NSCL Epics support software

Ron Fox
NSCL Epics support software
by Ron Fox

Revision History
Revision 1.0 March 28, 2007 Revised by: RF
Original Release
Table of Contents

I. NSCL Epics support for Tcl/Tk (1tcl) ................................................................. v
    NSCL Epics support .............................................................................................. 6
    epics tcl package .............................................................................................. 8
    BCM Meter widget ............................................................................................ 14
    epicsButton ......................................................................................................... 16
    epicsCommandButton ......................................................................................... 18
    Epics enumerated control .................................................................................. 19
    epicsGraph ........................................................................................................ 20
    Epics Label Widget ............................................................................................ 34
    epicsLabelWithUnits ......................................................................................... 35
    epicsLed ............................................................................................................. 35
    epicsMeter .......................................................................................................... 36
    epicsPullDown .................................................................................................... 37
    epicsStripChart .................................................................................................. 40
    typeNGo bound to epics ..................................................................................... 63
    epicsspinbox ....................................................................................................... 64
    Vertical meter widget ......................................................................................... 65
    LED Widget ........................................................................................................ 67
    typeNGo compound widget ............................................................................... 68
List of Examples

1. Setting TCLLIBPATH to /usr/opt/epicstcl/TclLibs ................................................................. 7
2. Adding /usr/opt/epicstcl/TclLibs to auto_path in a Tcl Script: ............................................ 8
1. Linking an epics channel to a Tcl variable ............................................................................. 11
2. Selecting the LinuxThreads thread library to prevent hangs ........................................... 12
1. Monitoring Z001F-C with a BCM Meter ............................................................................... 15
1. Using a button pair to turn on/off D125DV ......................................................................... 18
I. NSCL Epics support for Tcl/Tk (1tcl)
NSCL Epics support

Name

Intro — Overview of Epics support for Tcl/Tk at the NSCL

DESCRIPTION

NSCL has developed base support for epics access from within Tk programs or Tcl programs that are based around the Tcl event loop. The base support consists of a package called `epics`. You can use this package to directly access EPICS channels. In many cases, however, when building pure control panel applications, you will be able to accomplish your objectives by writing your application using the NSCL epics widget set.

NSCL has developed several Tcl/Tk widgets that understand how to directly interface with the EPICS control system. These widgets are built to directly understand EPICS channels, record fields and how to display them.

The Widget set consists of the following:

epicsBCMMeter
A meter with range controls that knows how to display NSCL Beam Current meters and control their ranges. This widget requires an EPICS record of a specific subset of types.

epicsButton
A pushbutton that is connected to an epics field. This normally is used to control binary I/O records.

epicsEnumeratedControl
Controls an epics field that can have a value from a set of discrete pre-defined values.

epicsGraph.xml
Creates 2-d plots of epics channels vs. each other.

epicsLabel
Displays the value of an epics record field in a Tk label widget.

epicsLabelWithUnits
Displays the value of an epics record primary field in a Tk label widget along with the value of the record’s engineering unit’s field.

epicsLed
Displays an indicator which is lit when the epics field is nonzero and not when it isn’t the on and off colors of the LED can be configured.
epicsMeter

Displays the value of an arbitrary EPICS record field in a meter.

epicsStripChart

Wraps the very flexible BLT Stripchart widget in some code that allows it to produce strip charts of the time evolution of an arbitrary number of epics fields.

epicsTypeNGo

Provides a simple type-in entry field that allows users to control the value of an arbitrary epics field. New values are committed by pressing a button.

epicsspinbox

Provides a spinbox that can control an arbitrary EPICS record field.

Getting started

The NSCL epics software is made up of several packages. A base epics package provides raw support to the channel access layer. Each widget provides a separate package as well. In Tcl, packages are loaded using the:

package require package-name

command.

The package require searches a list of library directories for matching packages. In general it will be necessary to add the directory in which the epics packages are installed to this list of directories. This can be done either by setting the TCLLIBPATH environment variable prior to running your scripts, or by adding that directory to the Tcl auto_path variable in your script.

I cannot anticipate where the NSCL epics software will be installed on all systems, however when I (Ron Fox) install this package, I install it in /usr/opt/epicstcl which will put the packages in /usr/opt/epicstcl/TclLibs.

Example 1. Setting TCLLIBPATH to /usr/opt/epicstcl/TclLibs

sh, bash shells:

export TCLLIBPATH=/usr/opt/epicstcl/TclLibs

csh:

setenv TCLLIBPATH /usr/opt/epicstcl/TclLibs
Example 2. Adding /usr/opt/epicstcl/TclLibs to auto_path in a Tcl Script:

lappend auto_path /usr/opt/epicstcl/TclLibs

On windows, no environment variables are needed. Simply

1. If not already installed, install the NCSAPPS package on your PC. This makes the EPICS channel access layer required by the epics package available.
2. Download and install the ActiveTcl package available at no charge from http://www.activestate.com Install this package in its default location.
3. Download the epics installer from the NSCL anonymous FTP site. At the time this is being written it is: ftp://ftp.nscl.msu.edu/pub/epicstcl13-001.exe
4. Run the epics installer you downloaded.

epics tcl package

**Name**
epics — Loadable package to access epics.

**Synopsis**

package require epics

epicschannel *name*

*name* get ?count?

*name* set *value-list* ?*data-type*?

*name* updatetime

*name* delete
The epics package is a loadable package that supplies access to an epics control system. Loading epics will also load the shared libraries required for epics, so these must be installed on the system on which this package is being used.

The `epicschannel` command expresses an interest in a specific named channel, or database field. Once specified, this becomes a new command. The new command is an ensemble with several subcommands. These subcommands allow one to manipulate and inquire about the channel. When interest is declared epics events are requested to maintain the state of a channel. Epics events can only be processed, however, by entering the Tcl event loop. Either by running `wish`, or by doing a `vwait` in a pure Tcl interpreter.

It is perfectly possible and acceptable to do something like:

```tcl
epicschannel aaa
...
epicschannel aaa
```

Rather than creating a second, duplicate command, the epics package maintains a reference count for each distinct epics channel created. The first `epicschannel` in the example above creates the new command, with a reference count of 1. The second increments the reference count of the existing aaa command to 2.

Having done the sequence of commands shown above:

```tcl
aaa delete;       # Decrements the refcount to 1 aaa still exists.
...
aaa delete;       # refcount becomes 0 so aaa is deleted.
```
Hopefully this reference count scheme will make large programs easier to build, as sections will not have to worry about other sections yanking existing commands out from underneath them.

The subcommands for an epics channel are:

**get size?**
Retrieve the value of the channel or field. Note that if a connection event has not yet been received and processed, this will return an error. This can happen either if the channel is not an epics channel or if the event loop has not yet been entered enough times to allow the event to be seen. Note that epics events are processed prior to executing this command so it is possible for this command to fail first and then work a few tries later.

If the channel is an array, the entire array is returned as a Tcl list unless the optional size parameter is provided. In that case, the first size elements are returned or all elements depending on which is fewer.

**set value-list data-type?**
Sets the value of the channel or database field (if changeable) to value-list. All the remarks about the set subcommand apply here too.

If the channel is an array, value-list is a list of values that will be used to set the first elements of the array. The number of elements set is the smaller of the size of the list and the number of array elements managed by the channel.

If the optional data-type keyword is present, it provides the data type to be used to do the set. The data type can be any of string (default), real or int. It is an error for value-list to contain a value that cannot be converted to the type specified.

**updatetime**
Returns the time of the last update received for the channel. The time is returned as an integer suitable for use in the Tcl clock command. This allows the result to be formatted as a time and date, or used arithmetically to calculate time differences in seconds.

**delete**
Deletes the command and attachment to an epics channel. All resources associated with the command are also destroyed.

**link varname**
Links a variable to the epics proces variable (channel). Changes to the channel get reflected into the linked variable. Changes to the variable from Tcl scripts are traced and result in attempts to modify the epics channel.
Additional links are allowed and create a 1 to many link between an epics channel and several Tcl Variables.

At present, only the first element of array process variables is linked to the Tcl variable. Array process variables must be handled via the get and set sub-commands.

**unlink varname**

Removes the link between the epics channel and the Tcl variable varname. It is an error to attempt to unlink from a variable that is not linked.

**name listlinks ?pattern?**

Lists the set of links that match the optional pattern. If no pattern is supplied, it defaults to *.

**name values**

Lists the set of values that the process variable can legally accept. If this list is empty, the channel is either not connected or has not received its first value and therefore does not yet know its list of enumerated values. If the list size is one, this will be a textual encoding of the data types acceptable by the channel e.g. float, string or int. If the list size is greater than 1, this is a list of allowed values for the enumerated variable.

**name size**

Returns the number of elements in name. Epics process variables can be thought of as arrays, where a scalar value is just the special case of an array of size 1.

**EXAMPLES**

The code below creates a label widget that follows the value of the epics channel SOMECHANNEL:

**Example 1. Linking an epics channel to a Tcl variable**

```tcl
package require epics
epicschannel SOMECHANNEL
SOMECHANNEL link SOMECHANNELvar
label .l -textvariable SOMECHANNELvar
pack .l
```

Note that this can be done much more simply using the epics Tcl widgets. Those widgets understand how to display epics channels directly e.g.

```tcl
package require epicsLabelWithUnits
```

11
controlwidget::epicsLabelWithUnits .l -channel SOMECHANNEL
pack .l

Creates a GUI that displays SOMECHANNEL with its engineering units, updating as the value updates in Epics.

