DAQBench/X

Measurement and Automation Widgets for Linux X Window System

User's Guide

Manual Rev. 1.00: September 28, 2000

Trademarks

Getting Service from ADLINK

Customer Satisfaction is always the most important thing for ADLink Tech Inc. If you need any help or service, please contact us and get it.

ADLINK Technology Inc.			
Web Site	http://www.adlink.com.tw http://www.adlinktechnology.tw		
Sales & Service	service@adlink.com.tw		
Technical	NuDAQ	nudaq@adlink.com.tw	
Support	NuDAM	nudam@adlink.com.tw	
	NuIPC	nuipc@adlink.com.tw	
	NuPRO	nupro@adlink.com.tw	
	Software	sw@adlink.com.tw	
	AMB	amb@adlink.com.tw	
TEL	+886-2-82265877	FAX	+886-2-82265717
Address	9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan, R.O.C.		

Please inform or FAX us of your detailed information for a prompt, satisfactory and constant service.

Detailed Company Information			
Company/Organization			
Contact Person			
E-mail Address			
Address			
Country			
TEL		FAX	
Web Site			

Questions	
Product Model	
Environment to Use	☐OS: _____ ☐Computer Brand: _____ ☐M/B: ☐CPU: ☐Chipset: ☐Bios: ☐Video Card: ☐Network Interface Card: ☐Other:
Challenge Description	
Suggestions for ADLINK	

Contents

How to Use This Guide

This manual is designed to help you use the DAQBench/X software package for developing your measurement or automation applications. The manual describes how to install and use the software to meet your requirements and help you program your own software applications. It is organized as follows.

The DAQBench/X *User's Guide* is organized as follows:

- Chapter 1, "installation", describes how to install the software.
- Chapter 2, "Introducing the Widgets of DAQBench/X", simply describes all ActiveX controls of DAQBench; explains the individual controls, their object structure and different style of control.
- Chapter 3, "How to use DAQBench/x", describes how you can use DAQBench/X and introduce some function that you might use when you use DAQBench/X Widgets.
- Chapter 4, "Sample Programs" describes the sample programs in the software diskette.

1

Installation

1.1 Decompress DAQBench/X

Decompress DAQBench/X through a process that lasts approximately one minute.

1.1.1 System Requirements

- An IBM PC/AT or compatibles, running Linux Kernel 2.2.x and X window
- A hard disk with enough disk space to install DAQBench/X
- A CD-ROM drive or 1.44-MB, 3.5-inch floppy disk drive
- Application development system
- Using the appropriate C/C++ compiler (gcc or cc) to compile the program.

1.1.2 UNPACK

Decompress the DAQBench_x.tgz:

```
tar xvzf DAQBench_X.tgz
```

This will extract the 'DaqBench_X' directory with following subdirectories:

File/Sub-directory	Description
Widgets <DIR>	the files of widgets
Samples <DIR>	the sample programs for DAQBench/X
Docs <DIR>	documentations

2

Introducing the Widgets of DAQBench/X

2.1 XdbBoolean Widget

XdbBoolean widget is an UI component for operating Boolean functions. The maximum bit of the XdbBoolean is 32. It can be used to indicate the Boolean data like the LED signal. It also can be used to control the bit state of data like the switch. So, the XdbBoolean widget is very convenient to be used as the display of digital input and the control of digital output at data acquisition operation.

Pattern style


Square Button


Square Radio Button


Square Push Button


LED Button Round


Push Button


Round Button


Toggle Switch


Switch


Slide Switch

2.2 XdbSlide Widget

The XdbSlide widget represents different types of linear displays, such as the variant slide, thermometers and tank display. With XdbSlide w idget, users can input or output (display) individual or multiple scalar values. A XdbSlide can have multiple pointers (maximum eight) on the widget, Each pointer represents one scalar value.

Pattern style

Wide horizon slide

Wide vertical slide

Narrow horizon slide

Narrow vertical slide

Tank

Thermometer

2.3 XdbKnob Widget

The XdbKnob widget represents different types of circular displays, such as the knob, dial and different type of meters. With XdbKnob widget, users can input or output (display) individual or multiple scalar values. A XdbKnob can have multiple pointers (maximum eight) on the widget, Each pointer represents one scalar value.

