User's Guide

Publication number E2699-97001
October 2003

My Infiniium Integration Package

Contents

1

E2699A My Infiniium Integration Package Features

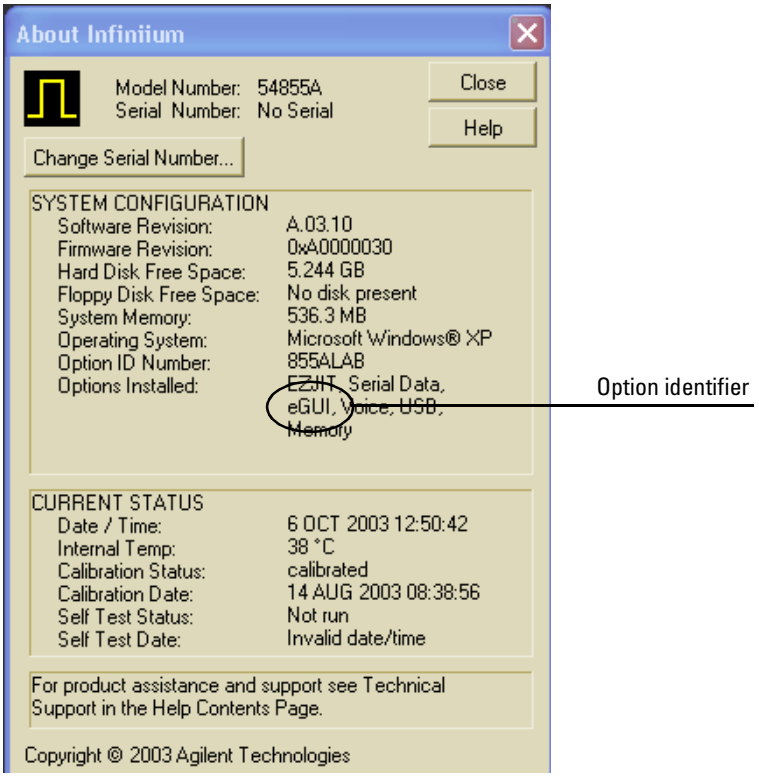
Checking for the Installed License

The My Infiniium Integration Package can only be installed on Infiniium 54830 series and 54850 series oscilloscopes with A.03.10 or later version of system software (Windows XP Professional).

Before you can use the My Infiniium Integration Package, the license must be installed on your Infiniium oscilloscope. To check for the installed license use the following instructions.

1. Select the Help menu on the Infiniium menu bar.
2. Select About Infiniium from the pull-down menu.
3. In the About Infiniium dialog box in the SYSTEM CONFIGURATION area there is an Options Installed line. If you see eGUI in the list of installed options then the My Infiniium Integration Package has been installed.

Figure 1-1


About Infiniium
Model Number: 54855A
Serial Number: No Serial
Close
Help
Change Serial Number...
SYSTEM CONFIGURATION
Software Revision: A.03.10
Firmware Revision: 0xA0000030
Hard Disk Free Space: 5.244 GB
Floppy Disk Free Space: No disk present
System Memory: 536.3 MB
Operating System: Microsoft Windows® XP
Option ID Number: 855ALAB
Options Installed: EZIIT, Serial Data, eGUI, Voice, USB, Memory
Option identifier
CURRENT STATUS
Date / Time: 6 OCT 2003 12:50:42
Internal Temp: 38 °C
Calibration Status: calibrated
Calibration Date: 14 AUG 2003 08:38:56
Self Test Status: Not run
Self Test Date: Invalid date/time
For product assistance and support see Technical Support in the Help Contents Page.
Copyright © 2003 Agilent Technologies

QuickExecute and Extensible Graphical User Interface

The My Infiniium Integration Package consists of two capabilities:

- QuickExecute
- Extensible Graphical User Interface (eGUI)

These capabilities allow you to extend the power of your oscilloscope by letting you launch your application directly from Infiniium's front panel or from Infiniium's graphical user interface.

QuickExecute is an additional user choice to the QuickMeas+ feature. When this option is selected, each time the QuickMeas+ button on the front panel is pressed, Infiniium will run an executable file that you have chosen.

The Extensible Graphical User Interface (eGUI) provides a method to add menu items to the Infiniium menu system. These menu entries that you add can run executable programs that you create.

You have great flexibility in the types of programs that can be run from QuickExecute or eGUI. Any program that can be run under Windows XP is a candidate for these features. You can develop and debug your programs on an external PC, using the language and tools of your choice.

Your programs can control the oscilloscope using the Agilent I/O Libraries which can be found on the "Manuals, Example Programs, and SICL/VISA Library Installation" CD-ROM that came with your oscilloscope. You can control the oscilloscope using either the GPIB or LAN physical interface. When run inside the oscilloscope, you use the special internal LAN address "lan[localhost]:inst0". For example, your program can instruct the oscilloscope to acquire data and then your program can retrieve it, via the hard disk or directly using the I/O commands. Then your program can perform the custom analysis and display the results via a dialog box.

For higher-level oscilloscope control, use the Infiniium IVI-COM driver in your program. The IVI-COM driver takes full advantage of industry-accepted standards and is compatible programming environments such as Microsoft Visual Studio, Agilent VEE Pro, and National Instruments LabVIEW. The Infiniium IVI-COM driver provides ease of use, higher performance, and interchangeability for your oscilloscope control program. You can download the Infiniium IVI-COM driver and documentation for free at the Agilent Developers Network, www.agilent.com/find/adn.

