```
W_LFSR -= 2^15
W_LFSR /= 2^15-1
```

Another way to do the same thing that avoids the Redimension operation, which could lead to fragmentation of memory:

```
Make/D/N=(2^16-1) LFSR_output
LinearFeedbackShiftRegister/N=16/DEST=LFSR_output
LFSR_output -= 2^15
LFSR_output /= 2^15-1
```

Make a bit stream with random +1 and -1 instead of 0 and 1:

```
LinearFeedbackShiftRegister/N=16/MODE=1
Redimension/B W_LFSR
W_LFSR = W_LFSR*2-1
```

See Also

If you really need random numbers, we provide high-quality RNG's that return random deviates from a number of distributions. See **enoise**, **gnoise**, and others.

References

A discussion of LFSR's can be found in "Generation of Random Bits" (Section 7.4) in Press, William H., *et al*., *Numerical Recipes in C*, 2nd ed., 994 pp., Cambridge University Press, New York, 1992. They refer to "primitive polynomials modulo 2" and do not use the name Linear Feedback Shift Register, but it is the same thing. We use an implementation equivalent to their Method I.

ListBox

ListBox [**/Z**] ***ctrlName*** [***keyword = value*** [**,** ***keyword = value*** …]]

The ListBox operation creates or modifies the named control that displays, in the target window, a list from which the user can select any number of items.

For information about the state or status of the control, use the **ControlInfo** operation.

Parameters

ctrlName is the name of the ListBox control to be created or changed.

The following keyword=value parameters are supported:

align=*alignment*

Sets the alignment mode of the control. The alignment mode controls the interpretation of the *leftOrRight* parameter to the pos keyword. The align keyword was added in Igor Pro 8.00.

If *alignment*=0 (default), *leftOrRight* specifies the position of the left end of the control and the left end position remains fixed if the control size is changed.

If *alignment*=1, *leftOrRight* specifies the position of the right end of the control and the right end position remains fixed if the control size is changed.

appearance={*kind* [, *platform*]}

Sets the appearance of the control. *platform* is optional. Both parameters are names, not strings.

kind can be one of `default`, `native`, or `os9`.

platform can be one of `Mac`, `Win`, or `All`.

See **Button** and **DefaultGUIControls** for more appearance details.

clickEventModifiers=*modifierSelector*

Selects modifier keys to ignore when processing clicks to start editing a cell or when toggling a checkbox. That is, use this keyword if you want to prevent a shift-click (for instance) from togging checkbox cells. Allows the action procedure to receive mousedown events with those modifiers without interfering with actions on the part of the listbox control.

modifierSelector is a bit pattern with a bit for each modifier key; sum these values to get the desired combination of modifiers:

modifierSelector=1:	Control key (Macintosh only)
modifierSelector=2:	Option (Macintosh) or Alt (Windows)
modifierSelector=4:	Context click (right click on Windows, control-click on Macintosh)
modifierSelector=8:	Shift key
modifierSelector=16:	Cmd key (Macintosh) or Ctrl key (Windows)
modifierSelector=32:	Caps lock key

See **Setting Bit Parameters** on page IV-12 for details about bit settings.

col=*c*

Sets the left-most visible column (user scrolling will change this). The list is scrolled horizontally as far as possible. Sometimes this won't be far enough to actually make column *c* the first column, but it will at least be visible. Use *c* =1 to put the left edge of column 1 (the *second* column) at the left edge of the list.

colorWave=*cw*

Specifies a 3-column (RGB) or 4-column (RGBA) numeric wave. Used in conjunction with planes in selWave to define foreground and background colors for individual cells. Values range from 65535 (full on) to 0.

disable=*d*

Sets user editability of the control.

d=0:	Normal.
d=1:	Hide.
d=2:	Draw in gray state; disable control action.

editStyle=*e*

Sets the style for cells designated as editable (see selWave, bit 1).

e=0:	Uses a light blue background (default).
e=1:	Draws a frame around the cell with a white background.
e=2:	Combines the frame with the blue background. The background in all cases can be overridden using the colorWave parameter.

focusRing=*fr*

Enables or disables the drawing of a rectangle indicating keyboard focus:

fr=0:	Focus rectangle will not be drawn.
fr=1:	Focus rectangle will be drawn (default).

