file (or Agilent.Infiniium.AppFW.Remote.dll, see "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide) to the "distrib" directory in the new "SimpleRemoteClient" folder.

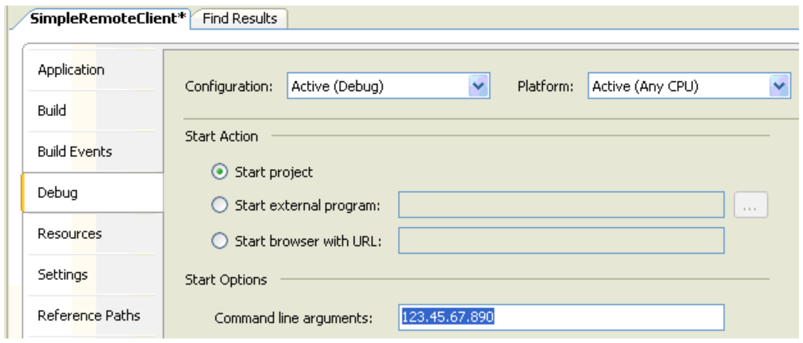
NOTE

Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.

3 Using Visual Studio 2013 or newer, open the file "SimpleRemoteClient.sln".

4 In Visual Studio, display the properties page for the SimpleRemoteClient project and go to the "Debug" tab. Enter the IP address of the oscilloscope in the "Command line arguments field":



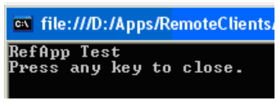


Configuration: Active (Debug)
Platform: Active (Any CPU)

[✓ Start project]
Start external program:
Start browser with URL:

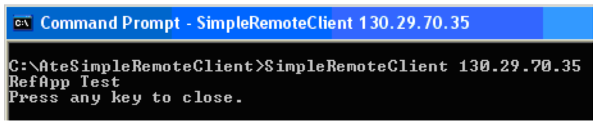
Command line arguments: 123.45.67.890

Build and run. You should see output similar to this:



```
RefApp Test
Press any key to close.
```

You can also run this from the command line:



```
C:\AteSimpleRemoteClient>SimpleRemoteClient 130.29.70.35
RefApp Test
Press any key to close.
```

File description

These are the key folders and files in the project directory:

- SimpleRemoteClient.sln — Visual Studio solution file
- SimpleRemoteClient\
    - SimpleRemoteClient.csproj — Visual Studio project file
    - Program.cs — C# source file containing the entire program
    - distrib\ — Directory where build outputs are copied to.

Source code description

Here is the function of each of the lines in the program:

1. This defines the namespace the program lives in:

   ```
   namespace SimpleRemoteClient
   ```

2. This names the class the program lives in:

   ```
   internal class Program
   ```

3. This is the method that gets executed when the program is run:

   ```
   private static void Main(string[] args)
   ```

4. This establishes a connection to the application running on the oscilloscope:

   ```
   Keysight.DigitalTestApps.Framework.Remote.IRemoteAte remoteAte =
   Keysight.DigitalTestApps.Framework.Remote.RemoteAteUtilities.GetRemot
   eAte(args[0]);
   ```

   The "Keysight.DigitalTestApps.Framework.Remote." prefix indicates that IRemoteAte and RemoteAteUtilities are defined in the Keysight.DigitalTestApps.Framework.Remote.dll.

5. This sends a remote command to the oscilloscope to execute a property query in the automated test application:

   ```
   string appName = remoteAte.ApplicationName;
   ```

   This particular property contains the application's name. Please see Chapter 1, Chapter 1, “Keysight DigitalTestApps Remote Interface,” starting on page 9, for information on other properties and methods available in the automated test application's remote interface.

6 This prints the result in the command window:

```
System.Console.WriteLine(appName + "\nPress any key to finish.");
System.Console.ReadKey();
```

The Simple Remote Client is intended for use as a remote programming learning and experimentation platform. For more realistic (and interesting) implementations, see the sections that follow.

ARSL Command Line Utility Implementation

You may also examine the implementation for, modify, and build the ARSL Command Line Utility.

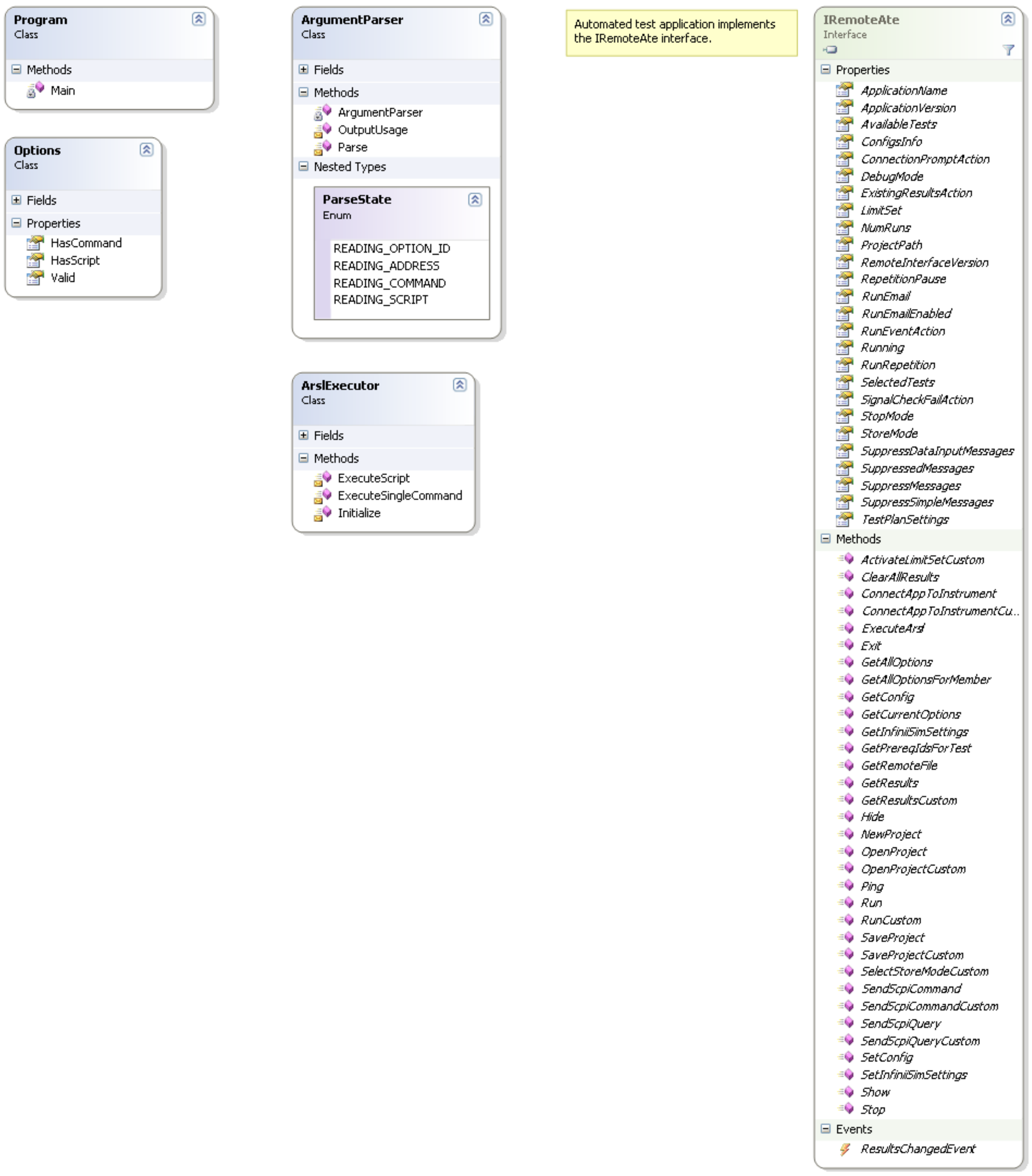
As with the Simple Remote Client, you will need to copy the file Keysight.DigitalTestApps.Framework.Remote.dll (or Agilent.Infiniium.AppFW.Remote.dll) to the distrib directory of the project.

NOTE **Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.**

Here are some of the differences from the Simple Remote Client:

- Design

  This is more complex, so the implementation uses multiple components: the Program, a type to hold the runtime options, an argument parser and an Arsl executor as summarized in the diagram below:



Notice that all of the interaction with the remote application is contained within the ArslExecutor.

- The ArslExecutor (ArslExecutor.cs) makes the connection to the remote application:

```
internal void Initialize(string ipAddress, bool useCustomPort,
    bool verbose)
{
  if (useCustomPort)
  {
    GetRemoteAteOptions options = new GetRemoteAteOptions();
    options.IpAddress = ipAddress;
    options.UseCustomPort = true;
    mRemoteAte = RemoteAteUtilities.GetRemoteAteCustom(options);
    …
  }
  else
  {
    mRemoteAte = RemoteAteUtilities.GetRemoteAte(ipAddress);
    …
  }

  …
}
```

- The ArslExecutor (ArslExecutor.cs) makes all the calls to the remote application. Even though the ARSL utility enables you to execute many different remote interface property and method actions, it only uses one member of the remote interface to accomplish this:

  ```
  internal void ExecuteSingleCommand(string command)
  {
    string result = mRemoteAte.ExecuteArsl(command);
    …
  }
  ```

  This particular method enables passing the utility a string representation of a property or method action and having it get executed as a real property or method action.

The rest of program is not specific to remote interface programming and is concerned only with parsing and validating the user input and generating the output.

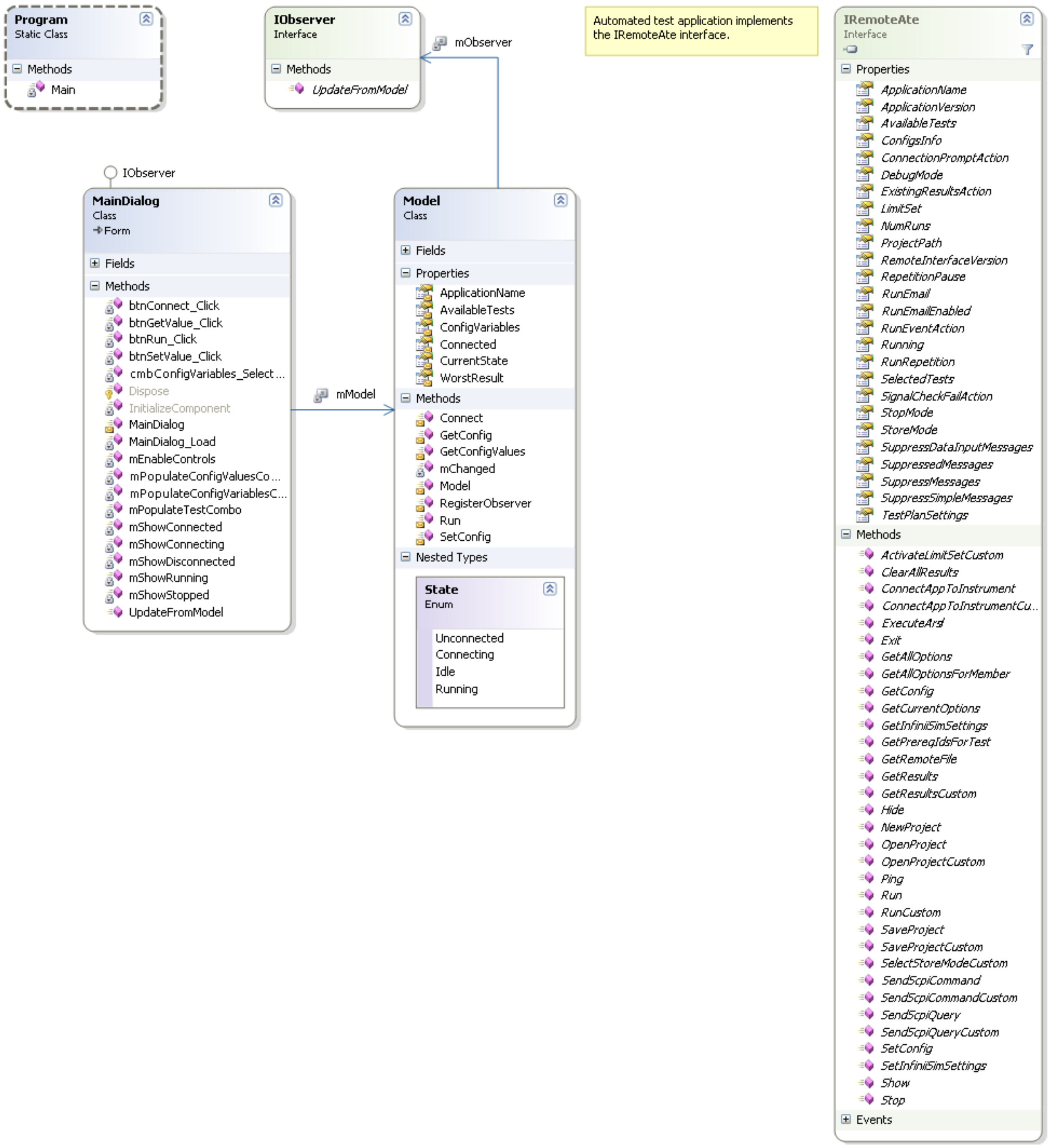
Simple GUI Remote Client Implementation

You may examine the implementation for, modify, and build the Simple GUI Remote Client.

As with the Simple Remote Client, you will need to copy the file Keysight.DigitalTestApps.Framework.Remote.dll (or Agilent.Infiniium.AppFW.Remote.dll) to the distrib directory of the project.

NOTE **Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.**

The design of this application is summarized in the diagram below.



There are two primary design patterns in use: Observer and State. The Observer pattern leads us to place business logic (interaction with the remote application) in a separate module from the main dialog. The State pattern leads us to implement well-structured transitions as the user performs the various steps described in the main dialog.

Here are some remote interface-related highlights from the code:

- The Model (Model.cs) configures the .NET Remoting Interface:

  ```
  Internal void Initialize()
  {
    string path =
      Path.getDirectoryName(Assembly.GetExecutingAssembly().Location);
    string configFullPath =
      Path.Combine(path,
      "Keysight.DigitalTestAps.Framework.Remote.config");
    RemotingConfiguration.Configure(configFullPath, false);
  }
  ```

- The Model (Model.cs) makes the connection to the remote application:

  ```
  internal void Connect(string ipAddress)
  {
    try
    {
      …
      mRemoteAte = RemoteAteUtilities.GetRemoteAte(ipAddress);
      mRemoteAte.Ping();
    }
    catch (Exception e)
    {
      MessageBox.Show("Could not connect to an application running at "
          + "IP Address: " + ipAddress + "\n\n" + "Reason: "
          + e.Message);
      …
      return;
    }
  }
  ```

  The Ping method is called because the GetRemoteAte method will always return successfully, even if there is no application at the specified address, while Ping (and all the other properties and methods of the remote interface) can only work if the connection is valid.

- The Model (Model.cs) makes all other calls to the remote application

  - Determining Available Tests

    As mentioned in "Determine required tasks by first using the graphical user interface" on page 86, you may only execute a test via the remote interface if the automated test application is in such a state that the test is available on the "Select Tests" tab of the user interface. Fortunately, the remote interface provides a property that tells you exactly which tests are available at any given time. The Simple GUI Remote Client uses this to build the test list that it presents to the user. Again, in the Connect method:

    ```
    internal void Connect(string ipAddress)
    {
      …
      mAvailableTests = mRemoteAte.AvailableTests;
    }
    ```

This stores the test availability information in the model so it will be accessible by the dialog.

