

# **UDT 531 Position Indicator LabWindows Instrument Driver**

**UDT Instruments**

**June, 2005**

1.	Introduction.....	5
2.	Known Problems .....	5
3.	Error Codes .....	5
4.	Function Tree Layout.....	6
5.	Function Reference.....	6
5.1.	Initialize.....	6
	Parameter List .....	7
	Address .....	7
	Reset.....	7
	Instrument ID .....	7
	Errors .....	7
5.2.	Set Backlight [B].....	7
	Parameter List .....	7
	Instrument ID .....	7
	Backlight.....	7
	Errors .....	7
5.3.	Set Board [N] .....	8
	Parameter List .....	8
	Instrument ID .....	8
	Board .....	8
	Errors .....	8
5.4.	Set Detector [D].....	8
	Parameter List .....	8
	Instrument ID .....	8
	Detector .....	8
	Errors .....	8
5.5.	Set Calibration [E, EX, EY].....	8
	Parameter List .....	9
	Instrument ID .....	9
	Axis .....	9
	Calibration.....	9
	Errors .....	9
5.6.	Set Responsivity [R, RX, RY].....	9
	Parameter List .....	9
	Instrument ID .....	9
	Axis .....	9
	Responsivity .....	9
	Errors .....	9
5.7.	Set Bias [V].....	10
	Parameter List .....	10
	Instrument ID .....	10
	Bias .....	10
	Errors .....	10
5.8.	Set Gain [A, AX, AY, MX, MY].....	10
	Parameter List .....	10
	Instrument ID .....	10
	Mode .....	10
	Gain.....	10
	Errors .....	11
5.9.	Set Integration [I].....	11
	Parameter List .....	11
	Instrument ID .....	11
	Integration Time .....	11
	Errors .....	11

5.10.	Set Axis Units [UX, UY].....	11
	Parameter List .....	11
	Instrument ID .....	11
	Axis .....	11
	Axis Units .....	11
	Errors .....	12
5.11.	Set Sum Units [USX, USY].....	12
	Parameter List .....	12
	Instrument ID .....	12
	Axis .....	12
	Sum Units.....	12
	Errors .....	12
5.12.	Set Ambient Zero [Q, QX, QY] .....	12
	Parameter List .....	13
	Instrument ID .....	13
	Ambient Zero .....	13
	Errors .....	13
5.13.	Set Position Zero [Z, ZX, ZY].....	13
	Parameter List .....	13
	Instrument ID .....	13
	Position Zero.....	13
	Errors .....	13
5.14.	Set Data Logger [L0, L1, LR].....	13
	Parameter List .....	14
	Instrument ID .....	14
	Data Logger .....	14
	Errors .....	14
5.15.	Set Logger Time [LE, LL] .....	14
	Parameter List .....	14
	Instrument ID .....	14
	Mode .....	14
	Setting.....	14
	Errors .....	14
5.16.	Set Trigger [T].....	14
	Parameter List .....	15
	Instrument ID .....	15
	Trigger .....	15
	Errors .....	15
5.17.	Set Update [J] .....	15
	Parameter List .....	15
	Instrument ID .....	15
	Display Update .....	15
	Errors .....	15
5.18.	Start/Stop [G, H] .....	15
	Parameter List .....	15
	Instrument ID .....	15
	Start/Stop .....	15
	Errors .....	16
5.19.	Read Data [Fx, Fy, FX, FY] .....	16
	Parameter List .....	16
	Instrument ID .....	16
	Mode .....	16
	Value.....	16
	Status.....	16
	Errors .....	16

5.20.	Read Logger Position [LX, LY].....	16
	Parameter List .....	17
	Instrument ID .....	17
	Axis .....	17
	Data Pointer .....	17
	Value.....	17
	Errors .....	17
5.21.	Read System Cfg [C].....	17
	Parameter List .....	17
	Instrument ID .....	17
	Ref Voltage .....	17
	Board 1 .....	17
	ID 1.....	18
	Board 2.....	18
	ID 2.....	18
	Release Date.....	18
	Errors .....	18
5.22.	Read Status [S] .....	18
	Parameter List .....	18
	Instrument ID .....	18
	Status.....	18
	Errors .....	19
5.23.	Close .....	19
	Parameter List .....	20
	Instrument ID .....	20
	Errors .....	20
6.	Test Program .....	21
6.1.	Program Operation .....	21
6.2.	Program Internals.....	25
	S531 Demo.h .....	25
S531 Demo.c.....		29
	About Panel.c .....	31
	Alarm.c.....	31
	Configuration Panel.c .....	33
	Continuous Measurements.c.....	40
	Digital Panel.c .....	46
	Logger Configuration Panel .....	47
	Menu Bar.c.....	47
	Numeric Format Panel .....	51
	Strip Plot Panel.....	52
	XY Plot Panel.....	53
	S531 Global Variables.....	53
7.	Index of UDT 531 Device Dependent Commands .....	55
7.	Index of UDT 531 Device Dependent Commands .....	55

## **UDT 531 Position Indicator LabWindows Instrument Driver**

### **1. Introduction**

This instrument module provides LabWindows/CVI programming support for the UDT 531 Position Indicator. 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 device-dependent 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 device-dependent 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 531. 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 531 are implemented in this driver.

### **3. Error Codes**

All error codes are returned in a global integer variable:

`udt531_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
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
300	Instrument error

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

#### **4. Function Tree Layout**

- UDT 531
  - Initialize
  - Configure
    - Set Backlight
    - Set Board
  - Detector
    - Set Detector
    - Set Calibration
    - Set Responsivity
    - Set Bias
    - Set Gain
    - Set Integration
    - Set Axis Units
    - Set Sum Units
  - Zero
    - Set Ambient Zero
    - Set Position Zero
  - Data Logger
    - Set Data Logger
    - Set Logger Time
  - Measure
    - Set Trigger
    - Set Update
    - Start/Stop
    - Read Data
    - Read Logger Position
    - Read System Cfg
    - Read Status
  - Close

#### **5. Function Reference**

##### **5.1. Initialize**

Description: Initializes the instrument in the following ways:

- 1). opens the instrument and sets the GPIB address,
- 2). device clear (if selected),
- 3). returns the instrument ID.

The instrument ID is used in all of the other function calls to distinguish between multiple 531 instruments connected to the bus. For example, if several 531's were used with device addresses of 5, 8, and 12, the corresponding instrument ID's would be 1, 2, and 3. For one 531 (the usual situation), the default ID = 1 is used.

```
int udt531_init(Address, Reset, Instrument_ID);  
int Address;  
int Reset;  
int *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.

Variable Type: Integer

Valid Range: 0 to 1

0 Off  
1 On (Default)

#### 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 531 is used, this value will be used to differentiate between them. The first device initialized will be assigned an Instrument ID of 1.

Variable Type: Integer

Expected Response: 1 to 10

The maximum number of instruments that can be controlled in a single system is set internally in the driver include file, and is usually 10.

### **Errors**

0 Success  
-1 Invalid Address  
-2 Invalid Reset

## **5.2. Set Backlight [B]**

Description: Turns the instrument backlight on or off.

```
int udt531_set_backlight(Instrument_ID, Backlight);  
int Instrument_ID;  
int Backlight;
```

### **Parameter List**

#### Instrument ID

Description: The Instrument ID is used to select the device at a given address (see Initialize). If more than one 531 is used, this value will be used to differentiate between them. For one 531, (the usual situation), the default ID = 1 is used.

Variable Type: Integer

Valid Range: 1 to 10

#### Backlight

Description: Turns the instrument backlight on or off.

Variable Type: Integer

Valid Range: 0 to 1

0 Off (Default)  
1 On

### **Errors**

0 Success  
-1 Invalid Instrument ID  
-2 Invalid Backlight

### 5.3. Set Board [N]

Description: Sets either board 1 or board 2 inside the 531.

```
int udt531_set_board(Instrument_ID, Board);  
int Instrument_ID;  
int Board;
```

#### Parameter List

Instrument ID

Board

Description: Sets either board 1 or board 2 inside the 531.

Variable Type: Integer

Valid Range: 1 to 2

1	1 (Default)
2	2

#### Errors

0	Success
-1	Invalid Instrument ID
-2	Invalid Board

### 5.4. Set Detector [D]

Description: Sets the detector type to dual axis, quadrant, single axis, or duo-lateral.

```
int udt531_set_detector(Instrument_ID, Detector);  
int Instrument_ID;  
int Detector;
```

#### Parameter List

Instrument ID

Detector

Description: Sets the detector type to dual axis, quadrant, single axis, or duo-lateral.

Variable Type: Integer

Valid Range: 1 to 4

1	Dual Axis (Default)
2	Quadrant
3	Single Axis
4	Duo Lateral

#### Errors

0	Success
-1	Invalid Instrument ID
-2	Invalid Detector

### 5.5. Set Calibration [E, EX, EY]

Description: Sets up the calibration for the selected axis. If a single axis detector is selected, the calibration for an individual axis may be set.

```
int udt531_set_calibration(Instrument_ID, Axis, Calibration);  
int Instrument_ID;  
int Axis;  
double Calibration;
```

## Parameter List

### Instrument ID

#### Axis

Description: Sets which axis is to be calibrated.

Variable Type: Integer

Valid Range: 0 to 2

0 X and Y (Default)

1 X

2 Y

#### Calibration

Description: The number corresponding to one half of the x or y axis calibration field. For example, if a one centimeter square active area photodiode is used, then a value of 0.5 should be used.

Variable Type: Real

Valid Range:

#### **Errors**

0 Success

-1 Invalid Instrument ID

-2 Invalid Axis

## 5.6. Set Responsivity [R, RX, RY]

Description: Sets the responsivity for the x and y axes. If a single axis detector is selected, the responsivity for an individual axis may be set.

```
int udt531_set_responsivity(Instrument_ID, Axis, Responsivity);
```

```
int Instrument_ID;
```

```
int Axis;
```

```
double Responsivity;
```

## Parameter List

### Instrument ID

#### Axis

Description: Selects the axis for setting the responsivity.

Variable Type: Integer

Valid Range: 0 to 2

0 X and Y (Default)

1 X

2 Y

#### Responsivity

Description: Sets the responsivity for the selected axis.

Variable Type: Real

Valid Range:

#### **Errors**

0 Success

-1 Invalid Instrument ID

-2 Invalid Axis

### 5.7. Set Bias [V]

Description: Sets the detector bias voltage.

```
int udt531_set_bias(Instrument_ID, Bias);  
int Instrument_ID;  
int Bias;
```

#### Parameter List

Instrument ID

Bias

Description: Sets the detector bias voltage.

Variable Type: Integer

Valid Range: 0 to 6

0	Off (Default)
1	-5 V
2	-2 V
3	-1 V
4	0 V
5	1 V
6	2 V

#### Errors

0	Success
-1	Invalid Instrument ID
-2	Invalid Bias

### 5.8. Set Gain [A, AX, AY, MX, MY]

Description: Sets the measurement gain to auto or manual with specific gain settings. If a single axis detector is selected, the gain for an individual axis may be set.

```
int udt531_set_gain(Instrument_ID, Mode, Gain);  
int Instrument_ID;  
int Mode;  
int Gain;
```

#### Parameter List

Instrument ID

Mode

Description: Sets the measurement gain to auto or manual with specific gain settings. If a single axis detector is selected, the gain for an individual axis may be set.

Variable Type: Integer

Valid Range: 0 to 5

0	Auto (Default)
1	Auto X
2	Auto Y
3	Manual
4	Manual X
5	Manual Y

Gain

Description: Sets the gain.

Variable Type: Integer

Valid Range: 3 to 7

### Errors

- 0 Success
- 1 Invalid Instrument ID
- 2 Invalid Mode
- 3 Invalid Gain

### 5.9. Set Integration [I]

Description: Sets the integration time.

```
int udt531_set_integration(Instrument_ID, Integration_Time);  
int Instrument_ID;  
int Integration_Time;
```

#### Parameter List

Instrument ID

Integration Time

Description: Sets the integration time.

Variable Type: Integer

Valid Range 1 to 32000 ms

### Errors

- 0 Success
- 1 Invalid Instrument ID
- 2 Invalid Channel
- 3 Invalid Integration Time

### 5.10. Set Axis Units [UX, UY]

Description: Sets the axis units for either the x or y axis.

