

UDT S370 Optometer LabWindows/CVI Instrument Driver

UDT Instruments

April, 2005

1.	Introduction.....	5
2.	Known Problems	5
3.	Error Codes	5
4.	Function Tree Layout.....	6
5.	Functions	6
5.1.	Initialize [B0].....	6
	Parameter List	6
	Address	6
	Reset.....	7
	Model	7
	Instrument ID	7
	Errors	7
5.2.	Set Calibration [Kx, K]	7
	Parameter List	7
	Instrument ID	7
	Calibration [Kx].....	7
	Errors	7
5.3.	Display Calibration [K].....	7
	Parameter List	8
	Instrument ID	8
	Errors	8
5.4.	Set Wavelength [Wx]	8
	Parameter List	8
	Instrument ID	8
	Wavelength [Wx].....	8
	Errors	8
5.5.	Increment Wavelength [WU, WD].....	8
	Parameter List	8
	Instrument ID	8
	Increment	8
	Errors	8
5.6.	Set Responsivity [R]	8
	Parameter List	9
	Instrument ID	9
	Responsivity [R]	9
	Errors	9
5.7.	Read Responsivity [R].....	9
	Parameter List	9
	Instrument ID	9
	Responsivity [R]	9
	Errors	9
5.8.	Set Mode [DL, DN, DR, DG].....	9
	Parameter List	9
	Instrument ID	9
	Mode	9
	Errors	10
5.9.	Set Range [A, M, Mx].....	10
	Parameter List	10
	Instrument ID	10
	Range	10
	Errors	10
5.10.	Read Status [S]	10
	Parameter List	10
	Instrument ID	10

	Mode	10
	Wavelength	11
	Auto/Man	11
	Gain	11
	Errors	11
5.11.	Display Status [X]	11
	Parameter List	11
	Instrument ID	11
	Errors	11
5.12.	Set Units [Vx]	11
	Parameter List	11
	Instrument ID	11
	Measurement Units	11
	Errors	12
5.13.	Read Units [Ux]	12
	Parameter List	12
	Instrument ID	12
	Measurement Units [Ux]	12
	Errors	12
5.14.	Set Screen [Bx]	12
	Parameter List	12
	Instrument ID	12
	Screen Mode	12
	Errors	13
5.15.	Set Analog [Ex]	13
	Parameter List	13
	Instrument ID	13
	Reading Mode	13
	Errors	13
5.16.	Set Message [*MESSAGE*]	13
	Parameter List	13
	Instrument ID	13
	Message [*MESSAGE*]	13
	Errors	13
5.17.	Set Zero [Zx]	13
	Parameter List	14
	Instrument ID	14
	Zero Mode	14
	Errors	14
5.18.	Start/Stop [G, H]	14
	Parameter List	14
	Instrument ID	14
	Start/Stop	14
	Errors	14
5.19.	Read Data [Fx]	14
	Parameter List	14
	Instrument ID	14
	Value [Fx]	14
	Status	15
	Errors	15
5.20.	Close	15
	Parameter List	15
	Instrument ID	15
	Errors	15
6.	Test Program	16

6.1.	Program Operation	16
6.2.	Program Internals.....	18
	S370 Demo.h	18
	S370 Demo.c	20
	About Panel.c	20
	Configuration Panel.c	20
	Continuous Measurements.c.....	22
	Digital Panel.c	24
	Initialization Panel.c	24
	Menu Bar.c	24
	S370 Global Variables.h.....	27
	Strip Panel.c.....	28
	Verbose Status Parameters.c.....	29
7.	Index of UDT S370 Device Dependent Commands	31

UDT S370 Optometer

1. Introduction

This instrument module provides LabWindows/CVI programming support for the UDT S370 Optometer. This driver contains functions for opening, configuring, taking measurements from, and closing the instrument.

To successfully use this module, the instrument should be connected to the National Instruments GPIB card installed in the PC, with the appropriate GPIB device address on the instrument selected. This device address appears as an argument to the initialize function and must match the address of the instrument.

UDT 3-letter commands are listed in square brackets next to the corresponding LabWindows/CVI functions listed below, and also appear in the on-line help for each function panel within the LabWindows/CVI environment. In addition, the UDT 3-letter commands are indexed by page number at the end of this document. Each LabWindows/CVI function corresponds to a specific instrument function.

This document is not intended to be a tutorial on setting up or running LabWindows/CVI. It is a summary of functions, their arguments, and intended use, specific to the UDT S370. For additional information about using LabWindows/CVI, consult the LabWindows/CVI documentation, or contact National Instruments at 512-683-0100 or UDT Instruments at (414) 342 - 2626.

2. Known Problems

There are no known problems in the instrument driver. All device-dependent commands associated with the UDT S370 are implemented in this driver.

3. Error Codes

All error codes are returned in a global integer variable whose name depends on the language being used:

udts370_err

The error code is also returned as a result of the function call, in order to maintain future compatibility with the Microsoft Windows environment.

Error	Description
0	Success
220	Unable to open instrument
221	Unable to close instrument
224	Error clearing instrument
230	Error writing to instrument
231	Error reading from instrument
232	Instrument not initialized
233	Error configuring GPIB address
236	Error interpreting instrument response

Application programs should test this variable after each function call is made to determine if any errors occurred.

4. Function Tree Layout

- UDT S370
 - Initialize
 - Configure
 - Calibration
 - Set Calibration
 - Display Calibration
 - Set Wavelength
 - Increment Wavelength
 - Set Responsivity
 - Read Responsivity
 - Set Mode
 - Set Range
 - Status
 - Read Status
 - Display Status
 - Units
 - Set Units
 - Read Units
 - Set Screen
 - Set Analog
 - Set Message
 - Set Zero
 - Measure
 - Start/Stop
 - Read Data
 - Close

5. Functions

5.1. Initialize [B0]

Description: Initializes the instrument in the following ways:

- 1). opens the instrument and sets the GPIB address,
- 2). device clear (if selected),
- 3). reads the model number of the instrument (370).
- 4). returns the instrument ID.

The instrument ID is used in all of the other function calls to distinguish between multiple S370 instruments connected to the bus. For example, if several S370's were used with device addresses of 5, 8, and 12, the corresponding instrument ID's would be 1, 2, and 3. For one S370 (the usual situation), the default ID = 1 would be used.

```
int Address;  
int Reset;  
int *Model;  
int *Instrument_ID;  
int udts370_init(Address, Reset, Model, Instrument_ID);
```

Parameter List

Address

Description: This is the device address which must match the internal setting of the instrument.

Variable Type: Integer

Valid Range: 0 to 30

Reset

Description: If reset is on, a device clear is performed during the initialization.