On Macintosh, regardless of this setting, the focus ring appears if you have enabled full keyboard access via the Shortcuts tab of the Keyboard system preferences.

font="*fontName*"

Sets the font used for the list box items, e.g., `font="Helvetica"`.

frame=*f*

Specifies the list box frame style.

f=0:	No frame.
f=1:	Simple rectangle.
f=2:	3D well.
f=3:	3D raised.
f=4:	Text well style.

fsize=s

Sets list box font size.

fstyle=*fs*

fs is a bitwise parameter with each bit controlling one aspect of the font style as follows:

Bit 0:	Bold
Bit 1:	Italic
Bit 2:	Underline
Bit 4:	Strikethrough

See **Setting Bit Parameters** on page IV-12 for details about bit settings.

hScroll=*h*

Scrolls the list to the right by *h* pixels (user scrolling will change this). *h* is the total amount of horizontal scrolling, not an increment from the current scroll position: *h* will be the value returned in the V_horizScroll variable by **ControlInfo**.

help={*helpStr*}

Sets the help for the control.

helpStr is limited to 1970 bytes (255 in Igor Pro 8 and before).

You can insert a line break by putting "\r" in a quoted string.

helpWave=*w*

Text wave containing text for tooltips for individual listbox cells. See ListBox Help below. Added in Igor Pro 8.00.

keySelectCol=*col*

Sets scan column number *col* when doing keyboard selection. Default is to scan column zero.

listWave=*w*

A 1D or 2D text wave containing the list contents.

mode=*m*

List selection mode specifying how many list selections can be made at a time.

m=0:	No selection allowed.
m=1:	One or zero selection allowed.
m=2:	One and only one selection allowed.
m=3:	Multiple, but not disjoint, selections allowed.
m=4:	Multiple and disjoint selections allowed.

When multiple columns are used, you can enable individual cells to be selected using modes 5, 6, 7, and 8 in analogy to *m*=1-4. When using *m*=3 or 4 with multiple columns, only the first column of the selWave is used to indicate selections. Checkboxes and editing mode, however, use all cells even in modes 0-4.

Modes 9 and 10 are the same as modes 4 and 8 except they use different selection rules and require testing bit 3 as well as bit 0 in selWave. In modes 4 and 8, a shift click toggles individual cells or rows, but in modes 9 and 10, the Command (*Macintosh*) or Ctrl (*Windows*) key toggles individual cells or rows whereas Shift defines a rectangular selection. T o determine if a cell is selected, perform a bitwise AND with `0x09`.

proc=*p*

Set name of user function proc to be called upon certain events. See discussion below.

pos={*leftOrRight*,*top*}

Sets the position in **Control Panel Units** of the top/left corner of the control if its alignment mode is 0 or the top/right corner of the control if its alignment mode is 1. See the align keyword above for details.

pos+={*dx*,*dy*}

Offsets the position of the list box in **Control Panel Units**.

row=*r*

r is desired top row (user scrolling will change this). Use a value of -1 to scroll to the first selected cell (if any). Combine with selRow to select a row and to ensure it is visible (modes 1 and 2).

selCol=*c*

Defines the selected column when mode is 5 or 6 and no selWave is used. To read this value, use **ControlInfo** and the V_selCol variable.

selRow=*s*

Defines the selected row when mode is 1 or 2; when no selWave is used, it is defined by modes 5 or 6. Use -1 for no selection.

To read this value, use **ControlInfo** and the V_value variable.

selWave=*sw*

sw is a numeric wave with the same dimensions as listWave. It is optional for modes 0-2, 5 and 6 and required in all other modes.

In modes greater than 2, *sw* indicates which cells are selected. In modes 1 and 2 use **ControlInfo** to find out which row is selected.

In all modes *sw* defines which cells are editable or function as checkboxes or disclosure controls.