With the My Infiniium Integration Package, your programs become an integral part of the oscilloscope. In fact, if you are currently running a custom analysis program on an external PC, it is very easy to move it into your oscilloscope using My Infiniium. If you are not experienced with using

applications programs with your Infiniium, Agilent provides some example programs to help you become familiar with the capabilities and benefits of My Infiniium. These programs also illustrate the wide variety of actions that are possible using My Infiniium.

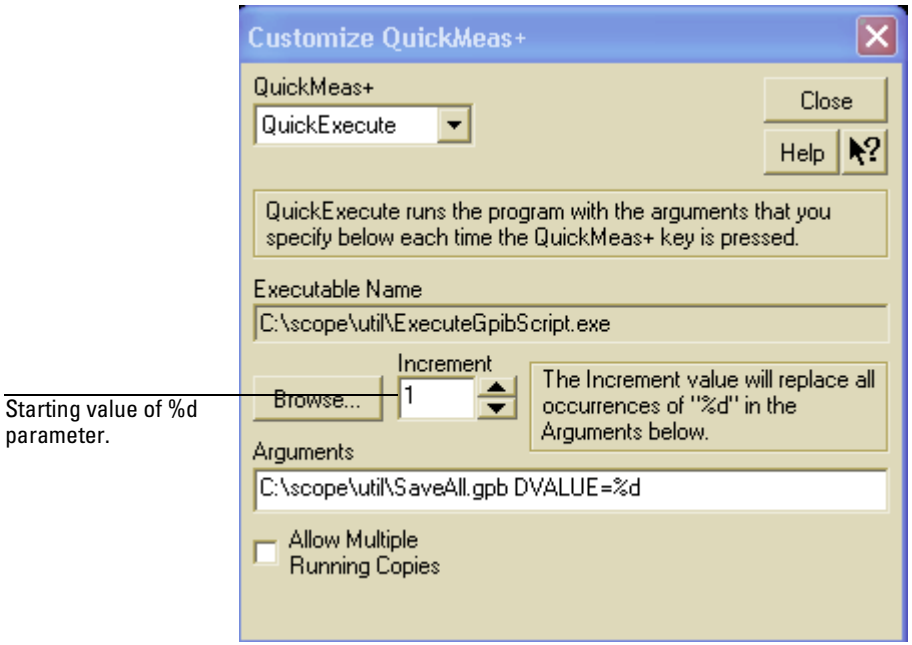
Example 1 - Scripting GPIB Commands

This example uses the ExecuteGpibScript.exe program, with the command file SaveAll.gpb. ExecuteGpibScript is a generic tool for executing groups of GPIB commands. The commands contained in SaveAll.gpb are shown below:

```
:Stop

:Disk:Store Chan1,"c:\scope\data\Acq%d_Ch1.csv",Text,XYPairs,On;
:Disk:Store Chan2,"c:\scope\data\Acq%d_Ch2.csv",Text,XYPairs,On;
:Disk:Store Chan3,"c:\scope\data\Acq%d_Ch3.csv",Text,XYPairs,On;
:Disk:Store Chan4,"c:\scope\data\Acq%d_Ch4.csv",Text,XYPairs,On;

:Disk:Store Setup,"c:\scope\data\Acq%d.set";
:Disk:SImage "c:\scope\data\Acq%d.bmp",BMP,Screen,On,Normal;
```

When ExecuteGpibScript.exe is mapped to QuickExecute or an eGUI menu item, these commands cause Infiniium to store all four channels of data as well as the scope setup and screen image to a set of files on the hard disk. The %d represents a numerical value that increments each time the script is run. The starting value can be set from the Customize QuickMeas+ dialog as shown in Figure 1-2.

Figure 1-2



Increment Control

Considering that nearly all of Infiniium's capabilities are controllable using GPIB, it should be clear that many valuable control sequences are possible using

ExecuteGpibScript.exe in this manner.

Example 2 - Mapping A Control To The Front Panel

This example uses a short C program, StepAveraging.cpp, to map a frequently used control to the oscilloscope's front panel for quick access. The source code is show below:

```
/* StepAveraging.cpp -- Sample QuickExecute application that steps through
                        several levels of averaging for fast viewing comparisons.
*/

#include <windows.h>
#include <sicl.h>

static INST Scope = 0;

static void ErrorHandler(INST Inst, int Error)
{
   /* This routine traps any I/O errors, so we don't have to check
      the return values of individual function calls.
   */

   char Text[1024];

   strcpy(Text, "This program has experienced a problem communicating \n");
  strcat(Text, "with the oscilloscope application.  (The error string is: \n");
   strcat(Text, igeterrstr(Error));
   strcat(Text, ").");

   MessageBox(NULL, Text, GetCommandLine(), MB_ICONEXCLAMATION);

   ionerror(0);    /* Clear the error handler. */
   iclose(Scope);  /* Close the session. */

   exit(EXIT_FAILURE);
}

static void WriteString(char *String)
{
   /* Write a command or query to the oscilloscope. */

   iwrite(Scope, String, strlen(String), 1, NULL);
}

static void ReadString(char *String)
{
   /* Read query results back from the oscilloscope. */

   unsigned long ActualCount = 0;

   iread(Scope, String, strlen(String), NULL, &ActualCount);

   String[ActualCount - 1] = '\0';
}
```

```
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine, int nCmdShow)
{
   char StateQueryResults[16];
   char NumberQueryResults[16];
   int  NumberOfAverages;

   ionerror(ErrorHandler);  /* Install the error handler. */
   itimeout(Scope, 10000);  /* Set the timeout value in milliseconds. */
   WriteString("*CLS");     /* Clear the error queue. */

   Scope = iopen("lan[localhost]:hpib7,7");  /* Open the session. */

   WriteString("SYSTEM:HEADER OFF");

   WriteString("ACQUIRE:AVERAGE?");
   ReadString(StateQueryResults);

   /* Cycle through 0, 4, 32, and 256 averages on successive invocations. */
   if (StateQueryResults[0] == '0')
   {
      WriteString("ACQUIRE:AVERAGE:COUNT 4");
      WriteString("ACQUIRE:AVERAGE ON");
   }
   else
   {
      WriteString("ACQUIRE:AVERAGE:COUNT?");
      ReadString(NumberQueryResults);
      NumberOfAverages = atoi(NumberQueryResults);
      if (NumberOfAverages <= 4)
      {
         WriteString("ACQUIRE:AVERAGE:COUNT 32");
         WriteString("ACQUIRE:AVERAGE ON");
      }
      else if (NumberOfAverages <= 32)
      {
         WriteString("ACQUIRE:AVERAGE:COUNT 256");
         WriteString("ACQUIRE:AVERAGE ON");
      }
      else
      {
         WriteString("ACQUIRE:AVERAGE OFF");
      }
   }
   iclose(Scope);
   return EXIT_SUCCESS;
}
```

