

Getting Started with IVI-COM and C# for the TDK-Lambda Z+ Power Supply

1. Introduction

This is a step-by-step guide to writing a program to remotely control the Z+ power supply using the TDK-Lambda IVI-COM drivers. This tutorial has instructions and sample code for a Microsoft Visual Studio C# project. It shows how to use the LAN option.

The C# code can be used as a guide for writing Visual Basic .NET and C++ programs.

2. Table of Contents

1.	INTRODUCTION	1
2.	TABLE OF CONTENTS.....	1
3.	REQUIREMENTS	2
4.	ADDITIONAL HELP.....	3
5.	YOUR FIRST LAN PROGRAM USING C#	4
6.	ADDING A COMMUNICATION ERROR HANDLER	12
7.	ADDING AN INSTRUMENT ERROR HANDLER	13

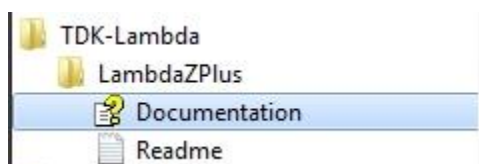
3. Requirements

- A. **Windows Operating System:** This tutorial is written using .NET version 2.0 for Windows 7. Other versions of Windows and .NET are supported.
- B. **Microsoft Visual Studio:** The examples use the Visual Studio 2010 development environment, although other versions are supported. Any language that uses COM or .NET objects should be compatible with the IVI drivers.
- C. **Prior Experience with C# Programming:** This tutorial assumes prior experience with Visual Studio C#. You should know how to create buttons and text boxes, modify their properties and handle button events. You should also know how to build, run and debug your application.
- D. **IVI Drivers:** The TDK-Lambda Z+ IVI driver package must be installed. It is available on the Z+ CD-ROM.
Install the file "LambdaZPlus 1.x.x.x.msi" on your computer
- E. **VISA Drivers:** The IVI technology is built upon a layer of hardware drivers called VISA (for "Virtual Instrument Software Architecture"). These drivers are available with the National Instruments Measurement and Automation Explorer (NI-MAX), the Agilent IO Libraries and from other companies.
- F. **Electrical Interface:** almost all personal computers have a local area network (LAN) port.

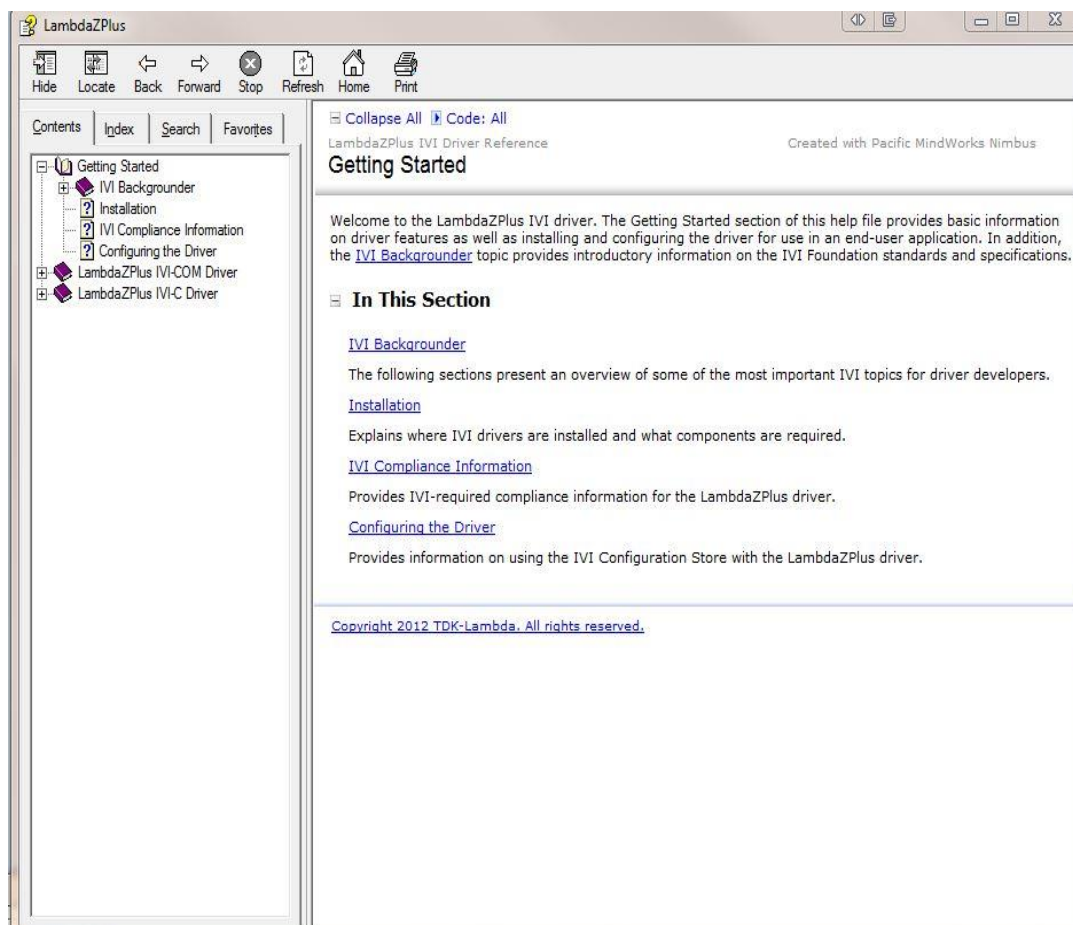
Getting Started with IVI-COM and C# for the TDK-Lambda Z+ Power Supply