- Test execution

  Tests are executed in the Run method (Model.cs) by setting the remote interface property 'SelectedTests' and calling the method 'Run':

```
internal void Run(int testIndex)
{
  try
  {
    int[] testToSelect =
        new int[] { mAvailableTests[testIndex].ID };
    mRemoteAte.SelectedTests = testToSelect;
    mRemoteAte.Run();
    …
  }
  catch (Exception e)
  {
    MessageBox.Show("An error occurred during the run:\n\n" +
        "Detail: " + e.Message);
  }
  …
}
```

- Result access

  The result of the test is accessed in the Run method (Model.cs) by calling the remote interface GetResults method:

```
internal void Run(int testIndex)
{
  …
  ResultContainer results = mRemoteAte.GetResults();

  // Remote Interface Versions 1.10 and prior (deprecated in 1.20)
:
  //mWorstResult = results.WorstResults[0];

  // Remote Interface Versions 1.20 and later only:
  mWorstResult = results.ExtremeResults[0];
  …
}
```

  This returns a ResultContainer object which contains various details concerning the result.

The rest of program is not specific to remote interface programming and is concerned only with managing the graphical user interface.

Simple Message Handling Remote Client Implementation

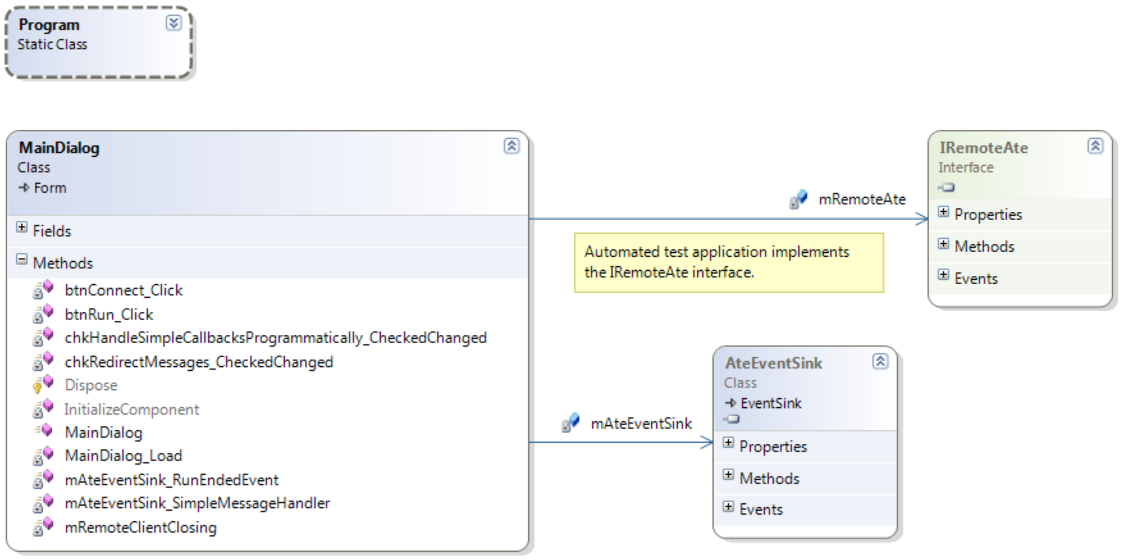
You may examine the implementation for, modify and build the Simple Message Handling Remote Client.

As with the Simple Remote Client, you will need to copy the file Keysight.DigitalTestApps.Framework.Remote.dll (or Agilent.Infiniium.AppFW.Remote.dll) to the distrib directory of the project. You will also need to copy this additional file: Keysight.DigitalTestApps.Framework.Remote.config (or Agilent.Infiniium.AppFW.Remote.config).

NOTE

Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.

The design of this application is summarized in the diagram below:



Note that the model-view pattern is not used in this application in order to more clearly highlight the message-handling elements. All relevant code is in the file MainDialog.cs. Here are some remote interface-related highlights from the code:

- Initialization occurs when you click the "Connect" button:

```
private void btnConnect_Click(object sender, EventArgs e)
{
  …
  // Enable forward remote interface
  // (remote client -> automated test app)
  mRemoteAte = RemoteAteUtilities.GetRemoteAte(ipAddress);
  // Enable asynchronous callbacks
  // (automated test app -> remote client)
  // User must place the .config file in the same location as the
  // remote client's .exe
  string path =
    Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
  string configFullPath =
    Path.Combine(path, "Keysight.DigitalTestApps.Framework.Remote.con
fig");
  RemotingConfiguration.Configure(configFullPath, false);
  mAteEventSink =
    RemoteAteUtilities.CreateAteEventSink(mRemoteAte, this,
      ipAddress);
}
```

- The AteEventSink object is the key to managing callback activity. It has properties for specifying whether you want prompts issued by the automated test app to be:
  - allowed to display on the scope
  - transported to the remote client PC and displayed there
  - never displayed (programmatically responded to by the remote client via an event handler)

```
private void chkRedirectMessages_CheckedChanged(
  object sender, EventArgs e)
{
  // Tell my AteEventSink to coordinate with app regarding message
  // redirection
  mAteEventSink.RedirectMessagesToClient =
    chkRedirectMessages.Checked;
  …
}
private void chkHandleSimpleCallbacksProgrammatically_CheckedChanged(
  object sender, EventArgs e)
{
  if (chkHandleSimpleCallbacksProgrammatically.Checked)
  {
    // Tell my AteEventSink to let me handle simple message callbacks
    // from the app myself
    mAteEventSink.SimpleMessageEvent +=
      mAteEventSink_SimpleMessageHandler;
  }
  else
  {
    // Tell my AteEventSink to automatically handle (display) simple
    // message callbacks from the app
    mAteEventSink.SimpleMessageEvent -=
      mAteEventSink_SimpleMessageHandler;
  }
}
```

- To programmatically respond to a prompt from the automated test application, the event handler sets a property corresponding to the button on the prompt that is to be "clicked", for example "OK":

```
private void mAteEventSink_SimpleMessageHandler(
  object sender, MessageEventArgs e)
{
  if (e.Message.Contains("completed")) // This is the prompt the app
                                     // displays at the end of a run
  {
    e.Response = DialogResult.OK; // Be sure to respond using a
      // DialgResult that is valid for the message being handled
    return;
  }
  …
}
```

- At the start of a run, the demo program chooses which thread to make the request in:
  - If no callbacks from the app can occur, the Run command may be safely executed in the main thread.
  - If callbacks are enabled, the Run command must be executed in a worker thread. For more details, please see the "multithreading (see page 112)" discussion.

```
private void btnRun_Click(object sender, EventArgs e)
{
  …
  if (mAteEventSink.RedirectMessagesToClient)
  {
    …
    // Must use worker thread to prevent deadlock when receiving
    // asynchronous callbacks from app during a run
    new Thread(mRemoteAte.Run).Start(); // This demo program will
      // immediately proceed to the next line of code regardless
      // of the state of the run
  }
  else
  {
    // No callbacks will be sent to the remote client, so it is
    // safe to call Run() in the main thread
    mRemoteAte.Run(); // This demo program will block at this line
      // of code until the run is complete
  }
}
```

- An automated test app will only allow one remote client to be registered to handle callbacks. Therefore, when the remote client exits it performs this cleanup action:

```
private void mRemoteClientClosing()
{
  if (mAteEventSink != null)
  {
    // Tell my AteEventSink to notify the app that I am no longer
    // the receiver for callbacks (so another remote client can
    // register with the app)
    mAteEventSink.Dispose();
  }
}
```

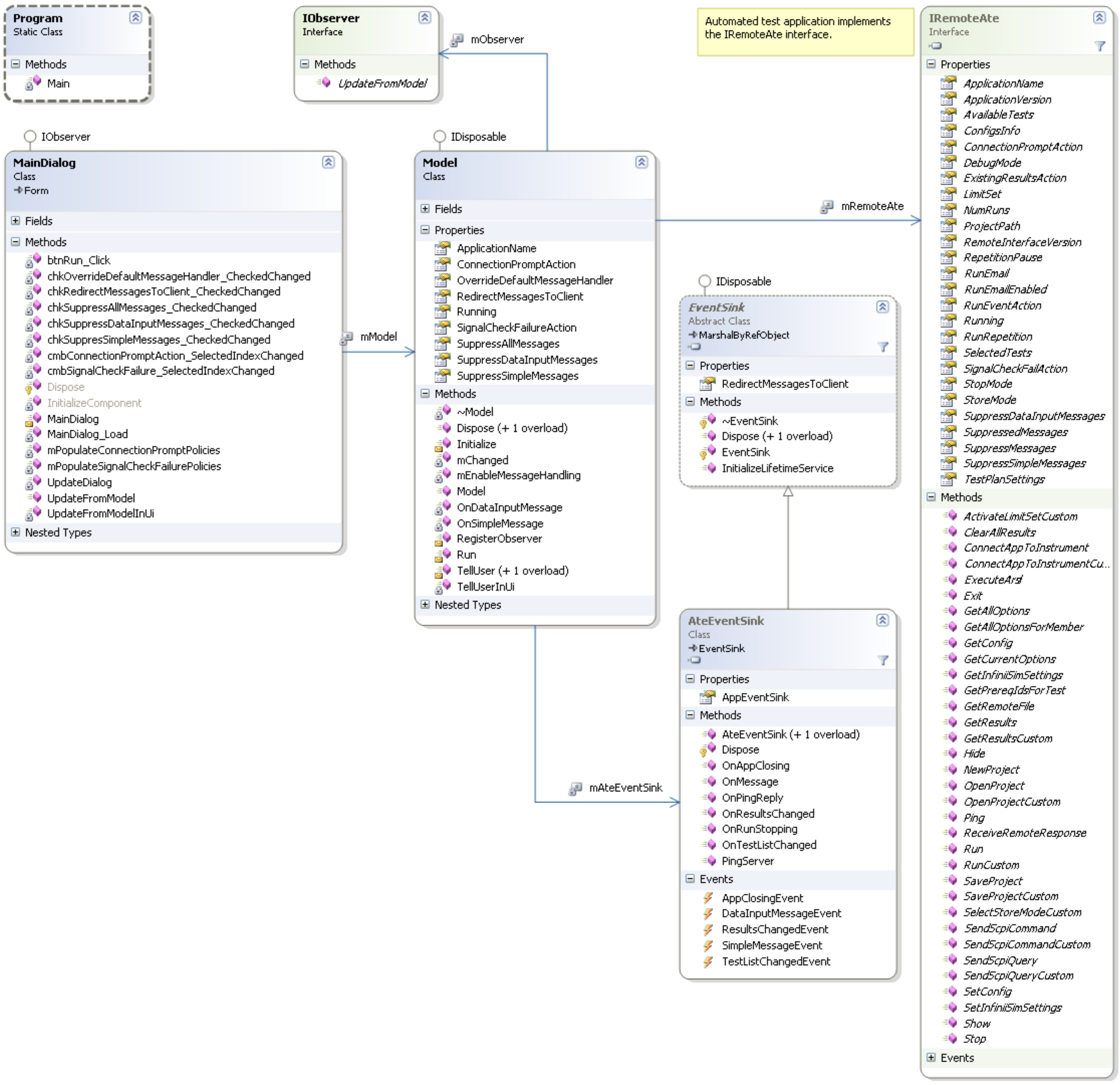
Message Handling Remote Client Implementation

You may examine the implementation for, modify and build the Message Handling Remote Client.

As with the Simple Remote Client, you will need to copy the file Keysight.DigitalTestApps.Framework.Remote.dll (or Agilent.Infiniium.AppFW.Remote.dll) to the distrib directory of the project. You will also need to copy this additional file: Keysight.DigitalTestApps.Framework.Remote.config.

NOTE **Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.**

The design of this application is summarized in the diagram below:



There is one primary design pattern in use: Observer. The Observer pattern leads us to place business logic (interaction with the remote application) in a separate module from the main dialog.

Here are some remote interface-related highlights from the code:

- .NET Remoting

The Model (Model.cs) configures the .NET Remoting Interface to enable the callback path (scope->client):

```
internal void Initialize(...)
{
  string path =
    Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
  string configFullPath =
    Path.Combine(path, "Keysight.DigitalTestApps.Framework.Remote.Con
fig");
  RemotingConfiguration.Configure(configFullPath, false);
  ...
}
```

NOTE **The configuration file's contents are standard and may be used for any remote client requiring callback capability.**

- Groups and Namespaces

  All of the controls in the "Core Features" group use remote interface members contained in the basic namespace, Keysight.DigitalTestApps.Framework.Remote:

  - Run: mRemoteAte.Run();
  - Suppress All Messages: mRemoteAte.SuppressMessages = value;
  - Suppress Simple Messages: mRemoteAte.SuppressSimpleMessages = value;
  - Suppress Data Input Messages: mRemoteAte.SuppressDataInputMessages= value;
  - Connect Prompt Action: mRemoteAte.ConnectionPromptAction = value;
  - Check Signal Prompt Action: mRemoteAte.SignalCheckFailAction= value;

  All of the controls in the "Advanced" group use remote interface members contained in the advanced namespace, Keysight.DigitalTestApps.Framework.Remote.Advanced:

  - Redirect Messages: mAteEventSink.RedirectMessagesToClient = value;
  - Override Default Handler: mAteEventSink.SimpleMessageEvent += OnSimpleMessage;

- Event Sink

  Both of the advanced features require the automated test application running on the oscilloscope to be able to call a function on the remote client. This is enabled by use of a class called "AteEventSink" (contained in the Keysight.DigitalTestApps.Framework.Remote.dll assembly). All the remote client has to do is instantiate an AteEventSink and keep it alive until the remote client closes:

```
mAteEventSink = RemoteAteUtilities.CreateAteEventSink(
    mRemoteAte,
    mMainDialog,
    mIpAddress);
```

The event sink uses its constructor parameters to establish a two-way link between the remote client and the application running on the oscilloscope. It also exposes properties and events that enable the remote client to activate advanced communication features, such as:

- Redirecting dialogs to the client:

  ```
  mAteEventSink.RedirectMessagesToClient = true;
  ```

  This property tells the EventSink to configure the application running on the oscilloscope to send user messages to the EventSink instead of displaying them on the oscilloscope. Then the EventSink will use a default algorithm to display the message on the client.

- Overriding the default message handler:

  ```
  mAteEventSink.SimpleMessageEvent += OnSimpleMessage;
  ```

  This event enables the remote client to register a client-defined method for EventSink to use instead of its default algorithm. So, when RedirectMessagesToClient is set to true and a handler is subscribed to the SimpleMessageEvent event, the EventSink will forward on messages received from the application running on the oscilloscope to your client to do with as you please. In the example program, the client simply appends some text to the original message and displays it:

  ```
  private void OnSimpleMessage(object sender, MessageEventArgs e)
  {
    e.Response = TellUser("Local handler processed this simple " +
      "message:\n\n" + e.Message + "\n\nClick one of the " +
      "buttons provided by the message sender:",
      e.Buttons,
      e.Icon,
      e.DefaultButton);
  }
  ```

NOTE If you need to detect a certain message in order to perform a related task, you could examine the e.Message property for words that appear in the message. However, this approach is fragile because the application may change the message slightly over time. Instead, use the e.ID property, which is stable over time.

- Mutlithreading

Message redirection is a powerful capability, but it adds complexity to the design of the remote client. In particular, while message redirection is enabled, you run the risk of deadlock. Here are two such scenarios (remember, these apply only when message redirection is enabled):

a Executing a remote interface property action or method call in the GUI thread. When the client sets a property or calls a method in the automated test application's remote interface, if the remote action leads the automated test application to issue a redirectable message, then deadlock will ensue:

- The client's GUI thread is occupied waiting for the property action or method call to return. It will not process the incoming redirected message.
- The automated test application's GUI thread is occupied waiting for the redirected message to get answered.