As the comments indicate, this program steps the oscilloscope through 0, 4, 32, and 256 levels of averaging for fast viewing comparisons. This program can be used as a pattern or template for accomplishing similar control mappings that may be desired for specific measurement situations. Also included as an example program with My Infiniium which is similar to StepAveraging.cpp and is called ToggleAveraging.cpp. Instead of stepping through successive levels of averaging, ToggleAveraging.cpp toggles averaging on or off using the front panel QuickMeas+ key.

Example 3 - Creating a Dialog Box with Custom Measurement Results

This example program, SlewRate.cpp, is a C program that performs a slew rate measurement and displays the result in a dialog box on the Infiniium display. The source code is shown below:

```
// SlewRate.cpp  -- Sample Infiniium eGUI application using
//                  VISA and the Agilent I/O Libraries.
//
//                  Usage: SlewRate [n]
//                         where n = channel number (default is 1)
//
// This program measures and displays the slew rate of the given channel
// based on the current upper and lower threshold voltages.
//
// To use this program as a template for similar operations, change the code
// in the last function listed here, PerformAction.  All other functions may
// be directly reused.  Remember to rename the program appropriately.
//
// This listing is provided as an example only without expressed or
// implied warranties.
//

#include <windows.h>
#include <stdio.h>
#include <math.h>
#include <visa.h>

#define SHORTBUF   80
#define LONGBUF  1024
#define SCOPE_ADDRESS  "GPIB0::7::INSTR"
#define TIMEOUT_SECONDS  10
static const char *WindowTitle = "Measure Slew Rate";
static const int GOOD_RESULT = 0;
static const int BAD_RESULT = 4;
static ViSession Scope = VI_NULL;
static void PerformAction(char *CommandLineArgs);
```

```
//************************ ErrorHandler *********************************
//
//  Description: The error handler for all Agilent I/O Library calls.
//               If an error occurs, it is trapped here and reported to the
//               user before the program terminates.  This allows us to avoid
//              checking the return values of individual I/O call for errors.
//
//   Parameters: Session -- the VISA session identifier.
//               EventType -- the logical event identifier.
//               Event -- the event handle.
//               UserHandle -- user handle for this event and session.
//
//      Returns: VI_SUCCESS for successful handling.
//
//         Note: It is critical that _VI_FUNCH be included in the declaration
//               for proper stack cleanup.
//

static ViStatus _VI_FUNCH ErrorHandler(ViSession   Session,
                                       ViEventType EventType,
                                       ViEvent     Event,
                                       ViAddr      UserHandle)
{
   ViStatus ErrorNumber;
   char FunctionName[LONGBUF];
   char ErrorString[LONGBUF];

   // Fetch the error value and the function name.
   viGetAttribute(Event, VI_ATTR_STATUS, &ErrorNumber);
   viGetAttribute(Event, VI_ATTR_OPER_NAME, FunctionName);

   // Map the error number to an error description.
   ErrorString[0] = '\0';
   viStatusDesc(Session, ErrorNumber, ErrorString);

   // Tell the user what happened.
   char Title[LONGBUF];
   char *CommandLine = ::GetCommandLine();
   strcpy(Title, "Error executing ");
   strcat(Title, CommandLine);

   char Text[LONGBUF];
   strcpy(Text, "This program has experienced a problem communicating ");
   strcat(Text, "with the oscilloscope application.\n");
   strcat(Text, "(The I/O function is: \"");
   strcat(Text, FunctionName);
   strcat(Text, "\", and the error string is: \n");
   strcat(Text, "\"");
```

```
   strcat(Text, ErrorString);
   strcat(Text, "\".) \n\n");
   strcat(Text, "Please make sure the Agilent I/O Libraries are installed ");
   strcat(Text, "and configured correctly.");

   MessageBox(NULL, Text, Title, MB_OK | MB_ICONEXCLAMATION);

   exit(EXIT_FAILURE);

   // This point is never reached but it makes the compiler happy.
   return VI_SUCCESS;
}

//************************************************************ WriteString
***
//
//  Description: Send a command or query string to the oscilloscope.
//
//   Parameters: String -- the character string to write out.
//

static void WriteString(char *String)
{
   viWrite(Scope, (unsigned char *)String, strlen(String), VI_NULL);
}

//************************************************************* ReadString
***
//
//  Description: Read query results back from the oscilloscope.
//
//   Parameters: String -- a buffer to place the results into.
//               MaxChars -- the size of the buffer.
//

static void ReadString(char *String, int MaxChars)
{
   unsigned long ActualCount = 0;

   viRead(Scope, (unsigned char *)String, MaxChars, &ActualCount);

   String[ActualCount - 1] = '\0';
}
```

```
//************************************************************* WinMain
***
//
//  Description: The entry point for the program.
//
//   Parameters: hInstance -- the handle to this instance of the program.
//              hPrevInstance -- the handle to the previous program instance.
//               lpCmdLine -- pointer to the command line characters.
//               nCmdShow -- the show state for the program.
//
//               These parameters are not typically required for Infiniium
//               programs, though lpCmdLine can be useful.  See the
//               Windows (tm) documentation for more details if required.
//
//    Returns: EXIT_SUCCESS for successful completion, EXIT_FAILURE otherwise.
//

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