4. Additional Help

In addition to this document, a “Getting Started” and language reference Help is installed by the TDK-Lambda Z+ IVI driver package. This help file may be opened through the Start menu as shown below:



Below is a sample of the Help file's contents:



5. Your First LAN Program Using C#

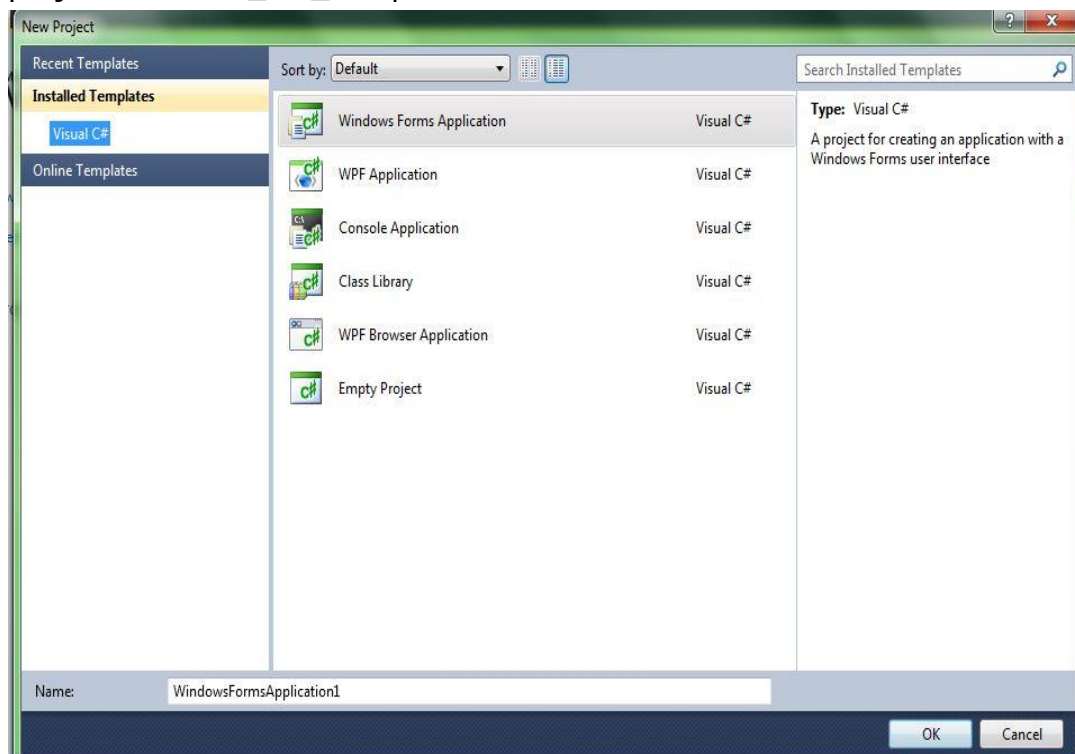
A. *Launch Visual Studio 2010.*

On the menu bar, select "File → New → Project..."

See the New Project dialog box open (picture below).

Select: Visual C# for Windows, and
Windows Application.

Enter your project name and folder name. The sample program uses the project name "Z_IVI_sample". Click "OK".