Pattern style

Knob

Dial

Upper meter

Down meter

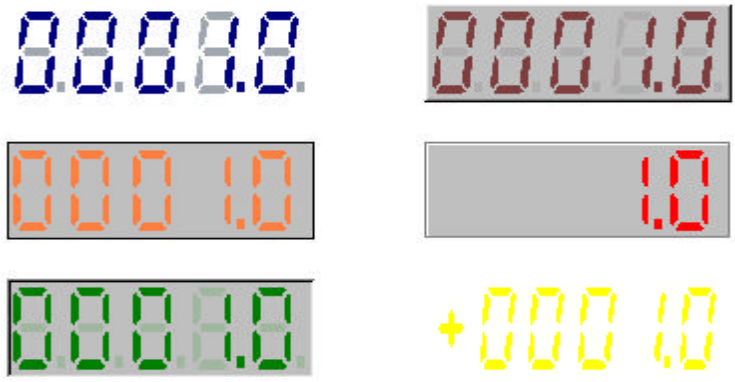
Left meter

Right meter

2.4 XdbSSegment Widget

XdbSSegnment widget is an UI component for display number using style of seven segment display. Users can configure the arguments of widget to specify the digit number, declined, digit number after point, color of segment, transparent and signed, etc.

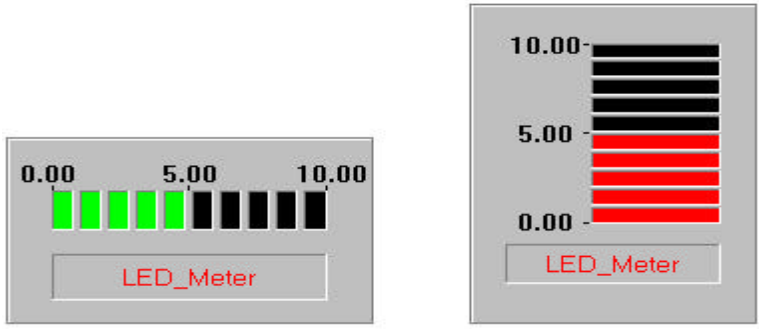
Pattern style



2.5 XdbLEDMeter Widget

XdbLEDMeter widget is an UI component for display number using style of LED Bar display. Users can configure the argument of widget to specify the bar number, direction, bar color, ticks, max value and min value, etc.

Pattern style



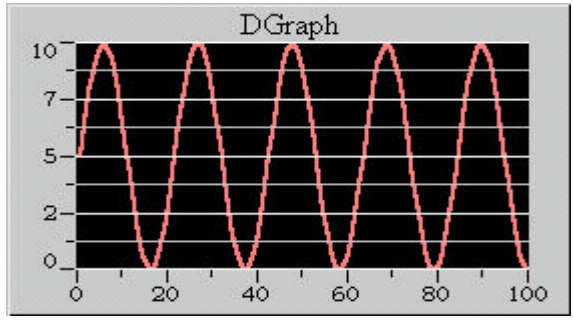
2.6 XdbGraph Widget

The XdbGraph widget is a flexible widget used for plotting data. It can display multiple plots (maximum eight plots). Plotting data refers to the process of taking a large number of points and updating one or more plots on the graph with new data.

The XAxis represents the input data points at horizon scale. Users can set the **`ViewNumber`** resource to specify the XdbGraph object how many data points will display on plot window. The XAxis object can display the time domain scale when the scale format is "Date" or "Time". The XAxis object includes one Ticks object that will process different style of ticks color, ticks mark and ticks label.
The YAxis represents the value of data points at vertical scale. Users can set the maximum and minimum resource to specify the XdbGraph object has the display range at plot window. The YAxis object has many scale formats to display scale label. The YAxis object includes one Ticks object that will process different style of ticks color ,ticks mark and ticks label.

The XdbGraph widget includes eight plots. Users can specify the resource of each plot object that includes line style, line width, pointer style, fill style, line color, fill color, pointer color, interpolation type.

