Infiniium Application Integration Utility

KEYSIGHT
TECHNOLOGIES

User's Guide

Notices

Revision

Version 11.30.00203

Edition

July 2022

Available in electronic format only

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

Trademarks

Warranty

Technology License

U.S. Government Rights

Safety Notices

CAUTION

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

WARNING

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

Application Integration Utility—At a Glance

The Application Integration Utility lets you launch applications directly from the oscilloscope's front panel or from the Infiniium oscilloscope application's graphical user interface. It provides these two capabilities:

- QuickExecute — This is a Multipurpose option that lets you run an executable program by pressing the oscilloscope's front panel **[Multipurpose]** key.
- Extensible Graphical User Interface (eGUI) — This lets you add menu items to the Infiniium oscilloscope application that run executable programs.

In This Book

This book has these chapters:

- Chapter 1, "Application Integration Utility Features," starting on page 7 — describes the Application Integration Utility features in more detail and provides a few examples.
- Chapter 2, "Reference Guide," starting on page 23 — describes the keywords and options that can appear in the eGUI.txt file.

Contents

Application Integration Utility—At a Glance / 3

1 Application Integration Utility Features

QuickExecute and Extensible Graphical User Interface (eGUI) / 8
Example 1 - Scripting SCPI Commands / 9
Example 2 - Mapping A Control To The Front Panel / 10
Example 3 - Creating a Dialog Box with Custom Measurement Results / 12
Example 4 - The Snapshot Measurement Utility / 20

Adding Menu Items to Infiniium / 21

2 Reference Guide

EGUI_END_ITEM; / 24
EGUI_FLAGS / 25
EGUI_ITEM_MENU / 26
MENU_PATH / 27
MENU_POS / 29
MENU_RUN / 30
MENU_RUN_ARG / 31
MENU_SEP / 32

Index

1 Application Integration Utility Features

QuickExecute and Extensible Graphical User Interface (eGUI) / 8
Adding Menu Items to Infiniium / 21

QuickExecute and Extensible Graphical User Interface (eGUI)

The Application Integration Utility consists of two capabilities:

- QuickExecute
- Extensible Graphical User Interface (eGUI)

These capabilities let you launch applications directly from Infiniium's front panel or from Infiniium's graphical user interface.

QuickExecute is a user choice to the Multipurpose feature. When this option is selected, each time the **[Multipurpose]** key 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 the Extensible Graphical User Interface (eGUI). Any program that can be run under Windows is a candidate for these features. You can develop and debug your programs on an external PC, using the programming language and tools of your choice.

For example, your programs can control the oscilloscope using the Keysight I/O Libraries. 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; then, your program can retrieve the data from the hard disk or directly or by using the I/O commands. Then, your program can perform custom analysis and display the results in 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, Keysight 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 Keysight Developers Network, www.keysight.com/find/adn.

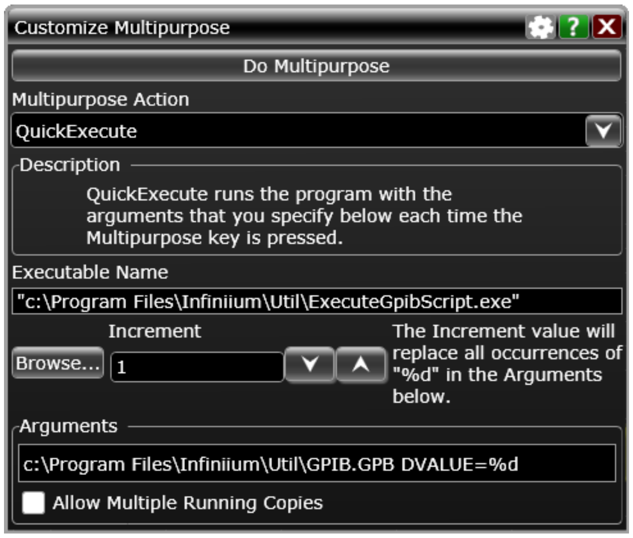
With the Application Integration Utility, your programs become an integral part of the oscilloscope. In fact, if you are currently running a custom analysis program on an external PC, it is very easy to move it into your oscilloscope using AIP. If you are not experienced with using applications programs with your Infiniium, Keysight provides some example programs to help you become familiar with the capabilities and benefits of AIP. These programs also illustrate the wide variety of actions that are possible using AIP.