Variable Type: Integer

Variable Range: 0 to 1

0 Off
1 On (Default)

Model

Description: The instrument model number is returned, primarily as a check that communications between the host and the instrument are working properly.

Variable Type: Integer

Valid Range: 370

Instrument ID

Description: Returns an Instrument ID that is used in all subsequent function calls to select the device at a given address. If more than one S370 is used, this parameter is used to differentiate between them. The first device initialized will automatically be assigned an Instrument ID of 1.

Variable Type: Integer

Valid Range: 1 to 10

Errors

0 Success
-1 Invalid Address
-2 Invalid Reset

5.2. Set Calibration [Kx, K]

Description: Sets up the instrument calibration.

int Instrument_ID;

int Calibration;

int udts370_set_calibration(Instrument_ID, Calibration);

Parameter List

Instrument ID

Description: The Instrument ID returned by the Initialize function is used here to select the desired S370 instrument.

Variable Type: Integer

Valid Range: 1 to maximum number of this instrument

The maximum number in a single system is set internally in the driver. The typical value is 10.

Calibration [Kx]

Description: The number corresponding to a pre-programmed instrument calibration is input here.

Variable Type: Integer

Valid Range:

Errors

0 Success
-1 Invalid Instrument ID

5.3. Display Calibration [K]

Description: Displays the instrument calibration.

int Instrument_ID;

int *udts370_display_calibration(Instrument_ID);*

Parameter List

Instrument ID

Errors

0 Success
-1 Invalid Instrument ID

5.4. Set Wavelength [Wx]

Description: Sets the wavelength. The wavelength must be consistent with the calibration curve previously selected.

int *Instrument_ID;*

int *Wavelength;*

int *udts370_set_wavelength(Instrument_ID, Wavelength);*

Parameter List

Instrument ID

Wavelength [Wx]

Description: Sets the wavelength. The wavelength must be consistent with the calibration curve previously selected.

Variable Type: Integer

Valid Range: 200 to 2000 nm depending on calibration.

Errors

0 Success
-1 Invalid Instrument ID
-2 Invalid Wavelength

5.5. Increment Wavelength [WU, WD]

Description: Increments or decrements the current wavelength.

int *Instrument_ID;*

int *Increment;*

int *udts370_increment_wavelength(Instrument_ID, Increment);*

Parameter List

Instrument ID

Increment

Description: Increments or decrements the current wavelength.

Variable Type: Integer

Valid Range: 0 to 1

0	Down (Default)	[WD]
1	Up	[WU]

Errors

0 Success
-1 Invalid Instrument ID
-2 Invalid Increment

5.6. Set Responsivity [R]

Description: Sets the responsivity.

```
int Instrument_ID;  
double Responsivity;  
int udts370_set_responsivity(Instrument_ID, Responsivity);
```

Parameter List

Instrument ID

Responsivity [R]

Description: Sets the responsivity.

Variable Type: Real

Valid Range:

Errors

0 Success
-1 Invalid Instrument ID

5.7. Read Responsivity [R]

Description: Reads the responsivity.

```
int Instrument_ID;  
double *Responsivity;  
int udts370_read_responsivity(Instrument_ID, Responsivity);
```

Parameter List

Instrument ID

Responsivity [R]

Description: Reads the responsivity.

Variable Type: Real

Valid Range:

Errors

0 Success
-1 Invalid Instrument ID

5.8. Set Mode [DL, DN, DR, DG]

Description: Sets the measurement mode to log, linear, ratio, and log ratio.

```
int Instrument_ID;  
int Mode;  
int udts370_set_mode(Instrument_ID, Mode);
```

Parameter List

Instrument ID

Mode

Description: Sets the mode.

Variable Type: Integer

Valid Range: 0 to 4

1	Log	[DL]
2	Lin (Default)	[DN]
3	Ratio	[DR]
4	LogRatio	[DG]

Errors

0 Success
-1 Invalid Instrument ID
-2 Invalid Mode

5.9. Set Range [A, M, Mx]

Description: Sets the measurement range to auto, manual (without changing range), and specific gain settings.

```
int Instrument_ID;  
int Range;  
int udts370_set_range(Instrument_ID, Range);
```

Parameter List

Instrument ID

Range

Description: Sets the range.

Variable Type: Integer

Valid Range: 1 to 9

1	Auto (Default)	[A]
2	Manual	[M]
3	3	[M3]
4	4	[M4]
5	5	[M5]
6	6	[M6]
7	7	[M7]
8	8	[M8]
9	9	[M9]

Errors

0 Success
-1 Invalid Instrument ID
-2 Invalid Range

5.10. Read Status [S]

Description: Reads the status.

```
int Instrument_ID;  
int *Mode;  
int *Wavelength;  
int *AutoMan;  
int *Gain;  
int udts370_read_status(Instrument_ID, Mode, Wavelength, AutoMan, Gain);
```

Parameter List

Instrument ID

Mode

Description: Reads the current mode.

Variable Type: Integer

Valid Range: 1 to 4

1	Log
---	-----

- 2 Linear
- 3 Ratio
- 4 Log Ratio

Wavelength

Description: Reads the current wavelength.

Variable Type: Integer

Valid Range: 200 to 2000

Auto/Man

Description: Reads the auto/manual state.

Variable Type: Integer

Valid Range: 0 to 1

- 0 Auto
- 1 Manual

Gain

Description: Reads the gain setting.

Variable Type: Integer

Valid Range: 3 to 9

Errors

- 0 Success
- 1 Invalid Instrument ID

5.11. Display Status [X]

Description: Displays the status.

int Instrument_ID;

int udts370_display_status(Instrument_ID);

Parameter List

Instrument ID

Errors

- 0 Success
- 1 Invalid Instrument ID

5.12. Set Units [Vx]

Description: Sets the measurement units.

int Instrument_ID;

int Measurement_Units;

int udts370_set_units(Instrument_ID, Measurement_Units);

Parameter List

Instrument ID

Measurement Units

Description: Sets the measurement units.

Variable Type: Integer

Valid Range: 1 to 8

- 1 W (Default) [V1]
- 2 Fc [V2]

3	Lux	[V3]
4	Fl	[V4]
5	W/cm2	[V5]
6	W/cm2-sr	[V6]
7	Cd/m2	[V7]
8	Lm	[V8]

Errors

0	Success
-1	Invalid Instrument ID
-2	Invalid Measurement Units

5.13. Read Units [Ux]

Description: Reads the measurement units.

```
int Instrument_ID;
int *Measurement_Units;
int udts370_read_units(Instrument_ID, Measurement_Units);
```

Parameter List

Instrument ID

Measurement Units [Ux]

Description: Reads the measurement units.