The example below finds out how many elements are in the channel K5RGA_HSCANT_DAT

package require epics
epicschannel K5RGA_HSCAN_DAT...
set elements [K5RGA_HSCAN_DAT size]

The following example, takes the channel K5RGA_HSCAN_DAT, already assumed to be connected, and clears its second array element. Note that the elements of Tcl lists number from 0.

set data [K5RGA_HSCAN_DAT get 2]; # get elements 0,1.
set data [lreplace $data 1 1 0]; # Replace second item with 0.
K5RGA_HSCAN_DAT set $data; # Set elements 0,1

**OPEN ISSUES**

On some linux systems a broken implementation of the Linux Native Posix Thread Library (NPTL) causes the tcl shell extended with the epics package to deadlock (hang). This is a known issue with Linux. If this is observed then prior to starting tcl/wish applications, select the LinuxThreads implementation of the threading library by (bash):

**Example 2. Selecting the LinuxThreads thread library to prevent hangs**

export LD_ASSUME_KERNEL=2.4.19

For the C shell:
setenv LD_ASSUME_KERNEL 2.4.19
Issues with enumerated variable types

Enumerated types have an interesting interaction with array sets. This is not an defect in the software package, it is simply a property of Tcl that interacts with some enumerated types, and the ability to set enumerated types by string values. Consider an enumerated type whose string values have spaces e.g one legitimate value is "a b". Let’s call this process variable `funky1` (we will have a `funky2` to show another interesting issue with enumerated process variables. Suppose that this value "a b" corresponds to enumerated index 0. Consider the following two chunks of Tcl (`funky1` is already assumed to be established as a channel).

```tcl
funky1 set [lindex [funky1 values] 0]
funky1 set "a b"
```

Both of these provide the parameter a b to the `set` subcommand. this looks like a two element list, but `funky1` is only a single element array, so the value a is set which may not be legal, in which case epics will throw an error or, even worse, may correspond to another legal value for the enumerated type.

There are two potential solutions to this problem. First, ensure that a single element list is received by the `set` command, second, use indices only:

```tcl
funky1 set [list [lindex [funky1 values] 0]]
funky1 set [lsearch [funky1 values] "a "b] int
```

The `list` command will add an additional level of quoting if necessary to ensure that each parameter it receives is a properly quoted list element. The `lsearch` command will return the index of "a b" in the list of allowed values for `funky1`. This is an integer that represents the enumerated index value. The `int` at the back end of the command forces the `set` to be done as an integer data type rather than a string. See the discussion below about pathological enumerated process variables.

For enumerated process variables there can also be an interesting pathology. Consider a process variable `funky2` for which the `values` subcommand returns the list: 5 4 3 2 1 0. It is not clear what the following does, or even what the intent is:

```tcl
funky2 set 2
```

Is the 2 the string 2 (which has enumerated index 3), or is it the index 2 which has the string value 3? Process variable designers should avoid such pathologies. If, however, a pathology like this does exist, that would imply that the only unambiguous way to set enumerated process variables is by index. The following is unambiguous:

```tcl
funky2 set 2 int
```
This forces an integer set of the process variable which selects the textual value 3. Note that this pathology may well be hidden from the programmer, who is just using the `values` command to get the list of legal values and selecting from amongst them. The above discussion should hopefully lead you to conclude that for enumerated epics variable types, you should probably only use the textual representation, relying on the index to set the value and ensuring that the index is treated as an index by using the `int` data type parameter on the `set` sub command to ensure that pathologically labelled variables are not a problem. e.g:

```tcl
someenum set [lindex [someenum values] $index]; # Avoid this!!!
someenum set $index int; # use this instead.
```

There is a further subtlety. For linked variables, modifications of the variable triggers a set in `string` form. This avoids the vector/list issue, but steps right into the issue with pathological value sets. Therefore once more enumerated process variables, following the plan of using the text (variable) for display only, but use the `someenum set some-integer int` form for setting the variable is the best policy.

## BCM Meter widget

**Name**

`epicsBCMMeter` — Provide a widget for displaying and controlling beam current monitors.

**Synopsis**

```tcl
package require epicsBCMMeter

controlwidget::epicsBCMMeter path ?options...?
```

**OPTIONS**

- `-meterheight dimension`

Requests a specific height for the meter part of the widget. This height can be specified using any of the legal Tk dimension specifications. The value is passed to the meter widget’s `-height` option without interpretation.
-meterwidth `dimension`
Requests a specific width for the meter part of the widget. This height can be specified using any of the legal Tk dimension specifications. The value is passed to the meter widget’s `-width` option without interpretation.

-channel `name`
Specifies the `name` of the epics channel to be monitored by this meter. Note that the channel must have an MRNG, MSRN, and MRRN field in its database.

Note that the meter ranges are not exposed to the API. The widget maintains appropriate ranges and ticks depending on the value of the range of the underlying device.

The `-channel` option is required at creation time and cannot be changed later.

METHODS

get
Returns the current value of the meter’s channel.

getRange
Returns the value of the meter range. For example, `1e-06` means the meter runs between `-1e06` and `1e06`.

incRange
Increments the range of the monitor and meter. This will usually make the meter more sensitive.

decRange
Decrements the range of the monitor and meter. This will usually make the meter less sensitive.

EXAMPLES

The example below displays a BCM Meter that monitors the current on `Z001F-C`

Example 1. Monitoring Z001F-C with a BCM Meter

```tcl
package require epicsBCMMeter

controlwidget::epicsBCMMeter .meter -channel Z001F-C
pack .meter
```
SEE ALSO

meter

epicsButton

Name
epicsButton — Provide a control for on/off values in epics.

Synopsis

package require epicsButton

controlwidget::epicsButton path ?options...?

DESCRIPTION

This widget provides a mechanism for controlling binary output style devices with the ability to monitor an optional associated input status channel. Two control styles are supported, a single button and a pair of buttons. The widget can also be labeled.

OPTIONS

-channel channel-name

Specifies the name of the channel that will be controlled by the pushbutton widget. Note that unless the -statechannel option is specified, this channel will also be used to reflect the state of the device. This option must be specified when the button is created.
-statechannel channel-name
  Specifies the name of the channel that reflects the state of the device. The channel state is assumed
to be 'on' if this channel is boolean true and off otherwise. If this option is not specified when the
button is constructed, the state will be read from the channel specified by -channel.

-onvalue value
  Specifies the value to write to the channel to turn the device to the on state. If not specified, this
defaults to 1.

-offvalue value
  Specifies the value to write to the channel to turn the device to the off state. If not specified, this
defaults to 0.

-onlabel string
  Specifies the string to use to label the button that turns the device on. If a single button
representation has been selected, this string will label the button when the device is off (the button
turns the device on), and the button will display the off color. In a double button representation, this
label will label the left button, which turns the device on.

-offlabel string
  Specifies a string to use to label the button that turns the device off. If a single button representation
is selected, this string will label the button when the device is on (the button turns the device off in
that case), and the button will display the on color. In a double button representation, this label will
label the right button which turns the device off.

-oncolor color
  Specifies the color to use to indicate the device is on. In a single button case, the color is the
background color of the single button, in a double button case, this color is the background color of
the button that is enabled (when the device is on, the on button is disabled).

-offcolor color
  Specifies the color to use to indicate the device is off. See the discussion of -oncolor for a hint about
how this works.

-modality keyword
  Selects the type of button presentation desired. The keyword can have the value single or double.
Selecting whether a single button or a pair of buttons will be used to control this device.

-showlabel boolean
  If true (default) a channel name label is placed above the button(s). If false, no channel label is
displayed.
EXAMPLES

The example below (to the best of my knowledge), creates a pair of buttons that can turn the D125DV power supply on and off:

Example 1. Using a button pair to turn on/off D125DV

    package require epicsButton
    controlwidget::epicsButton .d125dvonoff -channel D125DV.ONL -statechannel D125DV.SONL \ -modality double \ -onlabel {Turn On} \ -offlabel {Turn Off} \ -oncolor green -offcolor red
    pack .d125dvonoff

epicsCommandButton

Name

    epicsCommandButton — Button that sends a value to a channel

Synopsis

    package require epicsButton
    controlwidget::epicsCommandButton path ?options?...

DESCRIPTION

    The epicsCommandButton is wraps the Tk button widget so that clicking the button sends a specific value to an associated epics process variable. The appearance defaults for the widget are the same as ordinary Tk buttons, in contrast with epicsButton widgets.
OPTIONS

The epicsCommandButton inherits all options and methods from the Tk button widget. The -command option is, however disabled to prevent interference with the epicsCommandButton’s use of this feature in the underlying button.

-channel epicschannel

epicschannel is the channel controlled by this widget. This must be supplied when constructing the widget and cannot be dynamically modified (attempts to do so are silently ignored).

-value value

value is the value that will be written to the channel when the button is clicked. This defaults to an empty string, and can be dynamically modified after the widget is created.

EXAMPLES

The example below creates a button that, when pressed sets the channel IGAI0 to zero.

```
package require epicsButton
ccontrolwidget::epicsCommandButton .eb -channel IGAI0 -value 0 -text IGAI0=>0
pack .eb
```

Epics enumerated control

Name

epicsEnumeratedControl — Provide a control for epics channels with discrete enumerable values.

Synopsis

```
package require epicsEnumeratedControl
ccontrolwidget::epicsEnumeratedControl path ?options...?```
SUMMARY

The `epicsEnumeratedControl` command provides a widget that allows you to monitor and control epics channels that can take one of a list of possible settings values. The widget is based on a `radioMatrix` widget, but the variable is bound to an epics channel.

OPTIONS

All of the options associated with a `radioMatrix` widget are accepted by the `epicsEnumeratedControl` widget except the `-variable` option. In addition, the `-channel` option can be provided to bind the matrix to an epics channel.

METHODS

The `Get` and `Set` methods work as for the `radioMatrix`.

SEE ALSO

`radioMatrix(1tcl)`

epicsGraph

**Name**

`epicsgraph` — Wrap a BLT graph with code for plotting epics channels against each other.

**Synopsis**