   ViSession ResourceMgr;

   // Initialize the interface to the oscilloscope.
   viOpenDefaultRM(&ResourceMgr);
   viInstallHandler(ResourceMgr, VI_EVENT_EXCEPTION, ErrorHandler, VI_NULL);
   viEnableEvent(ResourceMgr, VI_EVENT_EXCEPTION, VI_HNDLR, VI_NULL);

   viOpen(ResourceMgr, SCOPE_ADDRESS, VI_NULL, VI_NULL, &Scope);
   viInstallHandler(Scope, VI_EVENT_EXCEPTION, ErrorHandler, VI_NULL);
   viEnableEvent(Scope, VI_EVENT_EXCEPTION, VI_HNDLR, VI_NULL);

   viSetAttribute(Scope, VI_ATTR_TMO_VALUE, TIMEOUT_SECONDS * 1000);
   viClear(Scope);

   WriteString("*CLS");
   WriteString("SYSTEM:HEADER OFF");

   // Take action as appropriate for this program.
   PerformAction(lpCmdLine);


   // Clean up.
   viDisableEvent(Scope, VI_EVENT_EXCEPTION, VI_HNDLR);
   viUninstallHandler(Scope, VI_EVENT_EXCEPTION, ErrorHandler, VI_NULL);
   viClose(Scope);
```

```
   viDisableEvent(ResourceMgr, VI_EVENT_EXCEPTION, VI_HNDLR);
  viUninstallHandler(ResourceMgr, VI_EVENT_EXCEPTION, ErrorHandler, VI_NULL);
   viClose(ResourceMgr);

   return EXIT_SUCCESS;
}

//************************************************************ GetChannel
***
//
//  Description: Helper function for PerformAction below.  Determine the
//               channel to make the measurement on, using the command-line
//               parameter if it was passed and is valid.
//
//      Returns: An integer from 1-4 indicating the channel, defaults to 1.
//               Channels greater than 4 are mapped down into the 1-4 range.
//

static int GetChannel(char *CommandLineArgs)
{
   int Channel = 1;

   if (CommandLineArgs != NULL)
   {
      Channel = atoi(CommandLineArgs);

      if (Channel < 1)
      {
         Channel = 1;
      }
      else if (Channel > 4)
      {
         // Wrap 5-8, 9-12, 13-16, etc. all down to 1-4.
         Channel = ((Channel - 1) % 4) + 1;
      }
   }

   return Channel;
}
```

```
//********************** MakeMeasurement ********************************
//
//  Description: Helper function for PerformAction below.  Make one of the
//               measurements we need in order to calculate the slew rate.
//
//   Parameters: QueryString -- the scope query for making the measurement.
//               MeasResults -- the numerical measurement result.
//
//      Returns: The measurement status code from the scope.
//
//               A value of 0 means a valid result, values from 1-3 indicate
//               questionable results, and values greater than or equal to 4
//               indicate an error of some sort.  See the scope Programmer's
//               Reference for details.
//

static int MakeMeasurement(char   *QueryString,
                           double *MeasResults)
{
   // We need these two values to differentiate from valid atoi and atof
   // conversions that return zero, and invalid ones, which unfortunately
   // also return zero.

   static const char *ZeroFloat = "0.0E+00";
   static const char *ZeroInt = "0";

   int StatusCode = BAD_RESULT;
   char RawResults[SHORTBUF];

   WriteString("*CLS");
   WriteString(QueryString);
   ReadString(RawResults, SHORTBUF);

   char *ResultsString = strtok(RawResults, ",");
   char *StatusString = strtok(NULL, " ");

   if (StatusString != NULL)
   {
      StatusCode = atoi(StatusString);
      if (StatusCode == 0 && strcmp(StatusString, ZeroInt))
      {
         StatusCode = BAD_RESULT;
      }
   }
```

```
   if (StatusCode < BAD_RESULT)
   {
      if (ResultsString != NULL)
      {
         *MeasResults = atof(ResultsString);
         if (*MeasResults == 0.0 && strcmp(ResultsString, ZeroFloat))
         {
            StatusCode = BAD_RESULT;
         }
      }
   }

   return StatusCode;
}

//*********************** ScalingInfo ***********************************
//
//  Description: Not a function, but some data used by FormatResults below.
//               This allows us to display the slew rate results with the
//               correct SI prefix, for example V/ns or V/us, depending on
//               its magnitude.
//

static const int N_PREFIXES = 11;
static const int NO_PREFIX = 5;

struct tScalingInfo
{
   double  LowerLimit;
   double  UpperLimit;
   double  Multiplier;
   char    PrefixString[2];
};

static const tScalingInfo ScalingInfo[N_PREFIXES] =
{  1E-15, 1E-12, 1E+15, "P",
   1E-12, 1E-9,  1E+12, "T",
   1E-9,  1E-6,  1E+9,  "G",
   1E-6,  1E-3,  1E+6,  "M",
   1E-3,  1E+0,  1E+3,  "k",
   1E+0,  1E+3,  1E+0,  "" ,
   1E+3,  1E+6,  1E-3,  "m",
   1E+6,  1E+9,  1E-6,  "u",
   1E+9,  1E+12, 1E-9,  "n",
   1E+12, 1E+15, 1E-12, "p",
   1E+15, 1E+18, 1E-15, "f"
};
```

```
//********************** FormatResults ***********************************
//
//  Description: Helper function for PerformAction below.  Format the
//               numerical slew rate value into a presentable format,
//               including the correct units and prefix.
//
//   Parameters: SlewRate -- the valid, numerical slew rate measurement.
//              Channel -- the number (1-4) of the channel that was measured.
//               FormattedResults -- a string to hold the formatted results.
//

static void FormatResults(double  SlewRate,
                          int     Channel,
                          char   *FormattedResults)
{
   // Fetch the units from the scope.
   char UnitsQuery[SHORTBUF];
   char Units[SHORTBUF];

   sprintf(UnitsQuery, "CHAN%d:UNITS?", Channel);
   WriteString(UnitsQuery);
   ReadString(Units, SHORTBUF);

   // Now scale and format the results.
   double ScaledResults = SlewRate;
   double AbsResults = fabs(SlewRate);

   int i = 0;
   BOOL Found = FALSE;

   while (i < N_PREFIXES && !Found)
   {
     Found = (AbsResults >= ScalingInfo[i].LowerLimit &&
              AbsResults <  ScalingInfo[i].UpperLimit);
     i++;
   }
```