Variable Type: Integer

Valid Range: 1 to 11

1	W
2	Fc
3	Lux
4	Fl
5	W/cm2
6	W/cm2-sr
7	Cd/m2
8	Lm
9	A
10	Cd
11	W/sr

Errors

0	Success
-1	Invalid Instrument ID

5.14. Set Screen [Bx]

Description: Sets the instrument screen mode.

```
int Instrument_ID;
int Screen_Mode;
int udts370_set_screen(Instrument_ID, Screen_Mode);
```

Parameter List

Instrument ID

Screen Mode

Description: Sets the screen mode.

Variable Type: Integer

Valid Range: 0 to 3

0	Restore (Default)	[B0]
1	Blank	[B1]
2	Light On	[B2]
3	Light Off	[B3]

Errors

0	Success
-1	Invalid Instrument ID
-2	Invalid Screen Mode

5.15. Set Analog [Ex]

Description: Sets the instrument screen to digital or analog bar reading.

```
int Instrument_ID;  
int Reading_Mode;  
int udts370_set_screen(Instrument_ID, Reading_Mode);
```

Parameter List

Instrument ID

Reading Mode

Description: Sets the instrument screen to digital or analog bar reading.

Variable Type: Integer

Valid Range: 0 to 1

0	Digital	[E0]
1	Analog	[E1]

Errors

0	Success
-1	Invalid Instrument ID
-2	Invalid Reading Mode

5.16. Set Message [*MESSAGE*]

Description: Displays a message on the instrument.

```
int Instrument_ID;  
char *Message;  
int udts370_set_message(Instrument_ID, Message);
```

Parameter List

Instrument ID

Message [*MESSAGE*]

Description: Displays a message on the instrument.

Variable Type: String

Valid Range: ASCII string of up to 32 characters.

Errors

0	Success
-1	Invalid Instrument ID

5.17. Set Zero [Zx]

Description: Sets the zero mode on the instrument.

```
int Instrument_ID;  
int Zero_Mode;  
int udts370_set_zero(Instrument_ID, Zero_Mode);
```

Parameter List

Instrument ID

Zero Mode

Description: Sets the zero mode.

Variable Type: Integer

Valid Range: 0 to 1

0	Off (Default)	[Z0]
1	On	[Z1]

Errors

0	Success
-1	Invalid Instrument ID
-2	Invalid Zero Mode

5.18. Start/Stop [G, H]

Description: Starts or stops the instrument from making measurements.

```
int Instrument_ID;  
int StartStop;  
int udts370_start_stop(Instrument_ID, StartStop);
```

Parameter List

Instrument ID

Start/Stop

Description: Starts or stops the instrument from making measurements.

Variable Type: Integer

Valid Range: 0 to 1

0	Off (Default)	[H]
1	On	[G]

Errors

0	Success
-1	Invalid Instrument ID
-2	Invalid Start/Stop

5.19. Read Data [Fx]

Description: Reads the measurement.

```
int Instrument_ID;  
double *Value;  
int *Status;  
int udts370_read_data(Instrument_ID, Value, Status);
```

Parameter List

Instrument ID

Value [Fx]

Description: Reads the signal for the selected channel.
Variable Type: Real
Valid Range: Depends on the detector head.

Status

Description: Returns the status of the measurement.
Variable Type: Integer
Valid Range: 1 to 4

1	Overrange	[O]
2	Previously read	[P]
3	Normal	[N]
4	Undefined	[U]

Errors

0	Success
-1	Invalid Instrument ID

5.20. Close

Description: Takes the instrument off-line. The instrument must be reinitialized to use it again.

int Instrument_ID;
int udts370_close(Instrument_ID);

Parameter List

Instrument ID

Errors

0	Success
-1	Invalid Instrument ID

6. Test Program

This program runs a UDT S370 Optometer. The program (called S370 Demo) does the following:

- initializes the optometer,
- configures the instrument,
- measures the signal, and shows the results on the PC screen.

6.1. Program Operation

Figure 1. This is the initialization screen.

Instructions:

- Select the instrument's GPIB address.
- From the 'Configure Instrument' menu option select 'Initialize Instrument'
- Once the instrument is initialized, the LED will turn green. If the initialization fails, the LED will turn red.
- Once the instrument is successfully initialized, the other menu options will be enabled.
- Select the 'Measurement Parameters' menu option to configure the instrument or press the 'Measure' menu option.

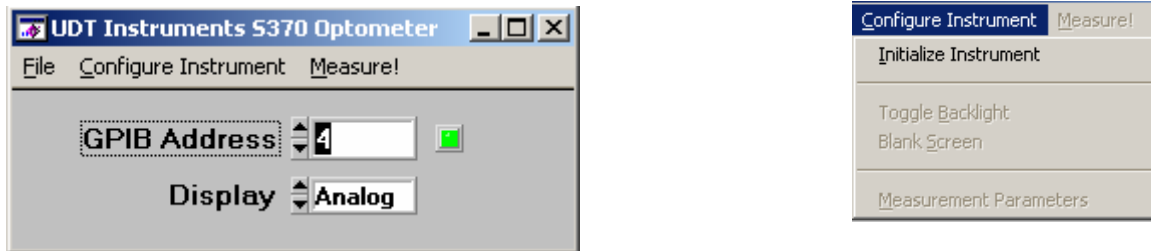


Figure 2. The configuration panel will configure the instrument.

Instructions:

- Set the desired parameters for the instrument
- Press the 'Measure' menu option. Depending on the 'Display' that was selected on the initialization panel either an analog or a digital data representation will be displayed.

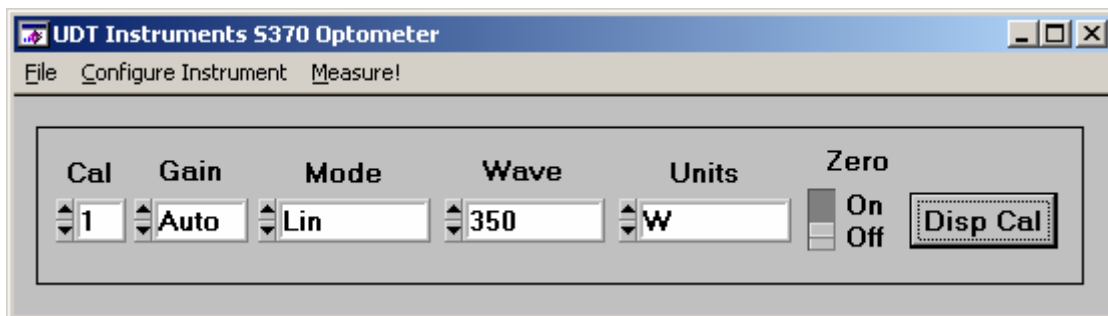


Figure 3. The digital data panel

Instructions:

- Press the 'Start' menu option to begin the measurement sequence.
- Press the 'Stop' menu option to stop the measurement sequence.
- While the measurement sequence is in progress the panel can not be closed and the measurement parameters can not be changed.