```
int udt531_set_axis_units(Instrument_ID, Axis, Axis_Units);  
int Instrument_ID;  
int Axis;  
int Axis_Units;
```

#### Parameter List

Instrument ID

Axis

Description: Sets the axis for setting the units.

Variable Type: Integer

Valid Range: 0 to 1

- 0 X (Default)
- 1 Y

Axis Units

Description: Sets the axis units.

Variable Type: Integer

Valid Range: 0 to 5

- 0 None (Default)
- 1 Inches
- 2 Meters
- 3 Arc Sec
- 4 Degrees

5 Radians

#### Errors

0 Success  
-1 Invalid Instrument ID  
-2 Invalid Axis  
-3 Invalid Axis Units

### 5.11. Set Sum Units [USX, USY]

Description: Sets the sum units for either the x or y axis. If a single axis detector is selected, the sum units for the y axis may be set.

```
int udt531_set_sum_units(Instrument_ID, Axis, Sum_Units);  
int Instrument_ID;  
int Axis;  
int Sum_Units;
```

#### Parameter List

Instrument ID

Axis

Description: Sets the axis for setting the sum units.

Variable Type: Integer

Valid Range: 0 to 1

0 X (Default)  
1 Y

Sum Units

Description: Sets the sum units.

Variable Type: Integer

Valid Range: 0 to 1

0 Amps (Default)  
1 Watts

#### Errors

0 Success  
-1 Invalid Instrument ID  
-2 Invalid Axis  
-3 Invalid Sum Units

### 5.12. Set Ambient Zero [Q, QX, QY]

Description: Sets the ambient zero. If a single axis detector is selected, the ambient zero for an individual axis may be set.

```
int udt531_set_ambient_zero(Instrument_ID, Ambient_Zero);  
int Instrument_ID;  
int Ambient_Zero;
```

### Parameter List

#### Instrument ID

#### Ambient Zero

Description: Sets the ambient zero.

Variable Type: Integer

Valid Range: 0 to 5

- 0 Delete (Default)
- 1 Set
- 2 Delete X
- 3 Set X
- 4 Delete Y
- 5 Set Y

#### Errors

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

### 5.13. Set Position Zero [Z, ZX, ZY]

Description: Sets the position zero for the x or y axis. If a single axis detector is selected, the position zero for an individual axis may be set.

```
int udt531_set_position_zero(Instrument_ID, Position_Zero);  
int Instrument_ID;  
int Position_Zero;
```

### Parameter List

#### Instrument ID

#### Position Zero

Description: Sets the position zero.

Variable Type: Integer

Valid Range: 0 to 5

- 0 Delete (Default)
- 1 Set
- 2 Delete X
- 3 Set X
- 4 Delete Y
- 5 Set Y

#### Errors

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

### 5.14. Set Data Logger [L0, L1, LR]

Description: Turns the data logger on or off, and also resets the data pointer to 0.

```
int udt531_set_data_logger(Instrument_ID, Data_Logger);  
int Instrument_ID;  
int Data_Logger;
```

### **Parameter List**

#### Instrument ID

#### Data Logger

Description: Turns the data logger on or off, or resets the data pointer to zero.

Variable Type: Integer

Valid Range: 0 to 2

- 0 Off (Default)
- 1 On
- 2 Reset

#### **Errors**

- 0 Success
- 1 Invalid Instrument ID
- 2 Invalid Data Logger

### **5.15. Set Logger Time [LE, LL]**

Description: Sets the time increment and total elapsed time for the data logger.

```
int udt531_set_logger_time(Instrument_ID, Mode, Setting);  
int Instrument_ID;  
int Mode;  
int Setting;
```

### **Parameter List**

#### Instrument ID

#### Mode

Description: Sets the data logger mode for setting the time.

Variable Type: Integer

Valid Range: 0 to 1

- 0 Increment (Default)
- 1 Total

#### Setting

Description: Sets the time for the selected mode in multiples of 0.1 second.

Variable Type: Integer

Valid Range:

There are 1024 memory locations in the data logger, so total time / time increment should be less than or equal to 1024.

#### **Errors**

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

### **5.16. Set Trigger [T]**

Description: Sets the trigger to off, internal, or external.

```
int udt531_set_trigger(Instrument_ID, Trigger);  
int Instrument_ID;  
int Trigger;
```

### **Parameter List**

#### Instrument ID

#### Trigger

Description: Sets the trigger.

Variable Type: Integer

Valid Range: 0 to 2

0 Off (Default)

1 Internal

2 External

#### **Errors**

0 Success

-1 Invalid Instrument ID

-2 Invalid Trigger

### **5.17. Set Update [J]**

Description: Enables or disables the display update. When disabled, measurement cycle time is reduced.

```
int udt531_set_update(Instrument_ID, Display_Update);
```

```
int Instrument_ID;
```

```
int Display_Update;
```

### **Parameter List**

#### Instrument ID

#### Display Update

Description: Turns the display update on or off.

Variable Type: Integer

Valid Range: 0 to 1

0 Off (Default)

1 On

#### **Errors**

0 Success

-1 Invalid Instrument ID

-2 Invalid Display Update

### **5.18. Start/Stop [G, H]**

Description: Starts or stops the instrument from making measurements.

```
int udt531_start_stop(Instrument_ID, StartStop);
```

```
int Instrument_ID;
```

```
int 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)

1 On

### Errors

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

### 5.19. Read Data [Fx, Fy, FX, FY]

Description: Reads the position or sum data for the x and y axis. If a single axis detector is not selected, the sum data for both axes will be the same.

```
int udt531_read_data(Instrument_ID, Mode, Value, Status);  
int Instrument_ID;  
int Mode;  
double *Value;  
int *Status;
```

### Parameter List

Instrument ID

Mode

Description: Determines whether position or sum data is read from the instrument.

Variable Type: Integer

Valid Range: 0 to 3

- 0 X Position (Default)
- 1 Y Position
- 2 X Sum
- 3 Y Sum

Value

Description: Reads the position or sum data for the x and y axis.

Variable Type: Real

Expected Response: Depends on the detector head.

Status

Description: Returns the status of the measurement.

Variable Type: Integer

Expected Response: 1 to 4

- 1 Overrange
- 2 Previously read
- 3 Normal
- 4 Undefined

### Errors

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

### 5.20. Read Logger Position [LX, LY]

Description: Reads the x or y position for the data logger.

```
int udt531_read_logger_position(Instrument_ID, Axis, Data_Pointer, Value);  
int Instrument_ID;  
int Axis;  
int Data_Pointer;  
double *Value;
```

## Parameter List

### Instrument ID

#### Axis

Description: Determines whether the x or y position is read from the data logger.

Variable Type: Integer

Valid Range: 0 to 1

0 X Position (Default)

1 Y Position

#### Data Pointer

Description: Sets the data pointer to one of 1024 data points stored in the data logger memory.

Variable Type: Integer

Variable Range: -1 to 1024

-1 Read position at current pointer location, and increment pointer

0 to 1023 Read position for data point at specified pointer location

#### Value

Description: Reads the position data from the data logger for the x or y axis and pointer location.

Variable Type: Real

Expected Response: Depends on the detector head and calibration.

## Errors

0 Success

-1 Invalid Instrument ID

-2 Invalid Axis

-3 Invalid Data Pointer

## 5.21. Read System Cfg [C]

Description: Reads the system configuration.

```
int udt531_read_system_cfg(Instrument_ID, Ref_Voltage,  
Board_1, ID_1, Board , ID , Release_Date);
```

```
int Instrument_ID;
```

```
double *Ref_Voltage;
```

```
int *Board_1;
```

```
int *ID_1;
```

```
int *Board ;
```

```
int *ID ;
```

```
char *Release_Date;
```

## Parameter List

### Instrument ID

#### Ref Voltage

Description: Reads the reference voltage

Variable Type: Real

Expected Response:

#### Board 1

Description: Reads whether board 1 is installed in the 531.

Variable Type: Integer

Expected Response: 0 to 1

0 Not installed

1 Installed

ID 1

Description: Reads the ID of board 1.

Variable Type: Integer

Expected Response:

Board 2

Description: Reads whether board 2 is installed in the 531.

Variable Type: Integer

Expected Response: 0 to 1

0 Not installed

1 Installed

ID 2

Description: Reads the ID of board 2.

Variable Type: Integer

Expected Response:

Release Date

Description: Reads the release date and revision level.

Variable Type String (length 20)

Expected Response:

Jul-29-1991 REV 3.0

**Errors**

0 Success

-1 Invalid Instrument ID

**5.22. Read Status [S]**

Description: Reads the status of the instrument.

*int udt531\_read\_status(Instrument\_ID, Status);*

*int Instrument\_ID;*

*int Status[];*

**Parameter List**

Instrument ID

Status

Description: Reads the status of the instrument.

Variable Type: Integer array (length 13)

Expected Response:

Mode (Index 0)

1 Daul axis

2 Quadrant

3 Single axis

4 Duo-lateral

Period (Index 1)

1 to 32000 ms

Auto/Manual X (Index 2)

0 Manual

1 Auto

Gain X (Index 3)  
3 to 7

Auto/Manual Y (Index 4)  
0 Manual  
1 Auto

Gain Y (Index 5)  
3 to 7

Bias (Index 6)  
1 -5V  
2 -2V  
3 -1V  
4 0V  
5 1V  
6 2V  
7 5V

Ambient Zero X (Index 7)  
0 Delete zero  
1 Set zero

Ambient Zero Y (Index 8)  
0 Delete zero  
1 Set zero

Position Zero X (Index 9)  
0 Delete zero  
1 Set zero

Position Zero Y (Index 10)  
0 Delete zero  
1 Set zero

Logger (Index 11)  
0 Off  
1 On

Logger Done (Index 12)  
0 Not done  
1 Done

#### Errors

0 Success  
-1 Invalid Instrument ID

### 5.23. Close

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

```
int udt531_close(Instrument_ID);  
int Instrument_ID;
```

**Parameter List**  
Instrument ID

**Errors**

- 0 Success
- 1 Invalid Instrument ID

## 6. Test Program

This program runs a UDT 531 Position Indicator. The program (called S531 Demo) does the following:

- initializes the instrument,
- 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.

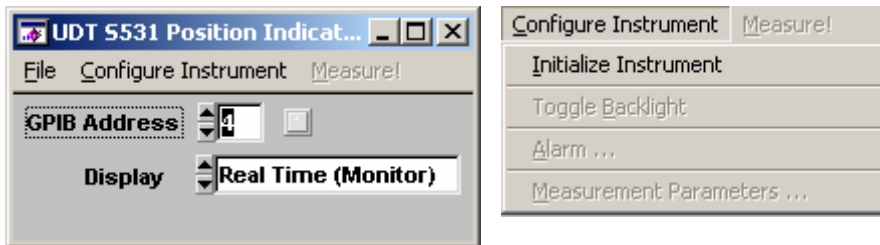


Figure 2. The Configuration panel

#### Instructions:

- Set the desired parameters for both boards.
- 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.

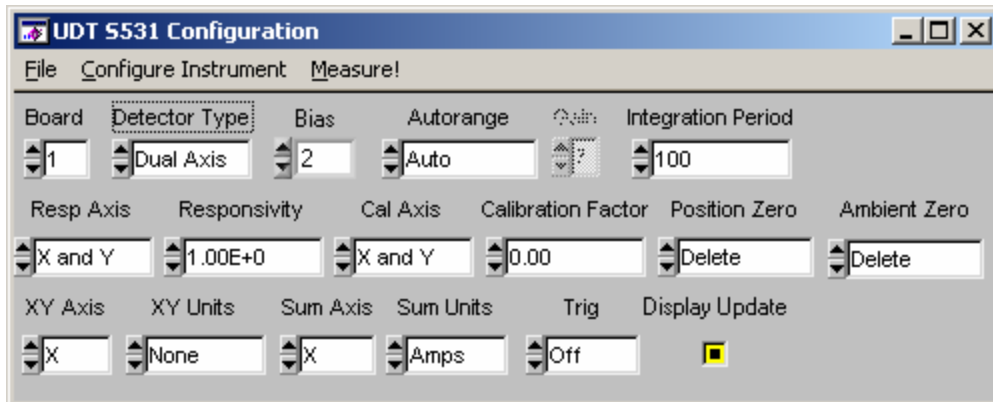


Figure 3. The Real Time Monitor panel

**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.
- Double clicking on any of the reading fields will display 'Numeric Display Format' panel which can be used to change the display type and precision (Figure 6).