```
package require epicsGraph
control::epicsStripChart name ?options?
nname addseries sname x-channel y-channel interval ?options?
nname removeSeries sname
```
DESCRIPTION

This widget is a thin wrapping of the BLT Graph widget. The wrapping allows you to easily create graphs of epics channel pairs (e.g. one channel on the x axis, one on the y axis). Any number of pairs of channels can be plotted on the same widget if desired with line colors symbol shapes and line types distinguishing between them.

OPTIONS

All options for the blt::graph widget are supported and passed to that widget without any interpretation. See the summary (section blt::graph summary) below or alternatively: http://man-wiki.net/index.php/N:blt_graph for a full description of that widget.

METHODS

All blt::graph widget methods are supported. See the summary (section blt::graph summary) below or alternatively, http://man-wiki.net/index.php/N:blt_graph The blt::graph widget methods are passed without any interpretation on to that widget.

In addition the following methods are also defined:

\texttt{name addseries surname x-channel y-channel interval ?options?}

Adds a data series to the graph. A data series consists of a blt::graph element that displayse the data and channelPairHistory object to automatically maintain the element’s data.

\textit{sname} is the name of the series to create. It must be unique and will also be used as the blt::graph element name.

\textit{x-channel y-channel} are the names of the EPICS process variables that will be the X and Y parameters of the data series respectively.

\textit{interval} is the number of milliseconds between samples on the plot.

\textit{options} are optional option value pairs that are passed as is to the blt::graph element add command and can be used to configure the appearance of the data series e.g.

The command returns the name of the channel pair history object which can be saved and manipulated.
**name** removeseries **sname**

Removes the series *sname* from the graph and destroys the channel pair history object that was created for it.

### channelPairHistory objects

channelPairHistory objects are used to keep track of and manage the automatic update of data series. While intended for use with the epicsGraph widget, you may also find them useful in your applications. This section therefore summarizes the capabilities of the `channelPairHistory` `snit::type`.

#### OPTIONS

**-period**

Specifies the milliseconds between data updates

**-xchannel**

Name of the x channel. When used with an epicsGraph to produce a data series, this parameter will be on the x axis.

**-ychannel**

Name of the y channel. When used with an epicsGraph to produce a data series, this parameter will be on the y axis.

**-timebase**

[Clock seconds] at which the data series start. When the data are retrieved from the object, the times associated with each data points are offsets relative to this time.

#### METHODS

**clear**

Clears the entire data series.

**clearfirst n**

Clears the first *n* data points in the series.

**keep n**

Keeps only the first *n* data points in a series. This restriction is enforced on each update of the series.

**names**

The data series is maintained in a set of three `blt::vector` objects. This returns the names of the three vectors. The names are returned as a three element list. The first element of the list is the name
of the time vector. The second element of the list is the name of the x parameter vector. The third element of the list is the name of the y parameter vector.

```tcl
get
```

Returns the data stored in the series. The data are returned as a list of data points. Each data point is a three element list consisting of (in order), the time relative to the `-timebase` time, the x parameter value at that time, and the y parameter value at that time.

## blt::graph summary

See [http://man-wiki.net/index.php/N:blt_graph](http://man-wiki.net/index.php/N:blt_graph) for a full description of the blt::graph widget. This section provides a summary of the more useful features of the widget in an attempt to make this manpage close to self contained for most uses of the widget.

The blt::graph widget is a graph that plots X-Y data. The graph widget can be thought of as having many independently configurable components. Configuring each component can determine how the graph will appear.

### OPTIONS

- `-height measure`
  
  sets the requested height of the widget. This can be any valid Tk measurement. The default is 4i

- `-title text`
  
  Sets the title to `text`. If `text` is "", no title will be displayed.

- `-width measure`
  
  Specifies the requested width of the widget. The default is 5i.

### COMPONENTS

The graph widget can be thought of as made up of several components. Each component can be independently configured and, in some cases several components of each type can be created. This section summarizes the components and what they do. Subsequent sections will describe the most useful options of the most used components.

- `axis`
  
  The graph has four standard axes (x, x2, y, and y2), but you can create and display any number of axes. Axes control what region of data is displayed and how the data is scaled. Each axis consists of the axis line, title, major and minor ticks, and tick labels. Tick labels display the value at each major tick.
crosshairs

Cross hairs are used to position the mouse pointer relative to the X and Y coordinate axes. Two perpendicular lines, intersecting at the current location of the mouse, extend across the plotting area to the coordinate axes.

element

An element represents a set of data points. Elements can be plotted with a symbol at each data point and lines connecting the points. The appearance of the element, such as its symbol, line width, and color is configurable. Data series of epics channels are implemented as elements.

grid

Extends the major and minor ticks of the X-axis and/or Y-axis across the plotting area.

legend

The legend displays the name and symbol of each data element. The legend can be drawn in any margin or in the plotting area.

marker

Markers are used annotate or highlight areas of the graph. For example, you could use a polygon marker to fill an area under a curve, or a text marker to label a particular data point. Markers come in various forms: text strings, bitmaps, connected line segments, images, polygons, or embedded widgets.

pen

Pens define attributes (both symbol and line style) for elements. Data elements use pens to specify how they should be drawn. A data element may use many pens at once. Here, the particular pen used for a data point is determined from each element’s weight vector (see the element’s -weight and -style options).

postscript

The widget can generate encapsulated PostScript output. This component has several options to configure how the PostScript is generated.

axis

Four coordinate axes are automatically created: two X-coordinate axes (x and x2) and two Y-coordinate axes (y, and y2). By default, the axis x is located in the bottom margin, y in the left margin, x2 in the top margin, and y2 in the right margin.

An axis consists of the axis line, title, major and minor ticks, and tick labels. Major ticks are drawn at uniform intervals along the axis. Each tick is labeled with its coordinate value. Minor ticks are drawn at uniform intervals within major ticks.
The range of the axis controls what region of data is plotted. Data points outside the minimum and maximum limits of the axis are not plotted. By default, the minimum and maximum limits are determined from the data, but you can reset either limit.

You can have several axes. To create an axis, invoke the axis component and its create operation.

```tcl
# Create a new axis called "tempAxis"
g axis create tempAxis
```

You map data elements to an axis using the element’s -mapy and -mapx configuration options. They specify the coordinate axes an element is mapped onto.

```tcl
# Now map the tempAxis data to this axis.
g element create "el" -xdata $x -ydata $y -mapy tempAxis
```

Any number of axes can be displayed simultaneously. They are drawn in the margins surrounding the plotting area. The default axes x and y are drawn in the bottom and left margins. The axes x2 and y2 are drawn in top and right margins. By default, only x and y are shown. Note that the axes can have different scales.

To display a different axis or more than one axis, you invoke one of the following components: xaxis, yaxis, x2axis, and y2axis. Each component has a use operation that designates the axis (or axes) to be drawn in that corresponding margin: xaxis in the bottom, yaxis in the left, x2axis in the top, and y2axis in the right.

```tcl
# Display the axis tempAxis in the left margin.
g yaxis use tempAxis
```

The use operation takes a list of axis names as its last argument. This is the list of axes to be drawn in this margin.

You can configure axes in many ways. The axis scale can be linear or logarithmic. The values along the axis can either monotonically increase or decrease. If you need custom tick labels, you can specify a Tcl procedure to format the label any way you wish. You can control how ticks are drawn, by changing the major tick interval or the number of minor ticks. You can define non-uniform tick intervals, such as for time-series plots.
Axis components are manipulated using an ensemble of widgets commands (methods) of the form:

\[
\text{pathName axis subcommand ...}
\]

The most useful of the subcommands will be described below.

cget \text{axisName option}

Returns the current value of the option given by option for axisName. Option may be any option described below for the axis configure operation.

configure \text{axisName option value}

Sets a new value for a configuration option for the axis axisName. option and value are described in the 'most useful list of options' below:

- \text{color color}

Sets the color of the axis and tick labels. The default is black.

- \text{descending boolean}

Indicates whether the values along the axis are monotonically increasing or decreasing. If boolean is true, the axis values will be decreasing. The default is 0.

- \text{logscale boolean}

Indicates whether the scale of the axis is logarithmic or linear. If boolean is true, the axis is logarithmic. The default scale is linear.

- \text{majorticks majorlist}

Specifies where to display major axis ticks. You can use this option to display ticks at nonuniform intervals. majorList is a list of axis coordinates designating the location of major ticks. No minor ticks are drawn. If majorList is ””, major ticks will be automatically computed. The default is ””.

- \text{max value}

Sets the maximum limit of axisName. Any data point greater than value is not displayed. If value is ””, the maximum limit is calculated using the largest data value. The default is ””. Note that this calculation is performed again as data elements change in time.

- \text{min value}

Sets the minimum limit of axisName. Any data point less than value is not displayed. If value is ””, the minimum limit is calculated using the smallest data value. The default is ””.

- \text{minorticks minorList}

Specifies where to display minor axis ticks. You can use this option to display minor ticks at non-uniform intervals. MinorList is a list of real values, ranging from 0.0 to 1.0, designating the placement of a minor tick. No minor ticks are drawn if the -majortick option is also set. If minorList is ””, minor ticks will be automatically computed. The default is ””.
- `steplsize value`
  Specifies the interval between major axis ticks. If value isn’t a valid interval (must be less than the axis range), the request is ignored and the step size is automatically calculated.

- `subdivisions number`
  Indicates how many minor axis ticks are to be drawn. For example, if number is two, only one minor tick is drawn. If number is one, no minor ticks are displayed. The default is 2.

- `title text`
  Sets the title of the axis. If text is "", no axis title will be displayed.

`create axisName ?options...?`
Creates a new axis by the name `axisName`. No axis by the same name can already exist. `?options...?` are option value pairs described above under the `configure` subcommand.

`delete axisName`
Deletes the named axes. An axis is not really deleted until it is not longer in use, so it’s safe to delete axes mapped to elements.

`names ?pattern?`
Returns a list of axes matching zero or more patterns. If no `pattern` argument is give, the names of all axes are returned.

**element**

An element is what we refer to as a data series. Elements are displayed as a set of X/Y points on the surface of the graph, limited by the axes they are associated with. The points can be connected by lines that have various line styles and colors (see pen components as well).

When new data elements are created, they are automatically added to a list of displayed elements. The display list controls what elements are drawn and in what order.

The following operations are the most useful ones available for elements. All are of the form:

`pathName element subcommand ...`

`cget elemNamme option`
Returns the current value of the element configuration option given by `option`. `Option` may be any of the options described below for the element configure operation.
configure elemName ... ?option...

Queries or modifies the configuration options for elements. Several elements can be modified at the same time. If option isn’t specified, a list describing all the current options for elemName is returned. If an option is specified, but not its value, then a list describing the option option is returned. If one or more option and value pairs are specified, then for each pair, the element option option is set to value. The following options are the most commonly used ones valid for elements.

=color color

Sets the color of the traces connecting the data points.

dashes dashlist

Sets the dash style of element line. DashList is a list of up to 11 numbers that alternately represent the lengths of the dashes and gaps on the element line. Each number must be between 1 and 255. If dashList is "", the lines will be solid.

=label text

Sets the element’s label in the legend. If text is "", the element will have no entry in the legend. The default label is the element’s name.

=pen penname

Set the pen to use for this element. For more information about pens, see the pen component described later in this document.

=symbol symbol

Specifies the symbol for data points. Symbol can be either square, circle, diamond, plus, cross, splus, scross, triangle, "" (where no symbol is drawn), or a bitmap. Bitmaps are specified as "source ?mask?", where source is the name of the bitmap, and mask is the bitmap’s optional mask. The default is circle.

exists elemName

Returns 1 if an element elemName currently exists and 0 otherwise.

names ?pattern?...

Returns the elements matching one or more pattern. If no pattern is given, the names of all elements is returned.

grid

Grid lines extend from the major and minor ticks of each axis horizontally or vertically across the plotting area. While there are many options and operations associated with grid lines, the most common ones are:
grid on
  Turns on the display the grid lines.

grid off
  Turns off the display the grid lines.

legend

The legend displays a list of the data elements. Each entry consists of the element’s symbol and label. The legend can appear in any margin (the default location is in the right margin). It can also be positioned anywhere within the plotting area.

Legend operations are of the form:

`pathName legend operation ...`

The most frequently used legend operations are:

`cget option`

Returns the current value of a legend configuration option. `Option` may be any option described below in the legend configure operation.

`configure ?option...?`

Queries or modifies the configuration options for the legend. If `option` isn’t specified, a list describing the current legend options for `pathName` is returned. If `option` is specified, but not `value`, then a list describing `option` is returned. If one or more option and value pairs are specified, then for each pair, the legend option `option` is set to `value`. The following options (and others) are valid for the legend.

`-hide boolean`

Indicates whether the legend should be displayed. If `boolean` is `true`, the legend will not be drawn. The default is `no`.

`-position pos`

Specifies where the legend is drawn. The `-anchor` option also affects where the legend is positioned. If `pos` is `left`, `left`, `top`, or `bottom`, the legend is drawn in the specified margin. If `pos` is `plotarea`, then the legend is drawn inside the plotting area at a particular anchor. If `pos` is in the form "@x,y", where x and y are the window coordinates, the legend is drawn in the plotting area at the specified coordinates. The default is `right`. 
Pen

Pens define attributes (both symbol and line style) for elements. Pens mirror the configuration options of data elements that pertain to how symbols and lines are drawn. Data elements use pens to determine how they are drawn. A data element may use several pens at once. In this case, the pen used for a particular data point is determined from each element’s weight vector (see the element’s -weight and -style options).

One pen, called activeLine, is automatically created. It’s used as the default active pen for elements. So you can change the active attributes for all elements by simply reconfiguring this pen.