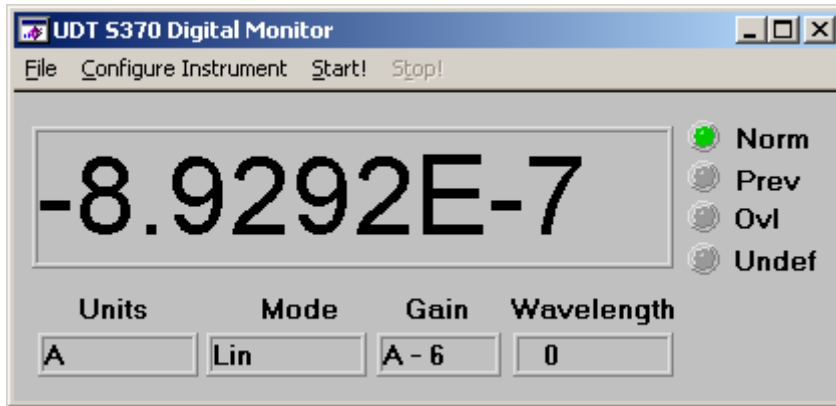
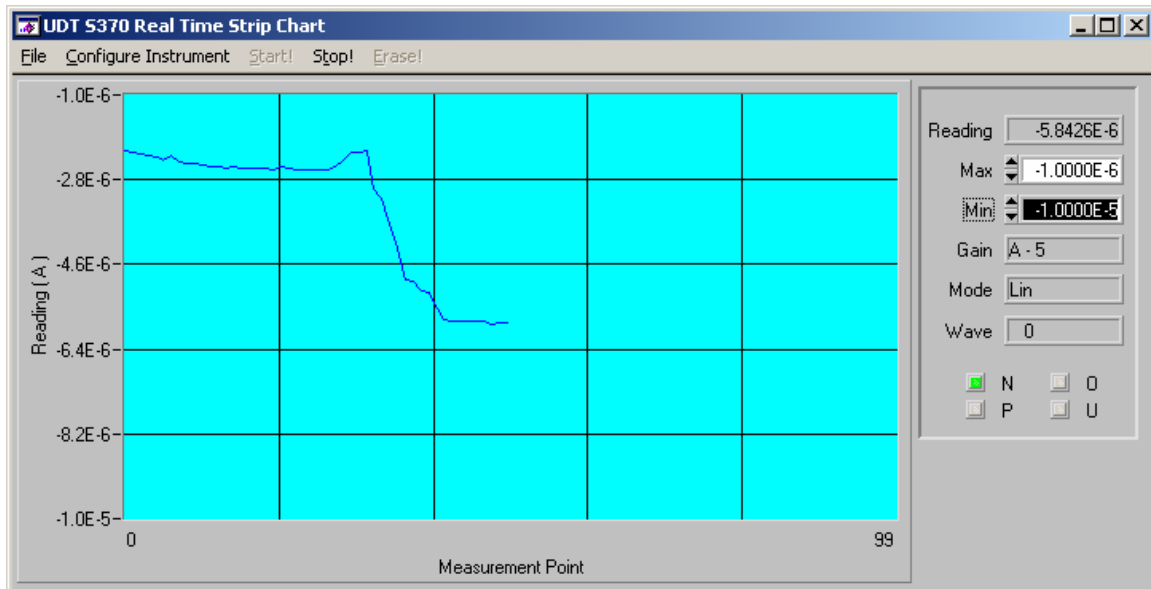


Figure 4. The analog data panel

Instructions:

- Press the 'Start' menu option to begin the measurement sequence.
- Press the 'Stop' menu option to stop the measurement sequence.
- Press the 'Erase' menu option to erase the strip chart traces.
- While the measurement sequence is in progress the panel can not be closed and the measurement parameters can not be changed.



6.2. Program Internals

The test program makes use of the LabWindows/CVI instrument driver `udts370`. Although not necessary to run the program, the instrument driver is necessary for making any modifications. There are three files associated with the instrument driver. These files are `udts370.c`, `udts370.h`, and `udts370.fp`.

The program `S370 Demo.exe` makes use of the LabWindows/CVI user interface library. The program is event-driven, in that it waits for an event such as a mouse click. After decoding the event to determine which control caused the event, the program branches to the appropriate function, and then waits for another event to occur. For example, clicking Initialize! on the menu-bar will cause the program to branch to the initialize subroutine.

In order to understand the program completely, it is best to use the LabWindows/CVI User Interface Editor on the file `S370 Demo.uir`, and examine how each control is defined within the editor environment. By using the editor, it will be apparent that the 'File' menu bar has associated with it, a tag defined as `BAR_FILE`. Other controls are handled similarly. Listing the file `S370 Demo.h` shows every control and the corresponding tags. (This file is generated automatically from within the LabWindows/CVI User Interface Editor).

