# PCM Interface Instrument

## Instrument

This lib’s purpose is offer an easy way to build an interface program for PCM, which can input parameters.

## Requirement

For Develop:

1. Visual Studio which support .NET 4.0 (C# or VB.NET or Express edition)  
   or any IDE develop tools for .NET like Sharp Developer
2. .Net FrameWork 4.0 or later version

For customer use

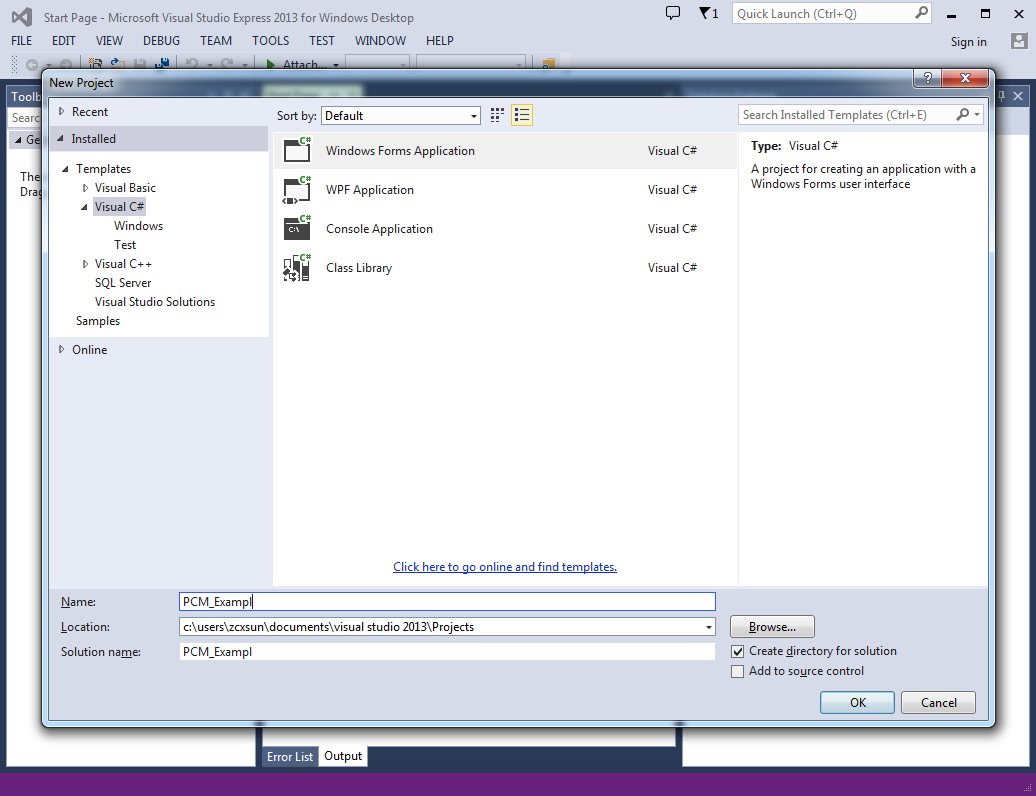
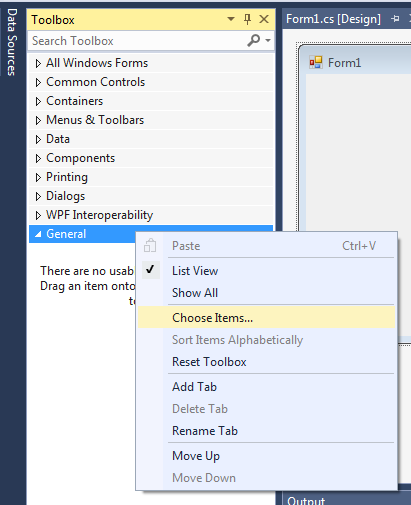
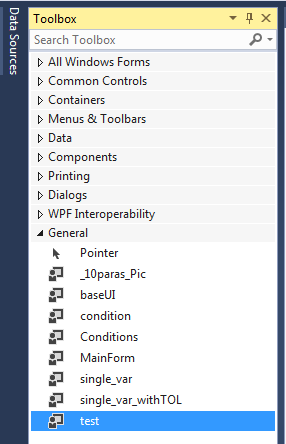
1. .Net FrameWork 4.0 or later version