Example 1 - Scripting SCPI Commands

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

```
:STOP
:DISK:SAVE:WAVeform CHAN1,"C:\Users\Public\Documents\Infiniium\data\
Acq%d_Ch1.csv",CSV,ON;
:DISK:SAVE:WAVeform CHAN2,"C:\Users\Public\Documents\Infiniium\data\
Acq%d_Ch2.csv",CSV,ON;
:DISK:SAVE:WAVeform CHAN3,"C:\Users\Public\Documents\Infiniium\data\
Acq%d_Ch3.csv",CSV,ON;
:DISK:SAVE:WAVeform Chan4,"C:\Users\Public\Documents\Infiniium\data\
Acq%d_Ch4.csv",CSV,ON;
:DISK:SAVE:SETup "C:\Users\Public\Documents\Infiniium\data\Acq%d.set";
:DISK:SAVE:IMAGe "C:\Users\Public\Documents\Infiniium\data\
Acq%d.png",PNG,SCReen,ON,NORMal;
```

When mapped to QuickExecute or an Extensible Graphical User Interface menu item, these commands cause Infiniium to store all four channels of data as well as the oscilloscope 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 of %d can be set using the **Increment** field in the Customize Multipurpose dialog box as shown in the following figure.



Multipurpose Action: QuickExecute

Description: QuickExecute runs the program with the arguments that you specify below each time the Multipurpose key is pressed.

Executable Name: "c:\Program Files\Infiniium\Util\ExecuteGpibScript.exe"

Increment: 1

The Increment value will replace all occurrences of "%d" in the Arguments below.

Arguments: c:\Program Files\Infiniium\Util\GPIB.GPB DVALUE=%d

Allow Multiple Running Copies

Figure 1 Increment Control

Considering that nearly all of Infiniium's capabilities are controllable using SCPI commands, it should be clear that many valuable control sequences are possible using ExecuteGpibScript in this manner.

Example 2 - Mapping A Control To The Front Panel

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

```
/* StepAveraging.c -- 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]:inst0");   /* 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 successive 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.

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 Keysight 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 Keysight 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 Keysight I/O Libraries are installe
d ");
  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.
```

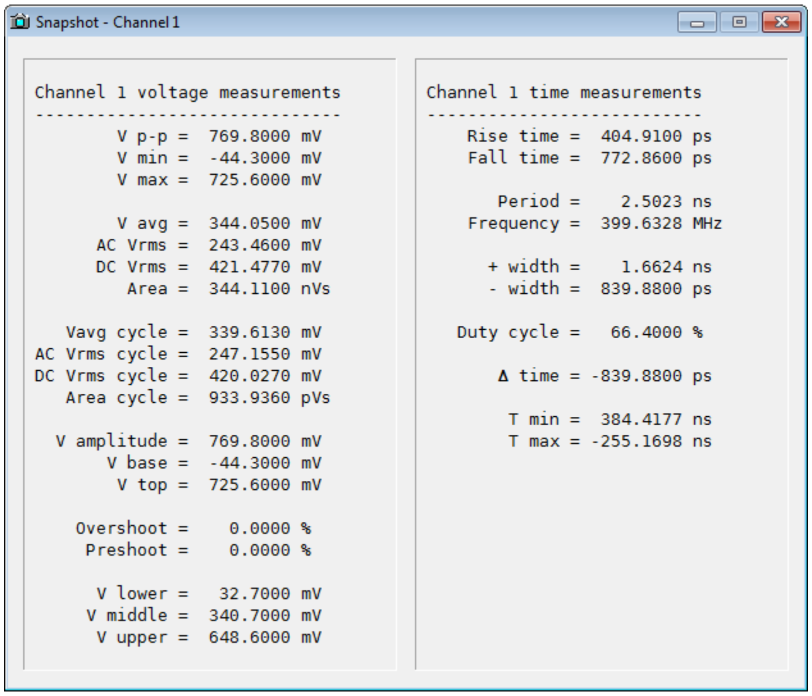
```
//

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 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_NUL
L);
  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_N
ULL);
  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 guide 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. The source code for Snapshot is not provided but the executable is located on Infiniium's hard disk, in the C:\Program Files\Infiniium\util directory. When mapped to QuickExecute, Snapshot provides a "measure all" capability on a source-by-source basis. Each invocation advances the measurement source, just like QuickMeas does. A screen shot of Snapshot's results dialog is shown in the following figure.