Numeric values are treated as integers with individual bits defined as follows:

Bit 0 (0x01):	Cell is selected.
Bit 1 (0x02):	Cell is editable.
Bit 2 (0x04):	Cell editing requires a double click.
Bit 3 (0x08):	Current shift selection.
Bit 4 (0x10):	Current state of a checkbox cell.
Bit 5 (0x20):	Cell is a checkbox.
Bit 6 (0x40):	Cell is a disclosure cell. Drawn as a disclosure triangle (*Macintosh*) or a treeview expansion node (*Windows*).
Bit 7 (0x80):	Cell is disabled. Clicks on the cell are ignored and it is drawn with a grayed appearance. Added in Igor Pro 8.00. Because of technical issues, this bit currently does not prevent the selection of the cell using the arrow keys.

In modes 3 and 4 bit 0 is set only in column zero of a multicolumn listbox.

Other bits are reserved. Additional dimensions are used for color info. See the discussion for colorWave. selWave is not required for modes 5 and 6.

setEditCell={*row*,*col*,*selStart*,*selEnd*}

Initiates edit mode for the cell at *row*, *col*. An error is reported if *row* or *col* is less than zero. Nothing happens and no error is reported if *row*, *col* is beyond the limits of the listbox, or if the cell has not been made editable by setting bit 1 of *selWave*.

selStart and *selEnd* set the range of bytes that are selected when editing is initiated; 0 is the start of the text. If there are N bytes in the listbox cell, setting *selStart* or *selEnd* to N or greater moves the start or end of the selection to the point after the last character. Setting *selStart* and *selEnd* to the same value selects no characters and the insertion point is set to *selStart*. Setting *selStart* to -1 always causes all characters to be selected.

size={*width*,*height*}

Sets list box size in **Control Panel Units**.

special={*kind*,*height*,*style*}

Specifies special cell formatting or contents.

kind=0: Normal text but with specified *height* (if nonzero). Use a *style* of 1 to autocalculate widths based on the entire list contents. In this case, user widths are taken to be minimums and the last is not repeated.

kind=1: Text taken to be the names of graphs or tables. Images of the graphs or tables are displayed in the cells. Use a *style* of 0 to display just the presentation portion of the graph or 1 to display it entirely. For tables, only the presentation portion is displayed.

kind=2: Text taken to be the names of pictures. Images are displayed in the cells.

kind=3: Displays a PNG, TIFF, or JPEG image. You can obtain binary picture data using SavePICT.

kind=4: Displays raw text without any escape code processing. Added in Igor Pro 8.00 to allow display of LaTeX expressions which make heavy use of backslashes.

For *kind*=1 or 2, *height* may be zero to auto-set cell height to same as width or a specific value.

titleWave=*w* Specifies a text wave containing titles for the listbox columns, instead of using the list wave dimension labels. Each row is the title for one column; if you have N columns you must have a wave with N rows.

userColumnResize=*u* Enables resizing the list columns using the mouse.

u=0: Columns are not resizable (default). The widths parameter still works, though.

u=1: User can resize columns by dragging the column dividers.

When resizing a column without Option, Alt, or Shift modifiers (a "normal" resizing), any width added to the column is subtracted from the following column (if any).

When resizing while pressing Option (*Macintosh*) or Alt (*Windows*), only columns following the dragged divider will move (the same way table columns are resized).

When pressing Shift, all columns are set to the same width as the column being resized. If the total widths of all columns is less than the width of the listbox, then each column expands to fill the available width.

userdata=*UDStr* Sets the unnamed user data to *UDStr*.

userdata(*UDName*)=*UDStr*

Sets the named user data, *UDName*, to *UDStr*.

userdata+=*UDStr* Appends *UDStr* to the current unnamed user data.

userdata(*UDName*)+=*UDStr*

Appends *UDStr* to the current named user data, *UDName*.

widths={*w1*,*w2*,…} Optional list of minimum column widths in screen pixels. If more columns than widths, the last is repeated. If total of widths is greater than list box width then a horizontal scroll bar will appear. If total is less than available width then each expands proportionally.

widths+={*w1*,*w2*,…}	Additional column widths. Because only 2500 bytes fit on a command line, lists with many columns may require multiple widths+= parameters to define all the column widths. However if all the widths are the same, widths+= is not needed; just use: `ListBox ctrlName widths={sameWidth}`
win=*winName*	Specifies which window or subwindow contains the named control. If not given, then the top-most graph or panel window or subwindow is assumed. When identifying a subwindow with *winName*, see **Subwindow Syntax** on page III-92 for details on forming the window hierarchy.