```tcl
.g pen configure "activeLine" -color green
```

You can create and use several pens. To create a pen, invoke the pen component and its create operation.

```tcl
.g pen create myPen
```

You map pens to a data element using either the element’s -pen or -activepen options.

```tcl
.g element create "line1" -xdata $x -ydata $tempData \
        -pen myPen
```

An element can use several pens at once. This is done by specifying the name of the pen in the element’s style list (see the -styles option).

```tcl
.g element configure "line1" -styles { myPen 2.0 3.0 }
```

This says that any data point with a weight between 2.0 and 3.0 is to be drawn using the pen myPen. All other points are drawn with the element’s default attributes.

The following operations are available for pen components, and are of the form:

```tcl
pathName pen operation ...
```
Descriptions start with the \textit{operation}.

\texttt{cget penName option}

Returns the current value of the option given by \texttt{option} for \texttt{penName}. \texttt{Option} may be any option described below for the pen \texttt{configure} operation.

\texttt{configure penName ?penName... ?option...?}

Queries or modifies the configuration options of \texttt{penName}. Several pens can be modified at once. If \texttt{option} isn’t specified, a list describing the current options for \texttt{penName} is returned. If \texttt{option} is specified, but not \texttt{value}, then a list describing \texttt{option} is returned. If one or more \texttt{option} and \texttt{value} pairs are specified, then for each pair, the pen \texttt{option} \texttt{option} is set to \texttt{value}. The following options are valid for pens.

\texttt{-color color}

Sets the color of the traces connecting the data points.

\texttt{-dashes dashList}

Sets the dash style of element line. \texttt{DashList} is a list of up to 11 numbers that alternately represent the lengths of the dashes and gaps on the element line. Each number must be between 1 and 255. If \texttt{dashList} is "", the lines will be solid.

\texttt{-symbol symbol}

Specifies the symbol for data points. Symbol can be either \texttt{square}, \texttt{circle}, \texttt{diamond}, \texttt{plus}, \texttt{cross}, \texttt{splus}, \texttt{scross}, \texttt{triangle}, "" (where no symbol is drawn), or a bitmap. Bitmaps are specified as "source ?mask?", where \texttt{source} is the name of the bitmap, and \texttt{mask} is the bitmap’s optional mask. The default is \texttt{circle}.

\texttt{create penName ?option value?...}

Creates a new pen by the name \texttt{penName}. No pen by the same name can already exist. \texttt{Option} and \texttt{value} are described in above in the pen \texttt{configure} operation.

\texttt{delete ?penName?...}

Deletes the named pens. A pen is not really deleted until it is not longer in use, so it’s safe to delete pens mapped to elements.

\texttt{names ?pattern?...}

Returns a list of pens matching zero or more patterns. If no pattern argument is give, the names of all pens are returned.

\texttt{postscript}

The graph can generate encapsulated PostScript output. There are several configuration options you can specify to control how the plot will be generated. You can change the page dimensions and borders. The
plot itself can be scaled, centered, or rotated to landscape. The PostScript output can be written directly to a file or returned through the interpreter.

Postscript operations all have the form:

\[ \text{pathName postscript operation ...} \]

The following postscript operations are available.

\text{cget option}

Returns the current value of the postscript option given by \text{option}. \text{Option} may be any option described below for the postscript \text{configure} operation.

\text{configure ?option value?...}

Queries or modifies the configuration options for PostScript generation. If \text{option} isn’t specified, a list describing the current postscript options for \text{pathName} is returned. If \text{option} is specified, but not \text{value}, then a list describing \text{option} is returned. If one or more option and value pairs are specified, then for each pair, the postscript option \text{option} is set to \text{value}. The following postscript options are available.

\text{~center boolean}

Indicates whether the plot should be centered on the PostScript page. If \text{boolean} is false, the plot will be placed in the upper left corner of the page. The default is 1 (true), which centers the plot on the postscript page.

\text{~colormode mode}

Specifies how to output color information. \text{Mode} must be either \text{color} (for full color output), \text{gray} (convert all colors to their gray-scale equivalents) or \text{mono} (convert foreground colors to black and background colors to white). The default mode is \text{color}.

\text{~landscape boolean}

If \text{boolean} is true, this specifies the printed area is to be rotated 90 degrees. In non-rotated output the X-axis of the printed area runs along the short dimension of the page ("portrait orientation"); in rotated output the X-axis runs along the long dimension of the page ("landscape orientation"). Defaults to 0.

\text{~maxpect boolean}

Indicates to scale the plot so that it fills the PostScript page. The aspect ratio of the graph is still retained. The default is 0.

\text{output ?fileName? ?option value?...}

Outputs a file of encapsulated PostScript. If a \text{fileName} argument isn’t present, the command returns the PostScript. If any option-value pairs are present, they set configuration options.
controlling how the PostScript is generated. Option and value can be anything accepted by the postscript configure operation above.

EXAMPLES

The following simple application prompts for two channels and then builds/dispalyes a plot of the two channels updated every 100ms. The plot will have axes that autorange, with a grid and axis titles that reflect the channels plotted as well as a title that reflects the plot.

```tcl
package require epicsGraph

# Build the prompt for the channels:
frame .prompt
label .prompt.cxlb1 -text X Channel
entry .prompt.cx
label .prompt.cylbl -text Y Channel
entry .prompt.cy
button .prompt.ok -text Ok -command createPlot
grid .prompt.cxlb1 .prompt.cx
grid .prompt.cylbl .prompt.cy
grid .prompt.ok
pack .prompt

# Called when the OK button is clicked.
# get the x/y channel names and create the
# plot. A lot of error checking has been
# omitted for the sake of brevity (e.g. what
# if the user does not fill in a channel?
proc createPlot {} {
    set xName [.prompt.cx get]
    set yName [.prompt.cy get]
    destroy .prompt; # Now the top level is clear.
    set seriesName "$xName_v_[$yName]"
    set title "$xName vs. $yName"