The following discussion uses the Visual Studio solution file:

Z_IVI_sample.sln

which is included in the TDK-Lambda "Getting Started with IVI" download.

Getting Started with IVI-COM and C# for the TDK-Lambda Z+ Power Supply

B. Create the Form with Objects

On the program's "Form1", add buttons and text boxes as shown in the example below. The example includes the following objects:

- **InitButton**: click this to open the communication port to a Z+ power supply at a specified IP address. The example program is fixed for the default LAN address = 169.254.7.53.
- **IP Address Box**: enter IP Address.
- **ApplyButton**: apply IP Address.
- **ToggleOutButton**: click to toggle the supply's output ON or OFF.
- **ProgVoltTextBox**: enter a new voltage limit setting to be sent to the selected supply when the ProgVoltButton is clicked.
- **ProgVoltButton**: click this to send the text box's program voltage setting to the power supply.
- **MeasVoltTextBox**: displays the output voltage read from the power supply after the MeasVoltButton is clicked.
- **MeasVoltButton**: click to read the output voltage from the supply.
- **CloseButton**: click this to close the connection to the LAN port. This does not close the program.

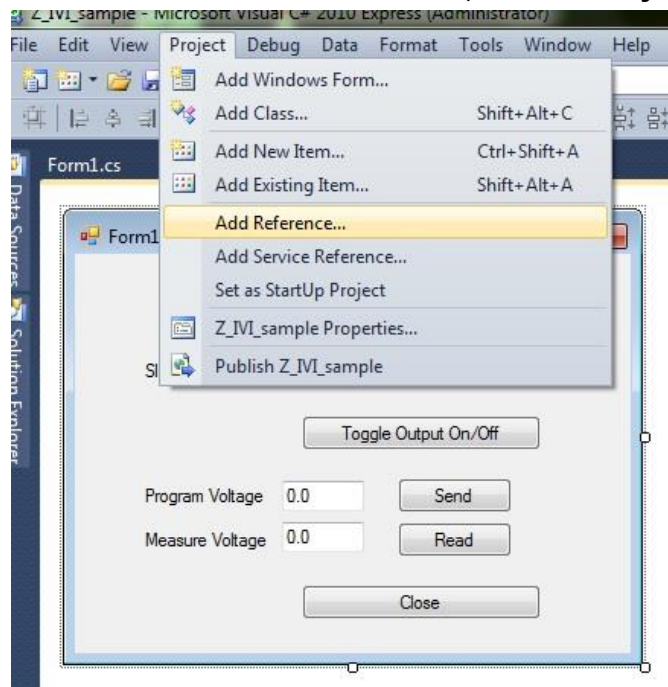
The screenshot shows a Windows form titled "Form1". The form contains the following elements:

- A label "IP Address = 169.254.7.53" and a blue "Initialize" button.
- An "IP Address" label followed by a text box containing "169.254.7.53" and an "Apply" button.
- A "Toggle Output On/Off" button.
- A "Program Voltage" label followed by a text box containing "0.0" and a "Send" button.
- A "Measure Voltage" label followed by a text box containing "0.0" and a "Read" button.
- A "Close" button at the bottom.

Getting Started with IVI-COM and C# for the TDK-Lambda Z+ Power Supply

C. Open the "Add Reference" Dialog

On the Visual Studio menu bar, select "Project → Add Reference..."