Example



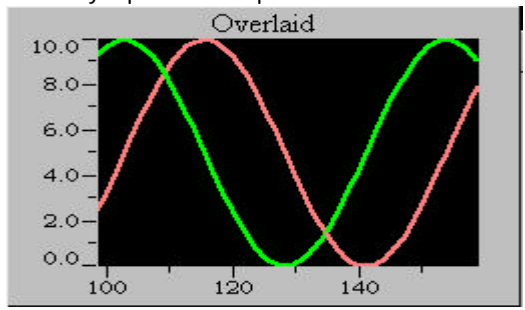
2.7 XdbChart Widget

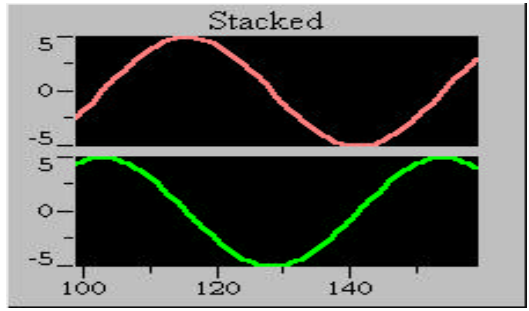
The XdbDChart widget is a flexible widget used for charting data. It can display multiple plots (maximum eight plots). Charting data appends new data points to an existing plot over time. Charting is used with slow processes where only few data points per second are added to the graph.

The XAxis represents the input data points at horizon scale. Users can set the **ViewNumber** resource to specify how many data points will display on plot window. The XAxis object can display the time domain scale when the scale format is "Date" or "Time". The XAxis object includes one Ticks object that will process different style of ticks color, ticks mark and ticks label.

The YAxis represents the value of data points at vertical scale. Users can set the **Maximum** and **Minimum** resource to specify the display range at plot window. The YAxis object has many scale formats to display scale label. The YAxis object includes one Ticks object that will process different style of ticks color, ticks mark and ticks label.
The XdbChart widget includes eight plots. Users can specify the resource of each plot object that includes line style, line width, pointer style, fill style, line color, fill color, pointer color, interpolation type.

Users can set the **PlotMode** argument of XdbChart to "Overlaid" or "Stacked" to specify different type for multiple plot data. The **UpdateMode** resource of XdbChart can determinate different update method while the charting data would be continuously input and the plot window would be scrolling.





3

How to use DAQBench/X

3.1 About the Widgets

When you want to use the widget of DAQBench/X in your program, you need to include its header file and link its object file.

Widget	Header File	Object File
XdbBoolean	Boolean.h	Boolean.o
XdbSlide	Slide.h	Slide.o
XdbKnob	Knob.h	Knob.o
XdbSSegment	SSegment.h	SSegment.o
XdbLEDMeter	LEDMeter.h	LEDMeter.o
XdbChart	Chart.h	Chart.o
XdbGraph	Graph.h	Graph.o

3.1.1 Creating a Widget

Creating a widget is a three-step process. First, the widget instance is allocated, and various instance-specific attributes are set by using **XtCreateWidget**. Second, the widget's parent is informed of the new child by using **XtManageChild**. Finally, X windows are created for the parent and all its children by using **XtRealizeWidget** and specifying the top-most widget.

The first two steps can be combined by using **XtCreateManagedWidget**. In addition, **XtRealizeWidget** is automatically called when the child becomes managed if the parent is already realized.

To allocate, initialize, and manage a widget, use **XtCreateManagedWidget**.

Widget XtCreateManagedWidget(*name*, *widget_class*, *parent*, *args*, *num_args*)

String *name*;

WidgetClass *widget_class*;

Widget *parent*;

ArgList *args*;

Cardinal *num_args*;

name Specifies the instance name for the created widget that is used for retrieving widget resources.

widget_class Specifies the widget class pointer for the created widget.

parent Specifies the parent widget ID.

args Specifies the argument list. The argument list is a variable-length list composed of name and value pairs that contain information pertaining to the specific widget instance being created.

num_args Specifies the number of arguments in the argument list. If the num_args is zero, the argument list is never referenced.

When a widget instance is successfully created, the widget identifier is returned to the application.

If an error is encountered, the **XtError** routine is invoked to inform the user of the error.

For further information, see X Toolkit Intrinsics — C Language Interface.

3.1.2 Retrieving Widget Resource Values

To retrieve the current value of a resource attribute associated with a widget instance, use **XtGetValues**.

void XtGetValues(w, *args*, *num_args*)

Widget w;

ArgList *args*;

Cardinal *num_args*;

w Specifies the widget.

args Specifies a variable-length argument list of name and **address** pairs that contain the resource name and the address into which the resource value is stored.

num_args Specifies the number of arguments in the argument list. The arguments and values passed in the argument list are dependent on the widget.

Note that the caller is responsible for providing space into which the returned resource value is copied; the **ArgList** contains a pointer to this storage (e.g. x and y must be allocated as Position).

For further information, see the X Toolkit Intrinsics — C Language Interface.