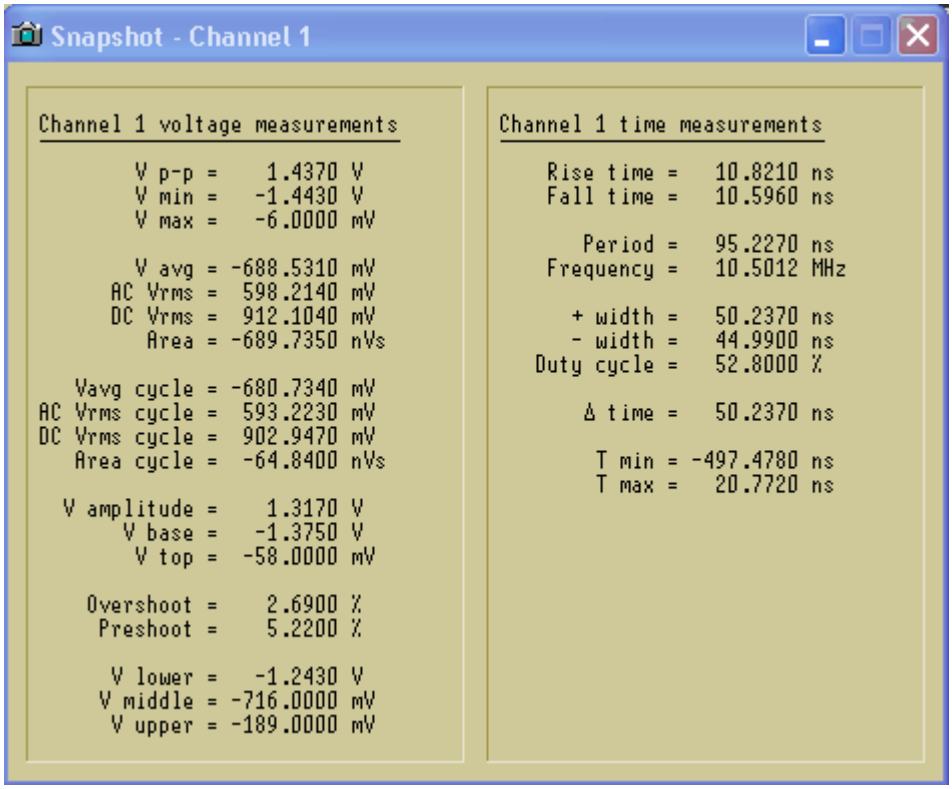
```
   if (Found)
   {
      i--;
   }
   else
   {
      // Make sure to provide for cases that are outside the table,
      // as well as the special "exactly 0" case.
      if (AbsResults > 0.0 && AbsResults < ScalingInfo[0].LowerLimit)
      {
         i = 0;
      }
      else if (AbsResults >= ScalingInfo[N_PREFIXES - 1].UpperLimit)
      {
         i = N_PREFIXES - 1;
      }
      else
      {
         i = NO_PREFIX;
      }
   }

   ScaledResults *= ScalingInfo[i].Multiplier;

   sprintf(FormattedResults, "%.2f %c/%ss",
                             ScaledResults,
                             Units[0],
                             ScalingInfo[i].PrefixString);
}

//****************** PerformAction ************************************
//
//  Description: This is a generalized routine for taking action based on
//               the requirements of the program.  In this case we are
//               calculating the slew rate of one of the scope's channels.
//

static void PerformAction(char *CommandLineArgs)
{
   // Make the three measurements that we need.  We need to stop the scope
   // so that all measurements are performed on the same acquisition data.
   int Channel = GetChannel(CommandLineArgs);
   double Risetime, Vlower, Vupper;

   WriteString("STOP");
   WriteString("MEASURE:SENDVALID ON");

   char RisetimeQuery[SHORTBUF];
```

```
   char VlowerQuery[SHORTBUF];
   char VupperQuery[SHORTBUF];

   sprintf(RisetimeQuery, "MEASURE:RISETIME? CHAN%d", Channel);
   sprintf(VlowerQuery, "MEASURE:VLOWER? CHAN%d", Channel);
   sprintf(VupperQuery, "MEASURE:VUPPER? CHAN%d", Channel);

   int RisetimeResultCode = MakeMeasurement(RisetimeQuery, &Risetime);
   int VlowerResultCode = MakeMeasurement(VlowerQuery, &Vlower);
   int VupperResultCode = MakeMeasurement(VupperQuery, &Vupper);

   // Calculate the slew rate if possible and format the results.
   double SlewRate;
   char ResultString[LONGBUF];

   sprintf(ResultString, "Slew rate (%d) ", Channel);

   if (RisetimeResultCode >= BAD_RESULT  ||
       VlowerResultCode >= BAD_RESULT    ||
       VupperResultCode >= BAD_RESULT)
   {
      strcat(ResultString, "cannot be calculated.");
      strcat(ResultString, "\nPlease verify that the channel is");
      strcat(ResultString, "\ndisplayed and scaled correctly.");
   }
   else
   {
      SlewRate = (Vupper - Vlower) / Risetime;
      if (RisetimeResultCode == GOOD_RESULT  &&
          VlowerResultCode == GOOD_RESULT    &&
          VupperResultCode == GOOD_RESULT)
      {
         strcat(ResultString, "= ");
      }
      else
      {
         strcat(ResultString, "? ");
      }

      char FormattedResults[LONGBUF];
      FormatResults(SlewRate, Channel, FormattedResults);
      strcat(ResultString, FormattedResults);
   }
   // Finally display the results to the user.
   MessageBox(NULL, ResultString, WindowTitle, MB_OK);
}
```