Getting Started with IVI-COM and C# for the TDK-Lambda Z+ Power Supply

D. Add the IVI Driver Reference

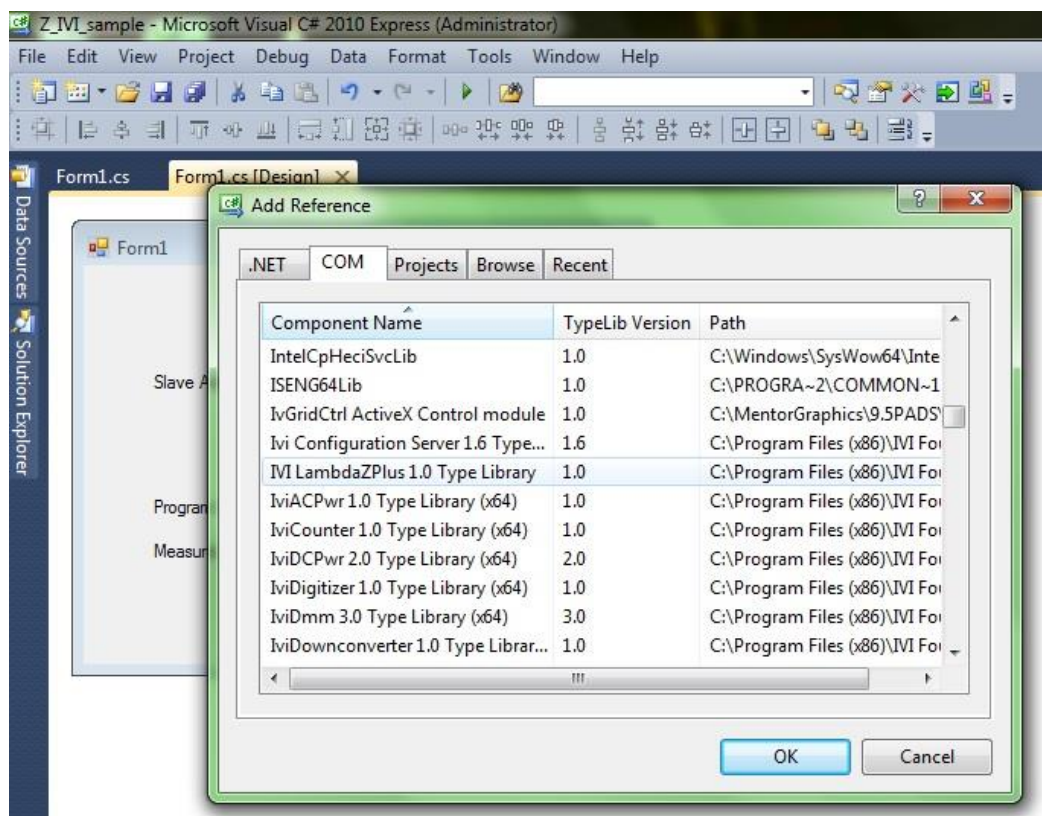
See the "Add Reference" dialog box open.

Select the "COM" tab (not the .NET tab).

Scroll down to find:

"IVI LambdaZPlus 1.0 Type Library"

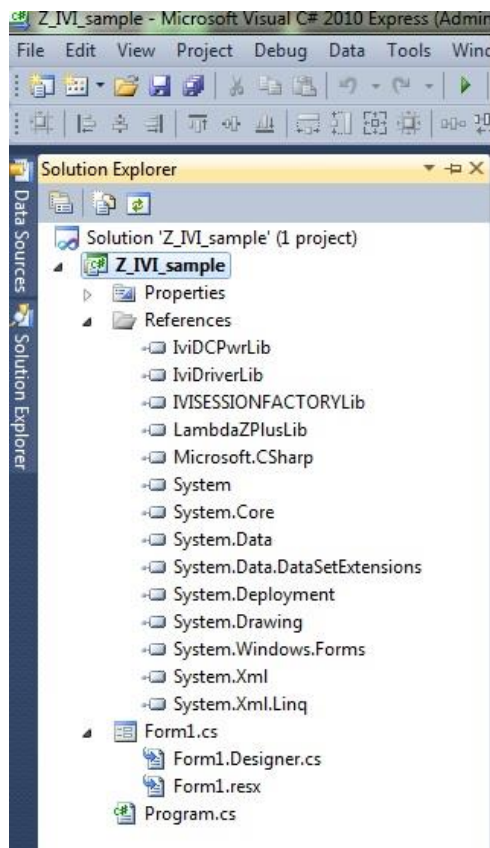
Click "OK".



Getting Started with IVI-COM and C# for the TDK-Lambda Z+ Power Supply

References explorer shows five new references have been added to the project:

- IviDCPwrLib
- IviDriverLib
- LambdaZPlusLib
- VisaComLib
- Ivi SessionFactoryLib



Getting Started with IVI-COM and C# for the TDK-Lambda Z+ Power Supply

E. Add an Instance of the Driver

In the "Form1" code section:

- In the code header, add a "using" directive to use the type:

```
using Lambda.LambdaZPlus.Interop;
```

- In the "Form1" class, create an instance of the IVI driver.
Name the IVI driver "ZplIvi" with this statement:

```
LambdaZplus ZplIvi;
```

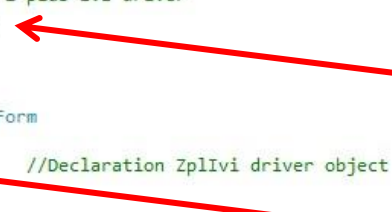
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