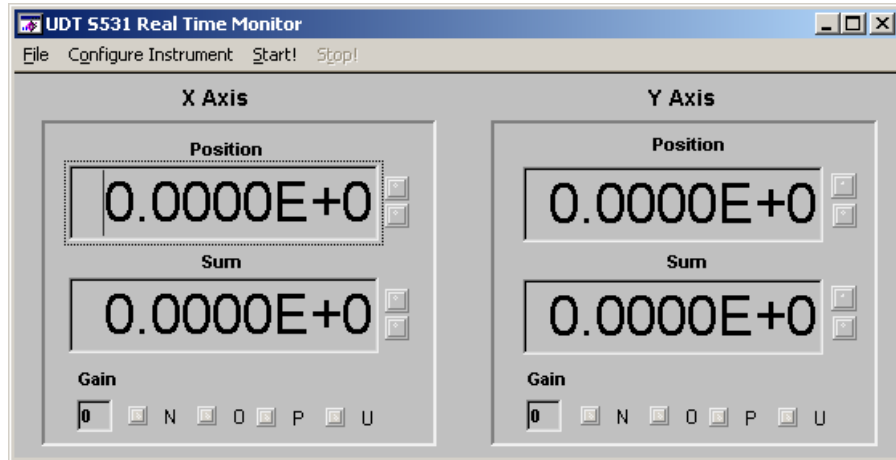


Figure 4. The Real Time Strip Chart panel

**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.

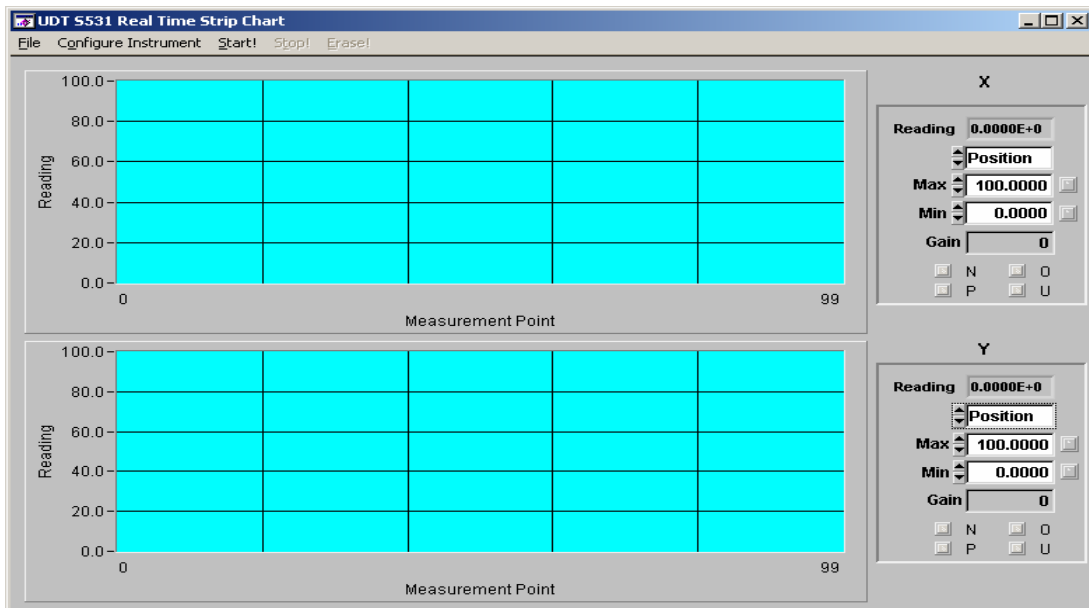


Figure 5. The Real Time X-Y Position panel

**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.

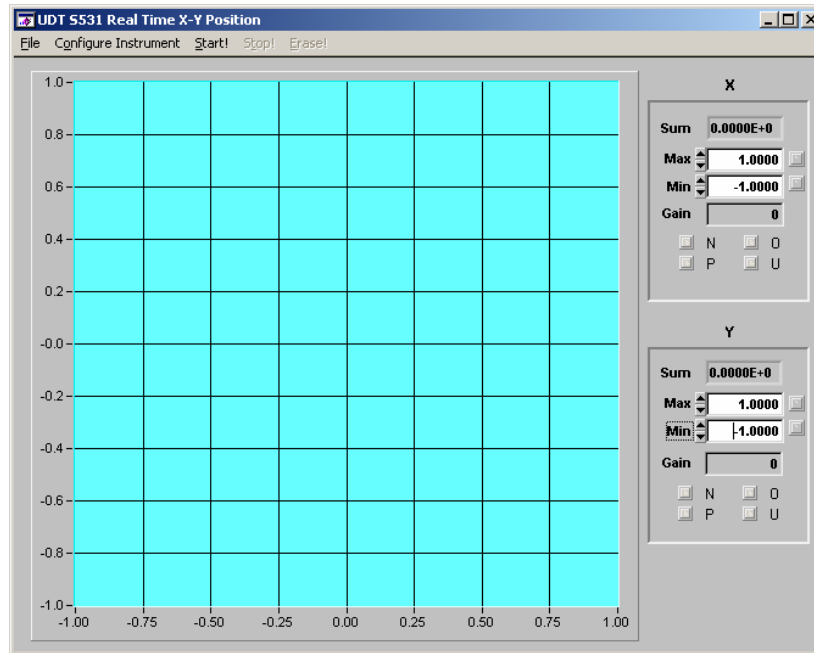


Figure 6. Numeric Display Format panel

**Instructions:**

- Select the display type, either Floating Point or Scientific.
- Select the data precision.

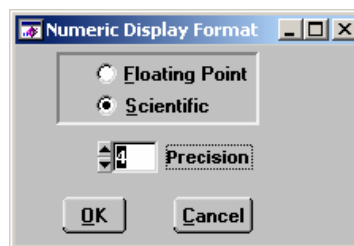


Figure 7. Alarm configuration panel

**Instructions:**

- Select whether the alarm is on.
- Select the high and low values for both the position and sum readings.
- Select whether to beep if the any reading falls out of the specified range. The sound will be generated through the pc builtin speaker.
- Select the beep frequency.

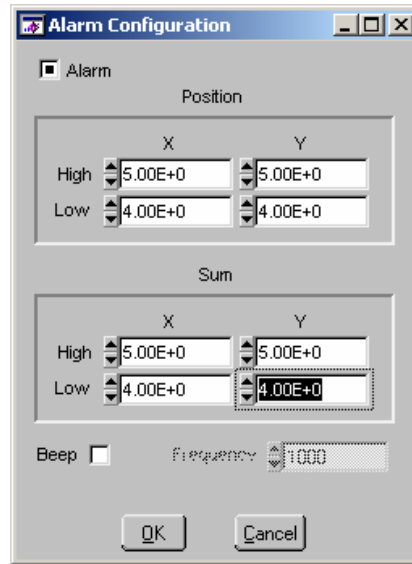
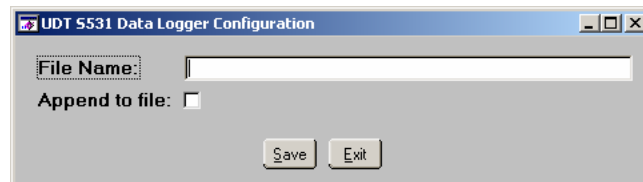


Figure 7. Data logger configuration panel

**Instructions:**

- Set the complete output filename.
- Select whether to append data to an existing file or whether to over write an existing file.



## 6.2. Program Internals

The test program makes use of the LabWindows/CVI instrument driver Udt531. 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 Udt531.c, Udt531.h, and UDT531.fp.

The program S531 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 S531 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 S531 Demo.h shows every control and the corresponding tags. (This file is generated automatically from within the LabWindows/CVI User Interface Editor).

### S531 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 ALARM 2 /* callback function: AlarmPanel */  
#define ALARM_ALARM_SUM_Y_LOW 2 /* callback function: Alarm */  
#define ALARM_ALARM_SUM_Y_HIGH 3 /* callback function: Alarm */  
#define ALARM_ALARM_SUM_X_LOW 4 /* callback function: Alarm */  
#define ALARM_CANCEL 5 /* callback function: Alarm */  
#define ALARM_OK 6 /* callback function: Alarm */  
#define ALARM_ALARM_SUM_X_HIGH 7 /* callback function: Alarm */  
#define ALARM_ALARM_POS_Y_LOW 8 /* callback function: Alarm */  
#define ALARM_ALARM_POS_Y_HIGH 9 /* callback function: Alarm */  
#define ALARM_ALARM_POS_X_LOW 10 /* callback function: Alarm */  
#define ALARM_ALARM_POS_X_HIGH 11 /* callback function: Alarm */  
#define ALARM_ALARM 12 /* callback function: Alarm */  
#define ALARM_ALARM_BEEP 13 /* callback function: Alarm */  
#define ALARM_FREQUENCY 14 /* callback function: Alarm */  
#define ALARM_DECORATION 15  
#define ALARM_TEXTMSG_3 16  
#define ALARM_TEXTMSG_4 17  
#define ALARM_TEXTMSG 18  
#define ALARM_TEXTMSG_2 19  
#define ALARM_TEXTMSG_5 20  
#define ALARM_DECORATION_2 21  
#define ALARM_TEXTMSG_6 22  
#define ALARM_TEXTMSG_7 23  
#define ALARM_TEXTMSG_8 24  
#define ALARM_TEXTMSG_9 25  
#define ALARM_TEXTMSG_10 26
```

```

#define CONFIG                3    /* callback function: ConfigurationPanel */
#define CONFIG_DET            2    /* callback function: DetectorType */
#define CONFIG_AUTORANGE     3    /* callback function: GainRange */
#define CONFIG_GAIN           4    /* callback function: GainRange */
#define CONFIG_INTEG         5    /* callback function: IntegrationPeriod */
#define CONFIG_RESP_AXIS     6    /* callback function: ResponsivityAxis */
#define CONFIG_RESP          7    /* callback function: Responsivity */
#define CONFIG_CAL_AXIS      8    /* callback function: CalibrateAxis */
#define CONFIG_CAL           9    /* callback function: CalibrationFactor */
#define CONFIG_PZERO         10   /* callback function: PositionZero */
#define CONFIG_AZERO         11   /* callback function: AmbientZero */
#define CONFIG_XY_AXIS       12   /* callback function: XYAxis */
#define CONFIG_XY_UNITS      13   /* callback function: XYUnits */
#define CONFIG_SUM_AXIS      14   /* callback function: SumAxis */
#define CONFIG_SUM_UNITS     15   /* callback function: SumUnits */
#define CONFIG_TRIG          16   /* callback function: Trigger */
#define CONFIG_UPDATE        17   /* callback function: Update */
#define CONFIG_BOARD         18   /* callback function: Board */
#define CONFIG_BIAS          19   /* callback function: Bias */

#define FORMAT                4    /* callback function: NumericFormatPanel */
#define FORMAT_NUMERIC        2    /* callback function: NumericType */
#define FORMAT_SCIENTIFIC    3    /* callback function: NumericType */
#define FORMAT_FLOAT          4    /* callback function: NumericType */
#define FORMAT_CANCEL         5    /* callback function: NumericFormat */
#define FORMAT_OK             6    /* callback function: NumericFormat */
#define FORMAT_DECORATION    7

#define INITIAL                5    /* callback function: MainPanel */
#define INITIAL_ADDR          2
#define INITIAL_LED           3
#define INITIAL_SET_DISPLAY   4

#define LOGGER                 6    /* callback function: LoggerPanel */
#define LOGGER_FILE_NAME      2
#define LOGGER_OVERWRITE_FILE 3
#define LOGGER_EXIT           4    /* callback function: LoggerConfiguration */
#define LOGGER_SAVE           5    /* callback function: LoggerConfiguration */

#define NUMERIC                7    /* callback function: RealTimeMonitorPanel */
#define NUMERIC_TEXTMSG       2
#define NUMERIC_DECORATION    3
#define NUMERIC_TEXTMSG_3     4
#define NUMERIC_ALARM_POS_X_HIGH 5
#define NUMERIC_ALARM_POS_X_LOW 6
#define NUMERIC_X_POSITION     7    /* callback function: SetNumericDisplayFormat */
#define NUMERIC_TEXTMSG_5     8
#define NUMERIC_ALARM_SUM_X_HIGH 9
#define NUMERIC_ALARM_SUM_X_LOW 10
#define NUMERIC_X_SUM         11   /* callback function: SetNumericDisplayFormat */
#define NUMERIC_GAIN_X        12
#define NUMERIC_LED_U_X       13
#define NUMERIC_LED_U_Y       14
#define NUMERIC_LED_O_X       15
#define NUMERIC_TEXTMSG_2     16
#define NUMERIC_LED_O_Y       17
#define NUMERIC_LED_P_X       18
#define NUMERIC_DECORATION_2  19
#define NUMERIC_LED_P_Y       20
#define NUMERIC_LED_N_X       21
#define NUMERIC_LED_N_Y       22
#define NUMERIC_TEXTMSG_4     23
#define NUMERIC_ALARM_POS_Y_HIGH 24
#define NUMERIC_ALARM_POS_Y_LOW 25
#define NUMERIC_Y_POSITION     26   /* callback function: SetNumericDisplayFormat */
#define NUMERIC_TEXTMSG_6     27
#define NUMERIC_ALARM_SUM_Y_HIGH 28
#define NUMERIC_ALARM_SUM_Y_LOW 29
#define NUMERIC_Y_SUM         30   /* callback function: SetNumericDisplayFormat */

```