S370 Demo.h

```
/******  
/* LabWindows/CVI User Interface Resource (UIR) Include File */  
/* Copyright (c) National Instruments 2005. All Rights Reserved. */  
/* */  
/* WARNING: Do not add to, delete from, or otherwise modify the contents */  
/* of this include file. */  
/******  
  
#include <userint.h>  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
/* Panels and Controls: */  
  
#define ABOUT 1 /* callback function: AboutPanel */  
#define ABOUT_TEXT 2  
  
#define CONFIG 2 /* callback function: ConfigurationPanel */  
#define CONFIG_DISP_CAL1 2 /* callback function: DisplayCalibration */  
#define CONFIG_UNITS1 3 /* callback function: SetUnits */  
#define CONFIG_MODE1 4 /* callback function: SetMode */  
#define CONFIG_GAIN1 5 /* callback function: SetGain */  
#define CONFIG_ZERO1 6 /* callback function: SetZero */  
#define CONFIG_CAL1 7 /* callback function: SetCalNumber */  
#define CONFIG_WAVE1 8 /* callback function: SetWavelength */  
#define CONFIG_DECORATION 9  
  
#define DIGITAL 3 /* callback function: DigitalPanel */  
#define DIGITAL_WAVE1 2  
#define DIGITAL_GAIN1 3  
#define DIGITAL_MODE1 4  
#define DIGITAL_READING1 5  
#define DIGITAL_UNIT1 6  
#define DIGITAL_OVL1 7  
#define DIGITAL_PREV1 8  
#define DIGITAL_NORM1 9  
#define DIGITAL_UNDEF1 10  
  
#define INITIAL 4 /* callback function: InitializationPanel */  
#define INITIAL_ADDR 2  
#define INITIAL_LED 3
```

```

#define INITIAL_SET_DISPLAY      4

#define STRIP                    5 /* callback function: StripPanel */
#define STRIP_MIN_CH1            2 /* callback function: SetMinValue */
#define STRIP_MAX_CH1           3 /* callback function: SetMaxValue */
#define STRIP_READING1          4
#define STRIP_WAVE1             5
#define STRIP_MODE1             6
#define STRIP_GAIN1             7
#define STRIP_LED_U1            8
#define STRIP_LED_O1           9
#define STRIP_LED_P1           10
#define STRIP_LED_N1           11
#define STRIP_DECORATION        12
#define STRIP_STRIP1           13

/* Menu Bars, Menus, and Menu Items: */

#define BAR                      1
#define BAR_FILE                 2
#define BAR_FILE_ABOUT           3 /* callback function: About */
#define BAR_FILE_SEPARATOR      4
#define BAR_FILE_EXIT           5 /* callback function: Exit */
#define BAR_CONFIG              6
#define BAR_CONFIG_INITIALIZE   7 /* callback function: InitializeInstrument */
#define BAR_CONFIG_SEPARATOR_2  8
#define BAR_CONFIG_BACKLIGHT    9 /* callback function: SetBacklightState */
#define BAR_CONFIG_BLANK       10 /* callback function: BlankScreen */
#define BAR_CONFIG_SEPARATOR_3  11
#define BAR_CONFIG_MEAS_PARMS  12 /* callback function: ConfigurationParameters */
#define BAR_MEASURE             13 /* callback function: Measure */

#define BAR2                     2
#define BAR2_FILE                2
#define BAR2_FILE_ABOUT          3 /* callback function: About */
#define BAR2_FILE_SEPARATOR      4
#define BAR2_FILE_EXIT          5 /* callback function: Exit */
#define BAR2_CONFIG              6
#define BAR2_CONFIG_BACKLIGHT    7 /* callback function: SetBacklightState */
#define BAR2_CONFIG_BLANK       8 /* callback function: BlankScreen */
#define BAR2_CONFIG_SEPARATOR_3  9
#define BAR2_CONFIG_MEAS_PARMS  10 /* callback function: ConfigurationParameters */
#define BAR2_START               11 /* callback function: Start */
#define BAR2_STOP                12 /* callback function: Stop */

#define REALTIME                 3
#define REALTIME_FILE            2
#define REALTIME_FILE_ABOUT      3 /* callback function: About */
#define REALTIME_FILE_SEPARATOR  4
#define REALTIME_FILE_EXIT       5 /* callback function: Exit */
#define REALTIME_CONFIG          6
#define REALTIME_CONFIG_BACKLIGHT 7 /* callback function: SetBacklightState */
#define REALTIME_CONFIG_BLANK    8 /* callback function: BlankScreen */
#define REALTIME_CONFIG_SEPARATOR_3 9
#define REALTIME_CONFIG_MEAS_PARMS 10 /* callback function: ConfigurationParameters */
#define REALTIME_START           11 /* callback function: Start */
#define REALTIME_STOP            12 /* callback function: Stop */
#define REALTIME_ERASE           13 /* callback function: Erase */

/* Callback Prototypes: */

void CVICALLBACK About(int menubar, int menuItem, void *callbackData, int panel);
int CVICALLBACK AboutPanel(int panel, int event, void *callbackData, int eventData1, int eventData2);
void CVICALLBACK BlankScreen(int menubar, int menuItem, void *callbackData, int panel);
int CVICALLBACK ConfigurationPanel(int panel, int event, void *callbackData, int eventData1, int eventData2);
void CVICALLBACK ConfigurationParameters(int menubar, int menuItem, void *callbackData, int panel);
int CVICALLBACK DigitalPanel(int panel, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK DisplayCalibration(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);

```

```

void CVICALLBACK Erase(int menubar, int menuitem, void *callbackData, int panel);
void CVICALLBACK Exit(int menubar, int menuitem, void *callbackData, int panel);
int CVICALLBACK InitializationPanel(int panel, int event, void *callbackData, int eventData1, int eventData2);
void CVICALLBACK InitializeInstrument(int menubar, int menuitem, void *callbackData, int panel);
void CVICALLBACK Measure(int menubar, int menuitem, void *callbackData, int panel);
void CVICALLBACK SetBacklightState(int menubar, int menuitem, void *callbackData, int panel);
int CVICALLBACK SetCalNumber(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK SetGain(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK SetMaxValue(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK SetMinValue(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK SetMode(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK SetUnits(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK SetWavelength(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK SetZero(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
void CVICALLBACK Start(int menubar, int menuitem, void *callbackData, int panel);
void CVICALLBACK Stop(int menubar, int menuitem, void *callbackData, int panel);
int CVICALLBACK StripPanel(int panel, int event, void *callbackData, int eventData1, int eventData2);

```

```

#ifdef __cplusplus
}
#endif

```

S370 Demo.c

```

#include <virte.h> /* Needed if linking in external compiler; harmless otherwise */
#include <userint.h>
#include "S370 Demo.h"
#include "S370 Global Variables.h"

int main (int argc, char *argv[])
{
    panelHandle[INITIAL] = LoadPanel (0, "S370 Demo.uir", INITIAL);
    panelHandle[ABOUT] = LoadPanel (0, "S370 Demo.uir", ABOUT);
    panelHandle[CONFIG] = LoadPanel (0, "S370 Demo.uir", CONFIG);
    panelHandle[DIGITAL] = LoadPanel (0, "S370 Demo.uir", DIGITAL);
    panelHandle[STRIP] = LoadPanel (0, "S370 Demo.uir", STRIP);

    configurationMenuId = GetPanelMenuBar (panelHandle[INITIAL]); /* load menubar */
    digitalDisplayMenuId = GetPanelMenuBar (panelHandle[DIGITAL]); /* load menubar */
    stripChartMenuId = GetPanelMenuBar (panelHandle[STRIP]); /* load menubar */

    DisplayPanel (panelHandle[INITIAL]);
    RunUserInterface ();
    return 0;
}

```

About Panel.c

```

#include "S370 Demo.h"

int CVICALLBACK AboutPanel (int panel, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_CLOSE:
            HidePanel(panel);
            break;
    }
    return 0;
}

```

Configuration Panel.c

```

#include "S370 Demo.h"
#include "S370 Global Variables.h"

int CVICALLBACK ConfigurationPanel (int panel, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_CLOSE:
            HidePanel (panelHandle[CONFIG]);
    }
}