    # Set up the plot:
    controlwidget::epicsGraph .eg -title $title
    .eg grid on
    .eg legend configure -position bottom
    .eg addseries $seriesName $xName $yName 100 -color black -symbol {} \
        -label $title
```
Epics Label Widget

**Name**
epicsLabel — Provide a label widget that connects to an epics channel.

**Synopsis**
package require epicsLabel

::controlwidget::epicsLabel path ?options?...

**OPTIONS**

All options supported by the Tcl label widget are supported by this widget. You should not, however use the -textvariable option as this is used to connect the widget to the channel.

-channel name

This option is required and can only be set at construction time. it provides the name of the epics channel to which the widget will be connected.

**METHODS**

All methods supported by the Tk label widget are supported by the epicsLabel widget.
SEE ALSO

epicsLabelWithUnits

Name
epicsLabelWithUnits — Show the value of an epics channel and its units if it has any.

Synopsis

package require epicsLabelWithUnits

controlwidget::epicsLabelWithUnits path ?options?

OPTIONS

See the epicsLabel(1tcl) man page for a description of the options acceptable to this widget.

METHODS

See the epicsLabel(1tcl) man page for a description of the methods recognized by this widget.

SEE ALSO

epicsLabel(1tcl)

epicsLed

Name
epicsLed — An LED bound to an epics channel.
Synopsis

package require epicsLed

::controlwidget::epicsLed path ?options?

OPTIONS

All options recognized by the led widget are recognized by this widget. In addition, the required option:
-channel epicsPV provides the name of the epics process variable to bind to the LED. The LED will be
‘on’ if the process variable is nonzero or any textual value that tcl recognizes as boolean, or ‘off’ if not.

METHODS

All methods recognized by the led widget are supported, however it is recommended that you not call on
or off.

KNOWN ISSUES

If the on or off colors are changed, this is not reflected until the channel next changes value.

SEE ALSO

led(1tcl)

epicsMeter

Name

epicsMeter — Provide a generic meter that can display an epics channel

Synopsis

package require epicsMeter
controlwidget::epicsMeter name -channel channel ?options?

DESCRIPTION

Provides a generic meter that can display any numeric epics process variable. The meter’s normal appearance is a vertical strip of subwidgets consisting of a textual label describing the widget contents (defaults to the channel name), An epicsLabel that shows the current value and units of the channel. A meter whose indicator shows the current value of the channel.

OPTIONS

All options supported by the controlwidget::meter widget are supported by this widget except the -variable option.

-channel channel-name
Provides the name of the epics process variable (channel) to display on the meter. This must be provided at construction time and cannot be changed.

-label string
Overrides the default widget label string, which is the channel name.

EXAMPLES

The example below displays the temperature outside the NSCL in degrees F, on a meter that goes from 0 to 100 degrees with tick marks every 20 degrees:

package require epicsMeter
controlwidget::epicsMeter .temp -from 0 -to 100 -majorticks 20 -channel Ti9400
pack .temp

SEE ALSO

controlwidget::meter(1tcl)
epicsPullDown

Name

epicsPullDown — Pull down menu connected to an epics channel

Synopsis

package require epicsPullDown

controlwidget::epicsPulldown path ?options...?

DESCRIPTION

The epicsPullDown widget provides a pull down menu that connects to an epics channel. The widget adapts a Tk menubutton widget and associates a menu with the widget. The menu represents a set of possible values that can be set in the process variable connected to the widget.

While the widget can easily be used for process variables with enumerated values it is not restricted to that use. The menu button face is labeled with the current value of the process variable. If the process variable has a value that is not represented by its menu choices the raw string value of the process variable labels the button.

OPTIONS

All options that are recognized by the Tk menubutton widget are supported. The application, however should not use the -menu option as that is used to connect the widget to the menu generated by the -items option described in the list of additional options below.

In addition to all of the menubutton widgets, the widget supports the following options:

-channel name

  Specifies the name of the epics process variable to which the menu will be connected. This option must be supplied when the widget is built and cannot be dynamically modified. Selecting entries in the widget will modify the specified process variable. The button face will reflect the current value of the process variable.

-items items

  Describes the menu entries. The items value is a Tcl list. Each list element describes a single item in the menu. The menu is populated top down in the order specified by the items list.
Each item in the list can have one of the following forms:

- Inserts a separator in the menu. A separator is a horizontal line that is used to visually group related sets of items.

- \textit{labelvalue} Inserts a radio button in the menu. The radio button has the label given by the text \textit{labelvalue}. This will also be the value of the process variable associated with this item. When the menu item created is selected, the process variable will be set to \textit{labelvalue}. When the process variable is \textit{labelvalue} the menu button will be labeled \textit{labelvalue}.

- \{\textit{label value}\} A two element Tcl list that creates a new radio button in the menu. The first element (\textit{label}) provides the text that labels the button. The second, \textit{value} provides the value associated with this label. When this menu entry is selected, the process variable will be set to \textit{value}. When the process variable is equal to \textit{value} the label of the menu button will be \textit{label}.

\textbf{-tearoff} \hspace{1em} \textit{true | false}

Determines whether or not the pull down menu can be torn off into a new top level widget. If \textit{true} (the default), the menu can be torn off. If \textit{false} not. Menus that can be torn off will have a dashed line across the top of them. Clicking on that dashed line makes a new top level widget that duplicates the menu. When the menu is torn off, you can still operate the menu button and, in fact, as many menu entries as desired can be torn off.

\section*{METHODS}

All of the widget commands of the Tk menubutton widget are supported.

\section*{EXAMPLES}

The example below creates an epics pull down menu connected to IGLI0. The first three menu items are values. The fourth a separator. The final two are label value pairs:

\begin{verbatim}
package require epicsPullDown

controlwidget::epicsPullDown .pd -channel IGLI0 -tearoff true
.pd configure -items {1 2 3 - {four 4} {{five units} 5}}

pack .pd
\end{verbatim}

The epics \texttt{list} command can also be used to build up the items list. The next example produces the same result, but uses \texttt{list} and defines the menu items when the drop down is constructed.

\begin{verbatim}
package require epicsPullDown
\end{verbatim}
controlwidget::epicsPullDown .pd -channel IGLI0 \  
-Teaoff true  \ 
-Items [list 1 2 3 -  \ 
[list four 4]  \ 
[list "five units" 5]]
pack .pd

The next example shows how to build the item list automatically for an epics enumerated channel. There are two complications.

• Individual strings may have spaces in them and not be interpreted as single item entries.
• The channel may not connect immediately so you can’t build up the item list until the connection completes.

The example below deals with all of these issues:

package require epicsPullDown

proc configureItems {widget channel} {
    if {([llength [K5RGA_M_O2.SCAN values]] != 0)} {
        foreach value [K5RGA_M_O2.SCAN values] {
            lappend itemlist [list $value]
        }
        $widget configure -items $itemlist
    } else {
        # Not connected yet reschedule.
        after 100 [list configureItems $widget $channel]
    }
}

controlwidget::epicsPullDown .pd -channel K5RGA_M_O2.SCAN
configureItems .pd K5RGA_M_O2.SCAN
pack .pd

pack .pd
epicsStripChart

Name
epicsStripChart — Wrap a BLT stripchart with code for plotting epics channel time evolutions.

Synopsis

package require epicsStripChart

controlwidget::epicsStripChart name ?options?

name addchannel channel milliseconds ?options?

name removechannel channel

DESCRIPTION

This widget adds machinery to the BLT Strip chart widget to support adding epics channels to a chart. For a summary of the BLT stripchart widget, see the section BLT STRIPCHART below. For full information about the BLT stripchart widget see the online man pages at e.g. http://man-wiki.net/index.php/N:blt_stripchart.

OPTIONS

All BLT stripchart widget options are supported. For some of the non-standard useful options see BLT STRIPCHART below, or the online man page referenced in the DESCRIPTION section.
METHODS

All blt stripchart methods are supported, in addition to the ones described below. For some of the more useful blt stripchart methods, see the section BLT STRIPCHART below, or refer to the online manual pages for the stripchart widget referred to in the DESCRIPTION section.

addchannel name milliseconds ?options?

Adds the channel name to the strip chart as a new channelHistory element. The strip chart will automatically trace the channel value updating every milliseconds milliseconds. The optional list of ?options, can be any of the options accepted by the BLT stripchart widget’s element component.

The method returns a name which is both the name of the new BLT strip chart element and a command that can be used to manipulate the channelHistory object. For more information about the channelHistory object, see the section channelHistory OBJECTS below.

removechannel name

Removes the specified channel from the strip chart. Note that this is not the name of the channelHistory object returned by the addchannel method.

Removing the specified channel from the strip chart destroys the element created for the trace. It also destroys the associated channelHistory object and all resources associated with that object.

channelHistory OBJECTS

Adding a channel to the stripchart creates a new object called a channelHistory object and returns the object name to the user. This object also has methods as described in the synopsis below:

set object [.stripchart addchannel name milliseconds ?options?]

object clear

object clearfirst points

object keep points

object updateperiod milliseconds

object get
**object names**

**channelHistory OPTIONS**

These options should be treated as readonly. That is you should always `cget` them and never `configure` them.

- **-period**
  The number of milliseconds between each update of the object

- **-channel**
  The name of the EPICS channel monitored by this object

- **-timebase**
  The time relative to which historical data time offsets are measured. This is the output of a `[clock seconds]` command. Note that time offsets are floating point seconds. It is possible to use a time offset in conjunction with `[clock format]`, a format string and a bit of arithmetic to produce a timestamp for an individual data item that is exact to the millisecond at which the data was updated.

**channelHistory METHODS**

**object clear**

Clears all the historical data in the object. On the stripchart this means the trace for the channel managed by this element will vanish and then start accumulating again as time passes.

In a larger application, you could clear the entire strip chart by iterating through the channelHistory elements you created and clearing them all. Since the channelHistory element object name is the same as the stripchart element name, the BLT stripchart element names can return all the channel history objects created as long as the chart only contains epics channels.

**object clearfirst points**

Removes the first `points` points of history data from the object. See the keep method for a better way to keep the history data size under control.

**object keep points**

Requires that the history object retain at most `points` data points. After each update interval, if the history object contains more than the specified number of points, the oldest points are discarded until the correct number of points are retained.
**object** updateperiod milliseconds

Changes the update period to the milliseconds milliseconds. this takes effect after the next update period.

**object** get

Returns the historical data for the parameter logged. The historical data is returned as a Tcl list of pairs. Each pair contains a time offset (floating point seconds) from the time base of the object (See the -timebase option), and the value of the channel at that time.

This can be used to, e.g. perform analysis, logging or serialization for later re-load, of the historical data.

**object** names

The historical data are stored in two BLT vectors, a time and a data vector. This method returns a two element list consisting of the name of the time and data vectors in that order.

---

**BLT STRIPCHART**

This section summarizes the BLT stripchart widget. This is intended just to provide an overview of the most useful options. It is not intended as a complete document for that widget. Complete documentation of the BLT strip chart widget can be found online at: http://man-wiki.net/index.php/N:blt_stripchart.

The stripchart widget is very flexible it can be configured via options, methods, and components. Components are named entities that can be added to the stripchart and then manipulated via their own options and components. The following component are supported:

**axis**

Coordinate axes control the region of data displayed and how the data are scaled. Up to four axes (2 x and 2 y) can be displayed on the chart.

**crosshairs**

crosshairs can be defined to get a better idea of where the cursor is

**element**

Elements are data point sets and their attributes. Each channel added to the chart creates an element.

**grid**

The grid extends major and minor ticks across the plotting area to make it easier to read the location of points off the plot by eye.
legend

Legends display labels for the elements and their styles. Note that while the strip chart default label is the element name, the epicsStripChart addchannel command labels the element’s entry in the legend with the EPICS channel name.

marker

Markers are used to annotate or highlight areas of the graph. Many different marker types are supported including text, bitmaps, polylines, images, polygons, and embedded widgets.

pen

Pens define attributes for elements. They can be thought of as attribute bundles that can be applied in a single configuration parameter.

postscript

The postscript component allows you to save the contents of the graph in a postscript file as well as to configure the way in which that file is produced.

Key Stripchart options

This section provides a summary of a few of the more interesting, non-standard options recognized by the strip chart widget. See the online docs for complete documentation.

- halo pixels
  Specifies a maximum distance to consider when searching for the closest data point (see the element’s closest operation below). Data points further than pixels away are ignored. The default is 0.5i. This is used e.g. to process mouse hits in event bindings.

- height measure
  Specifies the requested height of widget. The default is 4i.

- invertxy boolean
  Indicates whether the placement X-axis and Y-axis should be inverted. If boolean is true, the X and Y axes are swapped. The default is 0 (unswapped).

- tile image
  Specifies a tiled background. If image isn’t ‘’, the background is tiled using image. Otherwise, the normal background color is drawn. Image must be an image created using the Tk image command. The default is ‘’.

- title text
  Sets the title to text. If text is ‘’, no title will be displayed.

- width measure
  Specifies the requested width of the widget. The default is 5i.
**Key Stripchart methods**

This section is a summary of the most useful stripchart methods. See the online manpage for the BLT Stripchart for more complete documentation. (We don’t bother to document configure and cget as these are ‘well understood’ Tk methods).

**axis operation** ...

Manipulates axis components. See the section "Stripchart components" for more information about this.

**crosshairs operation** ...

Manipulates the crosshairs component of the stripchart. See the section "Stripchart components" below for more information.

**element operation** ...

Manipulates element components. Note that channels become element components of the stripchart. See the section "Stripchart components" below for more information.

**grid operation** ...

Manipulates the grid component of the stripchart. See the section "Stripchart components" below for more information.

**invtransform winX winY**

Performs a coordinate transform that maps the point defined by (winX winY) into the graph real coordinate system. Returns the transformed X/Y coordinates.

**legend operation** ...

Manipulates the legend component of the strip chart widget. See the section "Stripchart components" below for more information.

**marker operation** ...

Manipulates marker components of the strip chart widget. See the section "Stripchart components" below for more information.

**postscript operation** ...

Manipulates the postscript snapshot component of the widget.

**transform x y**

Transforms the point (x y) specified in graph coordinates to widget coordinates.

**xaxis | x2axis | yaxis | y2axis operation** ...

Manipulates an axis component of the graph. See the section "Stripchart components" below for more information.
Stripchart components

This section describes the various stripchart components and how to create and manipulate them.

Stripchart axes

Stripchart axes. Four coordinate axes are automatically created: two X-coordinate axes (x and x2) and two Y-coordinate axes (y, and y2). By default, the axis x is located in the bottom margin, y in the left margin, x2 in the top margin, and y2 in the right margin.

An axis consists of the axis line, title, major and minor ticks, and tick labels. Major ticks are drawn at uniform intervals along the axis. Each tick is labeled with its coordinate value. Minor ticks are drawn at uniform intervals within major ticks.

The range of the axis controls what region of data is plotted. Data points outside the minimum and maximum limits of the axis are not plotted. By default, the minimum and maximum limits are determined from the data, but you can reset either limit.

You can create and use several axes. To create an axis, invoke the axis component and its create operation.