```

#define NUMERIC_GAIN_Y          31

#define REAL_STRIP              8 /* callback function: RealTimeStripPlot */
#define REAL_STRIP_TYPE_Y       2 /* callback function: RealTimeStripChartType */
#define REAL_STRIP_MIN_Y        3 /* callback function: SetYRealTimeStrip */
#define REAL_STRIP_MAX_Y        4 /* callback function: SetYRealTimeStrip */
#define REAL_STRIP_ALARM_MIN_Y  5
#define REAL_STRIP_ALARM_MAX_Y  6
#define REAL_STRIP_GAIN_Y       7
#define REAL_STRIP_TYPE_X       8 /* callback function: RealTimeStripChartType */
#define REAL_STRIP_MIN_X        9 /* callback function: SetXRealTimeStrip */
#define REAL_STRIP_MAX_X       10 /* callback function: SetXRealTimeStrip */
#define REAL_STRIP_ALARM_MIN_X  11
#define REAL_STRIP_ALARM_MAX_X  12
#define REAL_STRIP_LED_U_Y      13
#define REAL_STRIP_LED_U_X      14
#define REAL_STRIP_LED_O_Y      15
#define REAL_STRIP_GAIN_X       16
#define REAL_STRIP_LED_P_Y      17
#define REAL_STRIP_LED_O_X      18
#define REAL_STRIP_LED_N_Y      19
#define REAL_STRIP_YSTRIP       20
#define REAL_STRIP_LED_P_X      21
#define REAL_STRIP_XSTRIP       22
#define REAL_STRIP_LED_N_X      23
#define REAL_STRIP_READING_Y    24
#define REAL_STRIP_READING_X    25
#define REAL_STRIP_TEXTMSG      26
#define REAL_STRIP_DECORATION   27
#define REAL_STRIP_DECORATION_2 28
#define REAL_STRIP_TEXTMSG_2    29

#define REAL_XY                 9 /* callback function: RealTimeXYPlot */
#define REAL_XY_MIN_Y           2 /* callback function: SetYRealTimeXY */
#define REAL_XY_MAX_Y           3 /* callback function: SetYRealTimeXY */
#define REAL_XY_ALARM_MIN_Y     4
#define REAL_XY_ALARM_MAX_Y     5
#define REAL_XY_GAIN_Y          6
#define REAL_XY_MIN_X           7 /* callback function: SetXRealTimeXY */
#define REAL_XY_MAX_X           8 /* callback function: SetXRealTimeXY */
#define REAL_XY_ALARM_MIN_X     9
#define REAL_XY_ALARM_MAX_X    10
#define REAL_XY_GAIN_X         11
#define REAL_XY_XY              12
#define REAL_XY_LED_U_Y         13
#define REAL_XY_DECORATION      14
#define REAL_XY_LED_O_Y         15
#define REAL_XY_LED_U_X         16
#define REAL_XY_LED_P_Y         17
#define REAL_XY_TEXTMSG         18
#define REAL_XY_LED_N_Y         19
#define REAL_XY_DECORATION_2    20
#define REAL_XY_TEXTMSG_2       21
#define REAL_XY_LED_O_X         22
#define REAL_XY_SUM_Y           23
#define REAL_XY_SUM_X           24
#define REAL_XY_LED_P_X         25
#define REAL_XY_LED_N_X         26

/* 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_QUIT           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_SEPARATOR_3    10
#define BAR_CONFIG_ALARM          11 /* callback function: ConfigureAlarm */
#define BAR_CONFIG_SEPARATOR_5    12
#define BAR_CONFIG_MEAS_PARMS     13 /* callback function: ConfigurationParameters */
#define BAR_MEASURE                14 /* 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_QUIT              5 /* callback function: Exit */
#define BAR2_CONFIG                 6
#define BAR2_CONFIG_BACKLIGHT       7 /* callback function: SetBacklightState */
#define BAR2_CONFIG_SEPARATOR_3     8
#define BAR2_CONFIG_DATA_LOGGER     9
#define BAR2_CONFIG_DATA_LOGGER_SUBMENU 10
#define BAR2_CONFIG_DATA_LOGGER_START 11 /* callback function: DataLogger */
#define BAR2_CONFIG_DATA_LOGGER_STOP 12 /* callback function: DataLogger */
#define BAR2_CONFIG_DATA_LOGGER_SEPARATOR 13
#define BAR2_CONFIG_DATA_LOGGER_CONFIGUR 14 /* callback function: DataLogger */
#define BAR2_CONFIG_SEPARATOR_4     15
#define BAR2_CONFIG_ALARM           16 /* callback function: ConfigureAlarm */
#define BAR2_CONFIG_SEPARATOR_5     17
#define BAR2_CONFIG_MEAS_PARMS      18 /* callback function: ConfigurationParameters */
#define BAR2_START                   19 /* callback function: Start */
#define BAR2_STOP                    20 /* 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_QUIT           5 /* callback function: Exit */
#define REALTIME_CONFIG              6
#define REALTIME_CONFIG_BACKLIGHT    7 /* callback function: SetBacklightState */
#define REALTIME_CONFIG_SEPARATOR_3  8
#define REALTIME_CONFIG_DATA_LOGGER  9
#define REALTIME_CONFIG_DATA_LOGGER_SUBM 10
#define REALTIME_CONFIG_DATA_LOGGER_STAR 11 /* callback function: DataLogger */
#define REALTIME_CONFIG_DATA_LOGGER_STOP 12 /* callback function: DataLogger */
#define REALTIME_CONFIG_DATA_LOGGER_SEPA 13
#define REALTIME_CONFIG_DATA_LOGGER_CONF 14 /* callback function: DataLogger */
#define REALTIME_CONFIG_SEPARATOR_5  15
#define REALTIME_CONFIG_ALARM        16 /* callback function: ConfigureAlarm */
#define REALTIME_CONFIG_SEPARATOR_4  17
#define REALTIME_CONFIG_MEAS_PARMS   18 /* callback function: ConfigurationParameters */
#define REALTIME_START               19 /* callback function: Start */
#define REALTIME_STOP                 20 /* callback function: Stop */
#define REALTIME_ERASE                21 /* 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);
int CVICALLBACK Alarm(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK AlarmPanel(int panel, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK AmbientZero(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK Bias(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK Board(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK CalibrateAxis(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK CalibrationFactor(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK Bias(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
void CVICALLBACK ConfigurationParameters(int menubar, int menuitem, void *callbackData, int panel);
void CVICALLBACK ConfigureAlarm(int menubar, int menuitem, void *callbackData, int panel);
void CVICALLBACK DataLogger(int menubar, int menuitem, void *callbackData, int panel);
int CVICALLBACK DetectorType(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 GainRange(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);

```

```

void CVICALLBACK InitializeInstrument(int menubar, int menuitem, void *callbackData, int panel);
int CVICALLBACK IntegrationPeriod(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK LoggerConfiguration(int panel, int control, int event, void *callbackData, int eventData1, int
eventData2);
int CVICALLBACK LoggerPanel(int panel, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK MainPanel(int panel, int event, void *callbackData, int eventData1, int eventData2);
void CVICALLBACK Measure(int menubar, int menuitem, void *callbackData, int panel);
int CVICALLBACK NumericFormat(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK NumericFormatPanel(int panel, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK NumericType(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK PositionZero(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK RealTimeMonitorPanel(int panel, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK RealTimeStripChartType(int panel, int control, int event, void *callbackData, int eventData1, int
eventData2);
int CVICALLBACK RealTimeStripPlot(int panel, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK RealTimeXYPlot(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK Responsivity(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK ResponsivityAxis(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
void CVICALLBACK SetBacklightState(int menubar, int menuitem, void *callbackData, int panel);
int CVICALLBACK SetNumericDisplayFormat(int panel, int control, int event, void *callbackData, int eventData1, int
eventData2);
int CVICALLBACK SetXRealTimeStrip(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK SetXRealTimeXY(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK SetYRealTimeStrip(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK SetYRealTimeXY(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 SumAxis(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK SumUnits(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK Trigger(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK Update(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK XYAxis(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK XYUnits(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);

```

```

#ifdef __cplusplus
}
#endif

```

### **S531 Demo.c**

```

#include <userint.h>
#include "S531 Global Variables.h"

```

```

void initializeVariables(void);

```

```

char buff[80];
char UIR[] = "S531 Demo.uir"; // uir name

```

```

int main (int argc, char *argv[])

```

```

{
    if (InitCVRTE (0, argv, 0) == 0) /* Needed if linking in external compiler; harmless otherwise */
        return -1; /* out of memory */
    if ((panelHandle[REAL_STRIP] = LoadPanel (0, UIR, REAL_STRIP)) < 0)
        return -1;
    if ((panelHandle[REAL_XY] = LoadPanel (0, UIR, REAL_XY)) < 0)
        return -1;
    if ((panelHandle[NUMERIC] = LoadPanel (0, UIR, NUMERIC)) < 0)
        return -1;
    if ((panelHandle[ALARM] = LoadPanel (0, UIR, ALARM)) < 0)
        return -1;
    if ((panelHandle[CONFIG] = LoadPanel (0, UIR, CONFIG)) < 0)
        return -1;
    if ((panelHandle[INITIAL] = LoadPanel (0, UIR, INITIAL)) < 0)
        return -1;
    if ((panelHandle[FORMAT] = LoadPanel (0, UIR, FORMAT)) < 0)
        return -1;
    if ((panelHandle[ABOUT] = LoadPanel (0, UIR, ABOUT)) < 0)
        return -1;
    if ((panelHandle[LOGGER] = LoadPanel (0, UIR, LOGGER)) < 0)
        return -1;
}

```

```

configurationMenuId = GetPanelMenuBar (panelHandle[INITIAL]);           // load menubar
digitalDisplayMenuId = GetPanelMenuBar (panelHandle[NUMERIC]);        // load menubar
stripChartMenuId = GetPanelMenuBar (panelHandle[REAL_STRIP]); // load menubar

initializeVariables();

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

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

void initializeVariables()
{
    detectorType[0] = 1;
    bias[0] = 0;
    autoRange[0] = 0;
    gain[0][0] = 7;
    gain[0][1] = 7;
    integrationPeriod[0] = 100;
    responsivityAxis[0] = 2;
    responsivity[0][0] = 1.0;
    responsivity[0][1] = 1.0;
    axisCalibration[0] = 2;
    axisCalibrationFactor[0][0] = 0.0;
    axisCalibrationFactor[0][1] = 0.0;
    positionZero[0] = 0;
    ambientZero[0] = 0;
    axis[0] = 0;
    axisUnits[0][0] = 1;
    axisUnits[0][1] = 1;
    sumAxis[0] = 0;
    sumAxisUnits[0][0] = 1;
    sumAxisUnits[0][1] = 1;
    trigger[0] = 0;
    update[0] = 1;
    stripChartMaxValue[0][0] = 100;
    stripChartMaxValue[0][1] = 100;
    stripChartMinValue[0][0] = 0;
    stripChartMinValue[0][1] = 0;

    detectorType[1] = 1;
    bias[1] = 0;
    autoRange[1] = 0;
    gain[1][0] = 7;
    gain[1][1] = 7;
    integrationPeriod[1] = 100;
    responsivityAxis[1] = 2;
    responsivity[1][0] = 1.0;
    responsivity[1][1] = 1.0;
    axisCalibration[1] = 2;
    axisCalibrationFactor[1][0] = 0.0;
    axisCalibrationFactor[1][1] = 0.0;
    positionZero[1] = 0;
    ambientZero[1] = 0;
    axis[1] = 0;
    axisUnits[1][0] = 1;
    axisUnits[1][1] = 1;
    sumAxis[1] = 0;
}

```

```

sumAxisUnits[1][0] = 1;
sumAxisUnits[1][1] = 1;
trigger[1] = 0;
update[1] = 1;
stripChartMaxValue[2][0] = 100;
stripChartMaxValue[2][1] = 100;
stripChartMinValue[2][0] = 0;
stripChartMinValue[2][1] = 0;
}

```

## About Panel.c