3.1.3 Modifying Widget Resource Values

To modify the current value of a resource attribute associated with a widget instance, use **XtSetValues**.

void XtSetValues(w, *args*, *num_args*)

Widget w;

ArgList *args*;

Cardinal *num_args*;

w Specifies the widget.

args Specifies an array of name and **value** pairs that contain the arguments to be modified and their new values.

num_args Specifies the number of arguments in the argument list. The arguments and values that are passed will depend on the widget being modified.

Some widgets may not allow certain resources to be modified after the widget instance has been created or realized. No notification is given if any part of a **XtSetValues** request is ignored.

For further information about these functions, see the *X Toolkit Intrinsics — C Language Interface*.

Note:

The argument list entry for XtGetValues specifies the address to which the caller wants the value copied. The argument list entry for XtSetValues, however, contains the new value itself, if the size of value is less than sizeof(XtArgVal) (architecture dependent, but at least sizeof(long)); otherwise, it is a pointer to the value. String resources are always passed as pointers, regardless of the length of the string.

3.1.4 Using the Client Callback Interface

Widgets can communicate changes in their state to their clients by means of a callback facility. The format for a client' s callback handler is:

void *CallbackProc*(w, *client_data*, *call_data*)

Widget w;

XtPointer *client_data*;

XtPointer *call_data*;

w Specifies widget for which the callback is registered.

client_data Specifies arbitrary client-supplied data that the widget should pass back to the client when the widget executes the client' s callback procedure. This is a way for the client registering the callback to also register client-specific data: a pointer to additional information about the widget, a reason for invoking the callback, and so on. If no additional information is necessary, NULL may be passed as this argument. This field is also frequently known as the *closure*.

call_data Specifies any callback-specific data the widget wants to pass to the client. For example, when Scrollbar executes its **jumpProc** callback list, it passes the current position of the thumb in *call_data*.

Callbacks can be registered either by creating an argument containing the callback list described below or by using the special convenience routines **XtAddCallback** and **XtAddCallbacks**. When the widget is created, a pointer to a list of callback procedure and data pairs can be passed in the argument list to **XtCreateWidget**. The list is of type **XtCallbackList**:

```
typedef struct {
    XtCallbackProc callback;
    XtPointer closure;
} XtCallbackRec, *XtCallbackList;
```

The callback list must be allocated and initialized before calling **XtCreateWidget**. The end of the list is identified by an entry containing NULL in callback and closure. Once the widget is created, the client can change or de-allocate this list; the widget itself makes no further reference to it. The closure field contains the client_data passed to the callback when the callback list is executed.

The second method for registering callbacks is to use **XtAddCallback** after the widget has been created.

void XtAddCallback(w, *callback_name*, *callbac*k, *client_dat*a)

Widget w;
String *callback_nam*e;
XtCallbackProc *callbac*k;
XtPointer *client_dat*a;

w Specifies the widget to add the callback to.
callback_name Specifies the callback list within the widget to append to.
callback Specifies the callback procedure to add.
client_data Specifies the data to be passed to the callback when it is invoked.

XtAddCallback adds the specified callback to the list for the named widget.

All widgets provide a callback list named **destroyCallback** where clients can register procedures that are to be executed when the widget is destroyed. The destroy callbacks are executed when the widget or an ancestor is destroyed. The *call_data* argument is unused for destroy callbacks.

4

Sample Programs

4.1 Sample Programs Included

There are several sample programs provided in this DAQBench/X. They can help you to program your own applications by using DAQBench/X easily. The brief descriptions of these programs are specified as follows:

S_XdbBoolean	An application using XdbBoolean Widget. GNU C program
S_XdbSlide	An application using XdbSlide Widget. GNU C program
S_XdbKnob	An application using XdbKnob Widget. GNU C program
S_XdbSSegment	An application using XdbSSegment Widget. GNU C program
S_XdbLEDMeter	An application using XdbLEDMeter Widget. GNU C program
S_XdbGraph	An application using XdbGraph Widget. GNU C program
S_XdbChart	An application using XdbChar Widget t. GNU C program

4.2 Sample Programs Developed Environment

We provide seven GNU C sample programs in this package. By default, they are located in directory DAQBENCH_X/SAMPLES.

You can use any editor to view or modify these source files. However, to build the executable file, you must have appropriated C/C++ compiler (gcc or cc). Please refer to related reference books to get the information about how to use C/C++ compiler (gcc or cc).