```

```

        DisplayPanel (panelHandle[INITIAL]);
        SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_MEAS_PARAMS, ATTR_DIMMED, FALSE);
        break;
    }
    return 0;
}

int CVICALLBACK SetCalNumber (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    int calibration;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panel, CONFIG_CAL1, &calibration);
            uds370_set_calibration (s370Id, calibration);
            break;
    }
    return 0;
}

int CVICALLBACK SetGain (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    int gain;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panel, CONFIG_GAIN1, &gain);
            uds370_set_range (s370Id, gain);
            break;
    }
    return 0;
}

int CVICALLBACK SetMode (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    int mode;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panel, CONFIG_MODE1, &mode);
            uds370_set_mode (s370Id, mode);
            break;
    }
    return 0;
}

int CVICALLBACK SetUnits (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    int units;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panel, CONFIG_UNITS1, &units);
            uds370_set_units (s370Id, units);
            break;
    }
    return 0;
}

int CVICALLBACK SetWavelength (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    int wavelength;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panel, CONFIG_WAVE1, &wavelength);
            uds370_set_wavelength (s370Id, wavelength);
            break;
    }
    return 0;
}

```

```

}

int CVICALLBACK SetZero (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    int zeroState;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panel, CONFIG_ZERO1, &zeroState);
            udts370_set_zero (s370Id, zeroState);
            break;
    }
    return 0;
}

int CVICALLBACK DisplayCalibration (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            udts370_display_calibration (s370Id);
            break;
    }
    return 0;
}

```

Continuous Measurements.c

```

#include <formatio.h>
#include <userint.h>
#include "S370 Demo.h"
#include "S370 Global Variables.h"

void getGain(int , int, char* );
void getMode(int , char* );
void getUnits(int , char* );

void ContinuousMeasurement ()
{
    double value;
    double a;
    int status;

    int channel;
    int wavelength;
    int autoManual;
    int mode;
    int gain;
    int units;
    char gainDescription[9];
    char modeDescription[10];
    char unitsDescription[9];
    char yAxisLabel[30];
    char wavelengthDescription[7];

    a = 1.0000;

    if(displayMode == 0)
    {
        udts370_read_units (s370Id,&units);
        getUnits(units, unitsDescription);
        SetCtrlVal(panelHandle[DIGITAL], DIGITAL_UNIT1, unitsDescription);
    }
    else
    {
        udts370_read_units (s370Id, &units);
        getUnits(units, unitsDescription);
        Fmt(yAxisLabel, "%s<Reading ( %s )",unitsDescription);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_STRIP1, ATTR_YNAME, yAxisLabel);
    }
}

```

```

while (stopMeasurements == OFF) // continue loop until stop measurements is encountered
{
    value = 0;
    status = 0;
    ProcessSystemEvents ();

    uds370_read_data (s370Id, &value, &status);

    if(displayMode == 0)
    {
        SetCtrlVal(panelHandle[DIGITAL], DIGITAL_READING1, value);

        SetCtrlVal(panelHandle[DIGITAL], DIGITAL_NORM1, OFF);
        SetCtrlVal(panelHandle[DIGITAL], DIGITAL_PREV1, OFF);
        SetCtrlVal(panelHandle[DIGITAL], DIGITAL_OVL1, OFF);
        SetCtrlVal(panelHandle[DIGITAL], DIGITAL_UNDEF1, OFF);

        switch(status)
        {
            case 1:
                SetCtrlVal(panelHandle[DIGITAL], DIGITAL_OVL1, ON);
                break;

            case 2:
                SetCtrlVal(panelHandle[DIGITAL], DIGITAL_PREV1, ON);
                break;

            case 3:
                SetCtrlVal(panelHandle[DIGITAL], DIGITAL_NORM1, ON);
                break;

            case 4:
                SetCtrlVal(panelHandle[DIGITAL], DIGITAL_UNDEF1, ON);
                break;
        }

        uds370_read_status (s370Id, &mode, &wavelength, &autoManual, &gain);
        getMode(mode, modeDescription);
        getGain(gain, autoManual, gainDescription);
        Fmt(wavelengthDescription, "%s<%i[w4]", wavelength);
        SetCtrlVal(panelHandle[DIGITAL], DIGITAL_MODE1, modeDescription);
        SetCtrlVal(panelHandle[DIGITAL], DIGITAL_GAIN1, gainDescription);
        SetCtrlVal(panelHandle[DIGITAL], DIGITAL_WAVE1, wavelengthDescription);
    }
    else
    {
        SetCtrlVal(panelHandle[STRIP], STRIP_READING1, value);
        PlotStripChartPoint (panelHandle[STRIP], STRIP_STRIP1, value);

        SetCtrlVal(panelHandle[STRIP], STRIP_LED_N1, OFF);
        SetCtrlVal(panelHandle[STRIP], STRIP_LED_P1, OFF);
        SetCtrlVal(panelHandle[STRIP], STRIP_LED_O1, OFF);
        SetCtrlVal(panelHandle[STRIP], STRIP_LED_U1, OFF);

        switch(status)
        {
            case 1:
                SetCtrlVal(panelHandle[STRIP], STRIP_LED_O1, ON);
                break;

            case 2:
                SetCtrlVal(panelHandle[STRIP], STRIP_LED_P1, ON);
                break;

            case 3:
                SetCtrlVal(panelHandle[STRIP], STRIP_LED_N1, ON);
                break;

            case 4:

```



```

    SetCtrlVal (panelHandle[ABOUT], ABOUT_TEXT, "ssavrda@hotmail.com\n");

    SetPanelPos (panelHandle[ABOUT], VAL_AUTO_CENTER, VAL_AUTO_CENTER);// set the panel position

    DisplayPanel (panelHandle[ABOUT]);
}

void CVICALLBACK Exit (int menuBar, int menuItem, void *callbackData, int panel)
{
    stopMeasurements = ON;
    udts370_close (s370Id);
    QuitUserInterface (0);
}

void CVICALLBACK SetBacklightState (int menuBar, int menuItem, void *callbackData, int panel)
{
    int value;
    GetMenuBarAttribute (configurationMenuId, BAR_CONFIG_BACKLIGHT, ATTR_CHECKED,&value);
    SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_BACKLIGHT,ATTR_CHECKED, !value);
    SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_BACKLIGHT,ATTR_CHECKED, !value);
    SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_BACKLIGHT,ATTR_CHECKED, !value);

    udts370_set_screen (s370Id, value + 2);
}

void CVICALLBACK BlankScreen (int menuBar, int menuItem, void *callbackData, int panel)
{
    int value;

    GetMenuBarAttribute (configurationMenuId, BAR_CONFIG_BLANK, ATTR_CHECKED,&value);
    SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_BLANK,ATTR_CHECKED, !value);
    SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_BLANK,ATTR_CHECKED, !value);
    SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_BLANK,ATTR_CHECKED, !value);

    udts370_set_screen (s370Id, !value);
}

void CVICALLBACK ConfigurationParameters (int menuBar, int menuItem, void *callbackData, int panel)
{
    HidePanel (panel);
    DisplayPanel (panelHandle[CONFIG]);
    SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_MEAS_PARS, ATTR_DIMMED, TRUE);
}

void CVICALLBACK Start (int menuBar, int menuItem, void *callbackData, int panel)
{
    int displayMode;

    stopMeasurements = OFF;
    udts370_start_stop (s370Id, ON);

    GetCtrlVal (panelHandle[INITIAL], INITIAL_SET_DISPLAY, &displayMode);
    if (displayMode == 0)
    {
        SetMenuBarAttribute (digitalDisplayMenuId, BAR2_STOP, ATTR_DIMMED, FALSE);
        SetMenuBarAttribute (digitalDisplayMenuId, BAR2_START, ATTR_DIMMED, TRUE);
        SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_MEAS_PARS, ATTR_DIMMED, TRUE);
    }
    else
    {
        SetMenuBarAttribute (stripChartMenuId, REALTIME_STOP, ATTR_DIMMED, FALSE);
        SetMenuBarAttribute (stripChartMenuId, REALTIME_START, ATTR_DIMMED, TRUE);
        SetMenuBarAttribute (stripChartMenuId, REALTIME_ERASE, ATTR_DIMMED, TRUE);
        SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_MEAS_PARS, ATTR_DIMMED, TRUE);
    }
    ContinuousMeasurement();
}

```

```

void CVICALLBACK Stop (int menuBar, int menuItem, void *callbackData, int panel)
{
    int displayMode;

    GetCtrlVal (panelHandle[INITIAL], INITIAL_SET_DISPLAY, &displayMode);
    stopMeasurements = ON;
    udts370_start_stop (s370Id, OFF);

    if (displayMode == 0)
    {
        SetMenuBarAttribute (digitalDisplayMenuId, BAR2_STOP, ATTR_DIMMED, TRUE);
        SetMenuBarAttribute (digitalDisplayMenuId, BAR2_START, ATTR_DIMMED, FALSE);
        SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_MEAS_PARAMS, ATTR_DIMMED, FALSE);
    }
    else
    {
        SetMenuBarAttribute (stripChartMenuId, REALTIME_STOP, ATTR_DIMMED, TRUE);
        SetMenuBarAttribute (stripChartMenuId, REALTIME_START, ATTR_DIMMED, FALSE);
        SetMenuBarAttribute (stripChartMenuId, REALTIME_ERASE, ATTR_DIMMED, FALSE);
        SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_MEAS_PARAMS, ATTR_DIMMED, FALSE);
    }
}

void CVICALLBACK Erase (int menuBar, int menuItem, void *callbackData, int panel)
{
    ClearStripChart (panel, STRIP_STRIP1);
}

void CVICALLBACK Measure (int menuBar, int menuItem, void *callbackData, int panel)
{
    int mode;

    parentPanel = panel;
    stopMeasurements = TRUE;
    HidePanel (parentPanel);
    GetCtrlVal (panelHandle[INITIAL], INITIAL_SET_DISPLAY, &displayMode);

    if (displayMode == 0)
    {
        GetCtrlVal(panelHandle[CONFIG], CONFIG_MODE1, &mode);

        if (mode == 0)
        {
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_READING1, ATTR_DIMMED, TRUE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_UNIT1, ATTR_DIMMED, TRUE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_MODE1, ATTR_DIMMED, TRUE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_GAIN1, ATTR_DIMMED, TRUE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_NORM1, ATTR_DIMMED, TRUE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_PREV1, ATTR_DIMMED, TRUE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_OVL1, ATTR_DIMMED, TRUE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_UNDEF1, ATTR_DIMMED, TRUE);
        }
        else
        {
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_READING1, ATTR_DIMMED, FALSE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_UNIT1, ATTR_DIMMED, FALSE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_MODE1, ATTR_DIMMED, FALSE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_GAIN1, ATTR_DIMMED, FALSE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_NORM1, ATTR_DIMMED, FALSE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_PREV1, ATTR_DIMMED, FALSE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_OVL1, ATTR_DIMMED, FALSE);
            SetCtrlAttribute (panelHandle[DIGITAL], DIGITAL_UNDEF1, ATTR_DIMMED, FALSE);
        }
        SetMenuBarAttribute (digitalDisplayMenuId, BAR2_STOP, ATTR_DIMMED, TRUE);
        DisplayPanel(panelHandle[DIGITAL]);
    }
    else
    {
        GetCtrlVal(panelHandle[CONFIG], CONFIG_MODE1, &mode);
        if (mode == 0)

```

```

    {
        SetCtrlAttribute (panelHandle[STRIP], STRIP_STRIP1, ATTR_DIMMED, TRUE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_MAX_CH1, ATTR_DIMMED, TRUE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_MIN_CH1, ATTR_DIMMED, TRUE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_GAIN1, ATTR_DIMMED, TRUE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_MODE1, ATTR_DIMMED, TRUE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_LED_N1, ATTR_DIMMED, TRUE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_LED_P1, ATTR_DIMMED, TRUE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_LED_O1, ATTR_DIMMED, TRUE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_LED_U1, ATTR_DIMMED, TRUE);
    }
    else
    {
        SetCtrlAttribute (panelHandle[STRIP], STRIP_STRIP1, ATTR_DIMMED, FALSE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_MAX_CH1, ATTR_DIMMED, FALSE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_MIN_CH1, ATTR_DIMMED, FALSE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_GAIN1, ATTR_DIMMED, FALSE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_MODE1, ATTR_DIMMED, FALSE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_LED_N1, ATTR_DIMMED, FALSE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_LED_P1, ATTR_DIMMED, FALSE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_LED_O1, ATTR_DIMMED, FALSE);
        SetCtrlAttribute (panelHandle[STRIP], STRIP_LED_U1, ATTR_DIMMED, FALSE);
    }
    SetMenuBarAttribute (stripChartMenuId, REALTIME_STOP, ATTR_DIMMED, TRUE);
    DisplayPanel(panelHandle[STRIP]);
}
}

void CVICALLBACK InitializeInstrument (int menuBar, int menuItem, void *callbackData, int panel)
{
    int IEEEAddress;
    int error;
    int model;

    uds370_close (s370Id);
    Delay(1);
    GetCtrlVal(panelHandle[INITIAL], INITIAL_ADDR, &IEEEAddress);
    error = uds370_init (IEEEAddress, 1, &model, &s370Id);
    if (error == 0)
    {
        if (model == 370)
        {
            SetCtrlVal (panelHandle[INITIAL], INITIAL_LED, 1);
            SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_BACKLIGHT, ATTR_DIMMED, FALSE);
            SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_BLANK, ATTR_DIMMED, FALSE);
            SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_MEAS_PARAMS, ATTR_DIMMED, FALSE);
            SetMenuBarAttribute (configurationMenuId, BAR_MEASURE, ATTR_DIMMED, FALSE);
            SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_INITIALIZE, ATTR_DIMMED, TRUE);
        }
        else
        {
            SetCtrlAttribute (panelHandle[INITIAL], INITIAL_LED, ATTR_OFF_COLOR, VAL_RED);
            MessagePopup ("Initialization Error!", "Not connected to a Model S370 instrument.");
        }
    }
    else
    {
        MessagePopup ("Initialization Error!", "Failed to initialize the Model S370 instrument. Aborting application.");
        uds370_close (s370Id);
        QuitUserInterface (0);
    }
}
}