```

#include <userint.h>
#include "S531 Global Variables.h"

```

```

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

```

## Alarm.c

```

#include <utility.h>
#include <userint.h>
#include "S531 Global Variables.h"

```

```
void setControlState(void);
```

```

int CVICALLBACK AlarmPanel (int panel, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_CLOSE:
            HidePanel(panelHandle[ALARM]);
            break;
    }
    return 0;
}

```

```

int CVICALLBACK Alarm (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    double value;

    switch (event)
    {
        case EVENT_COMMIT:
            switch(control)
            {
                case ALARM_OK:
                    SavePanelState (panelHandle[ALARM], "Alarm.uir", 0);
                    HidePanel(panelHandle[ALARM]);
                    break;

                case ALARM_CANCEL:
                    HidePanel(panelHandle[ALARM]);
                    break;

                case ALARM_ALARM:
                    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM, &alarmState);
                    setControlState();
                    break;

                case ALARM_ALARM_BEEP:
                    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM_BEEP, &beepState);

```

```

setControlState();
break;

case ALARM_ALARM_POS_X_HIGH:
    GetCtrlVal(panelHandle[ALARM],ALARM_ALARM_POS_X_HIGH, &value);
    if (value > alarmPosition[X][LOW])
    {
        alarmPosition[X][HIGH] = value;
    }
    else
    {
        Beep ();
        SetCtrlVal(panelHandle[ALARM],ALARM_ALARM_POS_X_HIGH, alarmPosition[X][HIGH]);
    }
    break;

case ALARM_ALARM_POS_X_LOW:
    GetCtrlVal(panelHandle[ALARM],ALARM_ALARM_POS_X_LOW, &value);
    if (value < alarmPosition[X][HIGH])
    {
        alarmPosition[X][LOW] = value;
    }
    else
    {
        Beep ();
        SetCtrlVal(panelHandle[ALARM],ALARM_ALARM_POS_X_LOW, alarmPosition[X][LOW]);
    }
    break;

case ALARM_ALARM_POS_Y_HIGH:
    GetCtrlVal(panelHandle[ALARM],ALARM_ALARM_POS_Y_HIGH, &value);
    if (value > alarmPosition[Y][LOW])
    {
        alarmPosition[Y][HIGH] = value;
    }
    else
    {
        Beep ();
        SetCtrlVal(panelHandle[ALARM],ALARM_ALARM_POS_Y_HIGH, alarmPosition[Y][HIGH]);
    }
    break;

case ALARM_ALARM_POS_Y_LOW:
    GetCtrlVal(panelHandle[ALARM],ALARM_ALARM_POS_Y_LOW, &value);
    if (value < alarmPosition[Y][HIGH])
    {
        alarmPosition[Y][LOW] = value;
    }
    else
    {
        Beep ();
        SetCtrlVal(panelHandle[ALARM],ALARM_ALARM_POS_Y_LOW, alarmPosition[Y][LOW]);
    }
    break;

case ALARM_ALARM_SUM_X_HIGH:
    GetCtrlVal(panelHandle[ALARM],ALARM_ALARM_SUM_X_HIGH, &value);
    if (value > alarmSum[X][LOW])
    {
        alarmSum[X][HIGH] = value;
    }
    else
    {
        Beep ();
        SetCtrlVal(panelHandle[ALARM],ALARM_ALARM_SUM_X_HIGH, alarmSum[X][HIGH]);
    }
    break;

case ALARM_ALARM_SUM_X_LOW:
    GetCtrlVal(panelHandle[ALARM],ALARM_ALARM_SUM_X_LOW, &value);

```

```

        if (value < alarmSum[X][HIGH])
        {
            alarmSum[X][LOW] = value;
        }
        else
        {
            Beep ();
            SetCtrlVal(panelHandle[ALARM],ALARM_ALARM_SUM_X_LOW, alarmSum[X][LOW]);
        }
        break;

    case ALARM_ALARM_SUM_Y_HIGH:
        GetCtrlVal(panelHandle[ALARM],ALARM_ALARM_SUM_Y_HIGH, &value);
        if (value > alarmSum[Y][LOW])
        {
            alarmSum[Y][HIGH] = value;
        }
        else
        {
            Beep ();
            SetCtrlVal(panelHandle[ALARM],ALARM_ALARM_SUM_Y_HIGH, alarmSum[Y][HIGH]);
        }
        break;

    case ALARM_ALARM_SUM_Y_LOW:
        GetCtrlVal(panelHandle[ALARM],ALARM_ALARM_SUM_Y_LOW, &value);
        if (value < alarmSum[Y][HIGH])
        {
            alarmSum[Y][LOW] = value;
        }
        else
        {
            Beep ();
            SetCtrlVal(panelHandle[ALARM],ALARM_ALARM_SUM_Y_LOW, alarmSum[Y][LOW]);
        }
        break;
    }
    break;
}
return 0;
}

void setControlState(void)
{
    SetCtrlAttribute(panelHandle[ALARM], ALARM_TEXTMSG_5, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_TEXTMSG, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_TEXTMSG_2, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_TEXTMSG_3, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_TEXTMSG_4, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_ALARM_POS_X_HIGH, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_ALARM_POS_X_LOW, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_ALARM_POS_Y_HIGH, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_ALARM_POS_Y_LOW, ATTR_DIMMED, !alarmState);

    SetCtrlAttribute(panelHandle[ALARM], ALARM_TEXTMSG_10, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_TEXTMSG_8, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_TEXTMSG_9, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_TEXTMSG_6, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_TEXTMSG_7, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_ALARM_SUM_X_HIGH, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_ALARM_SUM_X_LOW, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_ALARM_SUM_Y_HIGH, ATTR_DIMMED, !alarmState);
    SetCtrlAttribute(panelHandle[ALARM], ALARM_ALARM_SUM_Y_LOW, ATTR_DIMMED, !alarmState);

    SetCtrlAttribute(panelHandle[ALARM], ALARM_ALARM_BEEP, ATTR_DIMMED, !alarmState);

    SetCtrlAttribute(panelHandle[ALARM], ALARM_FREQUENCY, ATTR_DIMMED, !(alarmState && beepState));
}

```

## Configuration Panel.c

```

#include <userint.h>
#include "S531 Global Variables.h"

#define X 0
#define Y 1

void setPanelState(int);

// displays configuration parameters panel.
void CVICALLBACK ConfigurationParameters (int menuBar, int menuItem, void *callbackData, int panel)
{
    DisplayPanel(panelHandle[CONFIG]);          // display panel
}

// events generated from the configuration parameters panel.
int CVICALLBACK ConfigurationPanel (int panel, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_CLOSE:
            HidePanel(panelHandle[CONFIG]);      // hide panel
            break;
    }
    return 0;
}

// set board number.
int CVICALLBACK Board (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[CONFIG], CONFIG_BOARD, &board); // get current board
            udt531_set_board (s531Id, board);
            setPanelState(--board);
            break;
    }
    return 0;
}

// set the detector type
int CVICALLBACK DetectorType (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[CONFIG], CONFIG_DET, &detectorType[board]); // get current detector type
            udt531_set_detector (s531Id, detectorType[board]);
            break;
    }
    return 0;
}

// set the bias on/off
int CVICALLBACK Bias (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[CONFIG], CONFIG_BIAS, &bias[board]); // get current detector bias state
            udt531_set_bias (s531Id, bias[board]);
            break;
    }
    return 0;
}

// set the auto range
int CVICALLBACK GainRange (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)

```

```

{
case EVENT_COMMIT:
switch(control)
{
case CONFIG_AUTORANGE:
GetCtrlVal (panelHandle[CONFIG], CONFIG_AUTORANGE, &autoRange[board]);// get auto range value
if(autoRange[board] == 0 || autoRange[board] == 1 || autoRange[board] == 2)
{
SetCtrlAttribute (panelHandle[CONFIG], CONFIG_GAIN, ATTR_DIMMED, TRUE);
}
else
{
SetCtrlAttribute (panelHandle[CONFIG], CONFIG_GAIN, ATTR_DIMMED, FALSE);
switch (autoRange[board])
{
case 4:
SetCtrlVal (panelHandle[CONFIG], CONFIG_GAIN, gain[board][X]);// get auto range value
break;

case 5:
SetCtrlVal (panelHandle[CONFIG], CONFIG_GAIN, gain[board][Y]);// get auto range value
break;
}
}
break;

case CONFIG_GAIN:
switch (autoRange[board])
{
case 4:
GetCtrlVal (panelHandle[CONFIG], CONFIG_GAIN, &gain[board][X]);// get auto range value
break;

case 5:
GetCtrlVal (panelHandle[CONFIG], CONFIG_GAIN, &gain[board][Y]);// get auto range value
break;
}
break;
}
switch (autoRange[board])
{
case 5:
udt531_set_gain (s531Id, autoRange[board], gain[board][Y]);
break;

default:
udt531_set_gain (s531Id, autoRange[board], gain[board][X]);
break;
}
break;
}
return 0;
}

// set the integration period
int CVICALLBACK IntegrationPeriod (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
switch (event)
{
case EVENT_COMMIT:
GetCtrlVal (panelHandle[CONFIG], CONFIG_INTEG, &integrationPeriod[board]);// get integration period
udt531_set_integration (s531Id, integrationPeriod[board]);
break;
}
return 0;
}

int CVICALLBACK ResponsivityAxis (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
switch (event)

```

```

{
  case EVENT_COMMIT:
    GetCtrlVal (panelHandle[CONFIG], CONFIG_RESP_AXIS, &responsivityAxis[board]); // get responsivity axis
    switch (responsivityAxis[board])
    {
      case 0:
        SetCtrlVal(panelHandle[CONFIG], CONFIG_RESP, responsivity[board][X]);
        break;

      case 1:
        SetCtrlVal(panelHandle[CONFIG], CONFIG_RESP, responsivity[board][Y]);
        break;

      default:
        SetCtrlVal(panelHandle[CONFIG], CONFIG_RESP, responsivity[board][X]);
        break;
    }
    break;
}
return 0;
}

int CVICALLBACK Responsivity (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
  switch (event)
  {
    case EVENT_COMMIT:
      switch (responsivityAxis[board])
      {
        case 0:
          GetCtrlVal (panelHandle[CONFIG], CONFIG_RESP, &responsivity[board][X]); // get responsivity for selected
axis
          udt531_set_responsivity (s531Id, responsivityAxis[board], responsivity[board][X]);
          break;

        case 1:
          GetCtrlVal (panelHandle[CONFIG], CONFIG_RESP, &responsivity[board][Y]); // get responsivity for selected
axis
          udt531_set_responsivity (s531Id, responsivityAxis[board], responsivity[board][Y]);
          break;

        default:
          GetCtrlVal (panelHandle[CONFIG], CONFIG_RESP, &responsivity[board][X]); // get responsivity for selected
axis
          GetCtrlVal (panelHandle[CONFIG], CONFIG_RESP, &responsivity[board][Y]); // get responsivity for selected
axis
          udt531_set_responsivity (s531Id, responsivityAxis[board], responsivity[board][X]);
          udt531_set_responsivity (s531Id, responsivityAxis[board], responsivity[board][Y]);
          break;
      }
      break;
    }
  return 0;
}

int CVICALLBACK CalibrateAxis (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
  switch (event)
  {
    case EVENT_COMMIT:
      GetCtrlVal (panelHandle[CONFIG], CONFIG_CAL_AXIS, &axisCalibration[board]);
      switch(axisCalibration[board])
      {
        case 0:
          SetCtrlVal(panelHandle[CONFIG], CONFIG_CAL, axisCalibrationFactor[board][X]);
          break;

        case 1:
          SetCtrlVal(panelHandle[CONFIG], CONFIG_CAL, axisCalibrationFactor[board][Y]);
          break;
      }
    }
  }
}

```