To prevent this scenario, execute in a worker thread any remote interface property action or method call that may result in a redirected message.

b Updating the GUI directly from the remote interface. If you follow the design guideline from the previous step and execute remote actions from within a worker thread, you may still run into a similar deadlock situation. For example, consider this GUI code:

```
// GUI code
private void
chkRedirectMessagesToClient_CheckedChanged(object sender,
EventArgs e)
{
  mModel.RedirectMessagesToClient =
      chkRedirectMessagesToClient.Checked;
  UpdateDialog();
}

private void UpdateDialog()
{
  chkSuppresSimpleMessages.Checked =
      mModel.SuppressSimpleMessages;
  …
}

// Model code
public bool RedirectMessagesToClient
{
  set { return mRemoteAte. RedirectMessagesToClient = value; }
}

public bool SuppressSimpleMessages
{
  get { return mRemoteAte.SuppressSimpleMessages; }
}
```

This sequence of events may lead to deadlock:

- **i** If the application currently has a dialog displayed locally on the oscilloscope (for example "all tests completed").
- **ii** On the remote client, the user clicks the RedirectMessagesToClient checkbox to enable this mode.
- **iii** The callback code in chkRedirectMessagesToClient_CheckedChanged begins.
- **iv** A RedirectMessagesToClient property set is sent to the automated test application and control returns to the client.
- **v** The client begins its UpdateDialog method.
- **vi** Asynchronously, the dialog that was being displayed on the oscilloscope at the beginning of this scenario is redirected to the client.
- **vii** The client attempts to query the automated test application during UpdateDialog.

This would result in deadlock:

- The automated test application cannot respond to the client's query because it is busy waiting for a response from the client for the redirected message.
- The client will not display the redirected message because it is busy waiting for a response from the automated test application for the remote query.

To prevent this scenario, do not send remote commands to the automated test application while a redirected message needs attention on the remote client. In the above scenario, you would resolve the design problem by replacing the model's SuppressSimpleMessage getter implementation with one that caches the automated test application's state in a field and returns this field instead of querying the oscilloscope.

Here is actual message handling remote client implementation that demonstrates how design can avoid both of the above scenarios:

In MainDialog.cs:

```
// Define a delegate for a method that takes a single string
// parameter and returns void
private delegate void VoidStringDelegate(string s);

// Executed when the user clicks "Run"
private void btnRun_Click(object sender, EventArgs e)
{
  …
  // Call the Model's version of Run in a worker thread
  VoidStringDelegate dlgtRun = mModel.Run;
  dlgtRun.BeginInvoke(txtTestID.Text, null, null);
  // BeginInvoke immediately returns even though the run is not
  // completed.
```

```
    // Now the GUI is ready to handle any callbacks from the
    // automated test application that may occur during the run.
    …
  }
```

In Model.cs:

```
internal void Run(string testId)
{
  // Executing in a worker thread
  …
  mRemoteAte.RunTests(testId);
  …
}

public bool SuppressSimpleMessages
{
  get
  {
    // Return a field…don't query the oscilloscope
    return mSuppressSimpleMessages;
  }
  set
  {
    mSuppressSimpleMessages = value;
    mRemoteAte.SuppressSimpleMessages = mSuppressSimpleMessages;
  }
}
```

- Disposal

  To facilitate proper resource cleanup, always dispose of the event sink. The Model class (Model.cs) implements this using a standard pattern:

```
private bool mAlreadyDisposed;

public void Dispose()
{
  Dispose(true);
  GC.SuppressFinalize(this);
}

~Model()
{
  Dispose(false);
}

protected virtual void Dispose(bool isDisposing) {
  if (mAlreadyDisposed)
  {
    return;
  }

  if (isDisposing)
  {
    mAteEventSink.Dispose();
  }
```

```
      mAlreadyDisposed = true;
    }
```

LabVIEW Simple Remote Client

This section describes how to create your own remote client using the LabVIEW programming environment. In addition to requiring the Keysight.DigitalTestApps.Framework.Remote.dll file ((or Agilent.Infiniium.AppFW.Remote.dll, see "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide for a list of minimum requirements), LabVIEW clients use an interface adapter assembly, "Keysight.DigitalTestApps.RemoteTestClient.dll" (or Agilent.Infiniium.RemoteTestClient.dll). This file is also provided in this Toolkit in the desired Tools\LabVIEW\LabVIEW Remote Adapter subdirectory.

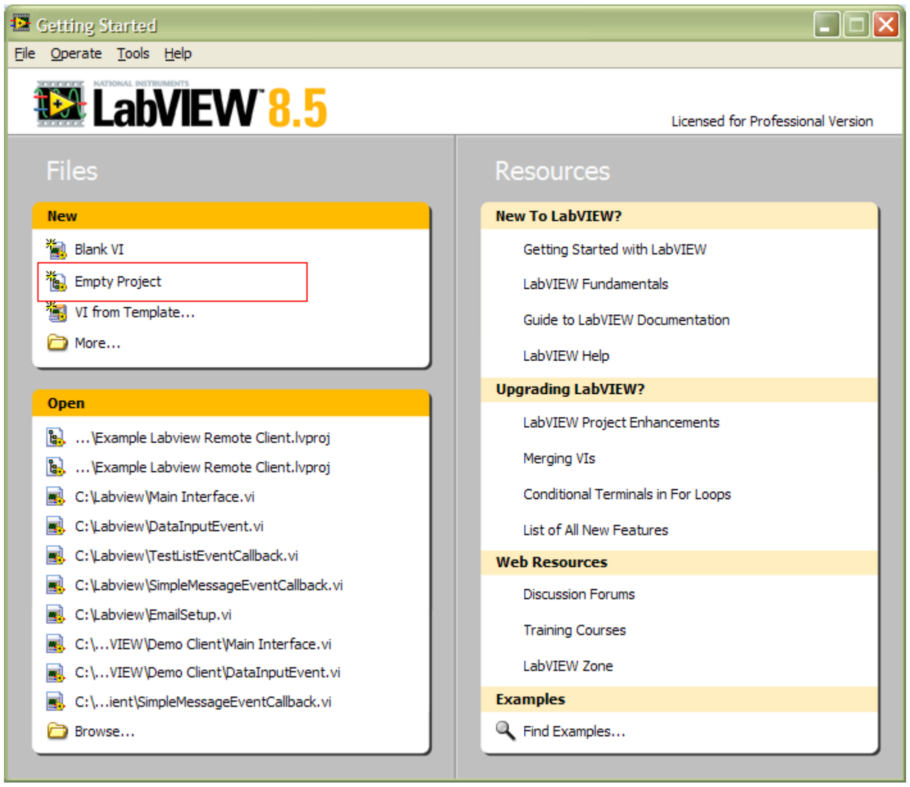
NOTE **Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.**

1 Create a new folder in C:\Labview.
2 Copy the following files to the folder created:
   a Keysight.DigitalTestApps.Framework.Remote.dll (or Agilent.Infiniium.AppFW.Remote.dll, see "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide).
   b On the computer, browse to the location where the toolkit is installed.

      From the desired "Tools\LabVIEW\LabVIEW Remote Adapter" subdirectory copy Keysight.DigitalTestApps.RemoteTestClient.dll (or Agilent.Infiniium.RemoteTestClient.dll) and Keysight.DigitalTestApps.RemoteTestClient.dll.config (or Agilent.Infiniium.RemoteTestClient.dll.config).

NOTE **Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.**

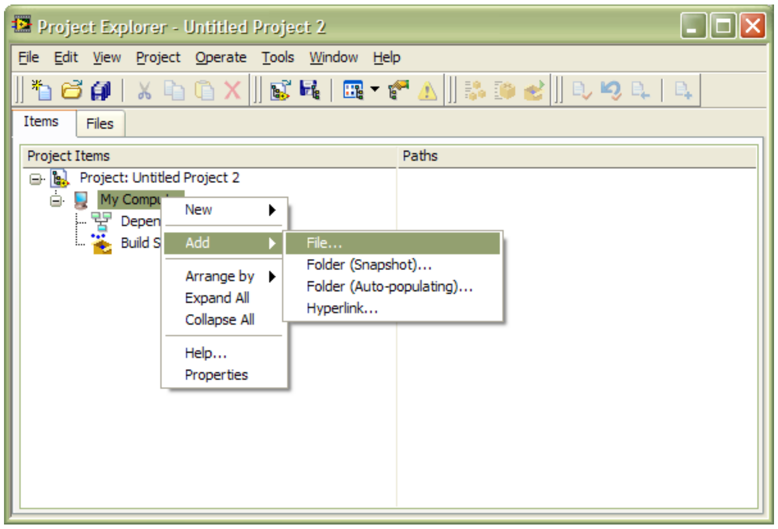
3 Launch Labview and create an empty project.

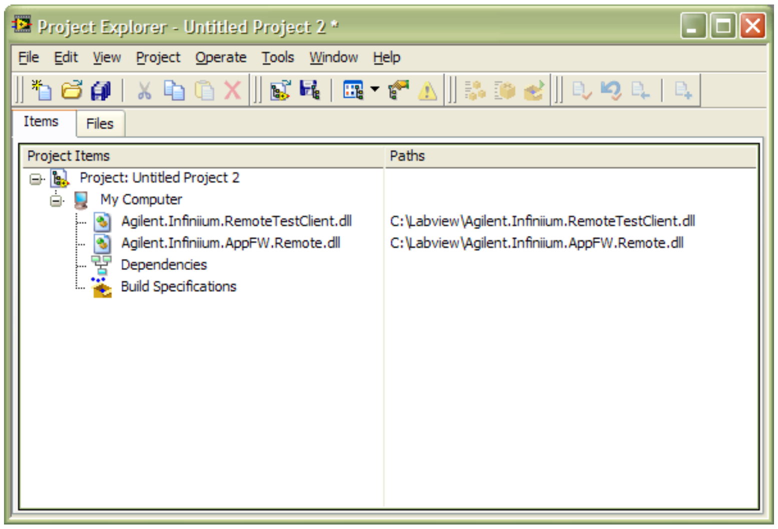

Getting Started
File Operate Tools Help
LabVIEW 8.5
Licensed for Professional Version
Files
New
Blank VI
Empty Project
VI from Template...
More...
Open
...\Example Labview Remote Client.lvproj
...\Example Labview Remote Client.lvproj
C:\Labview\Main Interface.vi
C:\Labview\DataInputEvent.vi
C:\Labview\TestListEventCallback.vi
C:\Labview\SimpleMessageEventCallback.vi
C:\Labview\EmailSetup.vi
C:\...VIEW\Demo Client\Main Interface.vi
C:\...VIEW\Demo Client\DataInputEvent.vi
C:\...ient\SimpleMessageEventCallback.vi
Browse...
Resources
New To LabVIEW?
Getting Started with LabVIEW
LabVIEW Fundamentals
Guide to LabVIEW Documentation
LabVIEW Help
Upgrading LabVIEW?
LabVIEW Project Enhancements
Merging VIs
Conditional Terminals in For Loops
List of All New Features
Web Resources
Discussion Forums
Training Courses
LabVIEW Zone
Examples
Find Examples...

4 Add the following files to the project:

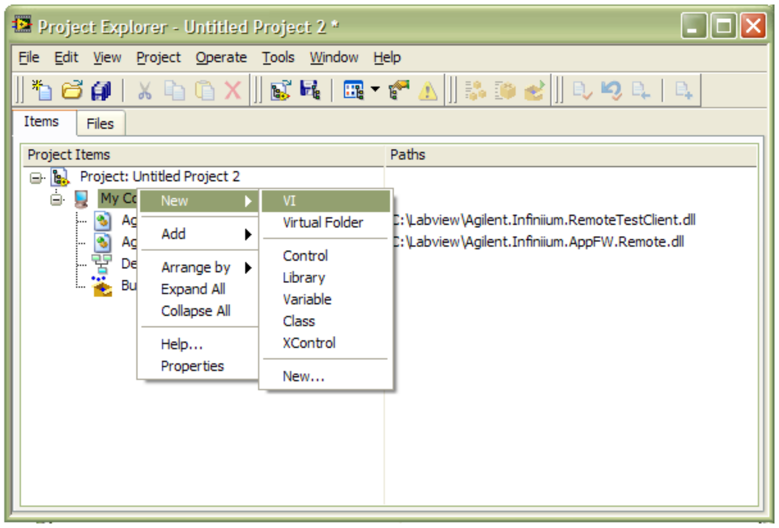
a Keysight.DigitalTestApps.Framework.Remote.dll

b Keysight.DigitalTestApps.RemoteTestClient.dll

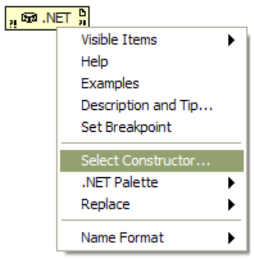


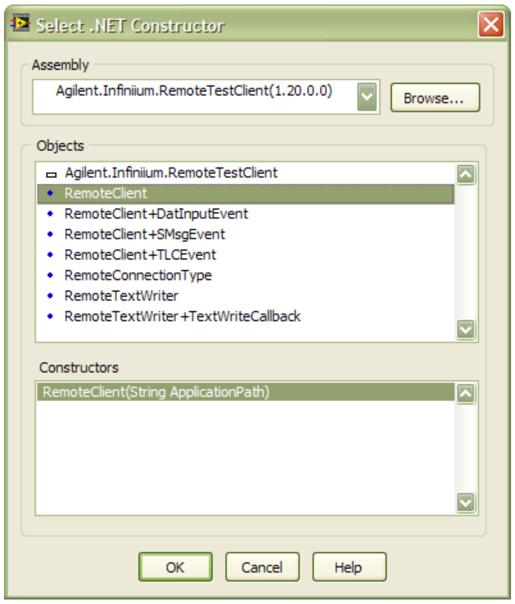

Project Explorer - Untitled Project 2 *
File Edit View Project Operate Tools Window Help
Items Files
Project Items
Paths
Project: Untitled Project 2
My Computer
Agilent.Infiniium.RemoteTestClient.dll
Agilent.Infiniium.AppFW.Remote.dll
Dependencies
Build Specifications
C:\Labview\Agilent.Infiniium.RemoteTestClient.dll
C:\Labview\Agilent.Infiniium.AppFW.Remote.dll

5 Add a new VI to the project.

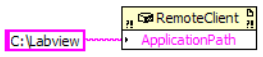

Project Explorer - Untitled Project 2 *
File Edit View Project Operate Tools Window Help
Items Files
Project Items
Paths
Project: Untitled Project 2
New
Add
Arrange by
Expand All
Collapse All
Help...
Properties
VI
Virtual Folder
Control
Library
Variable
Class
XControl
New...
C:\Labview\Agilent.Infiniium.RemoteTestClient.dll
C:\Labview\Agilent.Infiniium.AppFW.Remote.dll

6 Add a Constructor node to the Block Diagram of the VI and select the RemoteClient (String ApplicationPath) constructor from the Keysight.DigitalTestApps.RemoteTestClient.dll (or Agilent.Infiniium.RemoteTestClient.dll).