```

S370 Global Variables.h

```
#include "uds370.h"
```

```

#define TRUE 1
#define FALSE 0
#define ON 1

```

```
#define OFF 0
```

```
int panelHandle[6];  
int configurationMenuId;  
int digitalDisplayMenuId;  
int stripChartMenuId;  
int parentPanel;  
int stopMeasurements;  
int s370Id;  
int displayMode;
```

Strip Panel.c

```
#include <cvirte.h> /* Needed if linking in external compiler; harmless otherwise */  
#include <userint.h>  
#include "S370 Demo.h"  
#include "S370 Global Variables.h"
```

```
int CVICALLBACK StripPanel (int panel, int event, void *callbackData, int eventData1, int eventData2)  
{  
    switch (event)  
    {  
        case EVENT_CLOSE:  
            if(stopMeasurements == TRUE)  
            {  
                HidePanel(panel);  
                DisplayPanel(parentPanel);  
            }  
            break;  
    }  
    return 0;  
}
```

```
int CVICALLBACK SetMaxValue (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)  
{  
    double minValue;  
    double maxValue;  
  
    switch (event)  
    {  
        case EVENT_COMMIT:  
            GetCtrlVal (panel, STRIP_MIN_CH1, &minValue);  
            GetCtrlVal (panel, STRIP_MAX_CH1, &maxValue);  
            SetAxisScalingMode (panel, STRIP_STRIP1, VAL_LEFT_YAXIS, VAL_MANUAL, minValue,  
maxValue);  
            SetCtrlAttribute (panel, STRIP_MIN_CH1, ATTR_FRAME_COLOR, VAL_PANEL_GRAY);  
            break;  
    }  
    return 0;  
}
```

```
int CVICALLBACK SetMinValue (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)  
{  
    double minValue;  
    double maxValue;  
  
    switch (event)  
    {  
        case EVENT_COMMIT:  
            GetCtrlVal (panel, STRIP_MIN_CH1, &minValue);  
            GetCtrlVal (panel, STRIP_MAX_CH1, &maxValue);  
            SetAxisScalingMode (panel, STRIP_STRIP1, VAL_LEFT_YAXIS, VAL_MANUAL, minValue,  
maxValue);  
            SetCtrlAttribute (panel, STRIP_MAX_CH1, ATTR_FRAME_COLOR, VAL_PANEL_GRAY);  
            break;  
    }  
    return 0;  
}
```

Verbose Status Parameters.c

```
#include <formatio.h>
void getGain(int gain, int autoManual, char* gainDescription)
{
    switch(autoManual)
    {
        case 0:
            Fmt(gainDescription, "%s<A - ");
            break;

        case 1:
            Fmt(gainDescription, "%s<M - ");
            break;
    }
    switch(gain)
    {
        case 3:
            Fmt(gainDescription, "%s<%s3", gainDescription);
            break;

        case 4:
            Fmt(gainDescription, "%s<%s4", gainDescription);
            break;

        case 5:
            Fmt(gainDescription, "%s<%s5", gainDescription);
            break;

        case 6:
            Fmt(gainDescription, "%s<%s6", gainDescription);
            break;

        case 7:
            Fmt(gainDescription, "%s<%s7", gainDescription);
            break;

        case 8:
            Fmt(gainDescription, "%s<%s8", gainDescription);
            break;

        case 9:
            Fmt(gainDescription, "%s<%s9", gainDescription);
            break;
    }
}

void getMode(int mode, char* modeDescription)
{
    switch(mode)
    {
        case 0:
            Fmt(modeDescription, "%s<Off");
            break;

        case 1:
            Fmt(modeDescription, "%s<Log");
            break;

        case 2:
            Fmt(modeDescription, "%s<Lin");
            break;

        case 3:
            Fmt(modeDescription, "%s<Ratio");
            break;

        case 4:
            Fmt(modeDescription, "%s<Log Ratio");
            break;
    }
}
```

```

}

void getUnits(int units, char* unitsDescription)
{
    switch(units)
    {
        case 1:
            Fmt(unitsDescription, "%s<W");
            break;

        case 2:
            Fmt(unitsDescription, "%s<Fc");
            break;

        case 3:
            Fmt(unitsDescription, "%s<Lux");
            break;

        case 4:
            Fmt(unitsDescription, "%s<Fl");
            break;

        case 5:
            Fmt(unitsDescription, "%s<W/cm2");
            break;

        case 6:
            Fmt(unitsDescription, "%s<W/cm2-sr");
            break;

        case 7:
            Fmt(unitsDescription, "%s<Cd/m2");
            break;

        case 8:
            Fmt(unitsDescription, "%s<Lm");
            break;

        case 9:
            Fmt(unitsDescription, "%s<A");
            break;

        case 10:
            Fmt(unitsDescription, "%s<Cd");
            break;

        case 11:
            Fmt(unitsDescription, "%s<W/sr");
            break;
    }
}

```

7. Index of UDT S370 Device Dependent Commands

MESSAGE, 13

A, 10

B0, 6, 13

B1, 13

B2, 13

B3, 13

Bx, 12, 13

DG, 9

DL, 9

DN, 9

DR, 9

Fx, 14

G, 14

H, 14

K, 7

Kx, 7

M, 10

M3, 10

M4, 10

M5, 10

M6, 10

M7, 10

M8, 10

M9, 10

Mx, 10

N, 15

O, 15

P, 15

R, 8, 9

S, 10

U, 15

Ux, 12

V1, 11

V2, 11

V3, 11

V4, 12

V5, 12

V6, 12

V7, 12

V8, 12

Vx, 11

WD, 8

WU, 8

Wx, 8

X, 11

Z0, 14

Z1, 14

Zx, 13