//For creating an instance of the Z plus IVI driver
using Lambda.LambdaZPlus.Interop;

namespace Z_IVI_sample
{
    public partial class Form1 : Form
    {
        LambdaZPlus ZplIvi; //Declaration ZplIvi driver object

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
        }
    }
}
```



Getting Started with IVI-COM and C# for the TDK-Lambda Z+ Power Supply

F. Add a Step to Initialize the LAN Port

Now you are ready to insert calls to the IVI drivers that will communicate to the Z+ power supply over the LAN.

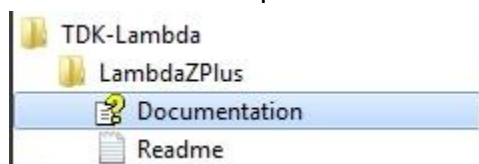
In the "InitButton_Click" event handler, create an instance of the "ZplIvi" driver.

Call the "ZplIvi.Initialize" method. It has four parameters.

```
private void InitButton_Click( object sender, EventArgs e )
{
    ZplIvi = new LambdaZplus( );

    ZplIvi.Initialize( "TCPIP::169.254.7.53::INSTR", true, true, ""
);
}
```

For more description of the Initialize method, the "F1" key context sensitive help does not work well, because Initialize has several levels of definition. Therefore, it is recommended you navigate directly to the LambdaZPlus help file as shown:



Navigate the Help file index for the entry "Initialize method". The parameter descriptions are given in detail. Code samples are given for Visual Basic, C# and C++.

Getting Started with IVI-COM and C# for the TDK-Lambda Z+ Power Supply

G. Add a Step to Toggle the Supply's Output

Now you can use any ZplIvi driver to configure and monitor the power supply. Notice, that after typing just a few letters of a method, the Intellisense will open a listbox of allowed properties and methods.

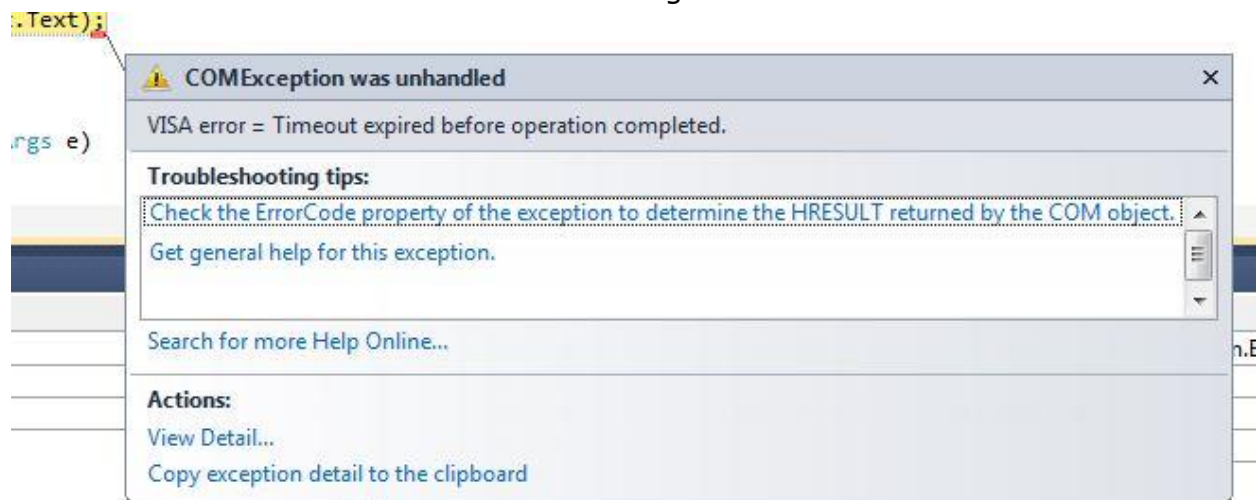
```
private void ToggleOutButton_Click( object sender, EventArgs e )
{
    Boolean OutState;
    OutState = ZplIvi.Output.Enabled;
    ZplIvi.Output.Enabled = !OutState;
}
```

As shown in the sample program, there is a "ToggleOutButton" that will read the output ON or OFF state and then send a command to change the state to the opposite.

H. No Error Handling

Although this is how a complete program is written, there is no error handler. If communication is lost, then the NET exception handler will display an error and then stop the program.

See the next section for better error handling.

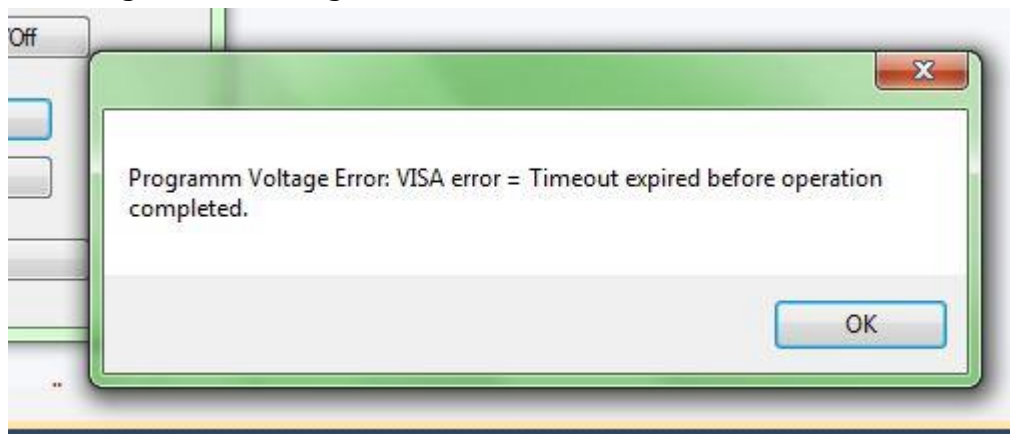


6. Adding a Communication Error Handler

The .NET framework handles errors by throwing exceptions. Any driver call should be in a "Try" statement followed by a "Catch" exception handler.