Example 4 - The Snapshot Measurement Utility

The final example involves a more complete measurement utility called Snapshot.exe. The source code for Snapshot is not provided but the executable is provided. When mapped to QuickExecute or eGUI menu, Snapshot provides a "measure all" capability on a source-by-source basis. Each invocation advances the measurement source. A screen shot of Snapshot's results dialog is shown in Figure 1-3.

Figure 1-3



Snapshot dialog box

Snapshot is a dialog-based MFC application that makes use of the GPIB commands in Infiniium's MEASURE subsystem. It is a good example of the integration value of My Infiniium.

Adding Menu Items to Infiniium

Menu items can be added to any of the existing Infiniium menus using the eGUI feature of My Infiniium. However, menu items cannot be added to the main menu bar. While you can add menu items to any menu, it is recommended that you add menu items to the Analyze or Utilities menus.

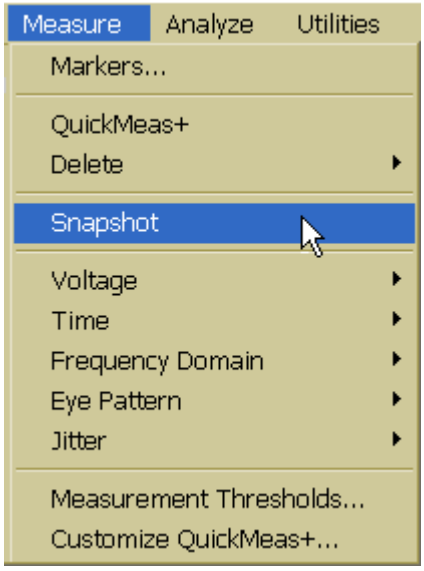
Menu items must be added to the file eGUI.txt found in the C:\SCOPE\CONFIG directory. This file is a text only ASCII file which can be edited using any text editor.

The following is an example of adding a menu item for SnapShot.exe.

```
EGUI_ITEM_MENU
   MENU_PATH={MAIN\Measure\Snapshot}
   MENU_RUN={C:\scope\util\Snapshot.exe}
   MENU_RUN_ARGS={}
   MENU_POS={4}
   MENU_SEP={BEFORE}
   EGUI_FLAGS={ALLOW_MULTIPLE_RUNNING}
EGUI_END_ITEM;
```

Figure 1-4 shows the resulting menu item that was create by preceding text in the eGUI.txt file.

Figure 1-4



Example eGUI Menu Item

Explanations for these eGUI commands are contained in the “Reference Guide” section of this manual.

2

Reference Guide

Comment Entries

Comment lines can be added to the eGUI.txt file by placing a # symbol at the beginning of a line. For example:

```
# This is a comment line.
```

EGUI_END_ITEM;

Keyword `EGUI_END_ITEM;`

A menu item must end with the EGUI_END_ITEM; keyword and must begin with the EGUI_ITEM_MENU keyword. The semi-colon (;) is required for this keyword.

EGUI_FLAGS

Keyword `EGUI_FLAGS={<flags>}`

The EGUI_FLAGS keyword is optional. When the EGUI_FLAGS keyword is not used, then the default behavior for each of the flags is used. When the EGUI_FLAGS keyword is used then the flag parameters are separated by spaces.