Source Code

You can download the code by yourself on <https://darksun190.visualstudio.com/DefaultCollection/PCMInterface>

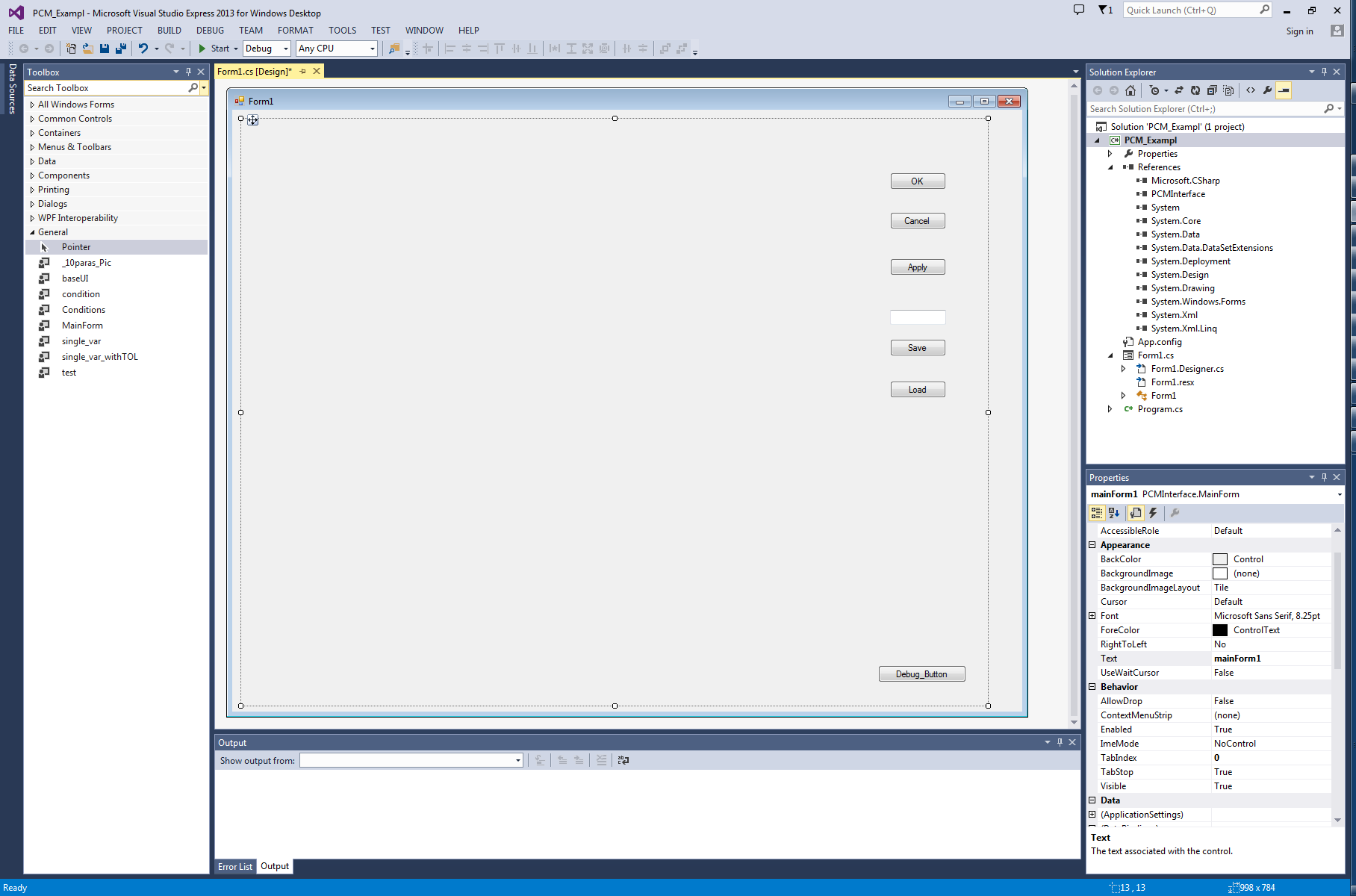
Or mail to me <mailto:xin.sun@zeiss.com> ask a newest copy.