```
private void ProgVoltButton_Click( object sender, EventArgs e )
{
    // Clicked the button send a new program voltage value
    try
    {
        ZplIvi.Output.Enabled = Double.Parse( ProgVoltTextBox.Text );
    }
    catch ( Exception ex )
    {
        MessageBox.Show( "Program Voltage error: " + ex.Message );
    }
}
```

If there is a communication error, using the sample code will produce the following error message box:



Note that the Try and Catch will only report communication errors. They will not report instrument errors, such as trying to program a setting out of range. To catch instrument errors, you must read the SYSTEM:ERROR messages from the supply. See the next section for this solution.

7. Adding an Instrument Error Handler

The communication error handler, in section 6 above, will only catch errors if the communication link is disconnected in some way. If your program tries to set the power supply to an illegal value, an exception (error) is not created.

To determine if the last command was accepted, the program must send the "SYSTEM:ERROR?" query. By enabling the QueryInstrStatus driver property, this query is done automatically after each command and an exception is generated if a command was not accepted.

A. *Create the Scope of COMException*

In the code header, add the statement as shown below.

```
using Lambda.LambdaZPlus.Interop;  
  
// For defining an error COMException class just for instruments  
using System.Runtime.InteropServices;  
  
namespace Z_IVI_sample
```

B. *Configure the Driver for Automatic Error Query*

In the Initialize method, the fourth parameter is an option string. Set "QueryInstrStatus" to TRUE to enable automatic queries for instrument errors, as shown below.

```
ZplIvi = new LambdaZPlus( );  
  
try  
{  
    ZplIvi.Initialize( "TCPIP::169.254.7.53::INSTR", true, true,  
        "QueryInstrStatus=True");  
}
```

Getting Started with IVI-COM and C# for the TDK-Lambda Z+ Power Supply

C. *Handle Instrument Errors*

A “try – catch” structure is used to handle cases where the power supply cannot do a command because something is wrong with the command.

Although catching “Exception” will handle all errors that the computer can generate, we are only interested in catching errors that the communication interface can generate. Therefore, the program catches “COMException”.

```
private void ProgVoltButton_Click( object sender, EventArgs e )
{
    // Clicked the button to send a new program voltage value
    try
    {
        ZplIvi.Output.Enabled = Double.Parse( ProgVoltTextBox.Text );
    }
    catch ( COMException )
    {
        Int32 ErrorCode = 0;
        String ErrorMessage = "";

        ZplIvi.Utility.ErrorQuery( ref ErrorCode, ref ErrorMessage );

        MessageBox.Show( "Program Voltage error: " +
                        ErrorCode + ", " + ErrorMessage );
    }
}
```

With this code in your program, if you try to program the voltage setting to any value that the power supply cannot accept, the new setting will be ignored, a COMException will be generated, and a messagebox will appear. For example, say an attempt is made to program a supply to 900 volts. Since this is above the supply’s rating, the command will not be accepted and the messagebox below will appear:

