Paper 002-2011

Windows Application Using .NET and SAS® to Produce Custom Rater Reliability Reports

Sailesh Vezzu, Educational Testing Service, Princeton, NJ, USA

ABSTRACT

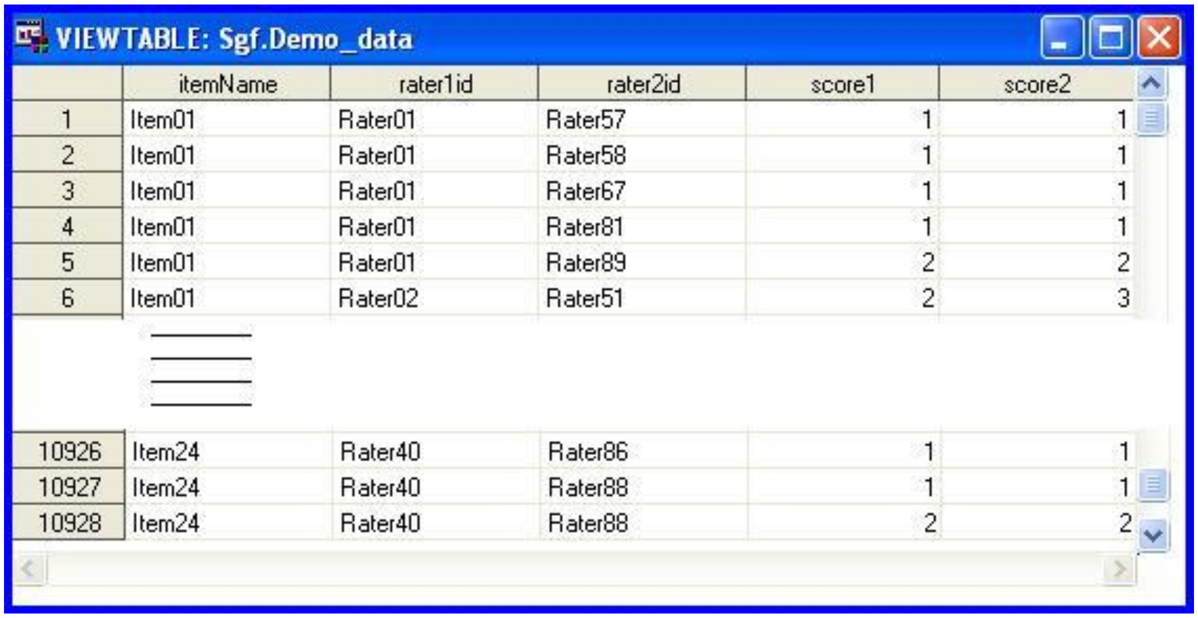
Some of the common measures used to monitor rater reliability of constructed response items include Percent Exact Agreement, Percent Adjacent Agreement, Simple Kappa Coefficient, Weighted (Cicchetti-Allison, Fleiss-Cohen) Kappa Coefficient, Pearson Correlation Coefficient, Mean Difference, Standardized Mean Difference, and Matched-Pair T-Test. This paper discusses the development of a Windows application using Visual C#.NET and SAS® to calculate various measures of rater reliability and produce custom reports. Users can select which measures to include in their reports and set threshold values to flag items with low rater reliability values.

INTRODUCTION

In the Research and Development division at Educational Testing Service (ETS), I work on several projects, which require me to calculate different measures of rater reliability in different ways. Depending on the purpose of the project, these measures sometimes need to be calculated per each item across several raters, per each rater across several items, or per each scoring session across all items and raters. In order to accommodate these scenarios, I built this application, which allows users to enter several parameters and produce customized reports with different measures of rater reliability.

The first section of this paper will discuss different measures of rater reliability and how to compute those using SAS. The second section of this paper will show how to build a Windows application using Visual C#.NET, get user input, and pass the input into SAS as parameters.

Figure 1 below shows the structure of a sample input data set for this application. A typical input data set includes the name of the item, the IDs of both rater 1 and rater 2, and the scores given by the raters. The data in this paper is for demonstration purposes only and is not actual data.



	itemName	rater1id	rater2id	score1	score2
1	Item01	Rater01	Rater57	1	1
2	Item01	Rater01	Rater58	1	1
3	Item01	Rater01	Rater67	1	1
4	Item01	Rater01	Rater81	1	1
5	Item01	Rater01	Rater89	2	2
6	Item01	Rater02	Rater51	2	3
10926	Item24	Rater40	Rater86	1	1
10927	Item24	Rater40	Rater88	1	1
10928	Item24	Rater40	Rater88	2	2

Figure 1. Sample Input Data Set

I recommend running the FREQ procedure on all the discrete variables and the MEANS procedure on all the continuous variables in the data set to check for missing and out of range values.

APP-FRIENDLY SAS CODING

To easily incorporate the SAS code into various applications, SAS code is organized into the following modules depending on SAS procedures used to compute these measures:

1. To calculate means, standard deviations, absolute mean difference and standardized mean difference
2. To calculate simple Kappa Coefficient and (Cicchetti-Allison, Fleiss-Cohen) weighted Kappa Coefficient
3. To calculate matched pair t-value and the corresponding p-value
4. To calculate Percent Exact Agreement and Percent Adjacent Agreement
5. To calculate Pearson Correlation Coefficient
6. To generate custom reports

All of these modules have an input data set and output data set as parameters. Any temporary data sets created during the module will be deleted at the end of the module using the DATASETS procedure.

MODULE 1: MEANS, STANDARD DEVIATIONS, MEAN DIFFERENCES

PROC MEANS can be used to calculate all the means, standard deviations, absolute mean differences, and standardized mean differences.

Absolute mean difference is the absolute value of the difference between the 2 scores, and the standardized mean difference is the difference between the means expressed in terms of the pooled standard deviation.

```
%Macro get_ns_means_diffs(indat, outdat, var1, var2, byvar);

proc means data=&indat. noprint;
       var &var1. &var2. ;
       by &byvar.;
       output out=t_1 (drop=_type_ _freq_)
                     n=N_&var1._&var2. mean=Mean_&var1. Mean_&var2.
                     STD=SD_&var1. SD_&var2.;
run;

data &outdat.;
       set t_1;
       mnd_&var1._&var2. = abs(Mean_&var1. - Mean_&var2.);
       stdd_&var1._&var2. = (Mean_&var2. - Mean_&var1.)/(sqrt(((SD_&var1.**2) +
       (SD_&var2.**2))/2));
       keep &byvar. N_&var1._&var2. Mean_&var1. Mean_&var2. SD_&var1. SD_&var2.
       mnd_&var1._&var2. stdd_&var1._&var2;
run;

proc datasets nolist;
delete t_1;
run; quit;

%Mend;
```

As you can see in the code above, module 1 has input and output data sets as parameters. Temporary data set “t_1” created during the module will be deleted at the end of the module. All of the modules used in this application expect the input data set to be sorted by the “byvar”. This is more efficient than sorting the input data set in each module. My personal preference in writing app-friendly SAS code is to never alter or even sort the input data set inside a module. If for some reason I have to make changes to the input data set, I create a separate temporary work data set and make changes to it. Doing this can help avoid unnecessary hassles when working with several modules within the same application.

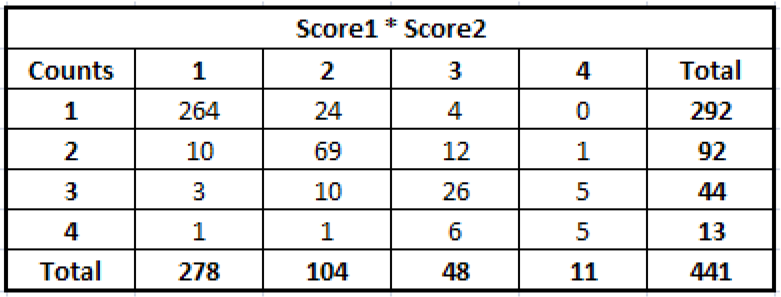
MODULE 2: KAPPA COEFFICIENTS

Kappa coefficients are measures of association between score 1 and score 2 after correcting for chance agreement rates. Kappa is a more appropriate statistic to use when you want to compare reliability of items with different scales.

Agree option in PROC FREQ will produce the simple or unweighted Kappa coefficient and the Cicchetti-Allison or linear weighted Kappa coefficient. When (wt=fc) is used next to the Agree option in PROC FREQ, it produces the simple Kappa coefficient and the Fleiss-Cohen or Quadratic weighted Kappa coefficient.

Kappa coefficients for balanced data

When there is an equal number of rows and columns in a crosstab between score1 and score 2, as shown in Figure 2 below, you have a simple case of balanced data. In this case, SAS computes Kappa coefficients without any problems.



Score1 * Score2					
Counts	1	2	3	4	Total
1	264	24	4	0	292
2	10	69	12	1	92
3	3	10	26	5	44
4	1	1	6	5	13
Total	278	104	48	11	441

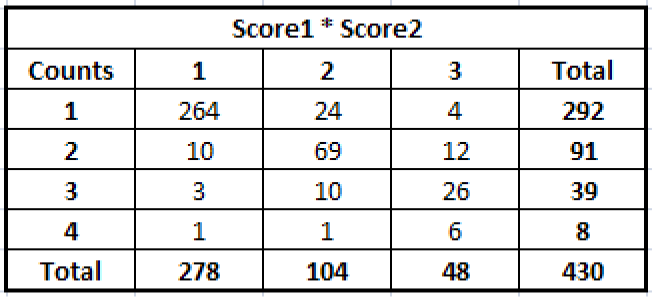
Figure 2. Balanced Data Example

```
%Macro get_kappas(indat, outdat, var1, var2, byvar);
…
…
proc freq data = &indat.;
       tables &var1. * &var2. / norow nocol nopercent nocum agree;
       output out = kappas_stats1 kappa wtkap;
run;
…
proc freq data = &indat.;
       tables &var1. * &var2. / norow nocol nopercent nocum agree (wt=fc);
       output out = kappas_stats2 kappa wtkap;
run;
…
%Mend;
```

The first version of the PROC FREQ above produces the simple and the linear weighted Kappa coefficients. The second version of the PROC FREQ above produced the simple and the quadratic weighted Kappa coefficients.

Kappa coefficients for unbalanced data

When there is an unequal number of rows and columns in a crosstab between score1 and score2, as shown in Figure 3, PROC FREQ doesn't produce the *agree* statistics.



Score1 * Score2				
Counts	1	2	3	Total
1	264	24	4	292
2	10	69	12	91
3	3	10	26	39
4	1	1	6	8
Total	278	104	48	430

Figure 3. Unbalanced Data Example

In the above crosstab between score1 and score2, there is a total of 8 observations where score1 equals 4, while there are 0 observations where score2 equals 4, hence resulting in a case of unbalanced data. The SAS support Web site published a solution for this by adding a dummy observation where score2 equals 4 and assigning it a weight of 0, while giving a weight of 1 for the rest of the observations. This solution is easy and simple to apply when you know in advance where the dummy observation needs to be added to your data. When you are running your program on several items, you don't want to manually check for all items to see if you have unbalanced data. This solution of having a weight of 0 is only possible with SAS versions 9 or later.

One other possible solution to this problem is to add a set of dummy observations for all possible cells in the crosstab to cover all bases instead of checking the crosstab each time to see how your data is unbalanced.

```
%Macro get_kappas(indat, outdat, var1, var2, byvar);
…
data for_dummy (drop=i j);
do i = 1 to 10 by 1;
       do j = 1 to 10 by 1;
              &var1 = i;
              &var2 = j;
              flag = "dummy";
              output;
       end;
end;
run;
…
data score;
       set &indat.;
       flag = "score";
       if &byvar. = "&tmp_by_var.";
run;
…
data t_1;
       set score dummy;
run;
proc freq data = t_1;
       tables flag * &var1. * &var2. / norow nocol nopercent nocum agree;
       output out = kappas_stats1 kappa wtkap;
run;
…
%Mend;
```

The snippets of code above show how to create a data set with 100 dummy observations, append the dummy data to the original data set, and use a flag variable to distinguish scoring observations from the dummy observations. Using the flag variable in the tables statement would produce Kappa statistics for each value of flag ”score ” and “dummy” and across all observations. To get the desired Kappa statistics, you just need to filter the output data set for flag=“score”.

MODULE 3: MATCHED PAIR T-TEST

A t-test is a commonly used hypothesis test for determining differences between 2 groups. Since the data here consists of pairs of scores for each paper, a matched pair t-test can be used to test for significant differences between 2 groups. The paired statement in the TTEST procedure produces the Matched Pair t-value and the corresponding p-value. A p-value of 0.05 or lower means that the 2 groups are significantly different at an Alpha level of 0.05.

```
%macro get_mptvalue(indat, outdat, var1, var2, byvar);

proc ttest data =&indat.;
       paired &var1. * &var2.;
       by &byvar.;
       ods output ttests = t_1(keep=&byvar. tvalue probt);
run;
…
%mend;
```

MODULE 4: PERCENT EXACT AND ADJACENT AGREEMENTS

To understand Percent Exact Agreement and Percent Adjacent Agreement, please see the crosstabs below in Figure 4 and Figure 5.

Score1 * Score2					
Counts	1	2	3	4	Total
1	264	24	4	0	292
2	10	69	12	1	92
3	3	10	26	5	44
4	1	1	6	5	13
Total	278	104	48	11	441

Figure 4. Crosstab of score1 * score2: Counts

Score1 * Score2					
Percents	1	2	3	4	Total
1	59.9%	5.4%	0.9%	0.0%	66.2%
2	2.3%	15.6%	2.7%	0.2%	20.9%
3	0.7%	2.3%	5.9%	1.1%	10.0%
4	0.2%	0.2%	1.4%	1.1%	2.9%
Total	63.0%	23.6%	10.9%	2.5%	100%

Figure 5. Crosstab of score1 * score2: Percents

Percent Exact Agreement is the sum of all percents in red on the diagonal from Figure 5. Percent Adjacent Agreement is the sum of all percents in red and green from Figure 5. To calculate these measures in SAS, get the absolute difference between score1 and score2, and then produce a frequency distribution of the difference variable. The percent of 0 differences is the Percent Exact Agreement, and the cumulative percent of 0s and 1s is the Percent Adjacent Agreement.

```
%Macro get_agree_0_adj_1(indat, outdat, var1, var2, byvar);
data t_1;
       set &indat.;
       ad_&var1._&var2. = abs(&var1. - &var2.);
run;
…
proc freq data = t_1;
       tables ad_&var1._&var2./out = t_2 outcum;
       by &byvar.;
run;
…
data t_3(rename=(CUM_PCT=agr_&var1._&var2.));
       set t_2;
       if ad_&var1._&var2. = 0;
       keep &byvar. cum_pct;
run;
…
data t_4(rename=(CUM_PCT=adj1_&var1._&var2.));
       set t_2;
       if ad_&var1._&var2. = 1;
       keep &byvar. cum_pct;
run;
…
%Mend;
```

MODULE 5: PEARSON CORRELATION COEFFICIENT

Pearson Correlation Coefficient is another measure of association between score1 and score2, which can easily be produced using PROC CORR in SAS.

```
%Macro get_correl(indat, outdat, var1, var2, byvar);
proc corr data=&indat. noprint outp=t_1 vardef=n;
       var &var1. &var2.;
       by &byvar.;
run;
…
%Mend;
```

MODULE 6: GENERATE CUSTOM REPORTS

Module 6 shows how to generate custom reports with only the measures specified by the user and flag cases with reliability values lower than the threshold values set by the user. In the next section of this paper, I will demonstrate how to design a Windows form and get input from the user. For now, assume that user input is available in the form of macro variables.

```
%let variable1 = score1;
%let variable2 = score2;
…
```

```
%let prt_means_sds = Yes; /* whether or not to print in the output file */
let prt_ckappa = Yes;
%let prt_fkappa = No;
…
%let kappa_threshold = 0.60; /* threshold values */
%let ckappa_threshold = 0.70;
%let fkappa_threshold = 0.75;
%let pea_threshold = 60;
%let paa_threshold = 75;
```

To highlight the stats with reliability lower than the threshold values set by the user, use PROC FORMAT to define formats.

```
Proc format;
       …
       value ckap_flags    low - < &ckappa_threshold.        ='graybb';
       value fkap_flags    low - < &fkappa_threshold.        ='graybb';
       value pea_flags     low - < &pea_threshold.           ='graybb';
       …
run;
```

PROC REPORT, along with ODS TAGSET EXCELXP, can be used to generate output files in XML format which can easily be saved as Excel files.

```
%macro gen_report(outdat, fl, fn, tt1, var1, var2, byvar);
/*&fl. and &fn. are macro variables containing the output file location and name*/
ods tagsets.Excelxp style=printer file =  "&fl.&fn.";
ods tagsets.Excelxp options
(      embedded_titles='yes'
       embedded_footnotes='yes'
       sheet_name ="&var1.&var2."
       zoom = '90'
       Frozen_Headers = '5'
);
proc report data = pro_to_rep nowd split= '~'  style=[just=center];
       …
       columns
       (&byvar.)
       ("&var1. by &var2."
       ("N" N_&var1._&var2.)
       …
       %if &prt_ckappa. = Yes %then %do;
              LWKap_&var1._&var2.
       %end;
       %if &prt_fkappa. = Yes %then %do;
              QWKap_&var1._&var2.
       %end;
       …
       %if &prt_ckappa. = Yes %then %do;
              define LWKap_&var1._&var2. / analysis mean "Linear wtd ~ kappa" center
              style={background=ckap_flags.};
       %end;
       %if &prt_fkappa. = Yes %then %do;
              define QWKap_&var1._&var2. / analysis mean "Quad wtd ~ kappa" center
              style={background=fkap_flags.};
       %end;
       …
run;
ods tagsets.Excelxp close;
…
%mend;
```

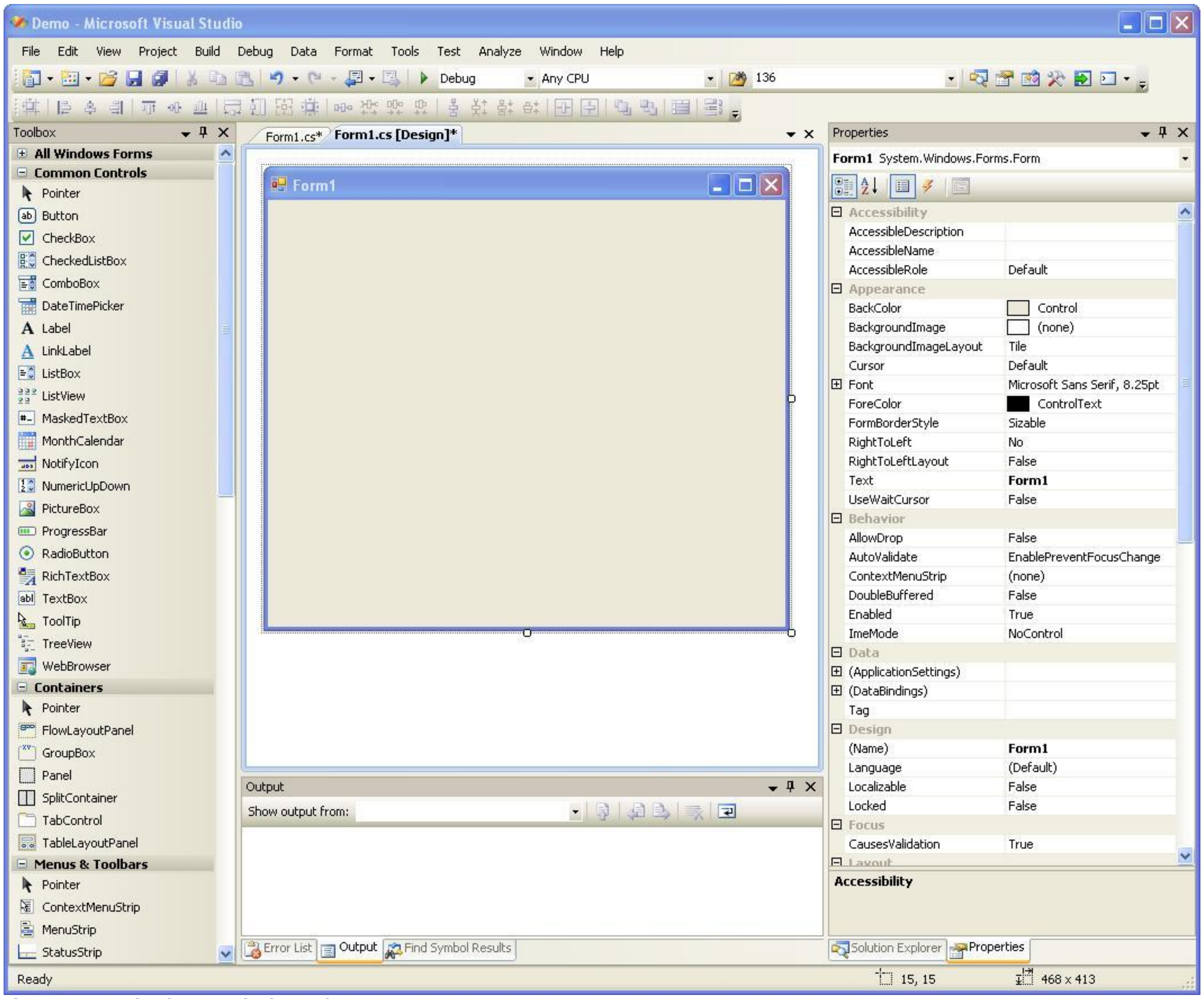
%if statements, along with the appropriate macro variables, can be used to control what is being printed in the output file as shown above. Also, %if statements can be used to control which define statements get submitted.

RUNNING SAS CODE FROM VISUAL C#.NET

In this section, I will discuss how to design a Windows form using Visual C#.NET, get input from the user and pass the input to SAS, and call SAS programs from Visual C#.NET using various parameters received from the user through this Windows form.

DESIGNING A WINDOWS FORM

Open Visual Studio 2008 and create a new project. Select Visual C# and Windows application when creating this new project. You will then see the Visual Studio 2008 interface to design a Windows form, as shown in Figure 6 below.

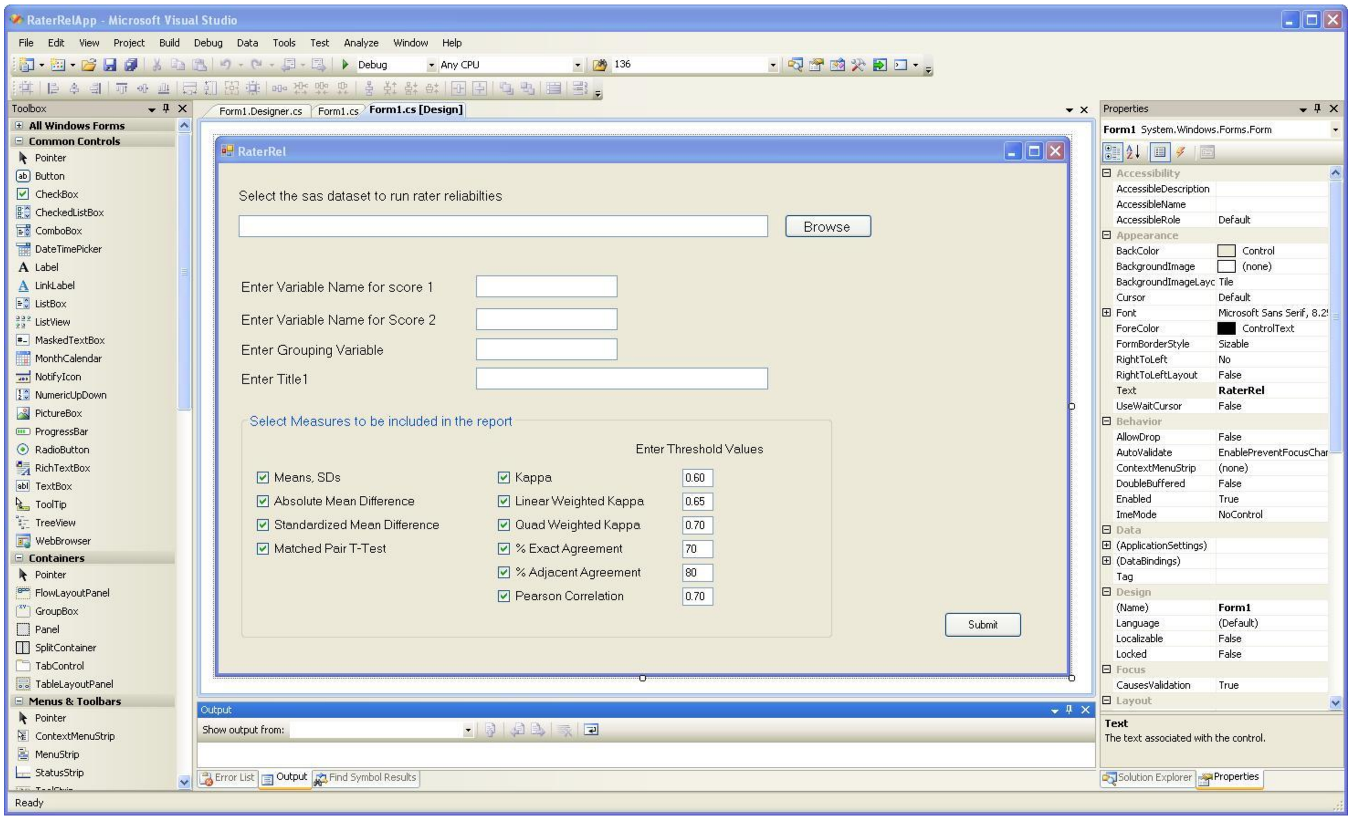


⊟ Accessibility	
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
⊟ Appearance	
BackColor	☐ Control
BackgroundImage	☐ (none)
BackgroundImageLayout	Tile
Cursor	Default
⊞ Font	Microsoft Sans Serif, 8.25pt
ForeColor	■ ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	**Form1**
UseWaitCursor	False
⊟ Behavior	
AllowDrop	False
AutoValidate	EnablePreventFocusChange
ContextMenuStrip	(none)
DoubleBuffered	False
Enabled	True
ImeMode	NoControl
⊟ Data	
⊞ (ApplicationSettings)	
⊞ (DataBindings)	
Tag	
⊟ Design	
(Name)	**Form1**
Language	(Default)
Localizable	False
Locked	False
⊟ Focus	
CausesValidation	True
⊟ Layout	

Figure 6. Designing a Windows form

On the left hand side of the screen, you will see common controls which can be dragged and dropped onto the blank Windows form. On the right hand side of the screen, you will see properties of the selected object. Start by dragging and dropping a textbox onto the form. As soon as you click on the textbox to select it, the properties windows on the right hand side will show the default properties of this textbox. You can easily rename it, change the size, add a default text, etc. by changing the properties window on the right. You can also drag and drop buttons to the form. You can assign a method to the event “button click”, by double clicking on the button. Double clicking a button in this interface will cause Visual C#.NET to create a method and associate it with that button click.

To design a frontend (Windows form) for the rater reliability application, drag and drop a few labels, checkboxes, textboxes, and buttons to your form. Once you have these controls on your Windows form, rename them, resize them, assign default properties, and arrange them nicely on the form by changing their properties on the right hand side of the screen. Figure 7 shows some default threshold values for the flags in the form, which a user can change before clicking the submit button.


RaterRelApp - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Analyze Window Help
Debug Any CPU 136
Toolbox
All Windows Forms
Common Controls
Pointer Button CheckBox CheckedListBox ComboBox DateTimePicker Label LinkLabel ListBox ListView MaskedTextBox MonthCalendar NotifyIcon NumericUpDown PictureBox ProgressBar RadioButton RichTextBox TextBox ToolTip TreeView WebBrowser
Containers
Pointer FlowLayoutPanel GroupBox Panel SplitContainer TabControl TableLayoutPanel
Menus & Toolbars
Pointer ContextMenuStrip MenuStrip StatusStrip
Form1.Designer.cs Form1.cs Form1.cs [Design]
Output
Show output from:
Error List Output Find Symbol Results
Solution Explorer Properties
Ready

RaterRel

Select the sas dataset to run rater reliabilties

Browse

Enter Variable Name for score 1

Enter Variable Name for Score 2

Enter Grouping Variable

Enter Title1

Select Measures to be included in the report

Enter Threshold Values

✓ Means, SDs	✓ Kappa	0.60
✓ Absolute Mean Difference	✓ Linear Weighted Kappa	0.65
✓ Standardized Mean Difference	✓ Quad Weighted Kappa	0.70
✓ Matched Pair T-Test	✓ % Exact Agreement	70
	✓ % Adjacent Agreement	80
	✓ Pearson Correlation	0.70

Submit

Properties

Form1 System.Windows.Forms.Form

Accessibility	
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
Appearance	
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayo	Tile
Cursor	Default
Font	Microsoft Sans Serif, 8.2
ForeColor	ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	RaterRel
UseWaitCursor	False
Behavior	
AllowDrop	False
AutoValidate	EnablePreventFocusChar
ContextMenuStrip	(none)
DoubleBuffered	False
Enabled	True
ImeMode	NoControl
Data	
(ApplicationSettings)	
(DataBindings)	
Tag	
Design	
(Name)	Form1
Language	(Default)
Localizable	False
Locked	False
Focus	
CausesValidation	True
Layout	

Text
The text associated with the control.

Figure 7. Windows form used in our application

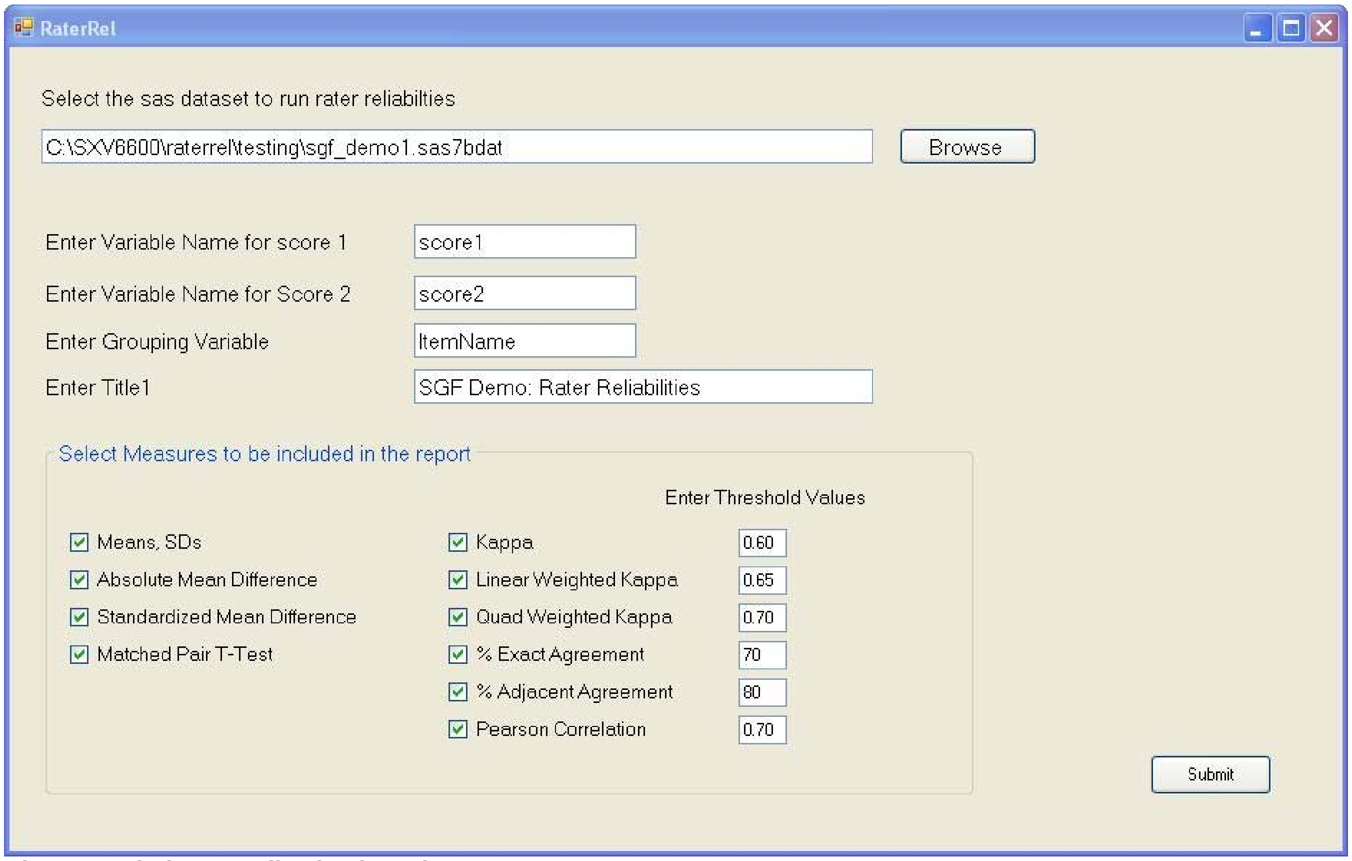
Once you double click the “Browse” button in the interface, Visual C#.NET will create a blank method in its code window. Then you just have to write the code to open and select the input data set.

```
private void browse1_Click(object sender, EventArgs e)
{
        OpenFileDialog fd1 = new OpenFileDialog();
        fd1.Title = "Select sas dataset";
        fd1.Filter = "sas datasets (*.sas7bdat)|*.sas7bdat|All files (*.*)|*.*";
        fd1.FilterIndex = 1;
        if (fd1.ShowDialog() == DialogResult.OK)
        {
                sasds_tb1.Text = fd1.FileName;
        }
}
```

The Visual C# code shown above creates a new file dialog window, asks the user to select a SAS data set, and sets the default filter for the file dialog window to “.sas7bdat” files. Once the user selects the appropriate input file, the filename is written in the textbox shown next to the “Browse” button. Once you double click the submit button, Visual C#.NET creates a blank method in the code window. Then you can write the code to get all the input from the user and submit the SAS modules with appropriate parameters.

GETTING USER INPUT

Figure 8 below shows a screenshot of the Windows application in action (running), where the user entered the necessary input.