## How to Import the library to a new Project

1. start the Visual studio and make a new project by “File -> New Project…”  
   select “Windows Forms Application” as type, and give a name  
     
   Click “OK”
2. If this is the first time, import the lib to the toolbox.  
   Right Click the “General” Label in the Toolbox, Click the “Choose Items…” in the context menu.  
   
3. On the tab page “.NET Framework Components”, Click “Browse…” and find out the lib file.  
   the controls will list on the Toolbox if OK.  
   
4. Then you can use these controls in your form.

## Generate a typically interface

Put a “MainForm” to the window, adjust the window size, let it be bigger than the control.



For example, you want to generate a file like this:

Test2 = 1

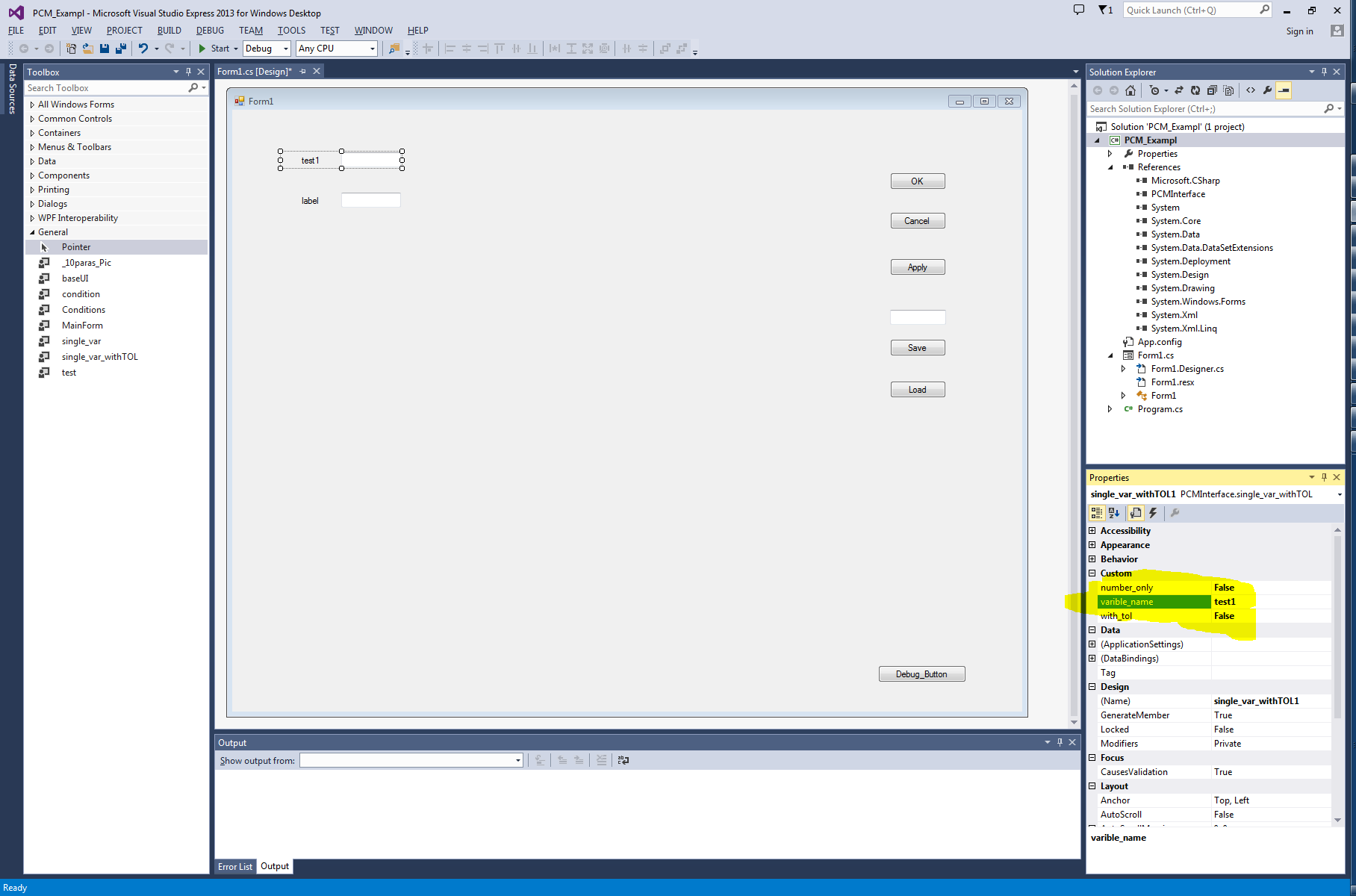
Test2\_UT = 0.1

Test2\_LT = -0.1

test1 = "a"