```

        default:
            SetCtrlVal(panelHandle[CONFIG], CONFIG_CAL, axisCalibrationFactor[board][X]);
            break;
    }
    break;
}
return 0;
}

int CVICALLBACK CalibrationFactor (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            switch(axisCalibration[board])
            {
                case 0:
                    GetCtrlVal (panelHandle[CONFIG], CONFIG_CAL, &axisCalibrationFactor[board][X]);
                    udt531_set_calibration (s531Id, axisCalibration[board], axisCalibrationFactor[board][X]);
                    break;

                case 1:
                    GetCtrlVal (panelHandle[CONFIG], CONFIG_CAL, &axisCalibrationFactor[board][Y]);
                    udt531_set_calibration (s531Id, axisCalibration[board], axisCalibrationFactor[board][Y]);
                    break;

                default:
                    GetCtrlVal (panelHandle[CONFIG], CONFIG_CAL, &axisCalibrationFactor[board][X]);
                    GetCtrlVal (panelHandle[CONFIG], CONFIG_CAL, &axisCalibrationFactor[board][Y]);
                    udt531_set_calibration (s531Id, axisCalibration[board], axisCalibrationFactor[board][X]);
                    udt531_set_calibration (s531Id, axisCalibration[board], axisCalibrationFactor[board][Y]);
                    break;
            }
            break;
    }
    return 0;
}

int CVICALLBACK PositionZero (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[CONFIG], CONFIG_PZERO, &positionZero[board]);
            udt531_set_position_zero (s531Id, positionZero[board]);
            break;
    }
    return 0;
}

int CVICALLBACK AmbientZero (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[CONFIG], CONFIG_PZERO, &ambientZero[board]);
            udt531_set_ambient_zero (s531Id, ambientZero[board]);
            break;
    }
    return 0;
}

int CVICALLBACK XYAxis (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[CONFIG], CONFIG_XY_AXIS, &axis[board]);
            switch(axis[board])

```

```

    {
        case 0:
            SetCtrlVal(panelHandle[CONFIG], CONFIG_XY_UNITS, axisUnits[board][X]);
            break;

        case 1:
            SetCtrlVal(panelHandle[CONFIG], CONFIG_XY_UNITS, axisUnits[board][Y]);
            break;
    }
    break;
}
return 0;
}

int CVICALLBACK XYUnits (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[CONFIG], CONFIG_XY_UNITS, &axisUnits[board][axis[board]]);
            switch(axis[board])
            {
                case 0:
                    GetCtrlVal (panelHandle[CONFIG], CONFIG_XY_UNITS, &axisUnits[board][X]);
                    udt531_set_axis_units (s531Id, axis[board], axisUnits[board][X]);
                    break;

                case 1:
                    GetCtrlVal (panelHandle[CONFIG], CONFIG_XY_UNITS, &axisUnits[board][Y]);
                    udt531_set_axis_units (s531Id, axis[board], axisUnits[board][Y]);
                    break;
            }
            break;
    }
    return 0;
}

int CVICALLBACK SumAxis (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[CONFIG], CONFIG_SUM_AXIS, &sumAxis[board]);
            switch(sumAxis[board])
            {
                case 0:
                    SetCtrlVal(panelHandle[CONFIG], CONFIG_SUM_UNITS, sumAxisUnits[board][X]);
                    break;

                case 1:
                    SetCtrlVal(panelHandle[CONFIG], CONFIG_SUM_UNITS, sumAxisUnits[board][Y]);
                    break;
            }
            break;
    }
    return 0;
}

int CVICALLBACK SumUnits (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[CONFIG], CONFIG_SUM_UNITS, &sumAxisUnits[board][sumAxis[board]]);
            udt531_set_sum_units (s531Id, sumAxis[board], sumAxisUnits[board][sumAxis[board]]);
            switch(sumAxis[board])
            {
                case 0:
                    GetCtrlVal (panelHandle[CONFIG], CONFIG_SUM_UNITS, &sumAxisUnits[board][X]);
                    udt531_set_sum_units (s531Id, sumAxis[board], sumAxisUnits[board][X]);

```

```

        break;

    case 1:
        GetCtrlVal (panelHandle[CONFIG], CONFIG_SUM_UNITS, &sumAxisUnits[board][Y]);
        udt531_set_sum_units (s531Id, sumAxis[board], sumAxisUnits[board][Y]);
        break;
    }
    break;
}
return 0;
}

int CVICALLBACK Trigger (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[CONFIG], CONFIG_TRIG, &trigger[board]);
            udt531_set_trigger (s531Id, trigger[board]);
            break;
    }
    return 0;
}

int CVICALLBACK Update (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[CONFIG], CONFIG_UPDATE, &update[board]);
            udt531_set_update (s531Id, update[board]);
            break;
    }
    return 0;
}

void setPanelState(int board)
{
    SetCtrlVal(panelHandle[CONFIG], CONFIG_DET, detectorType[board]);
    SetCtrlVal(panelHandle[CONFIG], CONFIG_BIAS, bias[board]);
    SetCtrlVal(panelHandle[CONFIG], CONFIG_AUTORANGE, autoRange[board]);
    SetCtrlAttribute(panelHandle[CONFIG], CONFIG_GAIN, ATTR_DIMMED, (autoRange[board] < 3));

    switch (autoRange[board])
    {
        case 4:
            SetCtrlVal(panelHandle[CONFIG], CONFIG_GAIN, gain[board][X]);
            break;

        case 5:
            SetCtrlVal(panelHandle[CONFIG], CONFIG_GAIN, gain[board][Y]);
            break;
    }

    SetCtrlVal(panelHandle[CONFIG], CONFIG_INTEG, integrationPeriod[board]);
    SetCtrlVal(panelHandle[CONFIG], CONFIG_RESP_AXIS, responsivityAxis[board]);

    switch (responsivityAxis[board])
    {
        case 0:
            SetCtrlVal(panelHandle[CONFIG], CONFIG_RESP, responsivity[board][X]);
            break;

        case 1:
            SetCtrlVal(panelHandle[CONFIG], CONFIG_RESP, responsivity[board][Y]);
            break;
    }
}

```

```

    default:
        SetCtrlVal(panelHandle[CONFIG], CONFIG_RESP, responsivity[board][X]);
        break;
}

SetCtrlVal(panelHandle[CONFIG], CONFIG_CAL_AXIS, axisCalibration[board]);

switch(axisCalibration[board])
{
    case 0:
        SetCtrlVal(panelHandle[CONFIG], CONFIG_CAL, axisCalibrationFactor[board][X]);
        break;

    case 1:
        SetCtrlVal(panelHandle[CONFIG], CONFIG_CAL, axisCalibrationFactor[board][Y]);
        break;

    default:
        SetCtrlVal(panelHandle[CONFIG], CONFIG_CAL, axisCalibrationFactor[board][X]);
        break;
}

SetCtrlVal(panelHandle[CONFIG], CONFIG_PZERO, positionZero[board]);
SetCtrlVal(panelHandle[CONFIG], CONFIG_AZERO, ambientZero[board]);
SetCtrlVal(panelHandle[CONFIG], CONFIG_XY_AXIS, axis[board]);

switch(axis[board])
{
    case 0:
        SetCtrlVal(panelHandle[CONFIG], CONFIG_XY_UNITS, axisUnits[board][X]);
        break;

    case 1:
        SetCtrlVal(panelHandle[CONFIG], CONFIG_XY_UNITS, axisUnits[board][Y]);
        break;
}

SetCtrlVal(panelHandle[CONFIG], CONFIG_SUM_AXIS, sumAxis[board]);

switch(axis[board])
{
    case 0:
        SetCtrlVal(panelHandle[CONFIG], CONFIG_SUM_UNITS, sumAxisUnits[board][X]);
        break;

    case 1:
        SetCtrlVal(panelHandle[CONFIG], CONFIG_SUM_UNITS, sumAxisUnits[board][Y]);
        break;
}

SetCtrlVal(panelHandle[CONFIG], CONFIG_TRIG, trigger[board]);
SetCtrlVal(panelHandle[CONFIG], CONFIG_UPDATE, update[board]);
}

```

## Continuous Measurements.c

```

#include "toolbox.h"
#include <utility.h>
#include <formatio.h>
#include "S531 Global Variables.h"

#define X_POSITION 0
#define Y_POSITION 1
#define X_SUM 2
#define Y_SUM 3

void ContinuousMeasurement (void)
{
    int status[13];

```

```

intfileId;                                /* id assigned to an open file */

double value1;
double value2;
double value3;
double value4;
double delay;

char buff[300];

if (logData == 1 && appendToFile == 0)
{
    fileId = OpenFile (fileName, VAL_WRITE_ONLY, VAL_TRUNCATE, VAL_ASCII);// create file
    if(fileId == -1)// if file could not be created display message
        MessagePopup ("File Error !", "Could not open file !");
    else
    {
        CloseFile(fileId);
    }
}

delay = (integrationPeriod[board] / 1000.0) + .1;
GetCtrlVal (panelHandle[INITIAL], INITIAL_SET_DISPLAY, &displayMode);

while (stopMeasurements == OFF)           // continue loop until stop measurements is encountered
{
    value1 = 0;
    value2 = 0;
    value3 = 0;
    value4 = 0;
    status[0] = 0;
    ProcessSystemEvents ();

    switch (displayMode)
    {
        case 0:// strip chart
            udt531_read_data (s531Id, X_POSITION, &value1, &status[0]);
            Delay(delay);
            udt531_read_data (s531Id, X_SUM, &value2, &status[0]);

            switch(stripChartType[X])
            {
                case 0://position
                    SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_READING_X, value1);
                    PlotStripChartPoint (panelHandle[REAL_STRIP], REAL_STRIP_XSTRIP, value1);
                    if(alarmState == 1)
                    {
                        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_ALARM_MAX_X, (value1 >
alarmPosition[X][HIGH]));
                        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_ALARM_MIN_X, (value1 <
alarmPosition[X][LOW]));
                        if(beepState == 1 && (value1 > alarmPosition[X][HIGH] || value1 < alarmPosition[X][LOW]))
                        {
                            StartPCSound (beepFrequency);
                            Delay(.1);
                            StopPCSound ();
                        }
                    }
                }
                break;

                case 2://sum
                    SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_READING_X, value2);
                    PlotStripChartPoint (panelHandle[REAL_STRIP], REAL_STRIP_XSTRIP, value2);
                    if(alarmState == 1)
                    {
                        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_ALARM_MAX_X, (value2 >
alarmSum[X][HIGH]));
                        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_ALARM_MIN_X, (value2 < alarmSum[X][LOW]));
                    }
                }
            }
    }
}

```

```

        if(beepState == 1 && (value2 > alarmSum[X][HIGH] || value2 < alarmSum[X][LOW]))
        {
            StartPCSound (beepFrequency);
            Delay(.1);
            StopPCSound ();
        }
    }
    break;
}

SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_O_X, OFF);
SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_P_X, OFF);
SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_N_X, OFF);
SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_U_X, OFF);
switch(status[0])
{
    case 1:
        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_O_X, ON);
        break;

    case 2:
        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_P_X, ON);
        break;

    case 3:
        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_N_X, ON);
        break;

    case 4:
        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_U_X, ON);
        break;
}

udt531_read_status (s531Id, status);
SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_GAIN_X, status[3]);

udt531_read_data (s531Id, Y_POSITION, &value3, &status[0]);
Delay(delay);
udt531_read_data (s531Id, Y_SUM, &value4, &status[0]);

switch(stripChartType[Y])
{
    case 0://position
        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_READING_Y, value3);
        PlotStripChartPoint (panelHandle[REAL_STRIP], REAL_STRIP_YSTRIP, value3);
        if(alarmState == 1)
        {
            SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_ALARM_MAX_Y, (value3 >
alarmPosition[Y][HIGH]));
            SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_ALARM_MIN_Y, (value3 <
alarmPosition[Y][LOW]));
            if(beepState == 1 && (value3 > alarmPosition[Y][HIGH] || value3 < alarmPosition[Y][LOW]))
            {
                StartPCSound (beepFrequency);
                Delay(.1);
                StopPCSound ();
            }
        }
        break;

    case 2://sum
        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_READING_Y, value4);
        PlotStripChartPoint (panelHandle[REAL_STRIP], REAL_STRIP_YSTRIP, value4);
        if(alarmState == 1)
        {
            SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_ALARM_MAX_Y, (value4 >
alarmSum[Y][HIGH]));
            SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_ALARM_MIN_Y, (value4 < alarmSum[Y][LOW]));
            if(beepState == 1 && (value4 > alarmSum[Y][HIGH] || value4 < alarmSum[Y][LOW]))

```