Flags

/Z	No error reporting.

Details

If the list wave has column dimension labels (see **SetDimLabel**), then those will be used as column titles. Note that a 1D wave is subtly different from a 1 column 2D wave. The former does not have any columns and therefore no column dimension labels.

Alternately, use a text wave with the titleWave keyword to specify column titles.

Using escape codes you can change the font, size, style, and color of the title. See **Annotation Escape Codes** on page III-53 or details.

For instance, to make a title bold use the \f01 escape sequence:

```
SetDimLabel 1, columnNum, $"\\f01My Label", textWave
```

If you can't fit title text within the 255 byte limit (styled text can be especially long), use the *titleWave* keyword with a text wave. The wave must have as many rows as the list wave has columns. When using a title wave, there are no restrictions on the number of bytes or on what characters you can use.

This example uses a title wave to add a red up-arrow graph marker to the end of a centered title:

```
Make/O/T/N=(numColumns) columnTitles
columnTitles[colNum]="\\JCThis is the title\\K(65535,0,0)\\k(65535,0,0)\\W523"
ListBox list0 titleWave=columnTitles
```

That's a 51-byte title that results in 19 characters or symbols that you actually see. \JC requests centered text, \K sets the text color (which colors the inside of the graph marker),\k sets the marker stroke color, and \W523 inserts a down-pointing triangular graph marker.

When using modes that allow multiple selections, use Shift to extend or add to the selection.

You can specify individual cells as being editable by setting bit 1 (counting from zero on the right) in selWave. The user can start editing a cell by either clicking in it or, if the cell is selected, by pressing Enter (or Return). When finished, the user can press Enter to accept the changes or can press Escape to reject changes. The user may also press Up or Down Arrow to accept changes and begin editing the next editable cell in a column. Likewise, Tab and Shift-Tab moves to the next or previous column in a row. If bit 2 of selWave is set then a double click will be required rather than a single click. **Note**: in edit mode, Tab and Shift-Tab are used to move left and right because the Left and Right Arrow keys are used to move the text entry cursor left and right.

When the listbox has keyboard focus (either by tabbing to the list box or by clicking in the box), the keyboard arrow keys move a cell selection (or row depending on mode). When not in cell edit mode, Tab and Shift-Tab move the keyboard focus to other objects in the window. The Home, End, Page Up, and Page Down keys affect the vertical scroll bar.

When the listbox has focus, the user may type the first few chars of an entry in the list to select that entry. Only the first column is used. If a match is not found then nothing is done. The search is case insensitive.

ListBox Help

You can provide a text wave with help strings to show a tooltip for individual cells or rows of a listbox in place of the generic help string provided by the help keyword. In general, the string in helpWave[*listbox row*][*listbox column*] is used. If your help wave is a 1D wave, then the listbox column is ignored, and helpWave[*listbox row*] is used in all columns.

Note that a 1D wave (Make/N=*rows*) is not the same as a 2D wave with one column (Make/N=(*rows*,1)). A 2D wave with one column provides help for column 0 only and the generic help string is shown for column 1 and beyond. A 1D wave provides help for all columns.

If the help wave has an empty string for a given row and column then the generic help string is shown for the corresponding listbox cell. If the help wave has too few rows or columns, then the generic help string is shown for the cells outside the range of the help wave.

Listbox Action Procedure

The action procedure for a ListBox control takes a predefined structure `WMListboxAction` as a parameter to the function:

```
Function ActionProcName(LB_Struct) : ListboxControl
   STRUCT WMListboxAction &LB_Struct
   …
   return 0
End
```

The "`: ListboxControl`" designation tells Igor to include this procedure in the Procedure pop-up menu in the List Box Control dialog.

See **WMListboxAction** for details on the WMListboxAction structure.

Although the return value is not currently used, action procedures should always return zero.

You may see an old format listbox action procedure in old code:

```
Function MyListboxProc(ctrlName,row,col,event) : ListboxControl
   String ctrlName     // name of this control
   Variable row        // row if click in interior, -1 if click in title
   Variable col        // column number
   Variable event      // event code
   …
   return 0            // other return values reserved
End
```

This old format should not be used in new code.