```
Channel 1 voltage measurements
------------------------------
        V p-p =  769.8000 mV
        V min =  -44.3000 mV
        V max =  725.6000 mV

        V avg =  344.0500 mV
      AC Vrms =  243.4600 mV
      DC Vrms =  421.4770 mV
         Area =  344.1100 nVs

   Vavg cycle =  339.6130 mV
AC Vrms cycle =  247.1550 mV
DC Vrms cycle =  420.0270 mV
   Area cycle =  933.9360 pVs

  V amplitude =  769.8000 mV
       V base =  -44.3000 mV
        V top =  725.6000 mV

    Overshoot =    0.0000 %
     Preshoot =    0.0000 %

      V lower =   32.7000 mV
     V middle =  340.7000 mV
      V upper =  648.6000 mV
```

```
Channel 1 time measurements
---------------------------
    Rise time =  404.9100 ps
    Fall time =  772.8600 ps

       Period =    2.5023 ns
    Frequency =  399.6328 MHz

      + width =    1.6624 ns
      - width =  839.8800 ps

   Duty cycle =   66.4000 %

       Δ time = -839.8800 ps

        T min =  384.4177 ns
        T max = -255.1698 ns
```

Figure 2 Snapshot dialog box

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

Adding Menu Items to Infiniium

Menu items can be added to any of the existing Infiniium menus. 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:\Users\Public\Documents\Infiniium\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.

```
EGUI_ITEM_MENU
   MENU_PATH={MAIN\Utilities\&My Menu Item}
   MENU_RUN={C:\Program Files\Infiniium\util\myprogram.exe}
   MENU_RUN_ARGS={/list}
   EGUI_FLAGS={SHOW_RUN_OPTIONS}
   MENU_SEP={BEFORE}
   MENU_POS={-1}
EGUI_END_ITEM;
```

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

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

The following figure shows the resulting menu item that was create by preceding text in the eGUI.txt file.

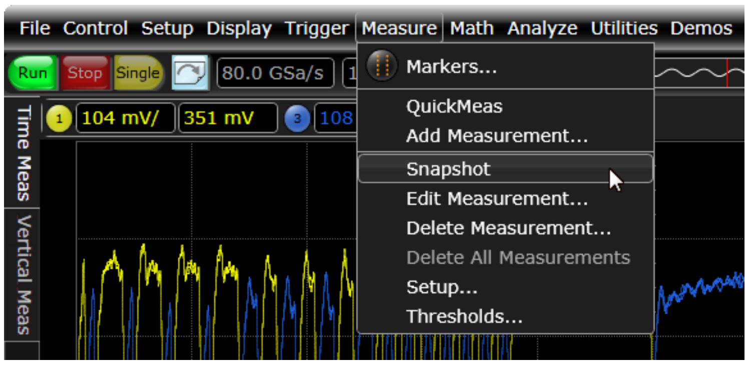


Figure 3 Example eGUI Menu Item

2 Reference Guide

EGUI_END_ITEM; / 24
EGUI_FLAGS / 25
EGUI_ITEM_MENU / 26
MENU_PATH / 27
MENU_POS / 29
MENU_RUN / 30
MENU_RUN_ARG / 31
MENU_SEP / 32

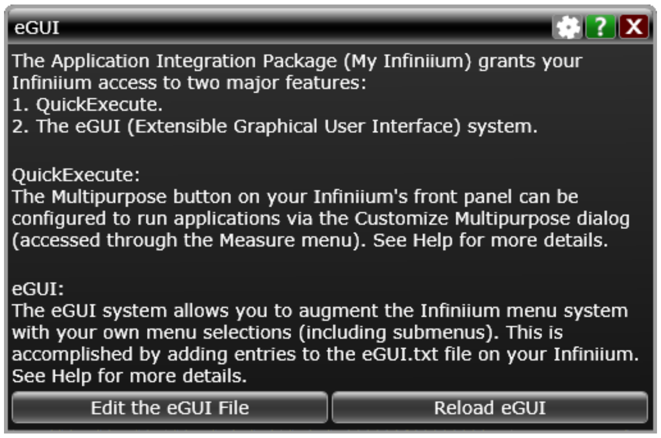
Comment Entries

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.
```

Programming Tips and Rules

1 You must reload the eGUI text file before your new menu item will appear. This is done by choosing **Utilities > eGUI...** and by clicking the **Reload eGUI** button.



The Application Integration Package (My Infiniium) grants your Infiniium access to two major features:
1. QuickExecute.
2. The eGUI (Extensible Graphical User Interface) system.

QuickExecute:
The Multipurpose button on your Infiniium's front panel can be configured to run applications via the Customize Multipurpose dialog (accessed through the Measure menu). See Help for more details.

eGUI:
The eGUI system allows you to augment the Infiniium menu system with your own menu selections (including submenus). This is accomplished by adding entries to the eGUI.txt file on your Infiniium. See Help for more details.

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.

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, the default behavior for each of the flags is used. When the `EGUI_FLAGS` keyword is used, 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}
```

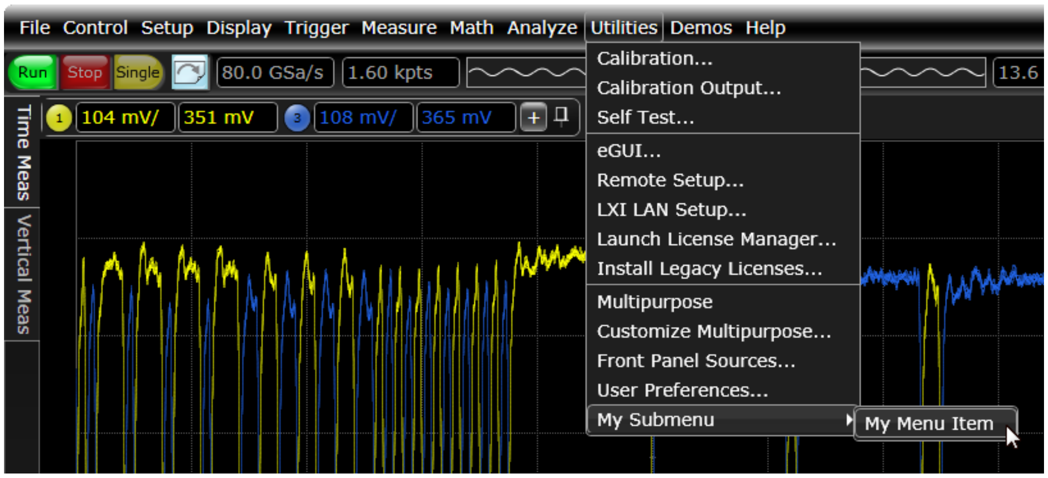
Nonterminal menu items look like the following.

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

Terminal menu items require that you 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.


File Control Setup Display Trigger Measure Math Analyze Utilities Demos Help
Run
Stop
Single
80.0 GSa/s
1.60 kpts
13.6
104 mV/
351 mV
108 mV/
365 mV
Time Meas
Vertical Meas
Calibration...
Calibration Output...
Self Test...
eGUI...
Remote Setup...
LXI LAN Setup...
Launch License Manager...
Install Legacy Licenses...
Multipurpose
Customize Multipurpose...
Front Panel Sources...
User Preferences...
My Submenu
My Menu Item

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> paramenter. 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 place 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:\Program Files\Infiniium\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; when is is 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.

NOTE **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}`

Index

A

Application Integration Utility, 3
Application Integration Utility features, 7
Application Integration Utility reference, 23

C

comment entries, 23
copyright, 2

E

eGUI, 3, 8
EGUI_END_ITEM;, 24
EGUI_FLAGS, 25
EGUI_ITEM_MENU, 26
Extensible Graphical User Interface, 8

I

in this book, 3

M

menu items, adding, 21
MENU_PATH, 27
MENU_POS, 29
MENU_RUN, 30
MENU_RUN_ARG, 31
MENU_SEP, 32

N

notices, 2

P

programming tips/rules, 23

Q

QuickExecute, 3, 8

T

trademarks, 2

W

warranty, 2