drag 2 “single\_var\_withTOL” class to the “MainForm”

select the first one, modify its property “variable\_name” to “test1”



Select another one, modify its property “variable\_name” to “Test2”, property “with\_tol” to “true”.

Press F7 to build your interface, it should be saved at ~\bin\debug\ folder.

More information about other controls, please see Chapter 2.

## Project PCMInterface

### Class MainForm (Top Level Control)

The top level control have to put on the top form windows, it is a container for other controls. You must have one top level control in the program, and only one.

It contains 5 buttons and 1 input field by default.

“Textbox” for input, define the name which to load or save. The end user didn’t need to add the extension name for the file, it was define as the property, which can’t be modified by the end user.

“Load” button, load the file defined by “Textbox” in subfolder, the subfolder is defined in the property by developer, which cannot be modified by end user.

“Save” button, save the file defined by “Textbox” in subfolder, the position is same as “Load”.

“Apply” check if everything was input by the end user, save the input to dictionary inside, usually for debug, you can hide it for end user.

“OK” active the save button then closed the whole program.

“Cancel” closed the program without do anything.

“Debug” button for developer, compare every variable was put into the mainform.

Properties

public string all\_text

Here is the sample text, for building a master library using for compare.  
you don’t need to define this text directly, use origin\_file to import a file, this text should be import automatically.

public string sub\_f\_name

define the subfolder which the files saved. Empty means the same folder as the exe file.

public string ext\_name

define the extension name, like “para” or “pcm”, without the dot “.”

public string origin\_file

select an example file to import the content.

Method

All the functions called by the buttons.

### Class MainformWithList(Top Level Control)

This is a similar class inherited the mainform, only add a list view to show all exists files in the specification folder.

Use it as the MainForm

### Class baseUI (abstract class, didn’t show in the toolbox)

This is the basic class, define the interface name and default return value;

Properties

None

Method

Check\_Validation()

Check everything was done by end user.

Read\_Parameter(Dictionary<string, string> dic)

Read content from the dictionary(dic), and find the correspondence value input to the field.

Write\_Parameter(Dictionary<string, string> dic)

Get the value from the field and write to the dictionary(dic).

### Class single\_var

A label and a textbox to input one variable

Properties

bool number\_only

if change this property to true, the end user can only input digit.

string variable\_name

define the variable name, for example “var1” in “var1 = 100”

Method

Same as baseUI

### Class single\_var\_withTOL (inherited from single\_var )

Properties

2 same properties as the “single\_var”

bool with\_tol

if change this property to true, will show 2 more textbox, and save 2 other variables as “var1\_UT” & “var1\_LT”

Method

Same as baseUI

### Class \_10paras\_Pic

Properties

int var\_no

the number of the variable, maximum is 10.

bool[] With\_tolerance

active tolerance for each variable

Color Highlight\_Color

When end user focus on one textbox, the label will be highlight, here defines the highlight’s color.

Image select\_pic

Select the picture which shows on the window.

Point[] Label\_Pos

Define the position where the highlight label put, the (0,0) is on the left up corner, all the coordinate is plus, like (120,60) means the label shift left 120 pixels, and shift down 60 pixels.

string[] Varible\_Name

define the names for the 10 variables.

Method

Same as baseUI

### Class Conditions

This class worked for some condition breaks, like

*Radius = 100*

If type = “Cone” then

Cone\_angle = 30

End if

Properties

String[] List\_str

Define the contents in the combobox. Maximum is 10.

String Varible\_Name

Define the condition control variable name. see the example below.

public int Current\_Selection

for developer, switch the combobox (and the panels) in design mode.

Method

Same as baseUI

Note

The panels can’t be moved.

Example

To be continue…