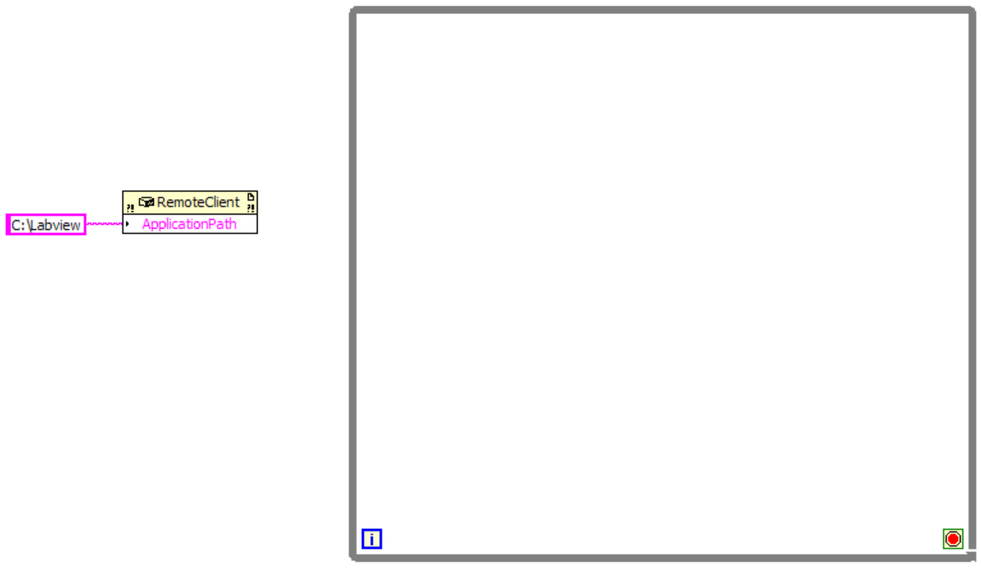



Select .NET Constructor
Assembly
Agilent.Infiniium.RemoteTestClient(1.20.0.0)
Browse...
Objects
Agilent.Infiniium.RemoteTestClient
RemoteClient
RemoteClient+DatInputEvent
RemoteClient+SMsgEvent
RemoteClient+TLCEvent
RemoteConnectionType
RemoteTextWriter
RemoteTextWriter+TextWriteCallback
Constructors
RemoteClient(String ApplicationPath)
OK
Cancel
Help

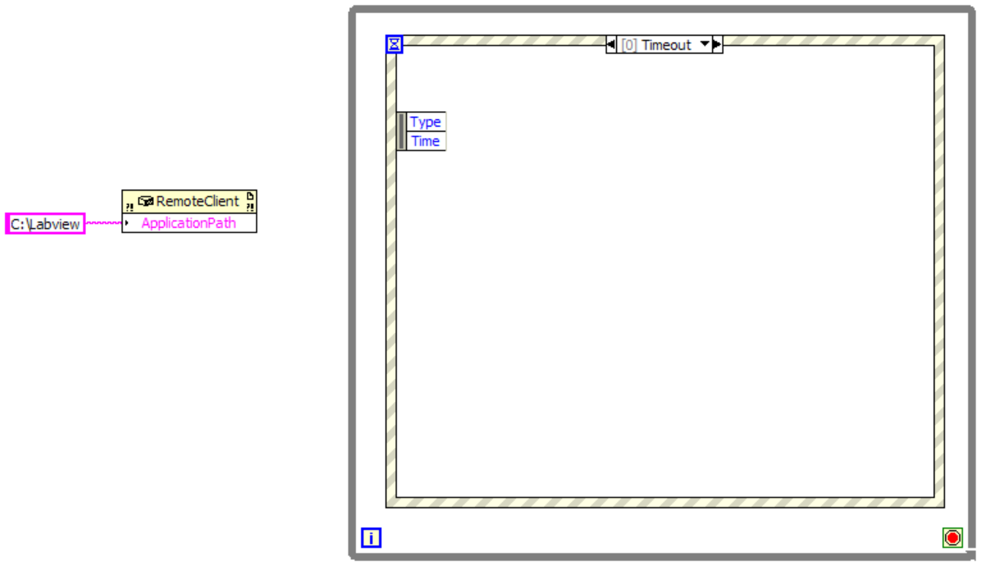
7 Add a string constant and link it to the constructor created earlier. The string constant value must be pointing to the directory where Keysight.DigitalTestApps.RemoteTestClient.dll.config (or Agilent.Infiniium.RemoteTestClient.dll.config) is located. In the case of this example it will be in C:\Labview as it copied there in step 1).



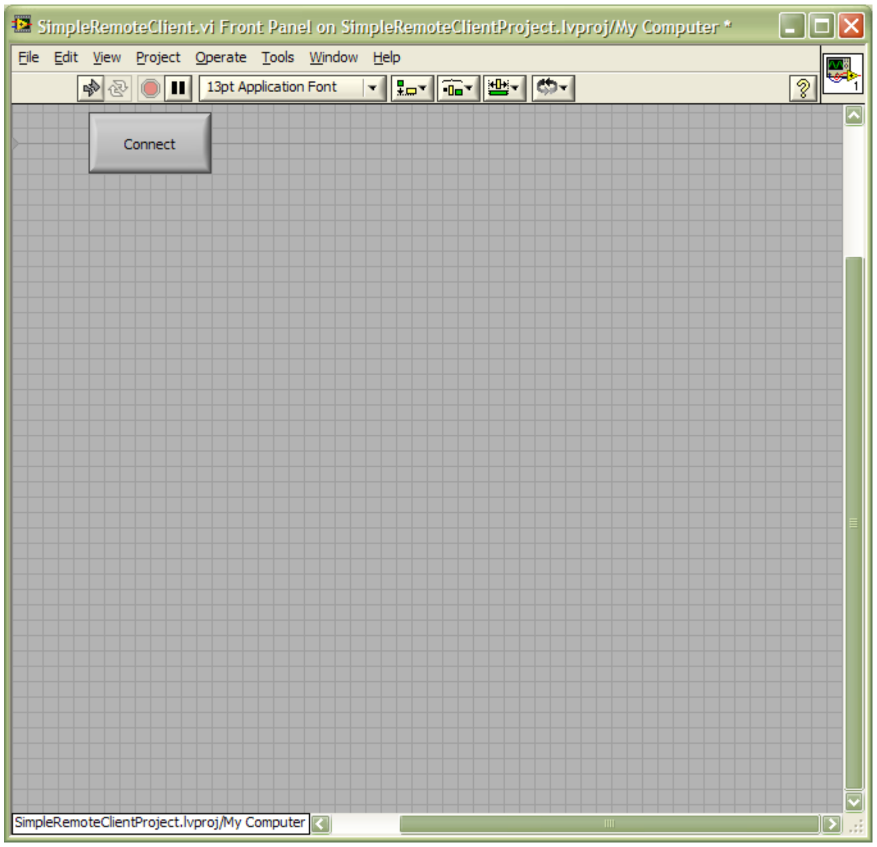
8 Add a While Loop node to the block diagram of the VI.



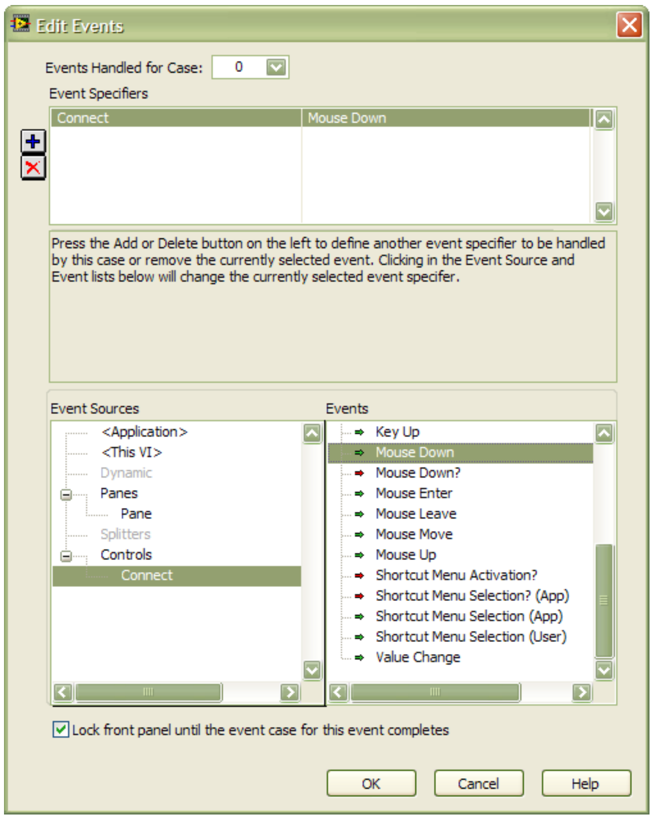
9 Add a Event Structure node into the While Loop node added in step 8.



10 Add an Ok Button to the Front Panel. Rename the button label to "Connect".


SimpleRemoteClient.vi Front Panel on SimpleRemoteClientProject.lvproj/My Computer *
File Edit View Project Operate Tools Window Help
13pt Application Font
Connect
SimpleRemoteClientProject.lvproj/My Computer

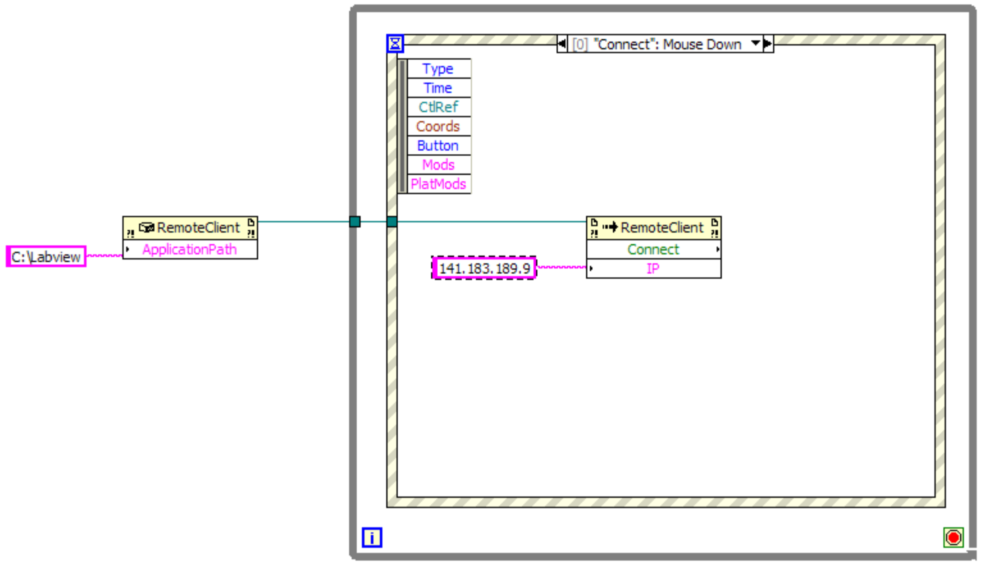
11 Edit the Timeout event in the Block Diagram to wait for a Mouse Down on the newly added Connect button.



Press the Add or Delete button on the left to define another event specifier to be handled by this case or remove the currently selected event. Clicking in the Event Source and Event lists below will change the currently selected event specifer.

☑ Lock front panel until the event case for this event completes

12 Add a Invoke node and select the Connect(String IP) method. Create a string constant with the IP of the scope where the automated test application is running.

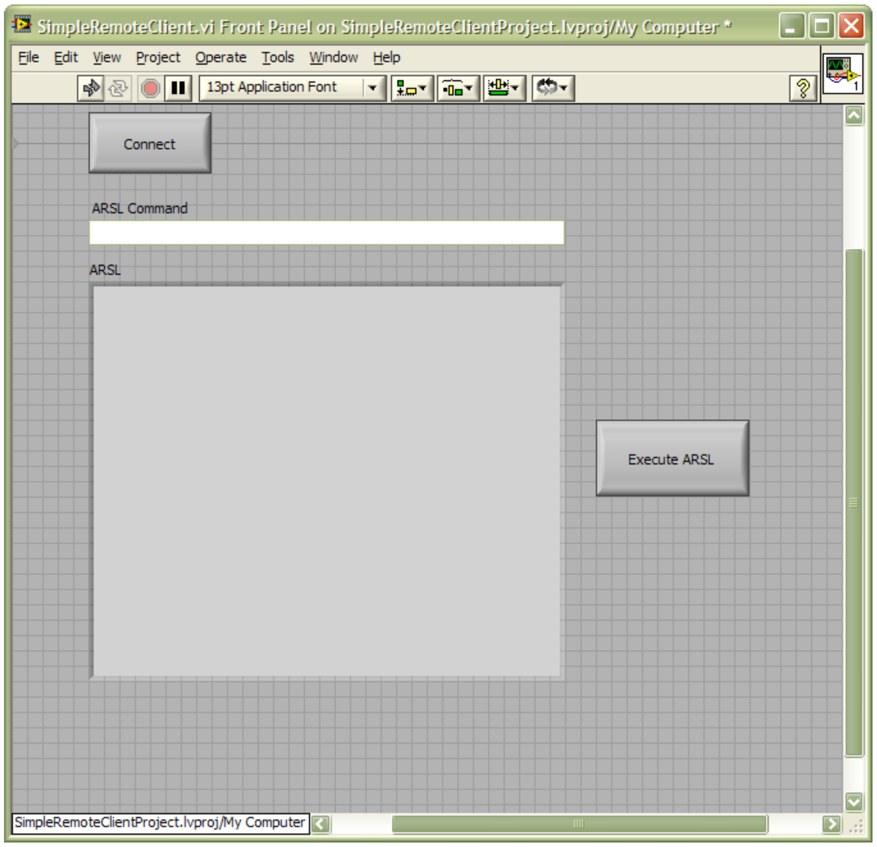


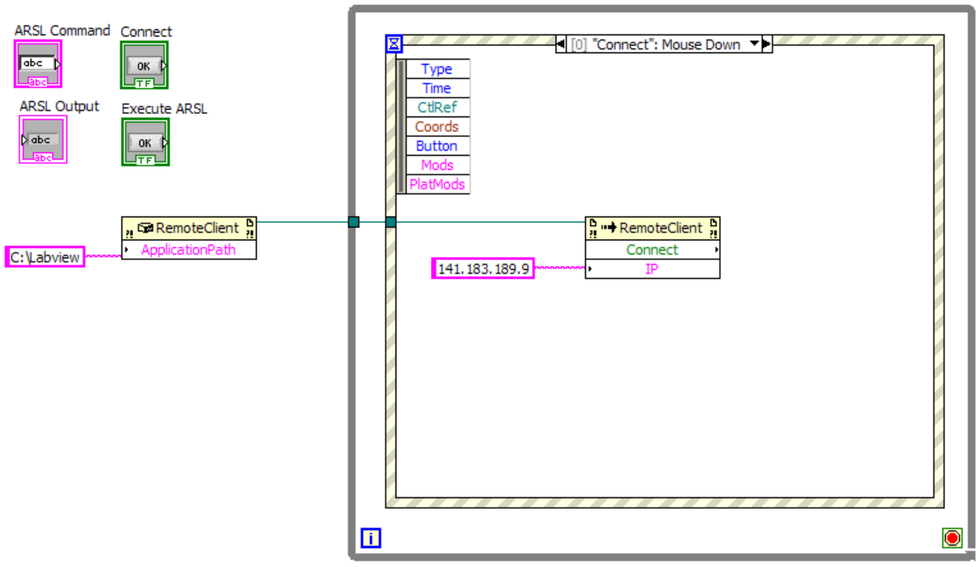
13 Create the following interface on the Front Panel using:

a One Ok Button

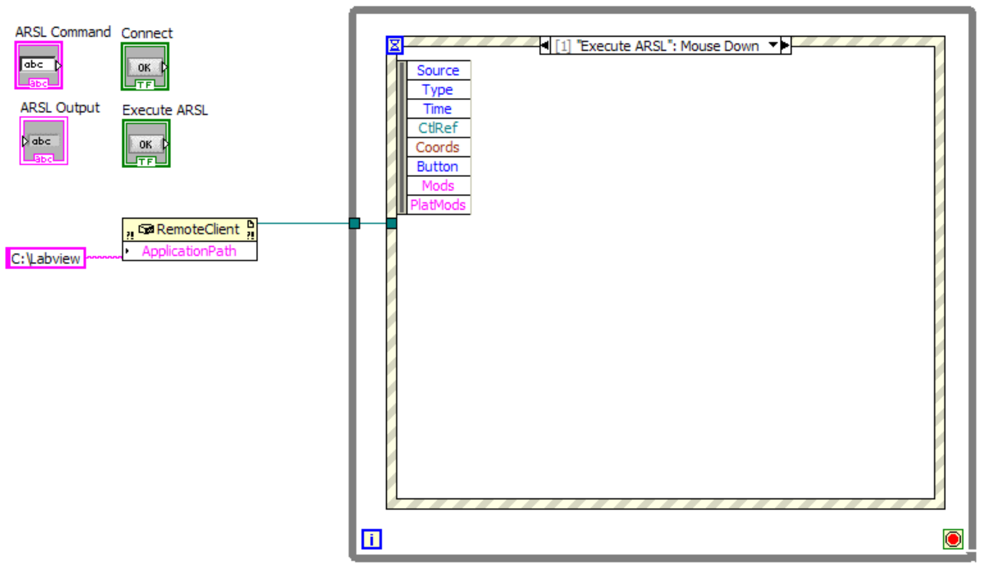
b One String Control

c One String Indicator

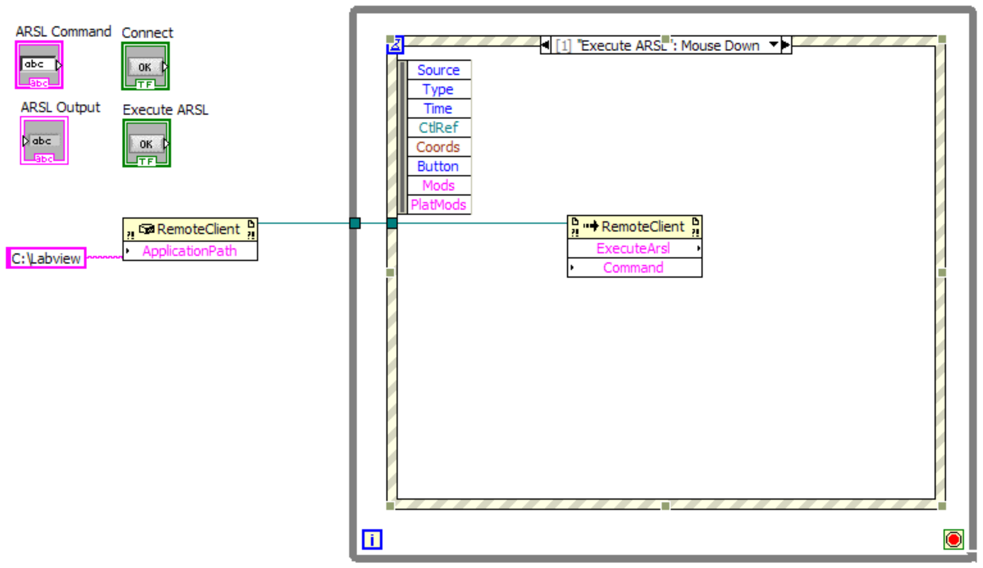

SimpleRemoteClient.vi Front Panel on SimpleRemoteClientProject.lvproj/My Computer *
File Edit View Project Operate Tools Window Help
13pt Application Font
Connect
ARSL Command
ARSL
Execute ARSL
SimpleRemoteClientProject.lvproj/My Computer



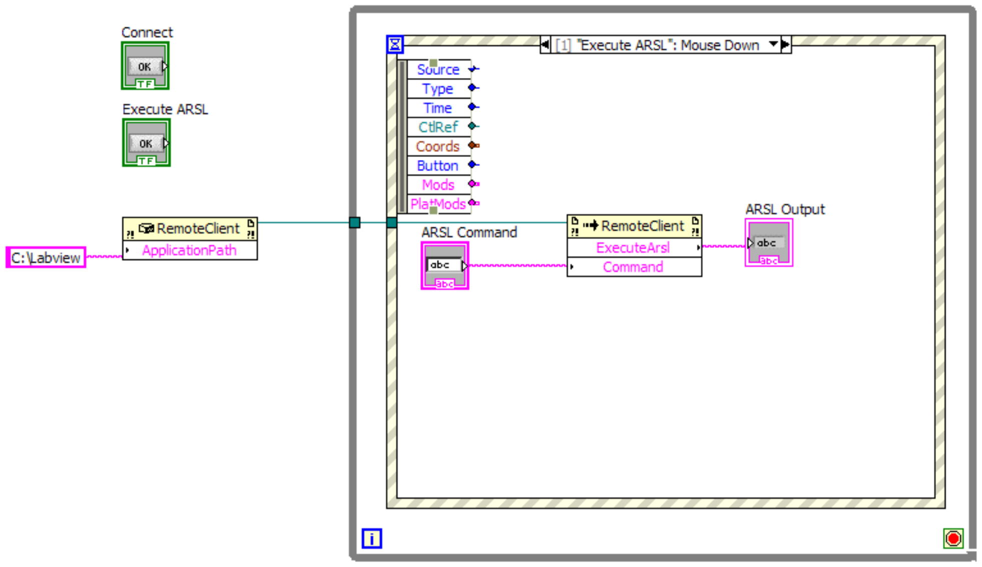
14 Add an event for the Execute ARSL button as you did for the Connect button.



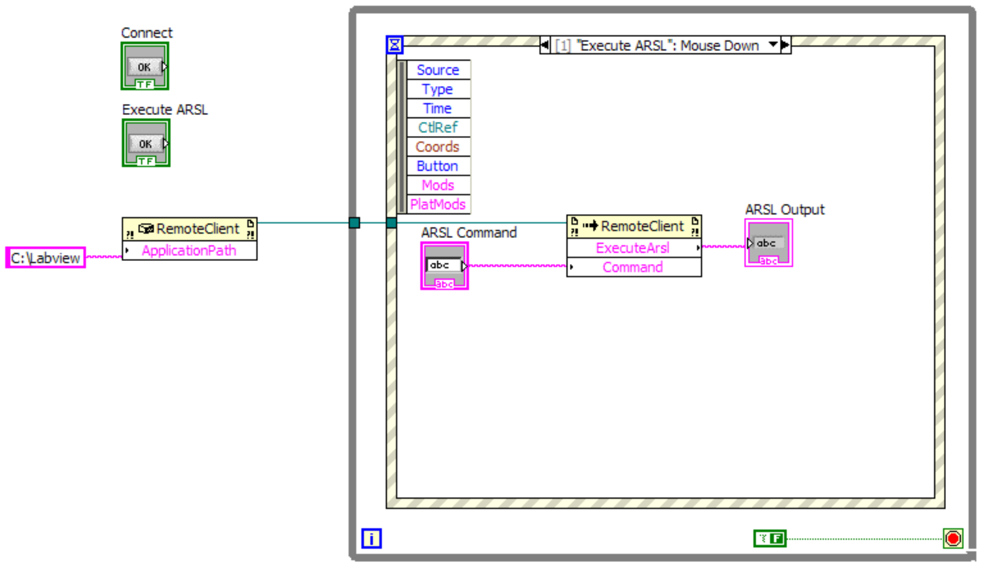
15 Add an Invoke node to the block diagram and select the Execute ARSL method.



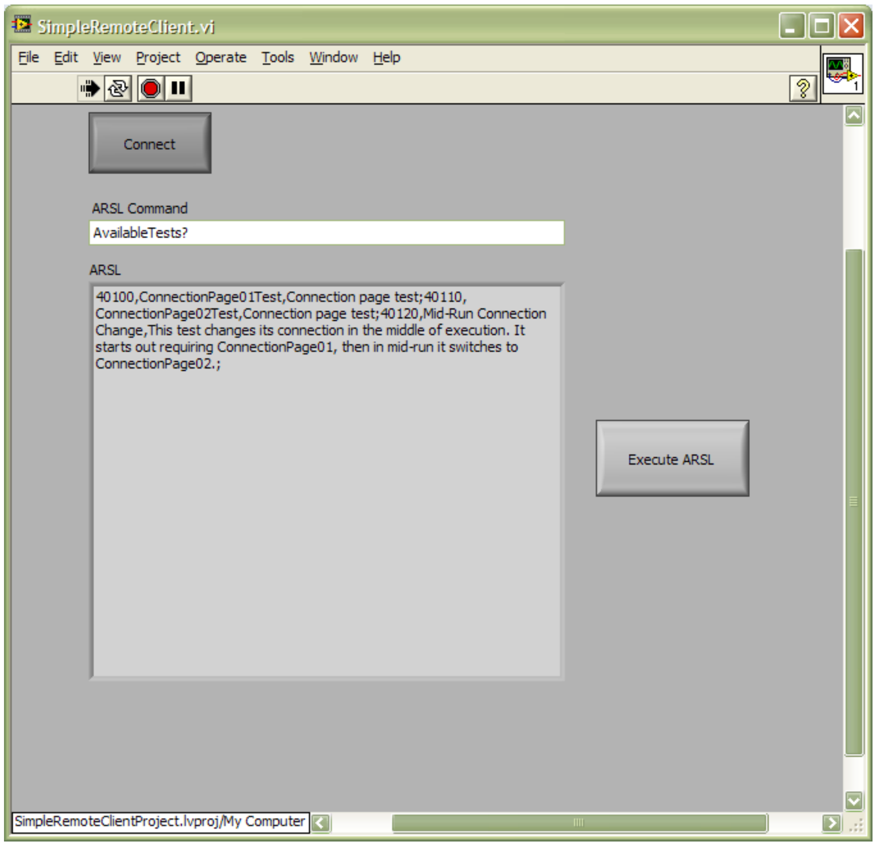
16 Link the ARSL Command and ARSL Output nodes to the ExecuteARSL node as shown below.



17 Add a False Constant node and link it to the While node Loop Condition as shown below.



18 Bring up the Front Panel VI and run it to test this example. Click on the **Connect** button and enter into the ARSL Command text "Available Tests?" minus the quotes. Click on the **Execute ARSL** button once the command has been entered.



ARSL Command

```
AvailableTests?
```

ARSL

```
40100,ConnectionPage01Test,Connection page test;40110,
ConnectionPage02Test,Connection page test;40120,Mid-Run Connection
Change,This test changes its connection in the middle of execution. It
starts out requiring ConnectionPage01, then in mid-run it switches to
ConnectionPage02.;
```

SimpleRemoteClientProject.lvproj/My Computer

19 If the connection and execution was successful, the list of available tests should be displayed in the ARSL Output Indicator.

LabVIEW Demo Client Construction

This section describes how to create your own remote client using the LabVIEW programming environment. In addition to requiring the Keysight.DigitalTestApps.Framework.Remote.dll file (or Agilent.Infiniium.AppFW.Remote.dll, see "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide for a list of minimum requirements), LabVIEW clients use an interface adapter assembly, "Keysight.DigitalTestApps.RemoteTestClient.dll" (or Agilent.Infiniium.RemoteTestClient.dll). This file is also provided in this Toolkit in the desired Tools\LabVIEW\LabVIEW Remote Adapter subdirectory.

NOTE

Please see the readme.txt file found in the "Agilent-Keysight Transition" or "Keysight Apps only" subdirectories for guidance on the files to use for your remote client.

The following topics describe how to create your own remote LabVIEW client. For a detailed description of the remote API for LabVIEW, see the documents described in "Remote Interface Documentation" on page 10.

NOTE

In order to import and use the Keysight.DigitalTestApps.RemoteTestClient.dll (or Agilent.Infiniium.RemoteTestClient.dll), all vi's that reference or use the dll need to be part of a Labview project. Once a project has been created, add a new or existing vi to it so the Keysight.DigitalTestApps.RemoteTestClient.dll (or Agilent.Infiniium.RemoteTestClient.dll) can be referenced correctly.

Creating a Remote Client

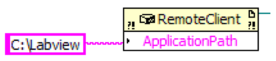


Figure 1 Remote Client Constructor

Insert a .Net Constructor node from the menu selection. Select RemoteClient(string) constructor from Keysight.DigitalTestApps.RemoteTestClient.dll (or Agilent.Infiniium.RemoteTestClient.dll). Supply to the constructor the application path which is a String.

Registering an Event

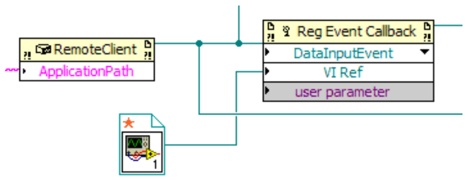


Figure 2 Registering an Event

Insert a .Net Register Event Callback node from the menu selection. There are three events available for registration:

1 DataInputEvent.

2 TestListEvent.

3 SimpleMessageEvent.

Select which event to register to from the drop down, and create a callback vi.

TestList Event

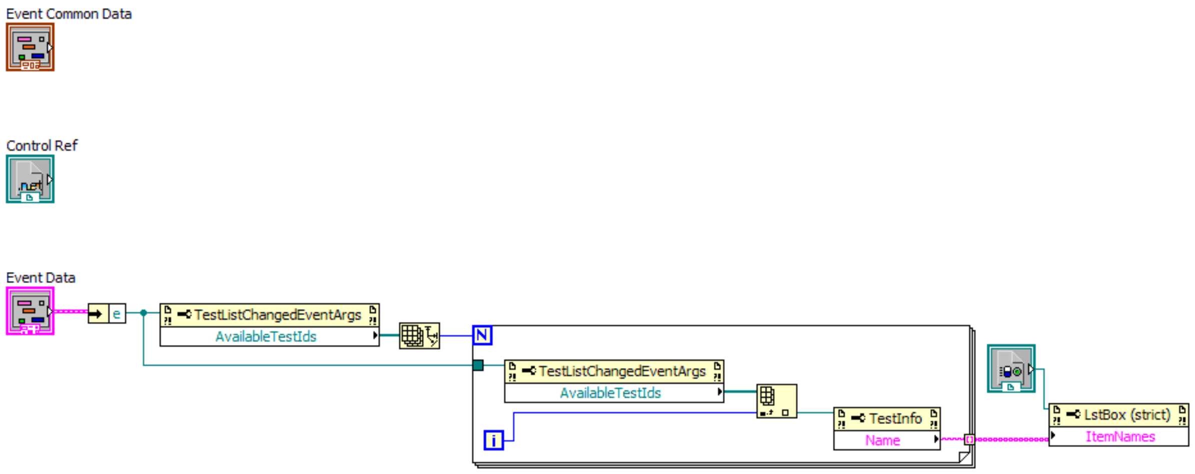


Figure 3 TestList Changed Event

The TestListChangedEventArgs parameter is passed during the TestListEvent. The example above shows how the values in a List box are updated with the names of the Available Tests. Please refer to *Keysight DigitalTestApps Remote Interface for LabVIEW* help file for more details on the TestListChangedEventArgs properties.

SimpleMessage Event

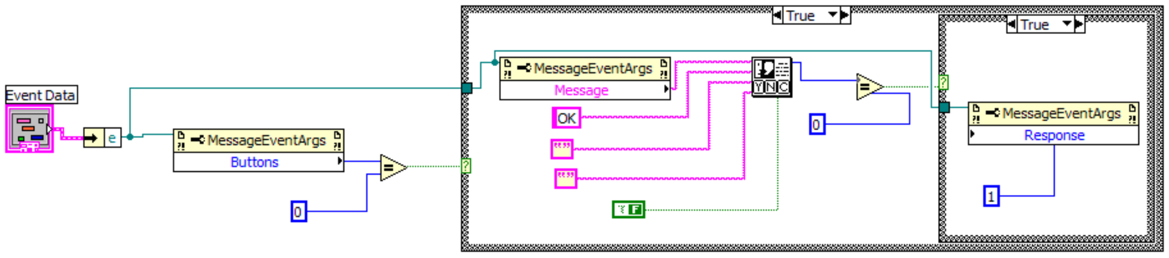


Figure 4 SimpleMessage Event

The MessageEventArgs parameter is passed during the SimpleMessageEvent. The example above shows how a single "OK" button message is handled. Different messages may have different expected return values. Please refer to the *Keysight DigitalTestApps Remote Interface for LabVIEW* help file for more details on the MessageEventArgs properties.

DataInput Event

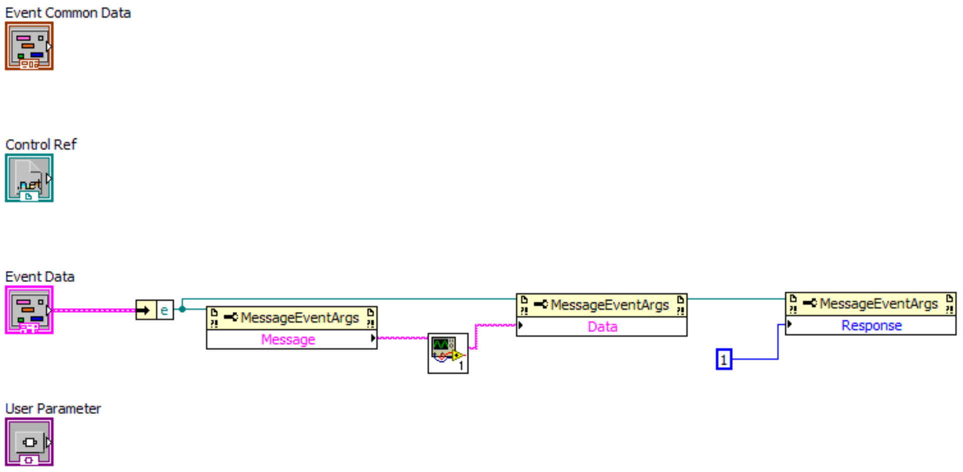


Figure 5 DataInput Event

The MessageEventArgs parameter is passed during the DataInputEvent. The example above shows a MessageEventArgs being passed to the Inputprompt.vi for user input. When the user has entered the data and clicked the "OK" button, the response will be sent back to the application.

Different messages may have different expected return values. Please refer to the *Keysight DigitalTestApps Remote Interface for LabVIEW* help file for more details on the MessageEventArgs properties.

Loading a Project

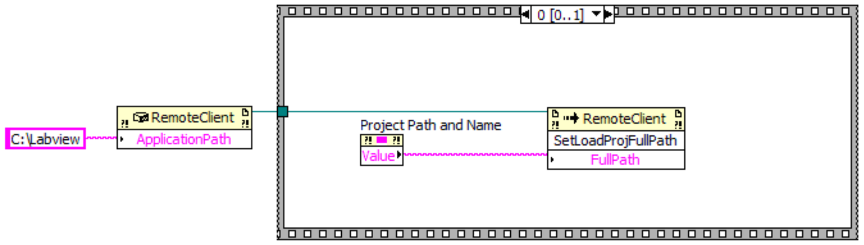


Figure 6 Loading a Project Part A

Insert an Invoke node from the menu and select "SetLoadProjFullPath(String)" as the method. Supply the method the full path with the project name.

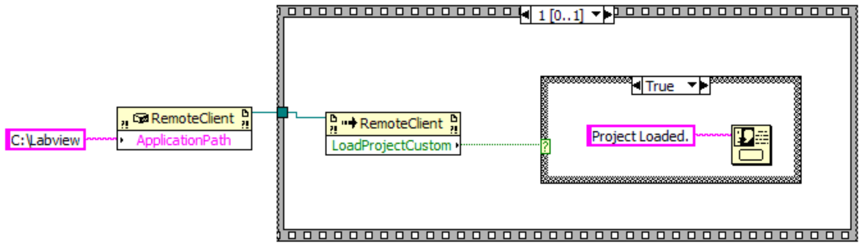


Figure 7 Loading a Project Part B

Invoke "LoadProjectCustom" once the project full path has been supplied.

Discarding Unsaved Changes

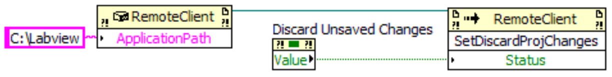


Figure 8 Discard Unsaved Changes

To discard unsaved changes when loading a project, invoke the "SetDiscardProjChanges(bool)" before invoking the "LoadProjectCustom" method.

Saving a Project

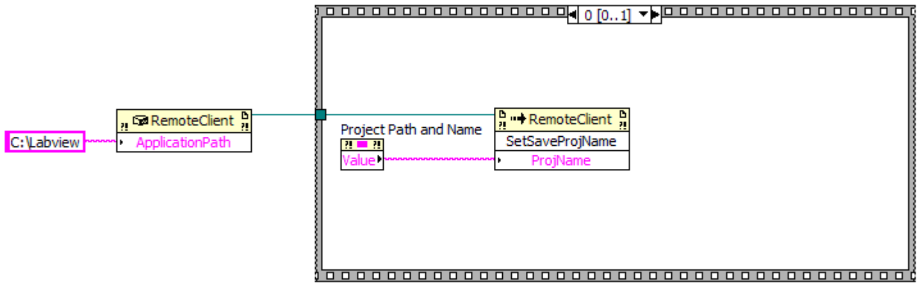


Figure 9 Saving a Project Part A

Insert an Invoke node from the menu and select "SetSaveProjName(string)" as the method. Supply the method the full path with the project name.

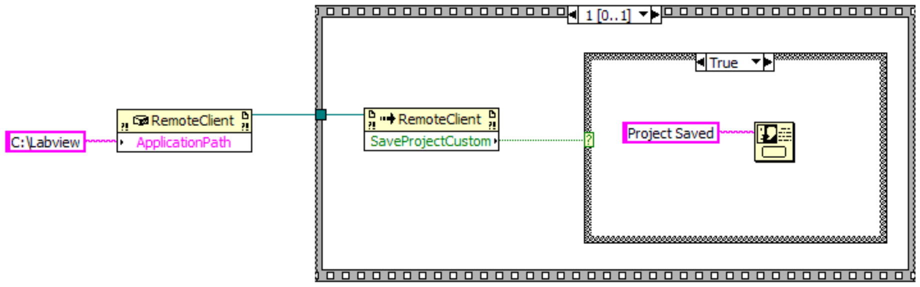


Figure 10 Saving a Project Part B

Invoke "SaveProjectCustom" once the full path has been supplied.

Setting a Config

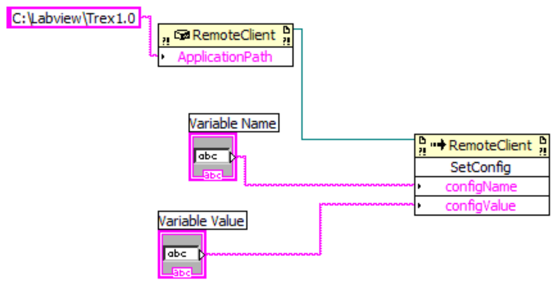


Figure 11 Setting a Configuration

The example above shows how to Set a Configuration value. Insert an Invoke node from the menu and select SetConfig(string,string) as the method. Supply the Variable Name and Variable Value to set the configuration.

Overwriting an Existing Project

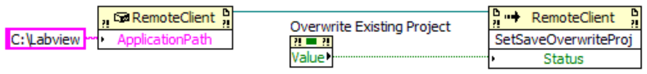


Figure 12 Overwrite an Existing Project

To overwrite an existing project during a save, invoke "SetSaveOverwriteProj(bool)" method with a Boolean value before invoking the "SaveProjectCustom" method.

Getting a Config

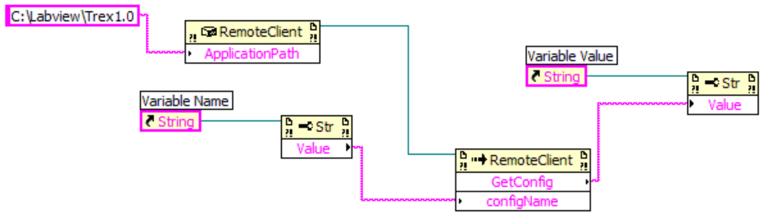


Figure 13 Getting a Configuration Value

The example above shows how to Get a Configuration value. Insert an Invoke node from the menu and select GetConfig(string) as the method. Supply the Variable Name to get the value for that variable.

Connection Prompt Action

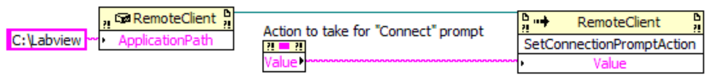

C:\Labview
RemoteClient
ApplicationPath
Action to take for "Connect" prompt
Value
RemoteClient
SetConnectionPromptAction
Value

Figure 14 Connection Prompt Action

When manual connection changes are required by the remote application, the user will be prompted. To select the response to these connection change prompts, invoke "SetConnectionPromptAction(String)" method with either "AutoRespond", "Display", or "Abort".

Signal Check Fail Action

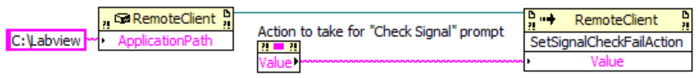


Figure 15 Signal Check Fail Action

Certain applications perform a series of checks before running a test to ensure that the signal provided is correct. In cases where the signal provided does not meet the criteria, a prompt will occur. To select the response to these connection change prompts, invoke "SetSignalCheckFailAction(String)" method with either "AutoRespond", "Display", or "Abort".

Running Tests

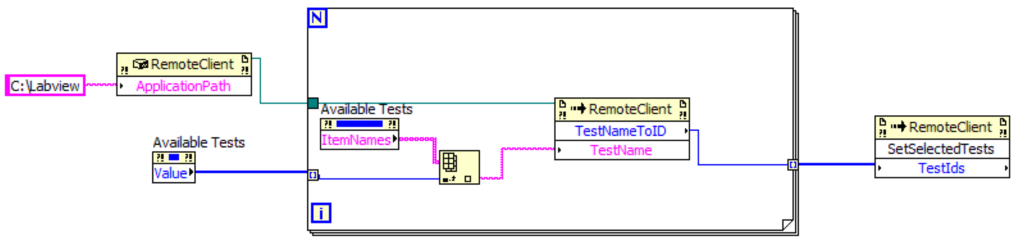


Figure 16 Running Tests Part A

The example above shows how to run a set of tests. Insert an Invoke node from the menu and select TestNameToID(string) as the method. Supply the test name to the method and the corresponding TestID will be returned. Pass the array of TestIDs to the "SetSelectedTests()" function.

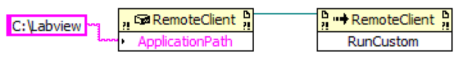


Figure 17 Running Tests Part B

After that, invoke "RunCustom()" to execute the tests.

Increasing Number of Trials

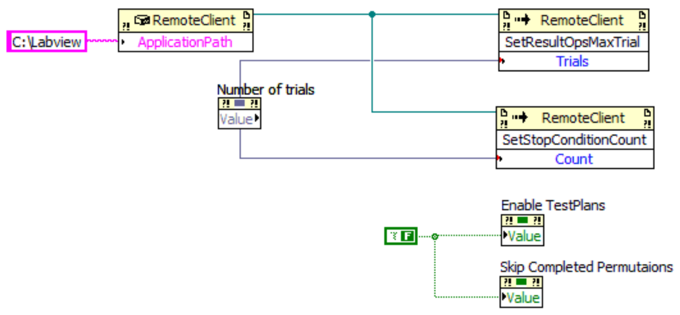


Figure 18 Increasing number of trials

The above example shows how to increase the number of trials to run each time the tests are executed (as well as the number of trials that are retrieved when obtaining results). Invoke the "SetStopConditionCount(int Trials)" method and set it to the number of trials each test is to be executed. Invoke "SetResultOpsMaxTrial(int Count)" to set the number of trials results to return. Note that when more than one trial is selected, the Test Plan feature is disabled (and the LabVIEW interface is updated).

Enabling Test Plan Feature

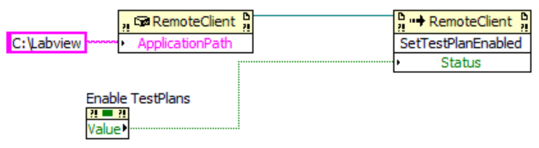


Figure 19 Enable Test Plans

The above example shows how to enable the Test Plan feature. The feature is only available if the remote application supports it. Invoke the "SetTestPlanEnabled(bool Status)" method and pass it a boolean value.

Skipping Completed Permutations

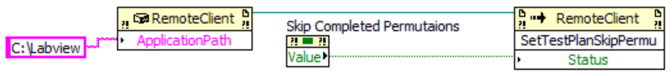


Figure 20 Skipping Test Plan Completed Permutations

The example above shows how to the application can be made to skip completed permutations in the Test Plan feature. Invoke the "SetTestPlanSkipPermu(bool status)" method and pass it a Boolean value.

Execute ARSL

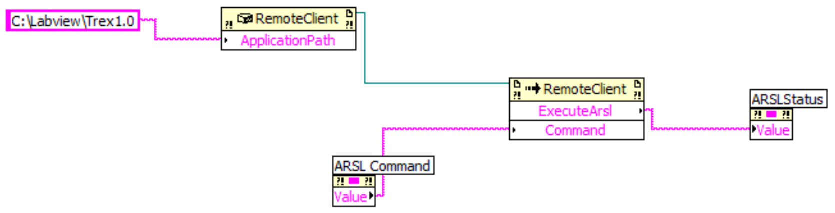


Figure 21 Executing ARSL Commands

The example above shows how to execute an ARSL command. Insert an Invoke node from the menu and select ExecuteARSL(string) as the method. Supply the ARSL command string to the method which will return the Status/Value as a String.

Connect

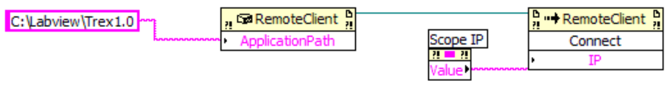


Figure 22 Connecting to the Application

The example above shows how to connect to the automated test application. Insert an Invoke node from the menu, and select Connect(string) as the method. Supply the IP address of the oscilloscope where the automated test application is running.

Appending/Replacing Test Results

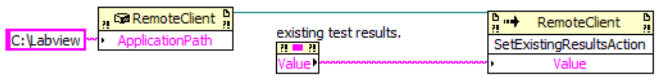


Figure 23 Appending/Replacing Test Results

The user is given a choice to either replace the test results of previous runs with the new run or append the previous test results with the new run. To do so, invoke the "SetExistingResultsAction" method with either "Replace" or "Append" as the parameter.

Getting Results

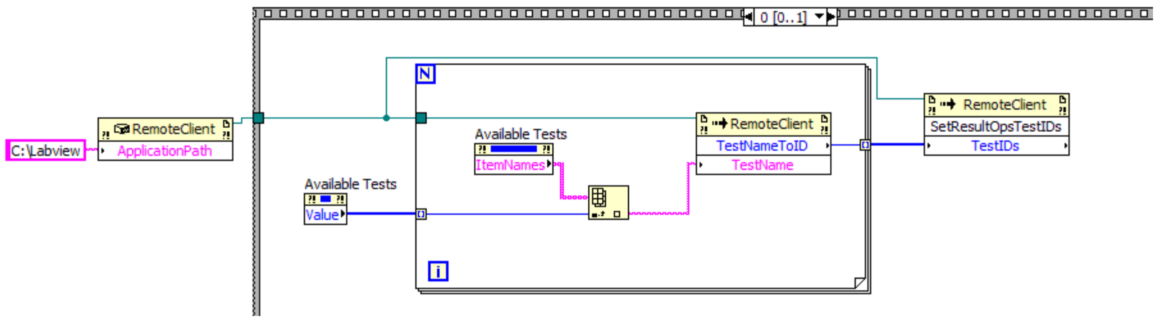


Figure 24 Getting Results Part A

The example above shows how to get test results for selected tests from the listbox. First, get the selected test names from the list box and pass them to the "TestNametoID" method to get the Test IDs. Next, pass the list of Test IDs to the "SetResultOpsTestIDs" method.