`<flags>` SHOW_RUN_OPTIONS

When this option is used, a Run Options dialog box is displayed before your program is run. This dialog box lets you choose a secondary monitor to display the program you are running.

Default: The Run Options dialog box is not displayed.

ALLOW_MULTIPLE_RUNNING

Allows you to launch more than one copy of your program even if one copy is already running.

Default: Only one copy of your program can be running. If you try to start another copy running while one is currently running, the currently running copy is brought to the top.

LEAVE_RUNNING_AFTER _EXIT

If the oscilloscope application is exited, your program will remain running if it was already running.

Default: When the oscilloscope application is exited, the last launched running copy of your program is exited.

Example

```
EGUI_FLAGS={SHOW_RUN_OPTIONS LEAVE_RUNNING_AFTER_EXIT}
```

EGUI_ITEM_MENU

Keyword

```
EGUI_ITEM_MENU
```

A menu item must begin with the EGUI_ITEM_MENU keyword and must end with the EGUI_END_ITEM; keyword.

MENU_PATH

Keyword

MENU_PATH={<scope_menu><menu_item>}

The MENU_PATH keyword is required and is the place where you want your menu item to appear in the Infiniium menu system. It is a good idea to put your menu items in either the Analyze or the Utilities menus.

<scope_menu>

The Infiniium oscilloscope menu where you want to place your menu item. For example:

MAIN\Analyze\

MAIN\Utilities\

<menu_item>

The name that you want to appear in the menu. You can also include a submenu if you do not want the menu item to appear in one of the main Infiniium menus. If you use an ampersand (&) character in front of one of the letters in a menu item, that letter is used as a shortcut key to the menu item.

There are two types of menu items: terminal and nonterminal. Terminal menu items look like the following.

```
MENU_PATH={MAIN\Analyze\My Menu Item}
```

Nonterminal menu items look like the following.

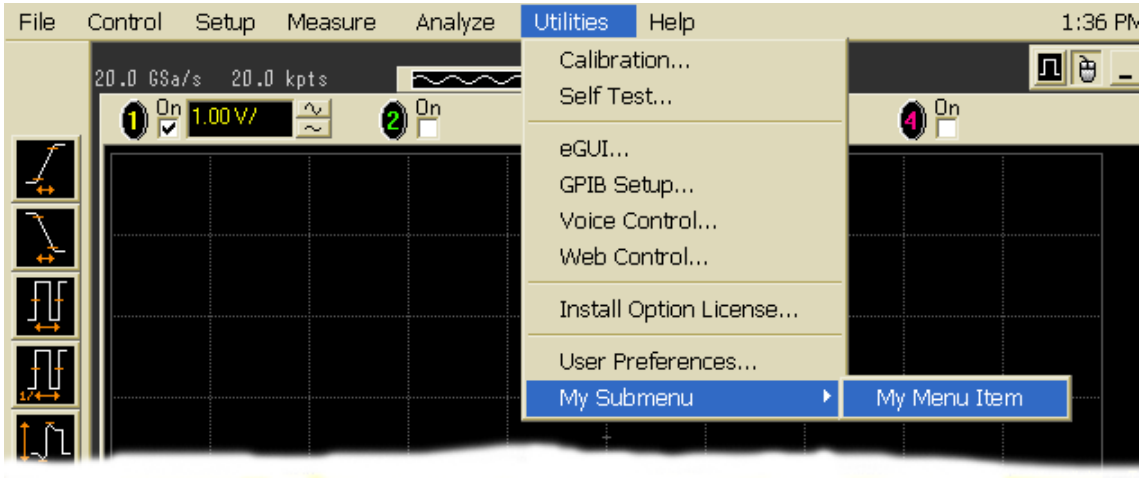
```
MENU_PATH={MAIN\Analyze\Sub Menu Item\}
```

Terminal menu items require you to use the MENU_RUN keyword while nonterminal menu items do not require the MENU_RUN keyword. The nonterminal menu items are used to add a sub menu item that does not execute a command but can be used to add separators to the sub menu item.

Example

```
MENU_PATH={MAIN\Utilities\My Submenu\My Menu Item}
```

The following picture shows the result of this keyword.



MENU_POS

Keyword

```
MENU_POS={<menu_position>}
```

The MENU_POS keyword positions your menu item at a point in the menu list specified by the <menu_position> parameter. The top position is position number 0 (zero) with positive integers having menu locations further down in the list. Minus integer values place the menu item at the bottom of the list. The MENU_POS keyword is not required and when not used the menu item is placed at the bottom of the menu list. It is recommended that you place your menu items at the bottom of the list.

<menu_position>	An integer from 0 (top) to maximum number (bottom) of list represents the position of your menu item in the list. Negative integers places the menu item at the bottom of the list.

Example `MENU_POS={2}`

MENU_RUN

Keyword `MENU_RUN={<run_file>}`

The MENU_RUN keyword specifies which program is launched when your menu item is selected. This is a required keyword when a terminal path has been specified by the MENU_PATH keyword.

<run_file>	The path and file name of your executable program.

Example `MENU_RUN={C:\scope\util\myprogram.exe}`

MENU_RUN_ARG

Keyword `MENU_RUN_ARG={<program_arg>}`

The MENU_RUN_ARG specifies the program arguments that are passed to your program when it is launched. This is not a required keyword and when not used nothing is passed to the program.

<program_arg>	An ASCII string of command line options to pass to your program.