Specifying Cell Color

The background and foreground (text) color of individual cells may be defined by providing colorWave in conjunction with specific planes in selWave. The planes in selWave are taken to be integer indexes into colorWave. The planes are defined by specific dimension labels and not by specific plane numbers. To provide foreground colors, define a plane labeled "foreColors" that contains the desired index values. Likewise define and fill a plane labeled "backColors" for background colors. The value 0 is special and indicates that the default colors should be used. Note that if you have a one column list for which you want to supply colors, the selWave needs to be three dimensional but with just one column. Here is an example:

```
Make/T/N=(5,1) tw= "row "+num2str(p)       // 5 row, 1 col text wave (2D)
Make/B/U/N=(5,1,2) sw                     // 5 row, 1 col, 2 plane byte wave
Make/O/W/U myColors={{0,0,0},{65535,0,0},{0,65535,0},{0,0,65535},{0,65535,65535}}
MatrixTranspose myColors      // above was easier to enter as 3 rows, 5 cols

NewPanel
ListBox lb,mode=3,listWave= tw,selWave= sw,size={200,100},colorWave=myColors
sw[][][1]= p                          // arbitrary index values into plane 1
```

Now, execute the following commands one at a time and observe the results:

```
SetDimLabel 2,1,backColors,sw       // define plane 1 as background colors
SetDimLabel 2,1,foreColors,sw       // redefine plane 1 as foreground colors
sw[][][%foreColors]= 4-p            // change the color index values
```

In the above example, the selWave was defined as unsigned byte. If you need more than 254 colors, you will have to use a larger number size.

The color wave may have four columns instead of three, with the fourth specifying transparency (65535=fully opaque, 0=fully transparent). Transparency of the background color is of limited utility as it just shows the white background color through the cell color, resulting in pastel shades. Transparency of the foreground color permits the cell background color to show through the text color.

Checkboxes in Cells

You can cause a cell to contain a checkbox by setting bit 5 in selWave. The title (if any) is taken from listWave and the results (selected/deselected) is bit 4 of selWave. If a checkbox cell is selected then the space bar will toggle the checkbox. (Clicking a checkbox cell does not select it — use the arrow keys.)

Errors

Your listbox may be drawn with a red X and an error code. The error codes are:

Error	Meaning
E1	The listbox is too small
E2	The listWave is invalid (missing, not text or no rows)
E3	The listWave and selWave do not match in dimensions
E4	Mode is greater than 2 but no selWave was specified
E5	The number of rows in titleWave does not match the number of columns in listWave
E6	The titleWave is not a textWave

Event Queue

It is possible for a single user action to produce more than one event. For instance, pressing the up arrow key while editing to select a cell that is not visible generates event codes 4, 8, and 6. Igor calls your action procedure separately for each code.

If your code tests for a nonzero value in eventCode2, and uses it only if it is nonzero, then the event queue method will not break your code. If you have determined empirically when you need to use eventCode2 and you do not test, your code will break.

Scroll Event Warnings

Events 8, 9, and 10 report to you that the listbox has been scrolled vertically or horizontally. These events are envisioned as allowing you to keep two listboxes synchronized (you may find other uses for these events). You might use an action procedure like this one to keep two listboxes (named list0 and list1) in sync:

```
Function ListBoxProc2(LB_Struct) : ListBoxControl
    STRUCT WMListboxAction &LB_Struct

    if (LB_Struct.eventCode == 8)
        String listname
        if (CmpStr(LB_Struct.ctrlName, "list1") == 0)
            listname = "list0"
        else
            listname = "list1"
        endif
        ControlInfo $listname
        if (V_startRow != LB_Struct.row)
            listbox $listname,row=LB_Struct.row
            ControlUpdate $listname
        endif
    endif

    return 0
End
```

It is very easy to create an infinite cascade of events feeding back between the two listboxes, especially if you use event 10. When this happens, you will see your listboxes jigging up and down endlessly. The test using ControlInfo is intended to make this unlikely.

The slow response of the old-style, nonstructure action procedure can defeat the ControlInfo test by delaying the action procedure execution. If you use events 8, 9, or 10, we recommended that you use the new-style action procedure.