```tcl
# Create a new axis called "temperature"
.s axis create temperature
```

You map data elements to an axis using the element’s -mapy and -mapx configuration options. They specify the coordinate axes an element is mapped onto.

```tcl
# Now map the temperature data to this axis.
.s element create "temp" -xdata $x -ydata $tempData \ 
 -mapy temperature
```

While you can have many axes, only four axes can be displayed simultaneously. They are drawn in each of the margins surrounding the plotting area. The axes x and y are drawn in the bottom and left margins.
The axes x2 and y2 are drawn in top and right margins. Only x and y are shown by default. Note that the axes can have different scales.

To display a different axis, you invoke one of the following components: xaxis, yaxis, x2axis, and y2axis. The use operation designates the axis to be drawn in the corresponding margin: xaxis in the bottom, yaxis in the left, x2axis in the top, and y2axis in the right.

```tcl
# Display the axis temperature in the left margin.
.s yaxis use temperature
```

You can configure axes in many ways. The axis scale can be linear or logarithmic. The values along the axis can either monotonically increase or decrease. If you need custom tick labels, you can specify a Tcl procedure to format the label as you wish. You can control how ticks are drawn, by changing the major tick interval or the number of minor ticks. You can define non-uniform tick intervals, such as for time-series plots.

This section describes the major operations on the strip chart axis component. All of these are invoked using the form

```
.stripchart axis operation ...
```

Where we will now describe the most useful operations.

**create name ?options?**

Creates a new axis name the optional options configure the axis as per the configuration options described below.

**delete name**

Deletes an existing axis.

**invtransform name screenCoords**

Transforms screenCoords from the widget coordinate system to the axis coordinate system defined by the axis name.
names ?pattern ... 

Returns the name of all defined axes that match at least one of the patterns provided. Patterns can contain *glob* wildcard characters. If no pattern is provided the command operates as if there was a single pattern: 

transform axisName axisCoord 

Returns axisCoord transformed to widget coordinates using the transformation defined by the axis axisName

Axes can be configured using the configure subcommand and their configuration can be inquired using the cget subcommand. For example:

```
.stripchart axis configure axisName ...
```

More useful configuration parameters for an axis include:

-logscale boolean 

Indicates whether the scale of the axis is logarithmic or linear. If boolean is true, the axis is logarithmic. The default scale is linear.

-majorticks ticklist 

Specifies where to display major axis ticks. You can use this option to display ticks at non-uniform intervals. ticklist is a list of axis coordinates designating the location of major ticks. No minor ticks are drawn. If ticklist is "", major ticks will be automatically computed. The default is "".

-max value 

Sets the maximum limit of axisName. Any data point greater than value is not displayed. If value is "", the maximum limit is calculated using the largest data value. The default is "".

-min value 

Sets the minimum limit of axisName. Any data point less than value is not displayed. If value is "", the minimum limit is calculated using the smallest data value. The default is "".

-minorticks ticklist 

Specifies where to display minor axis ticks. You can use this option to display minor ticks at non-uniform intervals. ticklist is a list of real values, ranging from 0.0 to 1.0, designating the
placement of a minor tick. No minor ticks are drawn if the -majortick option is also set. If minorList is "", minor ticks will be automatically computed. The default is "".

-shiftby value

Specifies how much to automatically shift the range of the axis. When the new data exceeds the current axis maximum, the maximum is increased in increments of value. You can use this option to prevent the axis limits from being recomputed at each new time point. If value is 0.0, then no automatic shifting is done. The default is 0.0.

-title text

Sets the title of the axis. If text is "", no axis title will be displayed.

The axis positions at the bottom, top, left and right can be selected and manipulated using the xaxis xaxis1 yaxis yaxis2 commands respectively. In this document, we only describe how to select an axis for use in that position:

```
.stripchart {xaxis | xaxis1 | yaxis | yaxis1} use axisName
```

For example, to create an axis named george and display it at the top part of the graph:

```
.stripchart axis create george

.stripchart xaxis1 use george
```

**Stripchart crosshairs**

**Crosshairs.** This section describes the crosshairs component of the stripchart. Cross hairs consist of two intersecting lines (one vertical and one horizontal) drawn completely across the plotting area. They are used to position the mouse in relation to the coordinate axes. Cross hairs differ from line markers in that they are implemented using XOR drawing primitives. This means that they can be quickly drawn and erased without redrawing the entire strip chart.

There is only a single crosshair. To turn crosshairs on for the stripchart .stripchart:

```
.stripchart crosshairs on
```
Similarly to turn crosshairs off:

```
.stripchart crosshairs off
```

**Stripchart elements (traces)**

Elements are data sets that are drawn on the stripchart. In the case of the `epicsStripChart` elements are created using the `addchannel` method. It is also possible to intermix elements created ‘manually’. We will not document how to do this. You will need to read the online BLT Stripchart widget to see how to do this.

When you create an element via `addchannel` the name of the element will be returned and can be captured via e.g.:

```
set elementName [.stripchart addchannel someChannel]
```

Once this is done you can use `$elementName` wherever an element name is required to refer to that element.

Elements can be configured with various options, and their configurations queried via e.g.:

```
.stripchart element configure name options...

.stripchart element cget name option-name
```

A useful subset of the options is:
-activepen penName

Specifies pen to use to draw active element. If penName is ",", no active elements will be drawn. The default is activeLine.

-color color

Sets the color of the traces connecting the data points.

-linewidth pixels

Specifies the line width of the element in pixels. If pixels is zero, no line will be drawn, between symbols.

-mapx axisName

Selects the X-axis to map the element’s X-coordinates onto. axisName must be the name of an axis. The default is xaxis.

-mapy axisName

Selects the Y-axis to map the element’s Y-coordinates onto. axisName must be the name of an axis. The default is yaxis.

-smooth style

Specifies how connecting line segments are drawn between data points. style can be either linear, step, natural, or quadratic. If style is linear, a single line segment is drawn, connecting both data points. When style is step, two line segments are drawn. The first is a horizontal line segment which steps the next x-coordinate. The second is a vertical line, moving to the next y-coordinate. Both natural and quadratic generate multiple segments between data points. If natural, the segments are generated using a cubic spline. If quadratic, a quadratic spline is used. The default is linear.

-symbol symbol

Specifies the symbol for data points. symbol can be either square, circle, diamond, plus, cross, splus, scross, triangle, "" (where no symbol is drawn), or a bitmap. Bitmaps are specified as "source ?mask?", where source is the name of the bitmap, and mask is the bitmap’s optional mask. The default is circle.

Element are invoked using the following general form:

..stripchart element methodname ...

Where the most useful methodnames and their parameters are:

closest x y varName ?option value?... ?elemName?...

Finds the data point closest to the window coordinates x and y in the element elemName. ElemName is the name of an element, that must not be hidden. If no elements are specified, then all
visible elements are searched. It returns via the array variable varName the name of the closest
element, the index of its closest point, and the graph coordinates of the point. Returns 0, if no data
point within the threshold distance can be found, otherwise 1 is returned. The following
option-value pairs are available.

- `-halo distance`

Specifies a threshold distance where selected data points are ignored. distance is a valid screen
distance, such as 2 or 1.2i. If this option isn’t specified, then it defaults to the value of the
stripchart’s `-halo` option.

- `-interpolate boolean`

Indicates that both the data points and interpolated points along the line segment formed should
be considered. If boolean is true, the closest line segment will be selected instead of the closest
point. If this option isn’t specified, boolean defaults to 0 which is a Tcl false value.

`exists elemName`

Returns 1 if an element elemName currently exists and 0 otherwise.

`names ?pattern? ...

Returns the elements matching one or more pattern. If no pattern is given, the names of all elements
is returned. Note that if the widget is only displaying epics channels, these names are the same as
the names of the channelHistory objects that contain and maintain the data that is being plotted.

**Stripchart grids**

**Stripchart Grids.** Stripchart grids extend the tick marks on the axes across the entire face of the graph
part of the widget. grids make it easier to read a point off the graph.

Grids have configuration options that are set and gotten via e.g.

