

သုတေသန

သုတေသန

သုတေသန

သုတေသန

သုတေသန

သုတေသန

သုတေသန

Contents

Contents

Contents

- Release Notes 4
 - Entering and leaving Logo 28
 - Tokenization 29
 - Data Structure Primitives 33
 - Communication 88
 - Arithmetic 133
 - Logical Operations 175
 - Graphics 178
 - Workspace Management 305
 - Control Structures 354
 - Conditional execution 364
 - Loops 371
 - Template Based Iteration 376
 - Macros 387
 - Error Processing 391
 - Special Variables 394
 - GUI programming 396
 - Sound programming 561
-

Release Notes

If you're new to aUCBLogo and experience problems, please read all Release Notes, the bug list, and try to read "Tokenization".

One of the most annoying things is incorrect documentation, so please tell me such things, best including a fix. Also tell me bugs, which are not listed in the bugs directory. This will be very helpful for the development, because stability is of course an issue.

Release Notes

- Release Notes for Version 4.69 4
 - Release Notes for Version 4.689 5
 - Release Notes for Version 4.688 5
 - Release Notes for Version 4.687 5
 - Release Notes for Version 4.686 6
 - Release Notes for Version 4.685 7
 - Release Notes for Version 4.684 13
 - Release Notes for Version 4.683 14
 - Release Notes for Version 4.682 15
 - Release Notes for Version 4.68 15
 - Release Notes for Version 4.672 16
 - Release Notes for Version 4.67 17
 - Release Notes for Version 4.66 18
 - Release Notes for Version 4.65 18
 - Buglist for version 4.65 21
 - Release Notes for Version 4.64 24
 - Berkeley Logo User Manual 27
-

Release Notes for Version 4.69

I wrote a color to HSB function, reHSB and reHSBA, to enable sorting the color database by hue. The color database can now be fetched with getColorDatabase.

I put most work into shadows this time: surfaces, polygons, tessellated polygons and partialEllipsoid's now can produce shadows, if you call enableShadows before their use and the primitive castShadows just before updateGraph. The shadow information stays persistent until you

Release Notes / Release Notes for Version 4.69

call `clearShadows`, so you can easily redraw the hole scene from a different view point with i.e. `rotatescene`, because `redraw` automatically calls `castShadows`, if shadows are enabled. The color of the shadows can be set with `setShadowColor` and queried with `ShadowColor`.

Release Notes for Version 4.689

For my new `IFS3D.lg` I needed `setPixelXYZ` and some fog primitives:

`enableFog` `disableFog` `setFogDensity` `setFogRange` `setFogColor` `setFogMode`

I wanted to be able to use the template based iteration tools with arrays, so I've added some array capabilities to:

`first` `butFirst` `last` `butLast` `firsts` `butFirsts` `fPut` `IPut` `combine`

`LogoVersion` now outputs also details about the operating system version.

To get the screen size I wrote the primitive `OSScreenSize`.

I've added some layouting primitives for the node type `Frame`:

`FrameMaximize` `FrameIconize` `FrameFullScreen`

To set the Stacks into noisy mode (for debugging) you can now call the new primitive `setStackNoisy`. By default the Stacks are not noisy now. Noisy means they print their size when they grow or shrink.

Release Notes for Version 4.688

Inspired by Ken Kahn's `extrapolate11.lg` I wrote an IFS morphing program `ifsmorph.lg`. For it I needed a fast shuffle primitive and `combine` for usage with array types, so I wrote them.

Release Notes for Version 4.687

The nodes persistence now works again in this version. Only `simstring3.lg` has been changed a bit in addition to that, so these Release Notes stay very short.

Release Notes for Version 4.686

This is mainly a bugfix release. The nodes persistence is broken in this version, because I introduced dynamic stacks, which can grow and shrink on demand, and those are not yet fully debugged. I've written a mandelbrot set zoom program, `mandel5.lg`. The new `ifs2.lg` is now much faster, as are the mandel programs, because of new array operations. A simple simulation of a flapping flag that I wrote is `flappingflag.lg`. I wrote an epiano simulation with computed wave files, using `echos` and `lowPassFilters`. Also a nice new demo is a string simulation `simstring3.lg`, which can create a `.wav` file output. And I played a bit with new PC speaker and Midi sound primitives, since they are easier to use than computing every sound for playwave in Logo (which involves complex simulations, for i.e. drums). I wrote a car racing simulation game, `carrace.lg` (but without cars). I've also ported a few games to aUCBLogo: `snake.lg`, `snakes.lg` and `pacman.lg` by David Costanzo, and `juniper9.lg` (Juniper Green) by David Peacock.

I've also included a few new array/list operations. Most important of them is "item" with a list index, which can be used as a lookup table (see `ifs2.lg`!).

Some new syntactic sugar are the new `setitem` operators: `a.X=b` and `a.X.Y=b`, where `a` must be something having elements, and `X` and `Y` can be expressions (without whitespace, still to do), for example `a.(i-2).(j+k)=b`.

A few new or changed primitives are also included:

`MidiCountDevices` `MidiDeviceInfo` `MidiOpen` `MidiClose` `MidiMessage` `MidiProgramChange`
`MidiNoteOn` `MidiNoteOff` `MidiAllSoundsOff` `MidiOutputStream` `MidiOutputStreamsStart`
`MidiOutputStreamsStop` `MidiOutputStreamsFinished`

`Tone` `Tones` `TonesStop` `TonesFinished`

`Sound` `Sounds` `SoundsStop` `SoundsFinished`

`LabelFont` `LabelSize` `LabelWeight` `LabelAlign`

Release Notes / Release Notes for Version 4.686

setPixel setPixelXY

addColorsMod

MandelIterate

BackslashEncode

ConsoleSetFocus

lowPassFilter saturateAbove saturateBelow

deepCopy

Item mdSetItem

shrinkStacks

You can always ask me if you experience problems with aUCBLogo!

<mailto:Andreas.Micheler@Physik.Uni-Augsburg.de> Andreas Micheler

Release Notes for Version 4.685

In trying to built more persistence into aUCBLogo, the console window text can now be automatically saved in the file USER-aucblogo-console, where USER is replaced by the name of the computer user currently logged in. There's a check item in the menu with which you can disable autoload and -save of the console text. The same autosave mechanism is now provided for the picture data, and loadPicture and savePicture now work (I hope) correctly. Additionally I made the new primitives loadPictureText and savePictureText to enable easy exporting and importing of picture data and better debugging.

Finally I achieved persistence of all Logo nodes, so you can run any aUCBLogo program, close aUCBLogo, do something else, start aUCBLogo again and recover your last state. The last state can be running or in midst of a debugging session. This persistent mode can of course be disabled by unchecking "Autosave Nodes" in the File menu, which can be highly wanted if there are many users logged in with the same username.

Printing does now work.

Support for arbitrary precision math using the MAPM library has been added:

BigFloat BigFloatSetPrecision

The primitives Time and FileTime now have the same output format and the MilliSeconds are an additional member.

The predicate FileP / File? is now a primitive.

The new primitive Files outputs the files of the current working directory.

The following primitives are for working with directories:

DirectoryP getWorkingDirectory makeDirectory

The constructors Array, IntArray, Int16Array and FloatArray have been expanded to include a copy constructor, so such data can easily be duplicated.

The primitives saveScreenBMP and saveBMP have been deleted, because there is now saveScreen, which can save the screen in all formats supported by wxWidgets.

Additionally there's now saveScreenVector which uses the GL2PS library to generate various vector graphic formats.

BitMaxX and BitMaxY are new primitives to get the maximal allowed coordinates of a Bitmap obtained by BitCopy or a file.

The shininess of a 3D filled shape can be set now with setMaterialShininess.

To the Sphere primitive has been expanded to optionally take slices and stacks parameters, which gives more control about the drawing finesse.

The primitives readCharExt and rCE have been changed to not output a word with length 1 (a char) but to output an Int, because in the section about KeyboardValue the wxWidgets Keycode constants have been added, which all have numbers ≥ 300 , and that's too big to fit in a char.

It is now possible to break or continue loops by using the break or continueLoop statements.

The nice primitive playWaveFast has been added for use in games and music programs. It can play

Release Notes / Release Notes for Version 4.685

multiple wave files simultaneously.

You now can create a Video for Windows (.avi) file using the new primitives
VideoStart..VideoFrame..VideoEnd.

The Color Database has been extended by lots of extra color names, but all still just in English.

Among other new GUI programming primitives, custom keyboard and mouse event handlers written in Logo are now supported. For the console are the following keyboard and mouse handlers:

OnChar OnKeyDown OnKeyUp KeyboardValue
OnTextMouseLeftDown OnTextMouseRightDown OnTextMouseMiddleDown
OnTextMouseLeftUp OnTextMouseRightUp OnTextMouseMiddleUp
OnTextMouseLeftDClick OnTextMouseRightDClick OnTextMouseMiddleDClick
OnTextMouseMotion

For the main graph window are those mouse handlers:

OnMouseLeftDown OnMouseRightDown OnMouseMiddleDown
OnMouseLeftUp OnMouseRightUp OnMouseMiddleUp
OnMouseLeftDClick OnMouseRightDClick OnMouseMiddleDClick OnMouseMotion

You can now set the mouse cursor to "busy" or normal and ask if mouse is busy with the following commands:

beginBusyCursor endBusyCursor BusyCursor?

The following standard dialogs for getting information from the user are now available:

DirSelector FileSelector getColorFromUser getFontFromUser getMultipleChoices
getNumberFromUser getPasswordFromUser getTextFromUser
getSingleChoice getSingleChoiceIndex MessageBox

For cleanup of texture memory there's now deleteTextures, so a texture memory overflow can be prevented, if you create many textures which you maybe only use once.

When you use Graph windows with a defined shape then setScreenRange might be useful to get away from the default [800,600] logical pixels.

setPixel can now also take 3D coordiantes as arguments.

For easier and faster plotting the new command setPixelXY has been written: It takes structured X data and equally structured Y data as separate arguments.

The DLCall primitive supports now argument-by-pointer when "IntPtr, "Int16Ptr, "Int8Ptr, "UInt8Ptr or "FloatPtr are specified as argument type.

For customizing the Logo environment, file and dir locations, the most important can now be read and changed:

LogoComspec setLogoComspec LogoEditor setLogoEditor LogoHelpDir setLogoHelpDir
LogoLibDir setLogoLibDir LogoTempDir setLogoTempDir

Windows (using the constructor Frame), multiple Graph windows (using the constructor Graph) and many Controls from the wxWidgets framework are now included.

New primitives for usage with Frame:

FrameDestroy FrameOnChar FrameOnKeyDown FrameOnKeyUp FrameSetFocus FrameEnable
FrameSetClientSize FrameSetColor FrameSetBackgroundColor
FrameSetFontSize FrameSetFontName FrameSetFontStyle FrameSetFontWeight
FrameSetShape FrameSetSizer

New primitives for usage with Graph:

GraphDestroy GraphCurrent GraphSetCurrent GraphOnChar GraphOnKeyDown GraphOnKeyUp
GraphOnMouseLeftDown GraphOnMouseRightDown GraphOnMouseMiddleDown
GraphOnMouseLeftUp GraphOnMouseRightUp GraphOnMouseMiddleUp
GraphOnMouseLeftDClick GraphOnMouseRightDClick GraphOnMouseMiddleDClick
GraphOnMouseMove

The main window layout mechanism of wxWidgets is supported with the constructor BoxSizer and the applicable primitives:

BoxSizerAdd BoxSizerDestroy

Be aware of the command FrameSetSizer: it enables a sizer on a Frame.

The new controls are (here follow the constructors alphabetically, the detailed listing of primitives afterwards):

Button CheckBox ChoiceBox ComboBox FloatControl

Release Notes / Release Notes for Version 4.685

Gauge IntControl ListBox ListControl RadioButton Slider StaticText TextControl ToggleButton

New primitives for usage with Button:

ButtonDestroy ButtonOnClick ButtonEnable

New primitives for usage with CheckBox:

CheckBoxDestroy CheckBoxOnClick CheckBoxValue CheckBoxSet CheckBoxEnable

New primitives for usage with ChoiceBox:

ChoiceBoxDestroy ChoiceBoxSelection ChoiceBoxSetSelection ChoiceBoxSetChoices
ChoiceBoxAppend ChoiceBoxSetItem ChoiceBoxRemoveItem ChoiceBoxCount
ChoiceBoxSetBackgroundColor ChoiceBoxSetColor
ChoiceBoxSetFontSize ChoiceBoxSetFontName
ChoiceBoxSetFontStyle ChoiceBoxSetFontWeight
ChoiceBoxOnChar ChoiceBoxOnKeyDown ChoiceBoxOnKeyUp ChoiceBoxOnSelect
ChoiceBoxEnable

New primitives for usage with ComboBox:

ComboBoxDestroy ComboBoxSelection ComboBoxSetSelection ComboBoxSetChoices
ComboBoxAppend ComboBoxSetItem ComboBoxRemoveItem ComboBoxCount
ComboBoxValue ComboBoxSetValue ComboBoxSetBackgroundColor ComboBoxSetColor
ComboBoxSetFontSize ComboBoxSetFontName
ComboBoxSetFontStyle ComboBoxSetFontWeight
ComboBoxOnChar ComboBoxOnKeyDown ComboBoxOnKeyUp
ComboBoxOnSelect ComboBoxOnChange ComboBoxOnEnter ComboBoxEnable

New primitives for usage with FloatControl:

FloatControlDestroy FloatControlValue FloatControlSetValue
FloatControlSetRange FloatControlOnChange FloatControlEnable

New primitives for usage with Gauge:

GaugeDestroy GaugeValue GaugeSetValue GaugeSetRange
GaugeSetColor GaugeSetBackgroundColor

New primitives for usage with IntControl:

IntControlDestroy IntControlValue IntControlSetValue IntControlSetRange IntControlOnChange
IntControlEnable

New primitives for usage with ListBox:

ListBoxDestroy ListBoxSelections ListBoxSetSelections ListBoxSetChoices ListBoxAppend
ListBoxSetItem ListBoxRemoveItem ListBoxCount ListBoxSetBackgroundColor ListBoxSetColor
ListBoxSetFontSize ListBoxSetFontName ListBoxSetFontStyle ListBoxSetFontWeight
ListBoxOnChar ListBoxOnKeyDown ListBoxOnKeyUp ListBoxOnSelect ListBoxOnDClick
ListBoxEnable

New primitives for usage with ListControl:

ListControlDestroy ListControlInsertColumn ListControlInsertItem
ListControlSetItem ListControlGetItem ListControlDeleteItem
ListControlSetRow ListControlSetColumn ListControlSet
ListControlGetRow ListControlGetColumn ListControlGet
ListControlItemCount ListControlColumnCount
ListControlColumn ListControlRow ListControlText ListControlSort
ListControlSetBackgroundColor ListControlSetColor
ListControlSetFontSize ListControlSetFontName
ListControlSetFontStyle ListControlSetFontWeight
ListControlOnChar ListControlOnKeyDown ListControlOnKeyUp
ListControlOnItemSelected ListControlOnItemActivated ListControlOnColClick
ListControlEnable

New primitives for usage with RadioButton:

RadioButtonDestroy RadioButtonOnClick RadioButtonValue RadioButtonSet RadioButtonEnable

New primitives for usage with Slider:

SliderDestroy SliderValue SliderSetValue SliderSetRange SliderOnScroll SliderEnable

New primitives for usage with StaticText:

StaticTextDestroy StaticTextLabel StaticTextSetLabel
StaticTextSetBackgroundColor StaticTextSetColor StaticTextSetFontSize StaticTextSetFontName
StaticTextSetFontStyle StaticTextSetFontWeight