Note on Keystroke Event

In a keystroke event passed to a listbox action procedure the eventCode is 12. A character code is stored in the row field of the WMListboxAction structure. This works only for ASCII characters and a few special

characters such as delete (8), forward delete (127) and escape (27). It does not work for non-ASCII characters such as accented characters which are represented as UTF-8 and require multiple bytes.

As explained below, many keystroke events are not sent to the listbox action procedure because they are consumed by the internal listbox code in Igor. For this reason the keystroke event for a listbox is of limited use.

The architecture of Igor controls is such that events are passed to an action procedure only after the control has used them. In the case of a keystroke event, that means that other uses of the keystroke may consume the event before the action procedure gets a chance at it. In particular, a listbox editable cell that is actively being edited consumes keystroke events, and the action procedure is not called. The only editing-related events your action procedure will get are event codes 6 and 7.

If an arrow key is pressed, and this results in the selected row or cell changing, your action procedure will not get a keystroke event. Instead, your action procedure will receive event code 4 or 5. If the arrow key causes scrolling to occur, the action procedure will also get event code 8 or 9.

Examples

Here is a simple Listbox example:

```
Make/O/T/N=30 tjack="this is row "+num2str(p)
Make/O/B/N=30 sjack=0
NewPanel /W=(19,61,319,261)
ListBox lb1,pos={42,9},size={137,94},listWave=tjack,selWave=sjack,mode= 3
Edit/W=(367,61,724,306) tjack,sjack
ModifyTable width(tjack)=148
```

Make selections in the list and note changes in the table and vice versa. Edit one of the list text values in the table and note update of the list.

Here is an example using a titleWave and styled text in the title cells. Note that the last title isn't very long when rendered, but requires a 63 byte specification.

```
Make/O/T/N=(4,3) ListWave="row "+num2str(p)+" col "+num2str(q)
Make/O/T/N=3 titles                    // three rows to match 3-column ListWave
titles[0] = "\f01Bold Title"
titles[1] = "title with semicolon;"
titles[2] = "Marker in Gray: \K(40000,40000,40000)\k(40000,40000,40000)\W517"
NewPanel /W=(515,542,1011,794)
ListBox list0,pos={1,2},size={391,120},listWave=ListWave
ListBox list0,titleWave=titles
```

It is often useful to be able to display a contextual menu in response to a click in a listbox cell. Here is a simple example:

```
Function ListBoxContextProc(lba) : ListBoxControl
    STRUCT WMListboxAction &lba


    Variable row = lba.row
    Variable col = lba.col
    WAVE/Z selWave = lba.selWave


    switch( lba.eventCode )
                        // Always display the contextual menu on the mouse-down event
                        // On Windows, the mouse-down is delivered only after the mouse is released
            case 1: // mouse down
                        if (row >= 0 && row < DimSize(selWave, 0))// click must be in a cell
                                    if (col == 0)                                   // click must be in the left
column
                                                if (lba.eventMod & 16)   // it's a contextual click
                                                            String menucontents =
"Nothing;Checkbox;Disclosure;Disabled;Enabled;"
```

```
                                                PopupContextualMenu menucontents
                                                if (V_flag > 0)// if nothing was selected,
V_flag = 0
                                                        StrSwitch(S_selection)//
S_selection is the text of the menu item
                                                        case "Nothing":
                                                        SelWave[row][1] = 0
                                                        break
                                                        case "Checkbox":
                                                        SelWave[row][1] = 0x20
                                                        break
                                                        case "Disclosure":
                                                        SelWave[row][1] = 0x40
                                                        break
                                                        case "Disabled":
                                                        SelWave[row][1] =
SelWave[row][1] | 0x80
                                                        break
                                                        case "Enabled":
                                                        SelWave[row][1] =
SelWave[row][1] & ~0x80
                                                        break
                                                        endswitch
                                                endif
                                        endif
                                endif
                        endif
                        break
                case 13:                                                // checkbox clicked
                        String msg
                        // test the value of SelWave for the clicked cell to see what it is, and its state
                        Variable isOn = (SelWave[row][1] & 0x10) ? 1 : 0
                        if (SelWave[row][1] & 0x20)
                                msg = "Checbox was turned "
                                if (isOn)
                                        msg += "on"
                                else
                                        msg += "off"
                                endif
                        else
                                msg = "Disclosure was "
                                if (isOn)
                                        msg += "opened"
```