RaterRel

Select the sas dataset to run rater reliabilties

C:\SXV6600\raterrel\testing\sgf_demo1.sas7bdat

Enter Variable Name for score 1	score1
Enter Variable Name for Score 2	score2
Enter Grouping Variable	ItemName
Enter Title1	SGF Demo: Rater Reliabilities

Select Measures to be included in the report

		Enter Threshold Values
✓ Means, SDs	✓ Kappa	0.60
✓ Absolute Mean Difference	✓ Linear Weighted Kappa	0.65
✓ Standardized Mean Difference	✓ Quad Weighted Kappa	0.70
✓ Matched Pair T-Test	✓ % Exact Agreement	70
	✓ % Adjacent Agreement	80
	✓ Pearson Correlation	0.70

Figure 8. Windows application in action

The user of this application picks the SAS data set, enters variable names, picks the statistics to be printed in the output file, changes the threshold values if needed, and clicks the submit button. All the input that the user entered in this form is gathered and read as C# string variables.

```
private void submit_button_Click(object sender, EventArgs e)
{
        // reading textboxes
        string input_sas_file_path_name = sasds_tb1.Text.ToString();
        string var1 = var1_tb.Text.ToString();
         string var2 = var2_tb.Text.ToString();
        …
        // reading checkboxes input
        if (ckappa_cb.Checked == true) ckappa = "Yes";
        if (fkappa_cb.Checked == true) fkappa = "Yes";
        …
        // reading threshold values
        string ckap_thvalue = ckap_th.Text.ToString();
        string fkap_thvalue = fkap_th.Text.ToString();
        …
}
```

Now that you have all the input from the user in the form of C# string variables, you have to pass them onto SAS and submit the SAS program.

SUBMITTING SAS CODE FROM VISUAL C#.NET

To work with SAS from Visual C#.NET, you need to add SAS references. To do this, open the solution explorer in Visual Studio, right click on “references” and choose “add reference”. Add the SAS Workspace Manager reference on the COM tab. Adding this reference will allow Visual C#.NET to communicate with SAS without actually opening the SAS environment in Windows.

Add the “using statements” in C# code for convenience.

```
using SAS;
using SASWorkspaceManager;
```

Define a new StringBuilder “sascode” in C# and append all the SAS code you wish to submit to that string.

```
private void submit_button_Click(object sender, EventArgs e)
{
        …
        …
        StringBuilder sascode = new StringBuilder();
        …
        sascode.Append("%let input_directory =" + input_directory + ";");
        …
        sascode.Append("%let variable1 = " + var1 + ";");
        sascode.Append("%let variable2 = " + var2 + ";");
        …
        sascode.Append("%let prt_ckappa = " + ckappa + ";");
        sascode.Append("%let prt_fkappa = " + fkappa + ";");
        …
        sascode.Append("%let ckappa_threshold = " + ckap_thvalue + ";");
       sascode.Append("%let fkappa_threshold = " + fkap_thvalue + ";");
        …
        …
}
```

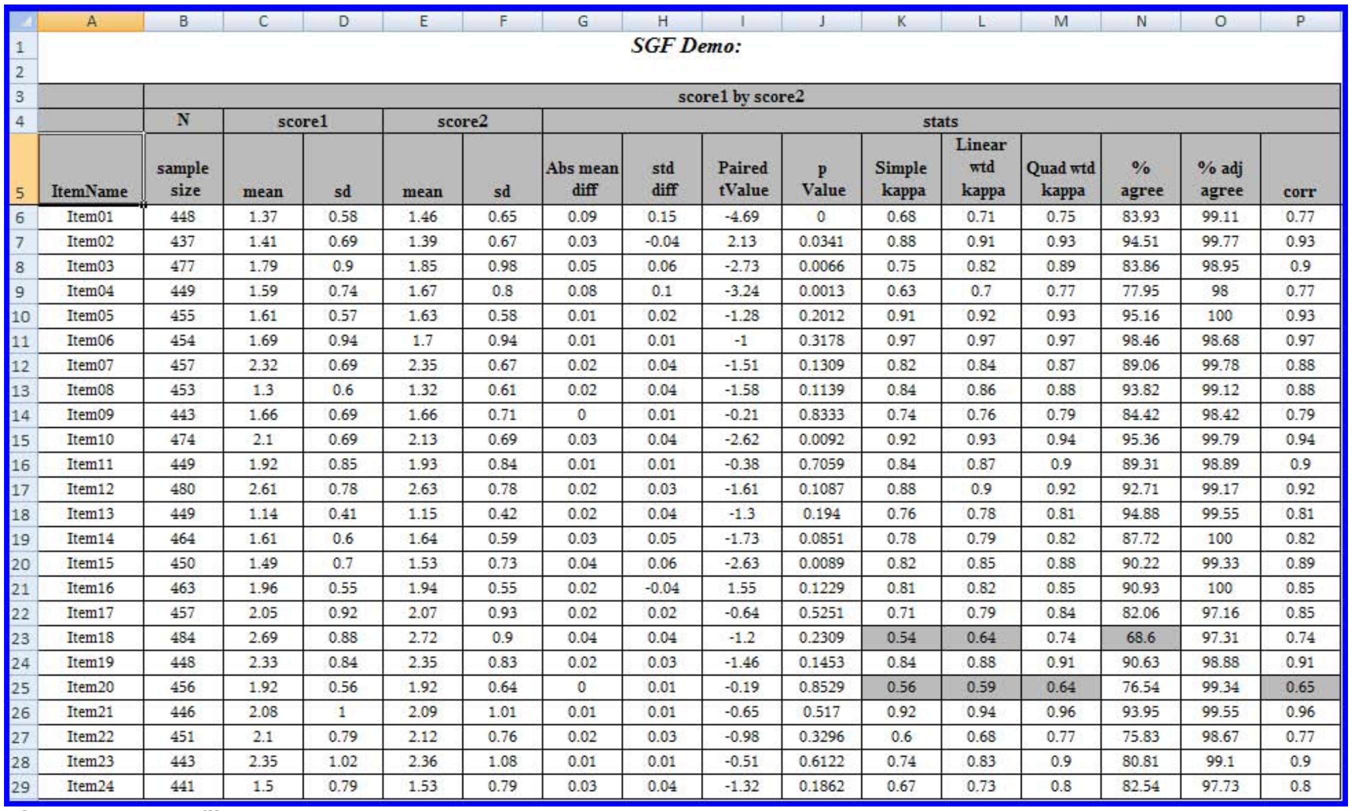
Now that you have all the input from the user assigned to global parameters in SAS, the next step is to submit the SAS program calling the modules with the necessary parameters. To do this, simply append the location of the SAS program calling the macros to the “sascode” with a %inc statement.

```
private void submit_button_Click(object sender, EventArgs e)
{
        …
        sascode.Append("%inc 'C:\\SXV6600\\raterrel\\eval_net.sas';");
        …
}
```

To submit the SAS code you appended to “sascode”, create a new instance of the SAS workspace and submit the “sascode” through the language service.

```
{
        …
        WorkspaceManager wsm = new WorkspaceManager();
        Workspace sasworkspace = Workspace)wsm.Workspaces.CreateWorkspaceByServer("",
        SASWorkspaceManager.Visibility.VisibilityProcess, null, "", "", out errorMessage);
        ILanguageService langservice = sasworkspace.LanguageService;
        langservice.Submit(sascode.ToString());
        …
}
```

Figure 9 below shows the output XML file that is created, which can easily be saved as an Excel file.



SGF Demo:

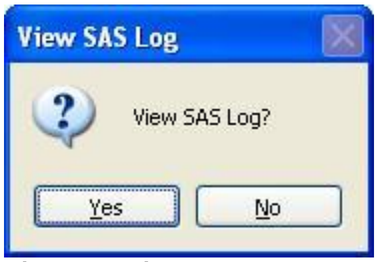
	score1 by score2														
	N	score1		score2		stats									
ItemName	sample size	mean	sd	mean	sd	Abs mean diff	std diff	Paired tValue	p Value	Simple kappa	Linear wtd kappa	Quad wtd kappa	% agree	% adj agree	corr
Item01	448	1.37	0.58	1.46	0.65	0.09	0.15	-4.69	0	0.68	0.71	0.75	83.93	99.11	0.77
Item02	437	1.41	0.69	1.39	0.67	0.03	-0.04	2.13	0.0341	0.88	0.91	0.93	94.51	99.77	0.93
Item03	477	1.79	0.9	1.85	0.98	0.05	0.06	-2.73	0.0066	0.75	0.82	0.89	83.86	98.95	0.9
Item04	449	1.59	0.74	1.67	0.8	0.08	0.1	-3.24	0.0013	0.63	0.7	0.77	77.95	98	0.77
Item05	455	1.61	0.57	1.63	0.58	0.01	0.02	-1.28	0.2012	0.91	0.92	0.93	95.16	100	0.93
Item06	454	1.69	0.94	1.7	0.94	0.01	0.01	-1	0.3178	0.97	0.97	0.97	98.46	98.68	0.97
Item07	457	2.32	0.69	2.35	0.67	0.02	0.04	-1.51	0.1309	0.82	0.84	0.87	89.06	99.78	0.88
Item08	453	1.3	0.6	1.32	0.61	0.02	0.04	-1.58	0.1139	0.84	0.86	0.88	93.82	99.12	0.88
Item09	443	1.66	0.69	1.66	0.71	0	0.01	-0.21	0.8333	0.74	0.76	0.79	84.42	98.42	0.79
Item10	474	2.1	0.69	2.13	0.69	0.03	0.04	-2.62	0.0092	0.92	0.93	0.94	95.36	99.79	0.94
Item11	449	1.92	0.85	1.93	0.84	0.01	0.01	-0.38	0.7059	0.84	0.87	0.9	89.31	98.89	0.9
Item12	480	2.61	0.78	2.63	0.78	0.02	0.03	-1.61	0.1087	0.88	0.9	0.92	92.71	99.17	0.92
Item13	449	1.14	0.41	1.15	0.42	0.02	0.04	-1.3	0.194	0.76	0.78	0.81	94.88	99.55	0.81
Item14	464	1.61	0.6	1.64	0.59	0.03	0.05	-1.73	0.0851	0.78	0.79	0.82	87.72	100	0.82
Item15	450	1.49	0.7	1.53	0.73	0.04	0.06	-2.63	0.0089	0.82	0.85	0.88	90.22	99.33	0.89
Item16	463	1.96	0.55	1.94	0.55	0.02	-0.04	1.55	0.1229	0.81	0.82	0.85	90.93	100	0.85
Item17	457	2.05	0.92	2.07	0.93	0.02	0.02	-0.64	0.5251	0.71	0.79	0.84	82.06	97.16	0.85
Item18	484	2.69	0.88	2.72	0.9	0.04	0.04	-1.2	0.2309	0.54	0.64	0.74	68.6	97.31	0.74
Item19	448	2.33	0.84	2.35	0.83	0.02	0.03	-1.46	0.1453	0.84	0.88	0.91	90.63	98.88	0.91
Item20	456	1.92	0.56	1.92	0.64	0	0.01	-0.19	0.8529	0.56	0.59	0.64	76.54	99.34	0.65
Item21	446	2.08	1	2.09	1.01	0.01	0.01	-0.65	0.517	0.92	0.94	0.96	93.95	99.55	0.96
Item22	451	2.1	0.79	2.12	0.76	0.02	0.03	-0.98	0.3296	0.6	0.68	0.77	75.83	98.67	0.77
Item23	443	2.35	1.02	2.36	1.08	0.01	0.01	-0.51	0.6122	0.74	0.83	0.9	80.81	99.1	0.9
Item24	441	1.5	0.79	1.53	0.79	0.03	0.04	-1.32	0.1862	0.67	0.73	0.8	82.54	97.73	0.8

Figure 9. Output file

The grey cells in the above output indicate the reliability values lower than the threshold levels set by the user.

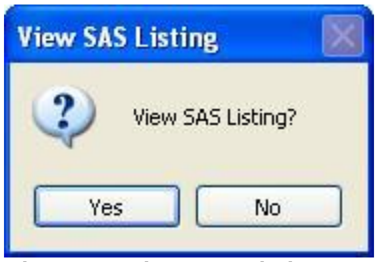
VIEWING SAS LOG AND LISTING

You can use Message Boxes in Visual C#.NET to ask the user if he or she wants to view the SAS log and listing.



View SAS Log?

Figure 10. View SAS Log?

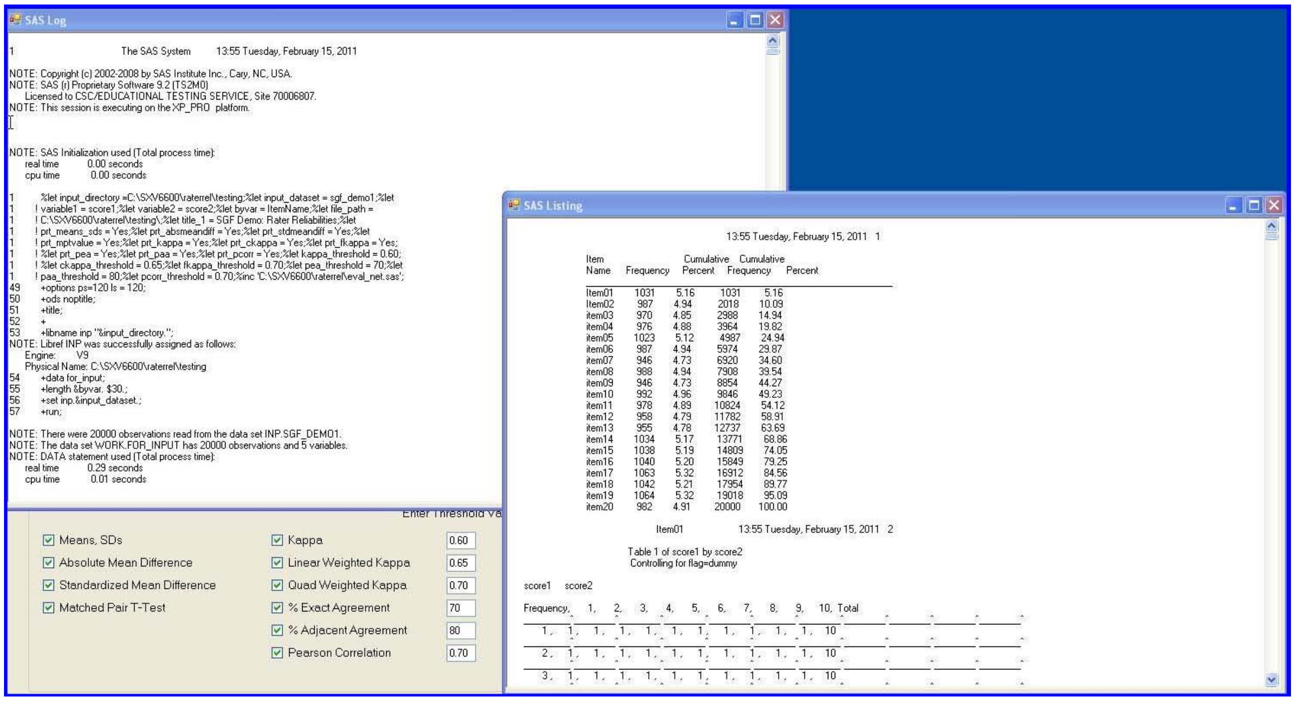


View SAS Listing?

Figure 11. View SAS Listing?

If the user clicked yes, you can create a new form, then add a rich textbox to display the SAS log and/or listing.

```
private void submit_button_Click(object sender, EventArgs e)
{
        …
        if (MessageBox.Show("View SAS Log?", "View SAS Log",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
        {
                string saslog = langservice.FlushLog(10000000).Replace("/f", "");
                RichTextBox rtb = new RichTextBox();
                rtb.Size = new System.Drawing.Size(895, 545);
                Form f2 = new Form();
                f2.Text = "SAS Log";
                f2.ClientSize = new System.Drawing.Size(900, 550);
                f2.Controls.Add(rtb);
                rtb.Text = saslog;
                f2.Show();
        }
        //To flush sas listing use this
        if (MessageBox.Show("View SAS Listing?", "View SAS Listing",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
        {
                string saslisting = langservice.FlushList(100000000).Replace("ƒ", "_");
                RichTextBox rtb2 = new RichTextBox();
                rtb2.Size = new System.Drawing.Size(895, 545);
                Form f3 = new Form();
                f3.Text = "SAS Listing";
                f3.ClientSize = new System.Drawing.Size(900, 550);
                f3.Controls.Add(rtb2);
                rtb2.Text = saslisting;
                f3.Show();
        }
        …
}
```



```
1                         The SAS System        13:55 Tuesday, February 15, 2011

NOTE: Copyright (c) 2002-2008 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software 9.2 (TS2M0)
      Licensed to CSC/EDUCATIONAL TESTING SERVICE, Site 70006807.
NOTE: This session is executing on the XP_PRO  platform.


NOTE: SAS Initialization used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

1         %let input_directory =C:\SXV6600\raterrel\testing;%let input_dataset = sgf_demo1;%let
1       ! variable1 = score1;%let variable2 = score2;%let byvar = ItemName;%let file_path =
1       ! C:\SXV6600\raterrel\testing\;%let title_1 = SGF Demo: Rater Reliabilities;%let
1       ! prt_means_sds = Yes;%let prt_absmeandiff = Yes;%let prt_stdmeandiff = Yes;%let
1       ! prt_mptvalue = Yes;%let prt_kappa = Yes;%let prt_ckappa = Yes;%let prt_fkappa = Yes;
1       ! %let prt_pea = Yes;%let prt_paa = Yes;%let prt_pcorr = Yes;%let kappa_threshold = 0.60;
1       ! %let ckappa_threshold = 0.65;%let fkappa_threshold = 0.70;%let pea_threshold = 70;%let
1       ! paa_threshold = 80;%let pcorr_threshold = 0.70;%inc 'C:\SXV6600\raterrel\eval_net.sas';
49        +options ps=120 ls = 120;
50        +ods noptitle;
51        +title;
52        +
53        +libname inp "&input_directory.";
NOTE: Libref INP was successfully assigned as follows:
      Engine:        V9
      Physical Name: C:\SXV6600\raterrel\testing
54        +data for_input;
55        +length &byvar. $30.;
56        +set inp.&input_dataset.;
57        +run;

NOTE: There were 20000 observations read from the data set INP.SGF_DEMO1.
NOTE: The data set WORK.FOR_INPUT has 20000 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.29 seconds
      cpu time            0.01 seconds
```

- [✓] Means, SDs
- [✓] Absolute Mean Difference
- [✓] Standardized Mean Difference
- [✓] Matched Pair T-Test
- [✓] Kappa 0.60
- [✓] Linear Weighted Kappa 0.65
- [✓] Quad Weighted Kappa 0.70
- [✓] % Exact Agreement 70
- [✓] % Adjacent Agreement 80
- [✓] Pearson Correlation 0.70

```
                                   13:55 Tuesday, February 15, 2011   1

              Item                      Cumulative    Cumulative
              Name      Frequency     Percent   Frequency     Percent
              ________________________________________________________
              Item01       1031        5.16        1031        5.16
              Item02        987        4.94        2018       10.09
              item03        970        4.85        2988       14.94
              item04        976        4.88        3964       19.82
              item05       1023        5.12        4987       24.94
              item06        987        4.94        5974       29.87
              item07        946        4.73        6920       34.60
              item08        988        4.94        7908       39.54
              item09        946        4.73        8854       44.27
              item10        992        4.96        9846       49.23
              item11        978        4.89       10824       54.12
              item12        958        4.79       11782       58.91
              item13        955        4.78       12737       63.69
              item14       1034        5.17       13771       68.86
              item15       1038        5.19       14809       74.05
              item16       1040        5.20       15849       79.25
              item17       1063        5.32       16912       84.56
              item18       1042        5.21       17954       89.77
              item19       1064        5.32       19018       95.09
              item20        982        4.91       20000      100.00

                       Item01              13:55 Tuesday, February 15, 2011   2

                    Table 1 of score1 by score2
                     Controlling for flag=dummy

score1     score2

Frequency,    1,    2,    3,    4,    5,    6,    7,    8,    9,   10,  Total
_____________________________________________________________________________
      1 ,     1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,    10
_____________________________________________________________________________
      2 ,     1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,    10
_____________________________________________________________________________
      3 ,     1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,   1 ,    10
```

Figure 12. Viewing SAS Log and Listing

CONCLUSION

Applying the same principle used in this paper, SAS and Visual C#.NET can be used to build Windows and Web applications for doing data analysis and producing custom reports. SAS is a great tool for doing data analysis, and Visual C#.NET provides a quick and easy way to build Windows and Web interfaces to interact with the user.

REFERENCES

SAS Institute Inc. 2009. *SAS® 9.2 Integration Technologies: Windows Client Developer's Guide*. Cary, NC: SAS Institute Inc

ACKNOWLEDGEMENTS

I would like to thank my colleagues and friends at ETS for supporting this paper and my professional development and my beautiful wife Meg for her endless support in all I do and her help with editing this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Sailesh Vezzu
Educational Testing Service
Rosedale Road, MS 20T
Princeton, NJ 08541
(609) 734 -5786
svezzu@ets.org

TRADEMARKS

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® Indicates USA registration. Other brand and product names are trademarks of their respective companies.