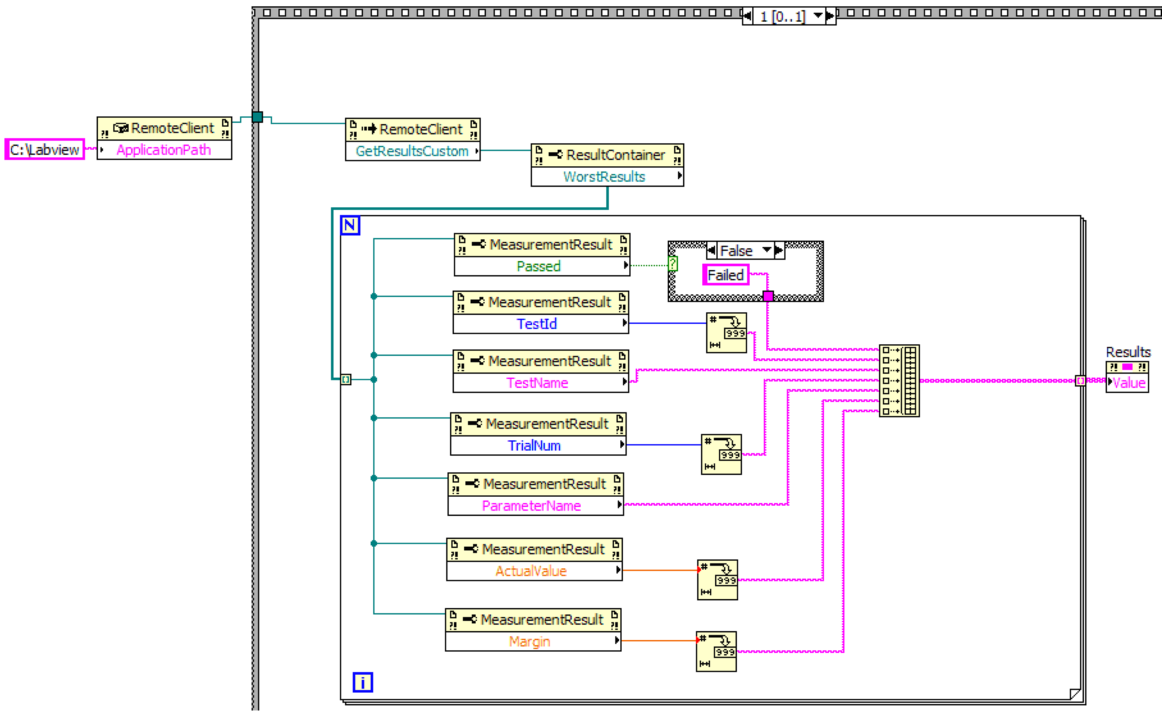


Figure 25 Getting Results Part B

Invoke "GetResultsCustom" which will return a "ResultContainer" object that has the property "WorstResults". The "WorstResults" property contains several details for the test results. Please refer to the *Keysight DigitalTestApps Remote Interface for LabVIEW* help file for more details on the various properties used by the Results.

Delete Results

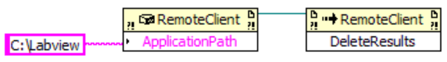


Figure 26 Deleting Results

The example above shows how to delete the results for all tests. Insert an Invoke node and select DeleteResults().

Refreshing the Test List

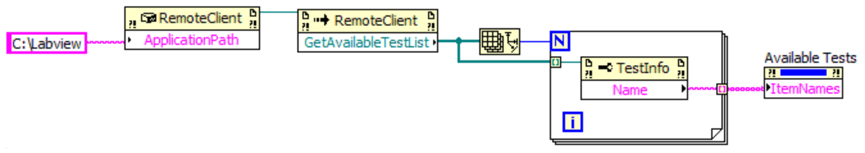


Figure 27 Getting Available Tests

The example above shows how to get the list of available tests. Insert an Invoke node and select GetAvailableTestList(). It will return an array of TestInfo objects. From the array of tests, extract the names of the tests to populate in the list box. Please refer to *Keysight DigitalTestApps Remote Interface for LabVIEW* help file.

Updating Your Automated Test Application

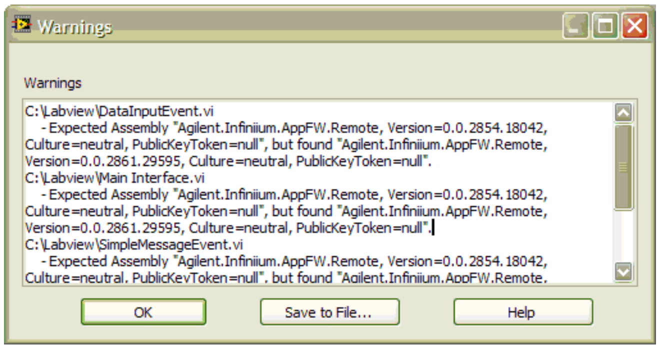
From time to time, automated test application updates are released. When this occurs, the associated remote interface version may be different as well (for example, in order to provide new functionality). Recall that you can use the ARSL Command Line Utility (see "ARSL Command Line Utility" on page 18) to quickly determine the remote interface version used by an application. If it has changed, you need to update these two files in your LabVIEW directory in order to use the new functionality:

1. Keysight.DigitalTestApps.Framework.Remote.dll (or Agilent.Infiniium.AppFW.Remote.dll)
2. Keysight.DigitalTestApps.RemoteTestClient.dll (or Agilent.Infiniium.RemoteTestClient.dll)

For more information, see Chapter 2, “Using Sample Remote Clients,” starting on page 17.

Use the following steps to update these files in your LabVIEW project:

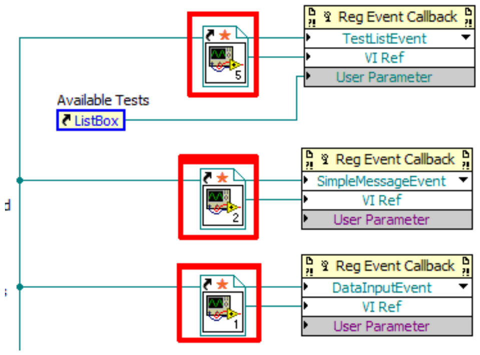
1. Remove all copies of the dlls from C:\LabView and any subfolders within them.
2. Place the new copies of the dlls in C:\LabView.
3. Launch the example as described in "LabVIEW Demo Remote Client" on page 31. If you receive the following prompt during the update, click OK to proceed:



Warnings

```
C:\Labview\DataInputEvent.vi
   - Expected Assembly "Agilent.Infiniium.AppFW.Remote, Version=0.0.2854.18042,
Culture=neutral, PublicKeyToken=null", but found "Agilent.Infiniium.AppFW.Remote,
Version=0.0.2861.29595, Culture=neutral, PublicKeyToken=null".
C:\Labview\Main Interface.vi
   - Expected Assembly "Agilent.Infiniium.AppFW.Remote, Version=0.0.2854.18042,
Culture=neutral, PublicKeyToken=null", but found "Agilent.Infiniium.AppFW.Remote,
Version=0.0.2861.29595, Culture=neutral, PublicKeyToken=null".
C:\Labview\SimpleMessageEvent.vi
   - Expected Assembly "Agilent.Infiniium.AppFW.Remote, Version=0.0.2854.18042,
Culture=neutral, PublicKeyToken=null", but found "Agilent.Infiniium.AppFW.Remote,
```

4 Open each of the sub-VIs that are marked in the red boxes and resave them.



5 Save the main VI

6 Save the project and exit.

6 Troubleshooting

Error Messages and Resolution / 142
How to Check Which Port an Application is Using / 145
How to Configure a Remote Interface User Port / 147
If your PC Has Two Network Interface Cards / 148
How To Test Two-way .NET Remoting Between Your PC and Your Scope / 149

The best place to start when troubleshooting remote interface difficulties is to verify the machine running the automated test application (oscilloscope or PC) is set up properly. The easiest way to do this is to run the ARSL command line utility on the remote computer and see if you can send the remote query "ApplicationName?" to the automated test application. See "ARSL Command Line Utility" on page 18 for instructions on using this utility.

Error Messages and Resolution

Here are some common error messages and resolution steps:

Could not load file or assembly 'Keysight.DigitalTestApps.Framework.Remote'

Check:

1 Ensure a copy of the file Keysight.DigitalTestApps.Framework.Remote.dll is placed in the same directory as the remote client executable (see "On the Client" in the *Keysight DigitalTestApps Programming Getting Started* guide).

The remote interface of the target application has been disabled by local user

Check:

1 The automated test application running on the target machine (oscilloscope or PC) must have its remote interface enabled (see **View->Preferences**::**Remote** tab). Another user may have manually disabled the remote interface to protect a critical and/or long-duration test running on the target machine from being accidentally interrupted by a remote user.

A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond

Check:

1 The Windows Firewall on the machine running the automated test application must have these port exceptions defined (this is automatically taken care of during the automated test app's installation):

```
Legacy (Agilent) apps require ports 9944 and 49944
Keysight apps require ports 49944 and 59944
Various names may exist for these exceptions (differences
  are unimportant).
```

The network address is invalid, No connection could be made because the target machine actively refused it

Check:

1 The IP address you provided should be the address of the target machine running the automated test application (oscilloscope or PC). Ensure the remote client computer can see this machine.

2 A remote-capable automated test application must be installed and running on the target machine.

3 (Real-time 9xxx and 9xxxx Series Infiniium versions 3.19 and older only, Remote Interface versions 1.90 and older only) An automated test application remote control license must be installed on the oscilloscope. You can verify by checking the Infiniium **Help>About Infiniium** dialog and looking for the name "App Remote" in the list of installed options.

4 The automated test application must have activated a remote interface network port (for instructions, see "How to Check Which Port an Application is Using" on page 145). If it activated a user-defined port, you must access the application using the helper method RemoteAteUtilities.GetRemoteAteCustom() with option UseCustomPort set to true. The helper method RemoteAteUtilities.GetRemoteAte() can only access applications via the two standard ports.

The target application did not respond

Do the following::

1 Close the automated test application.

2 Configure a user-defined remote interface port on the target machine (for instructions, see "How to Configure a Remote Interface User Port" on page 147).

3 Restart the automated test application and verify it has enabled its remote interface (for instructions, see "How to Check Which Port an Application is Using" on page 145).

4 In your remote client, access the automated test application using the helper method RemoteAteUtilities.GetRemoteAteCustom() with option UseCustomPort set to true.

The target machine has an invalid user port entry

Check:

1 When using RemoteAteUtilities.GetRemoteAteCustom() with option UseCustomPort set to true, the target machine running the automated test application must have a user-defined remote interface port (for instructions, see "How to Configure a Remote Interface User Port" on page 147).

Method not found

Check:

1 The client found a remote-capable automated test application running on the target machine (oscilloscope or PC), but the remote interface property or method you attempted to use was not supported by the application. Ensure the property or method is supported in the version of the remote interface version used by the application. The version can be found in the application's **Help>About** dialog (if it is not there, the version is prior to 1.20). Then check the help file Keysight DigitalTestApps Remote Interface for .NET.chm (see "Remote

Interface Documentation" on page 10): each property and method is also annotated with the versions it is supported in.

2 Please visit: www.keysight.com/find/scope-apps to see if there is a newer version of your automated test application available for download from Keysight Technologies.

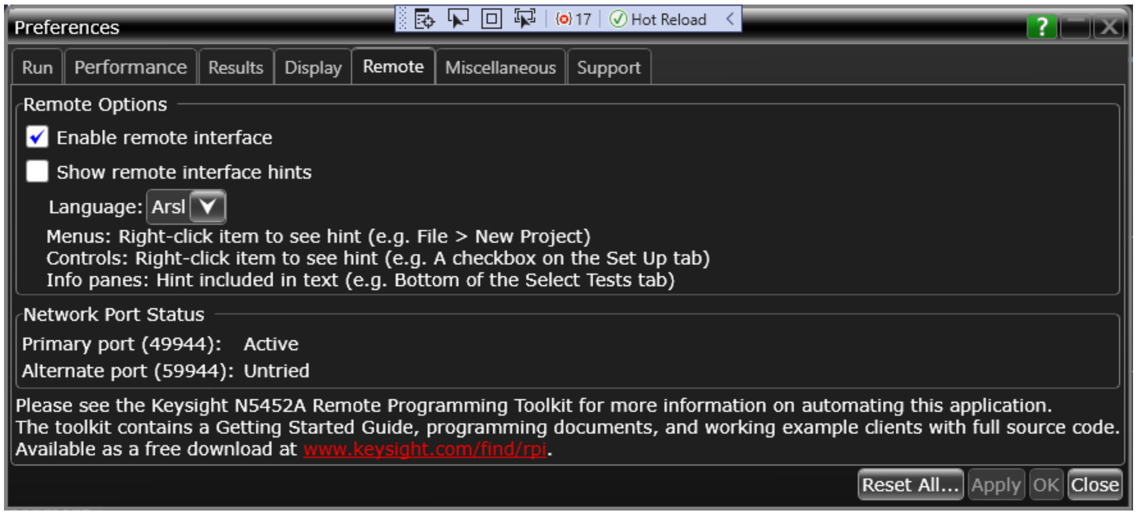
System.Runtime.Remoting.RemotingException: This remoting proxy has no channel sink which means either the server has no registered server channels that are listening, or this application has no suitable client channel to talk to the server.

Check:

1 Ensure a copy of the file Keysight.DigitalTestApps.Framework.Remote.dll.config is placed in the same directory as the Keysight.DigitalTestApps.Framework.Remote.dll.

How to Check Which Port an Application is Using

In the automated test application running on the target machine (oscilloscope or PC), open the **View >Preferences** dialog. On the **Remote** tab, in the Network Port Status group, the port being used by the remote interface subsystem will be marked "Active". If no user-defined remote interface port has been set on the target machine, then when the application launches it will attempt to activate one of the two standard ports:



Remote Options
- [✓] Enable remote interface
- [ ] Show remote interface hints

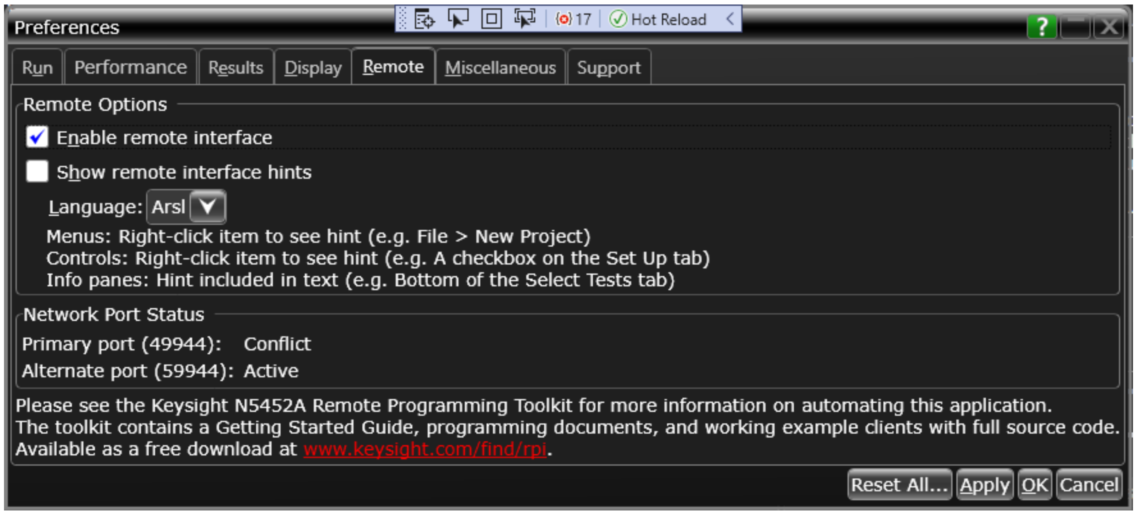
Language: Arsl

Menus: Right-click item to see hint (e.g. File > New Project)
Controls: Right-click item to see hint (e.g. A checkbox on the Set Up tab)
Info panes: Hint included in text (e.g. Bottom of the Select Tests tab)

Network Port Status
Primary port (49944): Active
Alternate port (59944): Untried

Please see the Keysight N5452A Remote Programming Toolkit for more information on automating this application.
The toolkit contains a Getting Started Guide, programming documents, and working example clients with full source code.
Available as a free download at www.keysight.com/find/rpi.