Release Notes / Release Notes for Version 4.685

New primitives for usage with TextControl:

TextControlDestroy TextControlValue TextControlSetValue TextControlWrite TextControlAppend
 TextControlSetInsertionPointEnd TextControlCursor TextControlSetCursor
 TextControlInsertMode TextControlOverwriteMode
 TextControlSetBackgroundColor TextControlSetColor
 TextControlSetFontSize TextControlSetFontName
 TextControlSetFontStyle TextControlSetFontWeight
 TextControlOnChar TextControlOnKeyDown TextControlOnKeyUp
 TextControlOnChange TextControlOnEnter TextControlEnable

New primitives for usage with ToggleButton:

ToggleButtonDestroy ToggleButtonOnClick ToggleButtonSetValue ToggleButtonValue
 ToggleButtonEnable

And for the debugging of the Garbage Collector (and for my amusement) I wrote a memory window, accessible through the view menu, where one can inspect every Node by pointing with the mouse to it: a tooltip window with a textual representation of that Node will show up then. It's pretty interesting to see the GC working!

You can always ask me if you experience problems with aUCBLogo!

 Andreas Micheler

Release Notes for Version 4.684

This Release is mainly because I've updated the Linux version of aUCBLogo.

I also changed the "Pause" and "Stop" hot keys from [trl-Q] and [Ctrl-P] to [Pause] and [Ctrl-Pause].

Additional hot keys are now [F2]="repeat last command" and [Ctrl-F2]="reset" (calls "reset.lg", so it's user-definable).

For the beginners there's now a popup menu on a right mouse click, with which one can insert primitives, choosen by category like in the help.

I added the new type FloatArray to enable faster array computing, which is sometimes highly wanted, because it's about five times as fast as with conventional Arrays. To support the new type I also wrote rSeqFloatArray, abbreviated rSeqFA, which does nearly the same as rSeq, but with a FloatArray output.

Inspired by Michael Malien I wrote a little Profiler (profile.lg, with demos testprofile.lg, testprofile2.lg and testprofile3.lg). To ask if a given character is in some subset of characters, I added the new predicates AINum?, Alpha?, ASCII?, Cntrl?, CSym?, Digit?, Graph?, Lower?, Print?, Punct?, Space?, Upper?, and xDigit?. They map directly to the C functions with similar names (isalpha etc.).

I improved bounce3.lg a bit. Now the ball is textured and rotates, depending on in which direction it flies. For this the new primitives spinX, spinY and spinZ have been written, and to enable the overwriting of drawn graphics I added setDepthFunc, supporting the choosing of the OpenGL depthFunc.

I also played with explosion.lg and added the demo 3drohre.lg, which is modeled after the screen saver "3D Pipes".

To demonstrate the speed of the new FloatArray type I developed makewav5.lg and makewav6.lg. In makewav6.lg is a realtime oscilloscope shown during the playing of the computed wave.

Mike Sandy inspired me to include NameInTable? .

And I've fixed a few very bad Array bugs in the unary and binary math functions.

Release Notes for Version 4.683

BitLoad & BitSave have been removed, because there are now loadImage and saveImage, which know all file formats which wxWidgets knows.

The texturing has been improved (see throwcoin.lg, house2.lg and landscape4.lg!).

New and changed primitives are:

FileTime

BitSetPixel BitPixel

Release Notes / Release Notes for Version 4.683

Texture

loadImage saveImage

setTexXY setTexPos

setTessWindingRule

Release Notes for Version 4.682

The IDE has been improved a lot.

F9 always runs the program from source code line, where the program is at that time.

F7 always sets the program into single stepping mode and executes one step. Buried procedures are stepped over.

Control-F8 sets a breakpoint in the source code line at the cursor's position.

The Vars and Calls windows now are much faster and don't flicker anymore.

Internally now my PtrDebug smart pointer checking templates work again, and memory leak checking is available in the debug built of aucblogo.exe. This is important to improve the stability, I found several related bugs.

And I found most of the bugs in bounce3.lg and am2.lg too.