```

        {
            StartPCSound (beepFrequency);
            Delay(.1);
            StopPCSound ();
        }
    }
    break;
}
SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_O_Y, OFF);
SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_P_Y, OFF);
SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_N_Y, OFF);
SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_U_Y, OFF);
switch(status[0])
{
    case 1:
        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_O_Y, ON);
        break;

    case 2:
        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_P_Y, ON);
        break;

    case 3:
        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_N_Y, ON);
        break;

    case 4:
        SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_LED_U_Y, ON);
        break;
}
udt531_read_status (s531Id, status);
SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_GAIN_Y, status[3]);
Delay(.1);
break;

case 1:// numeric
udt531_read_data (s531Id, X_POSITION, &value1, &status[0]);
SetCtrlVal(panelHandle[NUMERIC], NUMERIC_X_POSITION, value1);
if(alarmState == 1)
{
    SetCtrlVal(panelHandle[NUMERIC], NUMERIC_ALARM_POS_X_HIGH, (value1 > alarmPosition[X][HIGH]));
    SetCtrlVal(panelHandle[NUMERIC], NUMERIC_ALARM_POS_X_LOW, (value1 < alarmPosition[X][LOW]));
    if((beepState == 1 && (value1 > alarmPosition[X][HIGH] || value1 < alarmPosition[X][LOW]))
    {
        StartPCSound (beepFrequency);
        Delay(.1);
        StopPCSound ();
    }
}
}
Delay(delay);

udt531_read_data (s531Id, X_SUM, &value2, &status[0]);
if(alarmState == 1)
{
    SetCtrlVal(panelHandle[NUMERIC], NUMERIC_ALARM_SUM_X_HIGH, (value2 > alarmSum[X][HIGH]));
    SetCtrlVal(panelHandle[NUMERIC], NUMERIC_ALARM_SUM_X_LOW, (value2 < alarmSum[X][LOW]));
    if((beepState == 1 && (value2 > alarmSum[X][HIGH] || value2 < alarmSum[X][LOW]))
    {
        StartPCSound (beepFrequency);
        Delay(.1);
        StopPCSound ();
    }
}
}

SetCtrlVal(panelHandle[NUMERIC], NUMERIC_X_SUM, value2);

SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_O_X, OFF);
SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_P_X, OFF);
SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_N_X, OFF);
SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_U_X, OFF);

```

```

switch(status[0])
{
    case 1:
        SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_O_X, ON);
        break;

    case 2:
        SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_P_X, ON);
        break;

    case 3:
        SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_N_X, ON);
        break;

    case 4:
        SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_U_X, ON);
        break;
}

udt531_read_status (s531Id, status);
SetCtrlVal(panelHandle[NUMERIC], NUMERIC_GAIN_X, status[3]);
Delay(delay);

udt531_read_data (s531Id, Y_POSITION, &value3, &status[0]);
SetCtrlVal(panelHandle[NUMERIC], NUMERIC_Y_POSITION, value3);
if(alarmState == 1)
{
    SetCtrlVal(panelHandle[NUMERIC], NUMERIC_ALARM_POS_Y_HIGH, (value3 > alarmPosition[Y][HIGH]));
    SetCtrlVal(panelHandle[NUMERIC], NUMERIC_ALARM_POS_Y_LOW, (value3 < alarmPosition[Y][LOW]));
    if(beepState == 1 && (value3 > alarmPosition[Y][HIGH] || value3 < alarmPosition[Y][LOW]))
    {
        StartPCSound (beepFrequency);
        Delay(.1);
        StopPCSound ();
    }
}
Delay(delay);

udt531_read_data (s531Id, Y_SUM, &value4, &status[0]);
SetCtrlVal(panelHandle[NUMERIC], NUMERIC_Y_SUM, value4);
if(alarmState == 1)
{
    SetCtrlVal(panelHandle[NUMERIC], NUMERIC_ALARM_SUM_Y_HIGH, (value4 > alarmSum[Y][HIGH]));
    SetCtrlVal(panelHandle[NUMERIC], NUMERIC_ALARM_SUM_Y_LOW, (value4 < alarmSum[Y][LOW]));
    if(beepState == 1 && (value4 > alarmSum[Y][HIGH] || value4 < alarmSum[Y][LOW]))
    {
        StartPCSound (beepFrequency);
        Delay(.1);
        StopPCSound ();
    }
}

SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_O_Y, OFF);
SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_P_Y, OFF);
SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_N_Y, OFF);
SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_U_Y, OFF);
switch(status[0])
{
    case 1:
        SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_O_Y, ON);
        break;

    case 2:
        SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_P_Y, ON);
        break;

    case 3:
        SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_N_Y, ON);
        break;
}

```

```

        case 4:
            SetCtrlVal(panelHandle[NUMERIC], NUMERIC_LED_U_Y, ON);
            break;
    }

    udt531_read_status (s531Id, status);
    SetCtrlVal(panelHandle[NUMERIC], NUMERIC_GAIN_Y, status[3]);
    Delay(delay);
    break;

case 3:// xy plot
    udt531_read_data (s531Id, X_POSITION, &value1, &status[0]);
    Delay(delay);
    udt531_read_data (s531Id, X_SUM, &value2, &status[0]);
    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_SUM_X, value2);

    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_O_X, OFF);
    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_P_X, OFF);
    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_N_X, OFF);
    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_U_X, OFF);
    switch(status[0])
    {
        case 1:
            SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_O_X, ON);
            break;

        case 2:
            SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_P_X, ON);
            break;

        case 3:
            SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_N_X, ON);
            break;

        case 4:
            SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_U_X, ON);
            break;
    }
    udt531_read_status (s531Id, status);
    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_GAIN_X, status[3]);
    Delay(delay);

    udt531_read_data (s531Id, Y_POSITION, &value3, &status[0]);
    Delay(delay);
    udt531_read_data (s531Id, Y_SUM, &value4, &status[0]);
    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_SUM_Y, value4);

    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_O_Y, OFF);
    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_P_Y, OFF);
    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_N_Y, OFF);
    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_U_Y, OFF);
    switch(status[0])
    {
        case 1:
            SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_O_Y, ON);
            break;

        case 2:
            SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_P_Y, ON);
            break;

        case 3:
            SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_N_Y, ON);
            break;

        case 4:
            SetCtrlVal(panelHandle[REAL_XY], REAL_XY_LED_U_Y, ON);
            break;
    }
    udt531_read_status (s531Id, status);

```

```

    SetCtrlVal(panelHandle[REAL_XY], REAL_XY_GAIN_Y, status[3]);

    PlotPoint (panelHandle[REAL_XY], REAL_XY_XY, value1, value3, VAL_EMPTY_SQUARE, VAL_WHITE);
    Delay(.1);
    break;
}
if (logData == 1)
{
    fileld = OpenFile (fileName, VAL_WRITE_ONLY, VAL_APPEND, VAL_ASCII);/* create file */
    if(fileld == -1) /* if history file could not be created display message */
        MessagePopup ("File Error !", "Could not open file !");
    else
    {
        struct tm *localTime;
        time_t calendarTime;

        time (&calendarTime);
        localTime = localtime (&calendarTime);

        Fmt(buf, "%s<%i[w2p0]:%i[w2p0]:%i[w2p0],%f,%f,%f,%f", localTime->tm_hour, localTime->tm_min, localTime->tm_sec, value1, value2, value3, value4); // format reading

        WriteLine (fileld, buf, StringLength(buf)); // write reading
        CloseFile(fileld);
    }
}
}
}
}

```

## Digital Panel.c

```

#include <userint.h>
#include "S531 Demo.h"
#include "S531 Global Variables.h"

int CVICALLBACK RealTimeMonitorPanel (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 SetNumericDisplayFormat (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    int value;
    int precision;

    switch (event)
    {
        case EVENT_LEFT_DOUBLE_CLICK:
            GetCtrlAttribute (panelHandle[NUMERIC], control, ATTR_FORMAT, &value);
            switch(value)
            {
                case VAL_FLOATING_PT_FORMAT:
                    SetCtrlVal (panelHandle[FORMAT], FORMAT_FLOAT, ON);
                    SetCtrlVal (panelHandle[FORMAT], FORMAT_SCIENTIFIC, OFF);
                    break;
                case VAL_SCIENTIFIC_FORMAT:
                    SetCtrlVal (panelHandle[FORMAT], FORMAT_SCIENTIFIC, ON);
                    SetCtrlVal (panelHandle[FORMAT], FORMAT_FLOAT, OFF);
                    break;
            }
    }
}

```

```

    }
    GetCtrlAttribute (panelHandle[NUMERIC], control, ATTR_PRECISION, &precision);
    SetCtrlVal (panelHandle[FORMAT], FORMAT_NUMERIC, precision);

    SetPanelPos (panelHandle[FORMAT], VAL_AUTO_CENTER, VAL_AUTO_CENTER);// set the panel position
    DisplayPanel(panelHandle[FORMAT]);// display panel

    controlId = control;
    break;
}
return 0;
}

```

## Logger Configuration Panel

```

#include <ansi_c.h>
#include <userint.h>
#include "S531 Global Variables.h"

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

int CVICALLBACK LoggerConfiguration (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    size_t length;

    switch (event)
    {
        case EVENT_COMMIT:
            switch(control)
            {
                case LOGGER_SAVE:
                    GetCtrlVal(panel, LOGGER_FILE_NAME, fileName);
                    GetCtrlVal(panel, LOGGER_OVERWRITE_FILE, &appendToFile);
                    length = strlen (fileName);
                    if (length > 3)
                    {
                        SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_DATA_LOGGER_START, ATTR_DIMMED,
FALSE);
                        SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_DATA_LOGGER_STAR, ATTR_DIMMED,
FALSE);
                    }
                    break;
            }

            break;
    }
    HidePanel(panel);
    return 0;
}

```

## Menu Bar.c

```

#include "S531 Demo.h"
#include <userint.h>
#include <utility.h>
#include "S531 Global Variables.h"

void ContinuousMeasurement(void);

```

```

void CVICALLBACK About (int menuBar, int menuItem, void *callbackData, int panel)
{
    SetCtrlVal (panelHandle[ABOUT], ABOUT_TEXT, "Model S531 Data Aquisition\n");
    SetCtrlVal (panelHandle[ABOUT], ABOUT_TEXT, "Version 2.0.1.0\n");
    SetCtrlVal (panelHandle[ABOUT], ABOUT_TEXT, "Release Date June 26, 2005\n");
    SetCtrlVal (panelHandle[ABOUT], ABOUT_TEXT, "\n\n");
    SetCtrlVal (panelHandle[ABOUT], ABOUT_TEXT, "By Steven Savrda\n");
    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]); // display panel
}

void CVICALLBACK ConfigureAlarm (int menuBar, int menuItem, void *callbackData, int panel)
{
    RecallPanelState (panelHandle[ALARM], "Alarm.uir", 0);
    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM, &alarmState);
    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM_BEEP, &beepState);
    GetCtrlVal(panelHandle[ALARM], ALARM_FREQUENCY, &beepFrequency);

    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM_POS_X_HIGH, &alarmPosition[X][HIGH]);
    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM_POS_X_LOW, &alarmPosition[X][LOW]);
    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM_POS_Y_HIGH, &alarmPosition[Y][HIGH]);
    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM_POS_Y_LOW, &alarmPosition[Y][LOW]);
    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM_SUM_X_HIGH, &alarmSum[X][HIGH]);
    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM_SUM_X_LOW, &alarmSum[X][LOW]);
    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM_SUM_Y_HIGH, &alarmSum[Y][HIGH]);
    GetCtrlVal(panelHandle[ALARM], ALARM_ALARM_SUM_Y_LOW, &alarmSum[Y][LOW]);

    setControlState();
    DisplayPanel(panelHandle[ALARM]);
}

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

    udt531_close (s531Id);
    Delay(1);
    GetCtrlVal(panelHandle[INITIAL], INITIAL_ADDR, &IEEEAddress);
    error = udt531_init (IEEEAddress, RESET_ON, &s531Id);
    if (error == 0)
    {
        SetCtrlVal (panelHandle[INITIAL], INITIAL_LED, 1);
        SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_BACKLIGHT, ATTR_DIMMED, FALSE);
        SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_ALARM, ATTR_DIMMED, FALSE);
        SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_MEAS_PARMS, ATTR_DIMMED, FALSE);
        SetMenuBarAttribute (configurationMenuId, BAR_MEASURE, ATTR_DIMMED, FALSE);
        SetMenuBarAttribute (configurationMenuId, BAR_CONFIG_INITIALIZE, ATTR_DIMMED, TRUE);
    }
    else
    {
        MessagePopup ("Initialization Error!", "Failed to initialize the Model S531 instrument. Aborting application.");
        udt531_close (s531Id);
        QuitUserInterface (0);
    }
}

void CVICALLBACK Exit (int menuBar, int menuItem, void *callbackData, int panel)
{
    stopMeasurements = ON;
    udt531_close (s531Id);
    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, lvalue);
SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_BACKLIGHT,ATTR_CHECKED, lvalue);
SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_BACKLIGHT,ATTR_CHECKED, lvalue);