If the primary port is in use by another application, the application will indicate "conflict" and attempt to use the alternate port:



Remote Options
- [✓] Enable remote interface
- [ ] Show remote interface hints

Language: Arsl

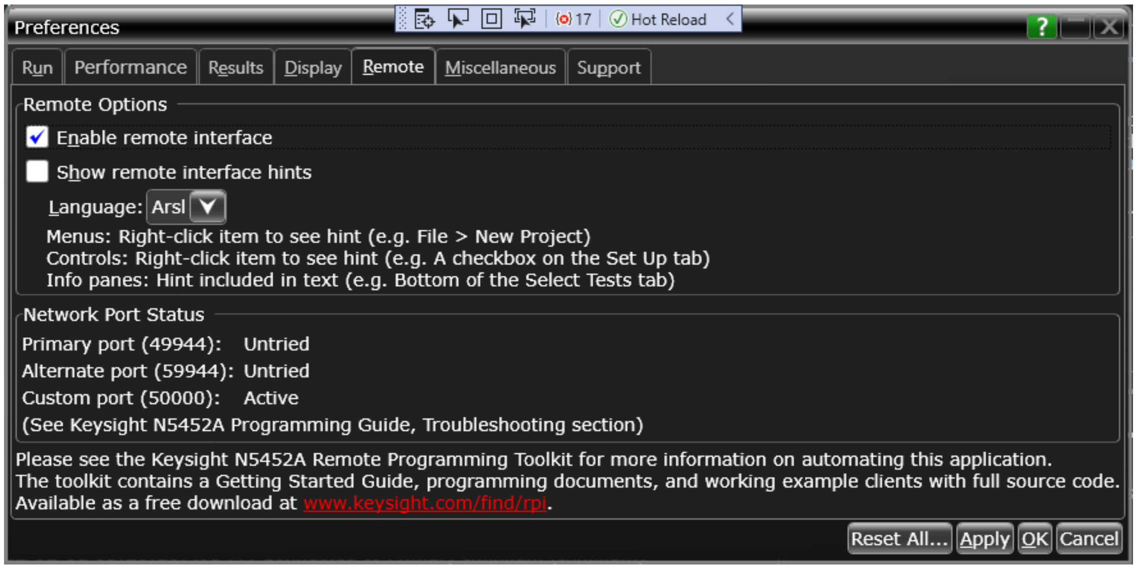
Menus: Right-click item to see hint (e.g. File > New Project)
Controls: Right-click item to see hint (e.g. A checkbox on the Set Up tab)
Info panes: Hint included in text (e.g. Bottom of the Select Tests tab)

Network Port Status
Primary port (49944): Conflict
Alternate port (59944): Active

Please see the Keysight N5452A Remote Programming Toolkit for more information on automating this application.
The toolkit contains a Getting Started Guide, programming documents, and working example clients with full source code.
Available as a free download at www.keysight.com/find/rpi.

To identify which other process is using the active port, try one of the methods described in this article: https://stackoverflow.com/questions/48198/how-can-you-find-out-which-process-is-listening-on-a-tcp-or-udp-port-on-windows

If a user-defined remote interface port (5000 in the example below) has been defined on the target machine, then when the application launches it will attempt to activate it first:



Remote Options

☑ Enable remote interface

☐ Show remote interface hints

Language: Arsl

Menus: Right-click item to see hint (e.g. File > New Project)
Controls: Right-click item to see hint (e.g. A checkbox on the Set Up tab)
Info panes: Hint included in text (e.g. Bottom of the Select Tests tab)

Network Port Status

```
Primary port (49944):   Untried
Alternate port (59944): Untried
Custom port (50000):    Active
(See Keysight N5452A Programming Guide, Troubleshooting section)
```

Please see the Keysight N5452A Remote Programming Toolkit for more information on automating this application.
The toolkit contains a Getting Started Guide, programming documents, and working example clients with full source code.
Available as a free download at www.keysight.com/find/rpi.

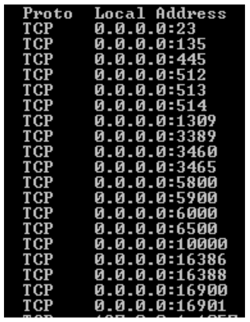
For more information, see "How to Configure a Remote Interface User Port" on page 147.

NOTE When you use the ARSL command line utility –up option, it will connect to the automated test app using RemoteAteUtilities.GetRemoteAteCustom() with option UseCustomPort set to true to. See "ARSL Command Line Utility" on page 18 for instructions on using this utility.

How to Configure a Remote Interface User Port

1 Identify an available network port on the target machine.

On the target machine, in a command window enter: `netstat -aon` (the letter o, not the number 0). The resulting output lists ports currently in use on that machine. In the TCP section, look for a port number in between 49152 and 65535 that does *not* appear in the list. In the example below, you could select any number from that range because none of them are in use:

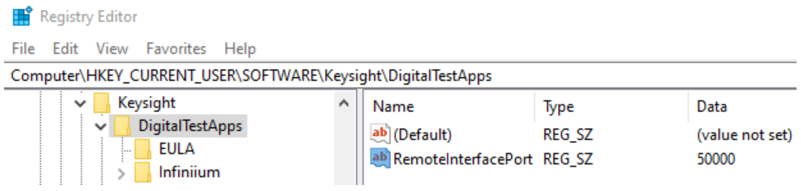


```
Proto  Local Address
TCP    0.0.0.0:23
TCP    0.0.0.0:135
TCP    0.0.0.0:445
TCP    0.0.0.0:512
TCP    0.0.0.0:513
TCP    0.0.0.0:514
TCP    0.0.0.0:1309
TCP    0.0.0.0:3389
TCP    0.0.0.0:3460
TCP    0.0.0.0:3465
TCP    0.0.0.0:5800
TCP    0.0.0.0:5900
TCP    0.0.0.0:6000
TCP    0.0.0.0:6500
TCP    0.0.0.0:10000
TCP    0.0.0.0:16386
TCP    0.0.0.0:16388
TCP    0.0.0.0:16900
TCP    0.0.0.0:16901
```

2 On the target machine, add a string value named "RemoteInterfacePort" to registry key:

HKEY_CURRENT_USER\SOFTWARE\Keysight\DigitalTestApps

Then assign the value of the port number you chose above to RemoteInterfacePort.



Computer\HKEY_CURRENT_USER\SOFTWARE\Keysight\DigitalTestApps

Name	Type	Data
(Default)	REG_SZ	(value not set)
RemoteInterfacePort	REG_SZ	50000

3 On the target machine, add a TCP port rule/exception (for the port number you chose above) to the Windows Firewall.

4 On the target machine, ensure the "Remote Registry" service is running.

NOTE When you use the ARSL command line utility –up option, it will connect to the automated test app using RemoteAteUtilities.GetRemoteAteCustom() with option UseCustomPort set to true to. See "ARSL Command Line Utility" on page 18 for instructions on using this utility.

If your PC Has Two Network Interface Cards

The remote interface uses Microsoft .NET Remoting technology. When a remote client (optionally) activates the callback path (see "Keysight.DigitalTestApps.Framework.Remote.Advanced" on page 14), .NET Remoting by default uses the IP address of the *primary* NIC configured in the client's operating system.

You have three options:

- Use the `RemoteAteUtilities.CreateAteEventSinkCustom` method, which will automatically select a NIC that is on the same subnet as the automated test application you are controlling. For more information on configuring the callback path, see "Message Handling Remote Client Implementation" on page 108. For more information on the `CreateAteEventSinkCustom` method, see the help file: Keysight DigitalTestApps Remote Interface for .NET.

  -OR-

- Add this property to your client's Keysight.DigitalTestApps.Framework.Remote.config file to tell .NET Remoting to use your private network IP address for compliance app callbacks:

  ```
  <channel ref="tcp" name="ATECallbackTCP" port="49946" machineName="196.168.0.2">
  ```

  -OR-

- Make the required NIC primary. Here is a website that describes how to assign which NIC on your client is considered by the OS to be primary:

  http://theregime.wordpress.com/2008/03/04/how-to-setview-the-nic-bind-order-in-windows/

NOTE

Replace "196.168.0.2" with your actual network IP address.

How To Test Two-way .NET Remoting Between Your PC and Your Scope

The remote interface uses Microsoft .NET Remoting technology. Included in this toolkit are a pair of sample client and server applications that exercise a bi-directional .NET Remoting interface. You may use these apps to verify .NET Remoting is working between your remote client PC and your oscilloscope.

1 Copy the **Miscellaneous\Tools\C#\Dot Net Remoting Server** folder to your oscilloscope desktop.

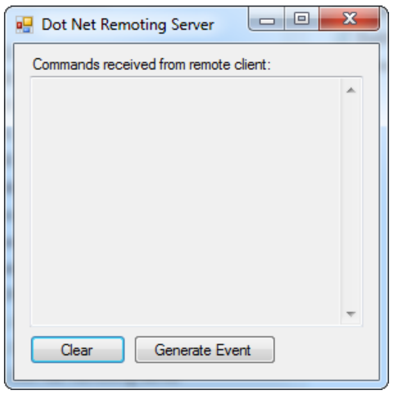
2 Copy the **Miscellaneous\Tools\C#\Dot Net Remoting Client** folder to your remote client PC desktop.

NOTE **The source code for both of these applications may be found in the Miscellaneous\Source\C#\ directory.**

3 On the oscilloscope:

   a (Optional) Edit the **DotNetRemotingServer.exe.config** file to use a different network port.

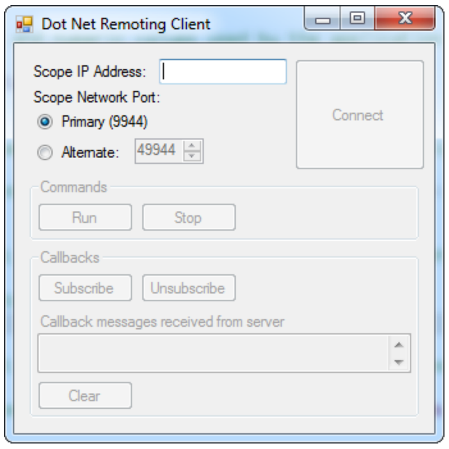
   b Run **DotNetRemotingServer.exe**. The following screen will display:



   When the server receives commands from the Dot Net Remoting Client, the server will display them in the command history listbox. Use the **Clear** button to clear this list at any time.

   The **Generate Event** button enables you to manually initiate a single callback event. If the Dot Net Remoting Client is subscribed to receive callbacks, then the client will receive a message from the server and acknowledge it in the client's user interface.

4 On the remote client PC, run **DotNetRemotingClient.exe**. The following screen will display:



5 Enter the **IP Address** of the scope the DotNetRemotingServer application is running on.

6 Select a port that matches the value found in the **DotNetRemotingServer.exe.config** file on the scope.

When you install any Keysight automated test application on your oscilloscope, it opens up ports 49944 and 59944 in the firewall (These are the primary and alternate ports a remote client will automatically try when connecting to one of those applications.) If you select any other port, ensure that port is opened in the scope's firewall.

7 Click **Connect**. If successful, the rest of the controls will become available and a "Ping" command will be sent to the server. The server will acknowledge receiving commands from the client in the server's user interface.

Use the **Run** and **Stop** buttons to send a command to the server to make it start and stop running its test program. This program is a simple loop that generates callback events once-per-second, which the server sends to the client if the client is subscribed to receive them.

Use the **Subscribe** and **Unsubscribe** buttons to cause the client to register/unregister to receive callback events from the server. When the client receives callback messages from the Dot Net Remoting Server, the client will show them in the callback history listbox.

Use the **Clear** button to clear this list at any time.

Index

Symbols

.NET Framework programming languages, 52
.NET Framework to ARSL syntax translation, 44
.NET Remoting, 109

A

Advanced group, 110
application did not respond, 143
Application Sequencer, 39
ARSL command line utility, 18, 98
ARSL syntax, translating from .NET Framework, 44
ArslExecutor, 99
AteEventSink class, 14, 110
Automated Test Engine Remote Scripting Language (ARSL), 44
available tests, 103

B

basic events handling example, Python, 80

C

class, program, 96
configuration variables and values, 10
connected host failed to respond, 142
connection failed, 142
connection refused, 142
connection, establishing, 96
copyright, 2
Core Features group, 110
custom types, 9, 45

D

deadlock, 111
default message handler, overriding, 111
default settings, automated test application, 26
disposal, 114
double quotes, 44
dummy thread actions, reassigning to Python's main thread, 81

E

error messages, 142
error messages in log, 89
event handling, .NET Framework, 58
event handling, Python, 78
event sink, 110
exception types, 15
ExecuteArsl() remote interface), 44
execution, test, 104

F

file description, 96

G

groups, 110

H

how to use this book, 3

I

in this book, 3
info messages in log, 89
invalid user port entry, 143
IP address of oscilloscope, 32
IP address of server, 18, 20, 24, 28
IRemoteAte interface, 14

K

Keysight KS8400A Test Automation for PathWave (TAP), 42
Keysight.DigitalTestApps.Framework.Remote, 14
Keysight.DigitalTestApps.Framework.Remote.Advanced, 14
Keysight.DigitalTestApps.Framework.Remote.dll, 142
Keysight.DigitalTestApps.Framework.Remote.Exceptions, 15

L

LabVIEW demo client, 128
LabVIEW demo remote client, 31
LabVIEW simple remote client, 27, 116
licensing requirements, 142
log, checking, 88

M

measurement server, .NET Framework, 60
message handler, overriding default, 111
message handling example, Python, 78
message handling remote client, 24, 108
messages, error, 142
method not found, 143
methods, 9
multithreading, 111

N

namespace, program, 96
namespaces, 110
network interface cards (NIC), two, 148
NIC, primary, 148
notices, 2

O

overriding the default message handler, 111

P

parallel testing, .NET Framework, 66
Ping method, 103
plugin for Keysight KS8400A Test Automation for PathWave, 92
primary NIC, 148
properties, 9
Python for .NET package, 67
Python programming language, 67
PyVISA package, 67

Q

quotes, 44

R

recommended remote programming practices, 85
redirecting messages to client, 111
remote client, developing, 91
remote command log code types, 89
remote commands, verifying, 87
remote interface, 9
remote interface disabled by local user, 142
remote interface user port, 147
RemoteAteUtilities class, 14
RemotingException, 144
reserved characters, 44
result access, 104
ResultContainer object, 104

S

sample remote clients, 17
Select Tests tab, 86
Set Up tab, 86
simple GUI remote client, 20, 101
simple message handling remote client, 22, 105
simple remote client, 94
single quotes, 44
source code description, 96
spaces, 44
switch matrix, .NET Framework, 58
switch matrix, .NET Framework, automatic mode, 59
switch matrix, .NET Framework, either mode, 60
switch matrix, .NET Framework, manual mode, 59
switch matrix, ARSL, 50
switch matrix, ARSL, automatic mode, 50
switch matrix, ARSL, either mode, 51
switch matrix, ARSL, manual mode, 50
switch matrix, Python, 82
switch matrix, Python, automatic mode, 82
switch matrix, Python, either mode, 83
switch matrix, Python, manual mode, 82

T

TAP, Keysight KS8400A Test Automation for PathWave, 42
target application did not respond, 143
tasks, determining, 86
threads, 111
translating .NET Framework to ARSL syntax, 44
troubleshooting, 141
two-way .NET remoting, 149

U

user message log code categories, 89
user message log codes, 89
user messges in log, 89
user port, remote interface, 147

W

warning messages in log, 89
warranty, 2
Windows Firewall exceptions, 142
working directory (LabVIEW), 32