Example `MENU_RUN_ARG={/myarg}`

MENU_SEP

Keyword

`MENU_SEP={<option>}`

The MENU_SEP keyword determines what menu separators are used around your menu item. This keyword is not required and when not used no separators are added around your menu item. If adding a menu separator would cause more than one separator to be added before or after a menu item then only one will be used.

If you perform a reload of the eGUI.txt file while the oscilloscope is still running, it is possible that the separator bars will not be exactly as you have specified. However, if you restart the oscilloscope application, the separator bars will be in the correct positions.

`<option>`

AFTER

A menu separator is added after your menu item.

BEFORE

A menu separator is added before your menu item.

BEFORE_AND_AFTER

A menu separator is added before and after your menu item.

NONE

No separators are used around your menu item.

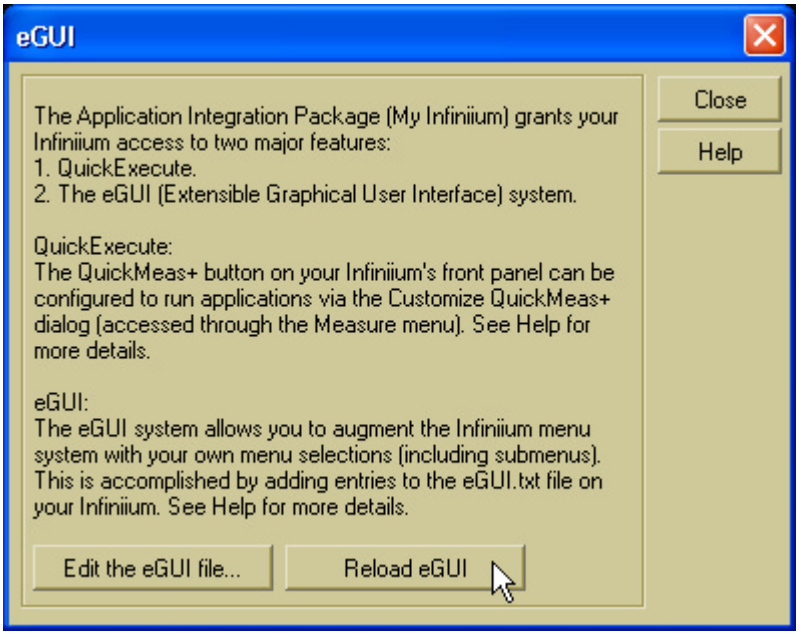
Example

```
MENU_SEP={BEFORE}
```

Programming Tips and Rules

1 You must reload the eGUI text file before your new menu item will appear. This is done by selecting **eGUI** in the **Utilities** menu and selecting the **Reload eGUI** button. See the following figure.

Figure 2-1



2 You must use upper case MAIN as the beginning of the menu path.

3 You may add multiple menu items by using multiple pairs of EGUI_ITEM_MENU and EGUI_END_ITEM; keywords

4 Use VISA instead of SICL driver for Visual C++ programs. SICL is simpler to use but VISA has the advantage that a program can be executed unchanged on either a PC or internal to the oscilloscope.

Safety Notices

This apparatus has been designed and tested in accordance with IEC Publication 1010, Safety Requirements for Measuring Apparatus, and has been supplied in a safe condition. This is a Safety Class I instrument (provided with terminal for protective earthing). Before applying power, verify that the correct safety precautions are taken (see the following warnings). In addition, note the external markings on the instrument that are described under "Safety Symbols."

Warnings

• Before turning on the instrument, you must connect the protective earth terminal of the instrument to the protective conductor of the (mains) power cord. The mains plug shall only be inserted in a socket outlet provided with a protective earth contact. You must not negate the protective action by using an extension cord (power cable) without a protective conductor (grounding). Grounding one conductor of a two-conductor outlet is not sufficient protection.

• Only fuses with the required rated current, voltage, and specified type (normal blow, time delay, etc.) should be used. Do not use repaired fuses or short-circuited fuseholders. To do so could cause a shock or fire hazard.

• If you energize this instrument by an auto transformer (for voltage reduction or mains isolation), the common terminal must be connected to the earth terminal of the power source.

• Whenever it is likely that the ground protection is impaired, you must make the instrument inoperative and secure it against any unintended operation.

• Service instructions are for trained service personnel. To avoid dangerous electric shock, do not perform any service unless qualified to do so. Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

• Do not install substitute parts or perform any unauthorized modification to the instrument.

• Capacitors inside the instrument may retain a charge even if the instrument is disconnected from its source of supply.

• Do not operate the instrument in the presence of flammable gasses or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

• Do not use the instrument in a manner not specified by the manufacturer.

To clean the instrument

If the instrument requires cleaning: (1) Remove power from the instrument. (2) Clean the external surfaces of the instrument with a soft cloth dampened with a mixture of mild detergent and water. (3) Make sure that the instrument is completely dry before reconnecting it to a power source.

Safety Symbols

Instruction manual symbol: the product is marked with this symbol when it is necessary for you to refer to the instruction manual in order to protect against damage to the product.

Hazardous voltage symbol.

Earth terminal symbol: Used to indicate a circuit common connected to grounded chassis.

Agilent Technologies Inc.
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901-2197, U.S.A.

Notices

Manual Part Number

E2699-97001, October 2003

Print History

E2699-97001, October 2003
E2699-97000, August 2003

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Document Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

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.

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.

Trademark Acknowledgements

Windows and MS Windows are U.S. registered trademarks of Microsoft Corporation.