udt531_set_backlight (s531Id, lvalue);
}

void CVICALLBACK DataLogger (int menuBar, int menuItem, void *callbackData, int panel)
{
    if (displayMode == 0)
    {
        switch(menuItem)
        {
            case BAR2_CONFIG_DATA_LOGGER_START:
                logData = 1;
                SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_DATA_LOGGER_START, ATTR_DIMMED,
TRUE);
                SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_DATA_LOGGER_STOP, ATTR_DIMMED,
FALSE);
                SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_DATA_LOGGER_CONFIGUR, ATTR_DIMMED,
TRUE);
                break;

            case BAR2_CONFIG_DATA_LOGGER_STOP:
                logData = 0;
                SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_DATA_LOGGER_START, ATTR_DIMMED,
FALSE);
                SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_DATA_LOGGER_STOP, ATTR_DIMMED,
TRUE);
                SetMenuBarAttribute (digitalDisplayMenuId, BAR2_CONFIG_DATA_LOGGER_CONFIGUR, ATTR_DIMMED,
FALSE);
                break;

            case BAR2_CONFIG_DATA_LOGGER_CONFIGUR:
                SetCtrlVal(panelHandle[LOGGER], LOGGER_FILE_NAME, fileName);
                SetCtrlVal(panelHandle[LOGGER], LOGGER_OVERWRITE_FILE, appendToFile);
                DisplayPanel (panelHandle[LOGGER]);
                break;
        }
    }
    else
    {
        switch(menuItem)
        {
            case REALTIME_CONFIG_DATA_LOGGER_STAR:
                logData = 1;
                SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_DATA_LOGGER_STAR, ATTR_DIMMED,
TRUE);
                SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_DATA_LOGGER_STOP, ATTR_DIMMED,
FALSE);
                SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_DATA_LOGGER_CONF, ATTR_DIMMED,
TRUE);
                break;

            case REALTIME_CONFIG_DATA_LOGGER_STOP:
                logData = 0;
                SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_DATA_LOGGER_STAR, ATTR_DIMMED,
FALSE);
                SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_DATA_LOGGER_STOP, ATTR_DIMMED,
TRUE);
                SetMenuBarAttribute (stripChartMenuId, REALTIME_CONFIG_DATA_LOGGER_CONF, ATTR_DIMMED,
FALSE);
                break;

            case REALTIME_CONFIG_DATA_LOGGER_CONF:
                SetCtrlVal(panelHandle[LOGGER], LOGGER_FILE_NAME, fileName);
                SetCtrlVal(panelHandle[LOGGER], LOGGER_OVERWRITE_FILE, appendToFile);
                DisplayPanel (panelHandle[LOGGER]);
                break;
        }
    }
}

```

```

    }
}

void CVICALLBACK Erase (int menuBar, int menuItem, void *callbackData, int panel)
{
    switch (displayMode)
    {
        case 0:// strip
            ClearStripChart (panel, REAL_STRIP_XSTRIP);
            ClearStripChart (panel, REAL_STRIP_YSTRIP);
            break;

        case 3:// xy
            DeleteGraphPlot (panel, REAL_XY_XY, -1, VAL_IMMEDIATE_DRAW);
            break;
    }
}

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);

    switch (displayMode)
    {
        case 0:
            DisplayPanel(panelHandle[REAL_STRIP]);
            break;
        case 1:
            DisplayPanel(panelHandle[NUMERIC]);
            break;
        case 3:
            DisplayPanel(panelHandle[REAL_XY]);
            break;
    }
}

void CVICALLBACK Start (int menuBar, int menuItem, void *callbackData, int panel)
{
    stopMeasurements = OFF;
    udt531_start_stop (s531Id, ON);

    switch (displayMode)
    {
        case 1:
            SetMenuBarAttribute (digitalDisplayMenuId, BAR2_STOP, ATTR_DIMMED, FALSE);
            SetMenuBarAttribute (digitalDisplayMenuId, BAR2_START, ATTR_DIMMED, TRUE);
            break;

        default:
            SetMenuBarAttribute (stripChartMenuId, REALTIME_STOP, ATTR_DIMMED, FALSE);
            SetMenuBarAttribute (stripChartMenuId, REALTIME_START, ATTR_DIMMED, TRUE);
            SetMenuBarAttribute (stripChartMenuId, REALTIME_ERASE, ATTR_DIMMED, TRUE);
            break;
    }
    ContinuousMeasurement();
}

void CVICALLBACK Stop (int menuBar, int menuItem, void *callbackData, int panel)
{
    stopMeasurements = ON;
    udt531_start_stop (s531Id, OFF);
}

```

```

switch (displayMode)
{
    case 1:
        SetMenuBarAttribute (digitalDisplayMenuId, BAR2_STOP, ATTR_DIMMED, TRUE);
        SetMenuBarAttribute (digitalDisplayMenuId, BAR2_START, ATTR_DIMMED, FALSE);
        break;

    default:
        SetMenuBarAttribute (stripChartMenuId, REALTIME_STOP, ATTR_DIMMED, TRUE);
        SetMenuBarAttribute (stripChartMenuId, REALTIME_START, ATTR_DIMMED, FALSE);
        SetMenuBarAttribute (stripChartMenuId, REALTIME_ERASE, ATTR_DIMMED, FALSE);
        break;
}
}

```

## Numeric Format Panel

```
#include "S531 Global Variables.h"
```

```

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

```

```

int CVICALLBACK NumericFormat (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    int formatType;
    int precision;

    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal (panelHandle[FORMAT], FORMAT_FLOAT, &formatType);
            switch(formatType)
            {
                case 0:
                    SetCtrlAttribute (panelHandle[NUMERIC], controlId, ATTR_FORMAT, VAL_SCIENTIFIC_FORMAT);
                    break;
                case 1:
                    SetCtrlAttribute (panelHandle[NUMERIC], controlId, ATTR_FORMAT, VAL_FLOATING_PT_FORMAT);
                    break;
            }

            GetCtrlVal(panelHandle[FORMAT], FORMAT_NUMERIC, &precision);
            SetCtrlAttribute (panelHandle[NUMERIC], controlId, ATTR_PRECISION, precision);

            HidePanel(panel);// hide panel
            break;
    }
    return 0;
}

```

```

int CVICALLBACK NumericType (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    int value;
    switch (event)
    {
        case EVENT_COMMIT:
            switch(control)
            {
                case FORMAT_FLOAT:
                    GetCtrlVal (panelHandle[FORMAT], FORMAT_FLOAT, &value);
                    SetCtrlVal (panelHandle[FORMAT], FORMAT_SCIENTIFIC, !value);
                    break;
            }
    }
}

```

```

        case FORMAT_Scientific:
            GetCtrlVal (panelHandle[FORMAT], FORMAT_Scientific, &value);
            SetCtrlVal (panelHandle[FORMAT], FORMAT_FLOAT, !value);
            break;
    }
    break;
}
return 0;
}

```

## Strip Plot Panel

```

#include <userint.h>
#include "S531 Demo.h"
#include "S531 Global Variables.h"

```

```

int CVICALLBACK RealTimeStripPlot (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 RealTimeStripChartType (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            switch(control)
            {
                case REAL_STRIP_TYPE_X:
                    GetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_TYPE_X, &stripChartType[X]);
                    SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_MAX_X, stripChartMaxValue[stripChartType[X]][X]);
                    SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_MIN_X, stripChartMinValue[stripChartType[X]][X]);
                    SetAxisRange (panelHandle[REAL_STRIP], REAL_STRIP_XSTRIP, VAL_NO_CHANGE, 0.0, 1.0,
                    VAL_MANUAL, stripChartMinValue[stripChartType[X]][X], stripChartMaxValue[stripChartType[X]][X]);
                    break;

                case REAL_STRIP_TYPE_Y:
                    GetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_TYPE_Y, &stripChartType[Y]);
                    SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_MAX_Y, stripChartMaxValue[stripChartType[Y]][Y]);
                    SetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_MIN_Y, stripChartMinValue[stripChartType[Y]][Y]);
                    SetAxisRange (panelHandle[REAL_STRIP], REAL_STRIP_YSTRIP, VAL_NO_CHANGE, 0.0, 1.0,
                    VAL_MANUAL, stripChartMinValue[stripChartType[Y]][Y], stripChartMaxValue[stripChartType[Y]][Y]);
                    break;
            }
            break;
    }
    return 0;
}

```

```

int CVICALLBACK SetXRealTimeStrip (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_MAX_X, &stripChartMaxValue[stripChartType[X]][X]);
            GetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_MIN_X, &stripChartMinValue[stripChartType[X]][X]);
            SetAxisRange (panelHandle[REAL_STRIP], REAL_STRIP_XSTRIP, VAL_NO_CHANGE, 0.0, 1.0,
            VAL_MANUAL, stripChartMinValue[stripChartType[X]][X], stripChartMaxValue[stripChartType[X]][X]);
            break;
    }
}

```

```

return 0;
}

```

```

int CVICALLBACK SetYRealTimeStrip (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_MAX_Y, &stripChartMaxValue[stripChartType[Y]][Y]);
            GetCtrlVal(panelHandle[REAL_STRIP], REAL_STRIP_MIN_Y, &stripChartMinValue[stripChartType[Y]][Y]);
            SetAxisRange (panelHandle[REAL_STRIP], REAL_STRIP_YSTRIP, VAL_NO_CHANGE, 0.0, 1.0,
            VAL_MANUAL, stripChartMinValue[stripChartType[Y]][Y], stripChartMaxValue[stripChartType[Y]][Y]);
            break;
    }
    return 0;
}

```

## XY Plot Panel

```
#include "S531 Global Variables.h"
```

```

int CVICALLBACK RealTimeXYPlot (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 SetXRealTimeXY (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    double min;
    double max;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal(panel, REAL_XY_MIN_X, &min);
            GetCtrlVal(panel, REAL_XY_MAX_X, &max);
            if (min < max)
                SetAxisScalingMode (panel, REAL_XY_XY, VAL_BOTTOM_XAXIS, VAL_MANUAL, min, max);
            break;
    }
    return 0;
}

```

```

int CVICALLBACK SetYRealTimeXY (int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    double min;
    double max;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal(panel, REAL_XY_MIN_Y, &min);
            GetCtrlVal(panel, REAL_XY_MAX_Y, &max);
            if (min < max)
                SetAxisScalingMode (panel, REAL_XY_XY, VAL_LEFT_YAXIS, VAL_MANUAL, min, max);
            break;
    }
    return 0;
}

```

## S531 Global Variables

```

#include "UDT531.h"
#include "S531 Demo.h"

#define TRUE 1
#define FALSE 0
#define ON 1
#define OFF 0
#define RESET_ON 1
#define RESET_OFF 0
#define X 0
#define Y 1
#define HIGH 1
#define LOW 0
#define POSITION 0
#define SUM 1

void setControlState(void);

int panelHandle[10]; // array of panel handles
int configurationMenuId;
int digitalDisplayMenuId;
int stripChartMenuId;
int controlId;
int stripChartType[2];
double stripChartMaxValue[3][2];
double stripChartMinValue[3][2];

int s531Id;
int parentPanel;
int stopMeasurements;
int displayMode;
int appendToFile;
int logData;
char fileName[255];
int realTimeStripChartType[2];

int board; // the current board that is to be configured
int detectorType[2]; // the current detector type
int bias[2]; // the current state of the detector bias
int autoRange[2]; // the current state of the auto range value
int gain[2][2]; // the manual gain level
int integrationPeriod[2]; // the integration period (ms)
int responsivityAxis[2];
int axisCalibration[2];
int positionZero[2];
int ambientZero[2];
int axis[2];
int axisUnits[2][2];
int sumAxis[2];
int sumAxisUnits[2][2];
int trigger[2];
int update[2];
double responsivity[2][2];
double axisCalibrationFactor[2][2];

int alarmState;
int beepState;
int beepFrequency;
double alarmPosition[2][2];
double alarmSum[2][2];

```

## **7. Index of UDT 531 Device Dependent Commands**

A, 10  
AX, 10  
AY, 10  
B, 7  
C, 17  
D, 8  
E, 8  
EX, 8  
EY, 8  
Fx, 16  
FX, 16  
Fy, 16  
FY, 16  
G, 15  
H, 15  
I, 11  
J, 15  
L0, 13  
L1, 13  
LE, 14  
LL, 14  
LR, 13  
LX, 16  
LY, 16  
MX, 10  
MY, 10  
N, 8  
Q, 12  
QX, 12  
QY, 12  
R, 9  
RX, 9  
RY, 9  
S, 18  
T, 14  
USX, 12  
USY, 12  
UX, 11  
UY, 11  
V, 10  
Z, 13  
ZX, 13  
ZY, 13