```
.stripchart grid configure options...

.stripchartgrid cget option-name
```
Grids also have methods that are invoked via e.g:

```
.stripchart grid methodname ...
```

The key configuration options for the grid are:

- `-color color`
  
  Sets the color of the grid lines. The default is black.

- `-mapx xAxis`
  
  Specifies the X-axis to display grid lines. `xAxis` must be the name of an axis. The default is `xaxis`.

- `-mapy yAxis`
  
  Specifies the Y-axis to display grid lines. `yAxis` must be the name of an axis. The default is `y`.

In addition to the `configure` and `cget` methods that manipulate grid configuration parameters, the two main methods are:

```
off
```

Turns off the display the grid lines.

```
off
```

Turns on the display the grid lines.

**Stripchart legends**

The legend displays a list of the data elements. Each entry consists of the element’s symbol and label. The legend can appear in any margin (the default location is in the right margin). It can also be positioned anywhere within the plotting area.

The legend documentation we will provide here are the legend options that can all be configured or queried via e.g.:

```
.stripchart configure options...
```

```
.stripchart cget option-name
```

Where the most common options are:
-hide boolean

Indicates whether the legend should be displayed. If boolean is true, the legend will not be drawn. The default is false, allowing the legend to be visible.

-postition pos

Specifies where the legend is drawn. If pos is left, left, top, or bottom, the legend is drawn in the specified margin. If pos is plotarea, then the legend is drawn inside the plotting area. If pos is in the form "@x,y", where x and y are the window coordinates, the legend is drawn in the plotting area at the specified coordinates. The default is right.

Stripchart Pens

Stripchart Pens. Stripchart pen components are bundles of attributes that can be applied to elements (traces). The construction and manipulation of pens is an advanced topic refer to the blt::stripchart online manpage for information about this component.

Stripchart Postscript output

The postscript component support the generation of a postscript file that allows printing the contents of the plot. The most commonly used postscript method is:

```
.stripchart postscript output filename options...
```

Where the most common options are:

-colormode mode

Specifies how to output color information. Mode must be either color (for full color output), gray (convert all colors to their gray-scale equivalents) or mono (convert foreground colors to black and background colors to white). The default mode is color.

-landscape boolean

If boolean is true, this specifies the printed area is to be rotated 90 degrees. In non-rotated output the X-axis of the printed area runs along the short dimension of the page ("portrait" orientation); in rotated output the X-axis runs along the long dimension of the page ("landscape" orientation). Defaults to 0.

Stripchart Markers

Stripchart Markers. Markers are simple drawing procedures used to annotate or highlight areas of the strip chart. Markers have various types: text strings, bitmaps, images, connected lines, windows, or polygons.
While the use of markers can be useful in many application we do not want to reproduce the entire blt::stripchart manpage about markers here. We will only show how to create markers of the various sorts and what each marker type’s configuration options are.

All markers understand a -coords option that contains a list of coordinates. The number of coordinates required and their meaning depends on the marker type.

All markers understand a -name option that provides a unique name for the marker. If not provided, the widget will generate a unique name for you.

Creating a bitmap marker. A bitmap marker displays a bitmap. The size of the bitmap is controlled by the number of coordinates specified. If two coordinates, they specify the position of the top-left corner of the bitmap. The bitmap retains its normal width and height. If four coordinates, the first and second pairs of coordinates represent the corners of the bitmap. The bitmap will be stretched or reduced as necessary to fit into the bounding rectangle.

Bitmap markers are created with the marker’s create operation in the form:

```
.stripchart marker create bitmap ?options?
```

This command returns the name of the marker created.

Bitmap specific options of interest are:

- **-bitmap bitmap**
  Specifies the bitmap to be displayed. If bitmap is "", the marker will not be displayed. The default is "".

- **-mask mask**
  Specifies a mask for the bitmap to be displayed. This mask is a bitmap itself, denoting the pixels that are transparent. If mask is "", all pixels of the bitmap will be drawn. The default is "".

- **-rotate theta**
  The marker is first rotated and then placed according to its anchor position. The default rotation is 0.0.

Image markers. A image marker displays an image. Image markers are created with the marker’s create operation in the form:

```
.stripchart marker create image -image image-name
```
The coordinates for an image have the same meaning as for a bitmap.

**Line Markers.** A line marker displays one or more connected line segments. Line markers are created with marker’s create operation in the form:

```
.stripchart marker create line ?options?
```

The coordinates in this case are the coordinates of the vertices of the polyline that make up the marker. There must be at least four coordinates, \(x_1,y_1\) and \(x_2,y_2\) of a single line segment, however there can be additional points to add additional line segments to the marker.

The commonly used options for the line marker are:

- **foreground color**
  Sets the foreground color. The default foreground color is black.

- **linewidth pixels**
  Sets the width of the lines. The default width is 0.

**Polygon Markers.** These are essentially Line markers with an added line connecting the last point to the first point, however just to be perverse, some of the options have different names.

```
.stripchart marker create polygon options...
```

Key options are:

- **fill color**
  Sets the fill color of the polygon. If \(color\) is "", then the interior of the polygon is transparent. The default is white.

- **outline color**
  Sets the color of the outline of the polygon. The default is black.

**Text Markers.** A text marker displays a string of characters on one or more lines of text. Embedded newlines cause line breaks. They may be used to annotate regions of the strip chart. One pair of coordinates must be supplied with the marker to specify where the text is modified by the \(-anchor\) options.

Text markers are created as follows:

```
.stripchart marker create text ?option? ...
```
Where the most commonly used options are:

- **anchor anchor**
  
  *Anchor* tells how to position the text relative to the positioning point for the text. For example, if *anchor is center* then the text is centered on the point; if *anchor is* then the text will be drawn such that the top center point of the rectangular region occupied by the text will be at the positioning point. This default is *center*.

- **foreground color**
  
  Sets the foreground color of the text. The default is black.

- **justify justify**
  
  Specifies how the text should be justified. This matters only when the marker contains more than one line of text. *Justify* must be *left*, *right*, or *center*. The default is *center*.

- **rotate theta**
  
  Specifies the number of degrees to rotate the text. *Theta* is a real number representing the angle of rotation. The marker is first rotated along its center and is then drawn according to its anchor position. The default is 0.0.

- **text text**
  
  Specifies the text of the marker. The exact way the text is displayed may be affected by other options such as *-anchor* or *-rotate*.

**Window markers.** Window markers allow you to place other widgets on the stripchart. The idea is that you create a widget that is a child of the stripchart. You then add it as a window marker using the coordinates to specify the position of the widget. e.g:

```
.stripchart marker create window -window .stripchart.w ...
```

**EXAMPLES**

This section will show some simple examples of how to create and use the epicsStripchart widget.
Creating a simple stripchart

This example shows how to create the simplest stripchart displaying a single control system parameter. The stripchart produced will display the time evolution of the parameter K5COILA-I, updated once a second. Each data point will be represented as a circle (default symbol). Black lines will connect the circles (default line color). Both the value and time scales will auto-scale to fit the parameter values and the time range. This will cause the time range to dynamically shrink to contain the entire data set:

```tcl
package require epicsStripChart

controlwidget::epicsStripChart .e
.e addchannel K5COILA-I 1000
pack .e
```

Setting stripchart trace attributes

This example is the same as the previous one, however we will configure the trace so that it has no symbols and is red in color

```tcl
package require epicsStripChart

controlwidget::epicsStripChart .e
set trace [.e addchannel K5COILA-I 1000]
.e element configure $trace -color red -symbol {}
pack .e
```

This example also shows how to capture the name of the element/history object for later manipulation. We can shorten this example, by using the fact that the addchannel method of epicsStripChart allows you to specify the configuration options for the element when you create it:

```tcl
package require epicsStripChart

controlwidget::epicsStripChart .e
.e addchannel K5COILA-I 1000 -color red -symbol {}
pack .e
```
Configuring the appearance of the plot.

The examples so far have suffered from a continuously shrinking time axis, no grid, a badly positioned legend, and a Y axis that does not really give you an idea of the absolute magnitude of the data. In this example we'll make the Y axis start at 0, and the X axis display only the last minute of data, shifting by 10 seconds when the trace goes out of range. We will also turn on a grid, and set the legend at the bottom of the graph.

```tcl
package require epicsStripChart

controlwidget::epicsStripChart .e
    .e yaxis configure -min 0.0
    .e xaxis configure -autorange 60.0 -shiftby 10.0
    .e grid on
    .e legend configure -position bottom
    .e addchannel K5COILA-I 1000 -color red -symbol {}

pack .e
```

Adding titles

This example builds on the previous example by adding a plot title and axis titles.

```tcl
package require epicsStripChart

controlwidget::epicsStripChart .e -title {NSCL Strip Chart}
    .e yaxis configure -min 0.0 -title {Parameter Values}
    .e xaxis configure -autorange 60.0 -shiftby 10.0 -title {Time}
    .e grid on
    .e legend configure -position bottom
    .e addchannel K5COILA-I 1000 -color red -symbol {}

pack .e
```

A graph with several traces

This example shows that you can add several traces to the graph. We will let the y axis go back to auto ranging, as the K800 B coil current was negative when this was tested:

```tcl
package require epicsStripChart

controlwidget::epicsStripChart .e -title {NSCL Strip Chart}
    .e yaxis configure -title {Parameter Values}
    .e xaxis configure -autorange 60.0 -shiftby 10.0 -title {Time}
```
Accessing historical data

This application adds a File menu. The File menu will have two menu items. The Exit menu item will exit the program. The Save... menu item will prompt for a filename and save the historical data on the plot.

Doing this involves using the channelHistory object created by the addchannel method. We will create a file that has a header that consists of a line containing the channel name, and a line containing the time of the first measurement. We will then provide the historical data as one line per measurement where each line consists of a pair of fields. The first field is the offset in seconds from the base time, and the second the parameter value at that time. We will also change the update time to once every 0.5 seconds (500 ms).

For simplicity, we will go back to a single trace of the K500 A coil current.

package require epicsStripChart

# First set up the strip chart:
package require epicsStripChart

controlwidget::epicsStripChart .e -title {NSCL Strip Chart}
.e yaxis configure -min 0.0 -title {Parameter Values}
.e xaxis configure -autorange 60.0 -shiftby 10.0 -title {Time}
.e grid on
.e legend configure -position bottom

# add the element, save the history object in the
# global variable 'history'.
set history [.e addchannel K5COILA-I 1000 -color red -symbol {}]

pack .e

# proc to save the history data; name of history object is
# passed in.
proc saveHistory {h} {
    set filename [tk_getSaveFile -defaultextension .trace  \
        -title {Save file as...}  \
        -filetypes {  \
            {{Trace files} {.trace} TEXT}  \
            {{Text Files} {.txt} TEXT}  \
            {{All Files}* } }
    ]
    # Blank filename means the user hit cancel
    if {$filename eq ""} {
        return
    }
    # If we can’t open the file put up a nice error dialog:
    if {[catch {open $filename w} fd]} {
        tk_messageBox -icon error -type ok -title {Open Failed}  \
            -message "Could not open $filename : $fd"
        return
    }
    # Now we can write the data to file:
    set channel [$h cget -channel]
    set startTime [$h cget -timebase]
    puts $fd $channel
    puts $fd [clock format $startTime]
    set data [$h get]
    foreach point $data {
        puts $fd "[lindex $point 0] [lindex $point 1]"
    }
    close $fd
}

# Proc to exit if the user confirms:
proc Exit {} {
    set answer [tk_messageBox -icon question -type yesno -title {Exit?}  \
        -message {Do you really want to exit?}]
    if {$answer eq "yes"} {
        exit
    }
}

# Create the file menu
menu .bar
menu .bar.file -tearoff 0
.bar add cascade -label File -menu .bar.file
SEE ALSO

blt::stripchart(3blt), epics(3tcl)

typeNGo bound to epics

Name
epicsTypeNGo — Provide epics bindings to a typeNGo widget.

Synopsis

package require epicsTypeNGo

controlwidget::epicsTypeNGo path ?options?...

SUMMARY

Links an epics channel to a typeNGo widget. Committing the value results in setting the value of the control channel, while the label continuously displays the up-to-date value of the channel. Validation serves to prevent a commit on nonsense values.

OPTIONS

All typeNGo options except -command are supported by this, however the use of the -textvariable option will break the binding of the label to the epics channel. There may or may not be good reasons to do this. The -channel option is added and selects which channel the widget will be bound to.
METHODS

All typeNGo methods are supported.

EXAMPLES

This example shows how to create a horizontally laid out epic type and go widget, that only allows floating point channel values to be entered. See the typeNGo widget’s -validate, -orient switches to understand this example.

package require epicsTypeNGo

controlwidget::epicsTypNGo .tng -channel Z001F-C -orient horizontal \
   -validate [list string is double -strict %V%]

pack .tng

SEE ALSO

typeNGo(1tcl)

epicsspinbox

Name

epicsspinbox — Connects a spinbox widget with an epics channel.

Synopsis

package require epicsSpinbox

::controlwidget::epicsSpinBox path ?options?...
OPTIONS

All options accepted by a Tk::spinbox are accepted by this widget, and apply to the spinbox piece of this widget. In addition, the following options have been implemented:

**-channel name**

Required at construction time, the value of this option specifies the EPICS process variable to be controlled/monitored by this widget. Any channel name or record field can be specified (although clearly it only pays to specify those that can be modified).

**-showsetting yes-no**

If the value of this option can be evaluated as boolean true, a label giving the actual value of the process variable will be displayed above the spinbox. If not, the label will be omitted. This can be dynamically modified.

METHODS

**Get**

Returns the current value of the process variable. Note that this is the value in the -showsetting label (if that would be displayed), rather than the setting from the spinbox itself.

**Set value**

Sets the value of the spinbox and the process variable to value.

Vertical meter widget

**Name**

meter — Provide a widget that is a vertical meter.

**Synopsis**

package require meter

controlwidget::meter path ?options...?
OPTIONS

-from value
Defines the lower end of the meter range and scale. If not provided, this defaults to -1.0. value should be a number that can be interpreted as a floating point value.

-to value
Defines the upper end of the meter range and scale. If not provided, this defaults to 1.0. value should be a number that can be interpreted as a floating point value.

-height value
Sets the height of the widget. This can be specified in pixels, centimeters or inches like any other tk dimension.

-width value
Specifies the width of the widget. This can be specified in pixels, centimeters or inches like any other tk dimension.

-variable name
Links the height of the meter’s indicator to a Tcl variable in global or namespace scope. As the value of this variable changes, the height of the meter indicator also changes. Setting to a blank name removes any linkage between the meter value and a variable.

-majorticks interval
Provides the interval between major ticks on the meter. Note that major ticks get labeled, so be sure that you have enough range between major ticks to allow the label to be legible.

-minorticks number
Specifies the number of minor tick intervals between major ticks (intervals in this case implies that there will be one fewer tick marks than you specify e.g. 5 intervals require 4 ticks). Minor tick marks are not labeled, and are somewhat shorter than major tick marks.

-log boolean
If the boolean is true, the meter will display in logscale. This has several other side effects:

- The user supplied values of -majorticks and -minorticks are ignored and chosen by the meter widget
- The -from and -to values are pushed to nearest decades below and above respectively, for example -from 55 and -to 750 will be pushed to -from 10 and -to 1000, resulting in two full decades of meter range. Ranges that encompass the negative direction are not supported and will result in an error.
- Data values that are zero are treated as .0001.
- Negative values will display as the lowest displayable value on the meter.
METHODS

set value
Sets the meter indicator height to a specific value. If the meter has a -variable specified, the variable is set as well.

get
Returns the current meter value.

SEE ALSO

controlwidget::bcmMeter

LED Widget

Name
led — Provide a widget that looks like an LED.

Synopsis
package require led

controlwidget::led path ?options?

OPTIONS

-size measure
Specifies the size of the widget (LED widgets are symmetric, so this specifies both the height and width of the widget.)
-on color

Specifies the color of the LED when it is on. By default this is green. The color can be specified in any way normally acceptable to Tk.

-off color

Specifies the color of the LED when it is off. By default, this is black.

-variable name

Specifies the name of a variable with permanent scope (global or in a namespace) that will control the value of the LED. If the value of the variable is 0 or a valid 'false' boolean, the LED will be off, otherwise, on.

METHODS

on

Turn the LED on. If there is a variable associated with the LED, it is set to 1.

off

Turn the LED off. If there is a variable associated with the LED, it is set to 0.

SEE ALSO

epicsLed(1tcl)

typeNGo compound widget

Name

typeNGo — provides a compound widget for entering text with an explicit commit.

Synopsis

package require typeNGo
controlwidget::typeNGo path ?options...?

SUMMARY

The typeNGo is a compound widget that consists of vertically stacked label, entry and button widgets. The idea is that this will typically be used to provide controlled updates of the value of the variable that controls the label widget. If you consider the case of an entry and a label both bound via -textvariable to the same variable, as you type in the entry, the variable value dynamically changes. This is not suitable for controls applications e.g.

The typeNGo widget provides explicit control over when the entry widget is a correct value worth propagating to the application. This is done either by clicking the button or by hitting the enter/return key while the focus is in the entry widget.

Validation scripts are also supported (see -validate in the OPTIONS section). Validation scripts are invoked when the button is clicked and must return a true or false value. If false is returned, the -command script is not invoked and the entry field is returned to its prior value.

OPTIONS

All label operations except -text are forwarded directly to the label widget contained by the megawidget. See, however the -label option.

-orient vertical | horizontal
  Only processed at widget creation time. This option determines the layout of the widget. If the value is vertical (the default), the label, entry and button are laid out vertically in that order. If the value is horizontal, the label, entry and button are laid out horizontally in that order. See however, -showlabel.

-showlabel bool
  Only processed at widget creation time. This option determines if the label widget is actually displayed. If the value is true (default), the label widget is displayed. If the value is false, the label widget is created but not displayed.

-text labelstring
  Provides a label for the button widget.

-label labelstring
  Provides a string for the label. This is overridden if a -textvariable is specified.
-command script

Provides a callback script that will be invoked when the entry is committed via a button click or enter key. In the script, %W is substituted with the widget command name. %V is substituted with the value of the entry.

-validate script

Provides a script to perform validation. The %V and %W substitutions described in -command are supported. If the script does not return a true value, the -command script will not be executed, and the entry field value will be returned to its prior value.

METHODS

Get

Gets the current value of the entry widget. This will be the text currently displayed in that widget, not the most recently committed value.

Set value

Sets the value of the entry widget to the value string. This does not commit it (see Invoke). This also does not do any validation.

Invoke

Simulates a button click. This will cause entry validation and, if permitted, a commit of the entry.

BINDINGS

<Return> - with focus in the entry widget.

does an Invoke on self.

<FocusOut> - With focus in the entry widget

restores the prior value to the widget.
EXAMPLES

This example shows how to use the -validate switch to ensure that the entry field has a legal floating point value when the -command script would be invoked. If the entry field is not a floating point value, the prior value of the field is restored.

```tcl
package require typeNGo
controlwidget::typeNGo .tng -validate [list string is double -strict %V]
pack .tng
```

This example uses the Tcl string is double command to determine if the new value (%V) is a double. If not, the validation fails, -command won’t be executed, and the prior value of the entry field will be restored.

SEE ALSO

epicsTypeNGo(1tcl)