Changed primitives are: edit editFile

A new primitive is updateVarsOnStep

Release Notes for Version 4.68

There's now an integrated debugging environment (IDE) included, where you can singlestep in the source code. The edit.exe is now obsolete and is not included any more because there is the internal

editor. Also errors jump directly to the source code line, where something was wrong. A simple example of the debugging capabilities shows `testedit.lg`. The lib files now all have a filename ending `".lg"`.

No new primitives and therefore no new help. For online html help use the 4.672 help version.

I can make a Linux version of the new 4.68 Release if anyone's interested, just send me an email (the adress is at the bottom of my web page).

The mastermind game by Brian Harvey now also runs fine with the new `delayed-list-evaluation-streams`.

Release Notes for Version 4.672

This is mainly a bugfix release.

There was a bug in `"for"`, one in the splitter window class, one in `setItem`, in `Sphere` and several other minor bugs.

New in this release are the primitives:

`DynamicLibrary` `DynamicLibraryCall` `DLCall`

`BitItem` `Items` `setItems`

`StringBuffer` `StringBufferToWord`

`setMaterialAmbient` `setMaterialDiffuse` `setMaterialSpecular` `setMaterialEmission`

`setVarsSplitter` `setCallsSplitter`

`TextMousePos` `TextMouseX` `TextMouseY`

`WordUnderCursor` `setTextSelection`

`enableTextMouseEvents` `disableTextMouseEvents`

And the primitive `splitScreen` and its abbreviation `ss` have been rewritten to optionally accept a

Release Notes / Release Notes for Version 4.672

splitting ratio. The window splitter class has been rewritten in parts, now the splitters should work correctly.

The window position and size including the splitter ratios are now saved into the registry on closing aUCBLogo.

PolyStart and TessStart have now been changed to not include the current pos into the polygon. PolyStart is now like in MSWLogo.

edit.exe is updated, too.

And I have been working through all the logo demos, they should work now all fine.

Release Notes for Version 4.67

The "edit" executable is new. It can replace Crimson Editor in most issues, and is for Logo programs even a bit nicer. It's based on wxStyledTextControl, better on the StcTest example of wxWidgets-2.6.1, but somewhat extended. I have written a Lexer module especially for aUCBLogo which highlights the syntax almost completely correct. The syntax highlighting can easily be deactivated by choosing View/Hilight language/<default>. The "Run" command bound to F9 can be changed, also the location of the help file bound to F1, so that aUCBLogo can be in any path location. F4 jumps to the procedure named like the word under the cursor, which is nice for symbol browsing. There are also some extras in the Extra menu: converting case of symbols to correct aUCBLogo case, and convert "make" to "=".

I've created some new Logo types for usage with binary data:

Int16 Int8 UInt8 IntArray Int16Array Struct

New and changed primitives are:

Int8 Int16 UInt8 IntArray Int16Array Struct

SizeOf TypeOf

toList

setWriter setReader

readIntBin readInt16Bin readInt8Bin readUInt8Bin readFloatBin readComplexBin
readIntArrayBin readInt16ArrayBin readStructBin

FileSize

LogoVersion

boldTextMode plainTextMode setTextSize CharUnderCursor

mandelIterate

GraphicStart GraphicEnd drawGraphic

allFullScreen notFullScreen

enableLighting disableLighting enableDither disableDither enablePointSmooth disablePointSmooth

playWave

PortOut PortIn setPortBit clearPortBit getPortBit notPortBit leftPortShift rightPortShift

waituS

setPC setFC setSC

New and changed library procedures are:

arc2

transposematrix

vowelp

Release Notes for Version 4.66

There's not much new to say. I only have ported aUCBLogo to wxWidgets and made a Linux version. Because this was yet a bit of work, therefore I released the new version.

Release Notes / Release Notes for Version 4.65

Release Notes for Version 4.65

aUCBLogo-4.65 is a port of my UCBLogo version to the OpenGL graphics library. The core has only slightly been changed since the last version (4.64): Two new Warnings have been added, so you can easier port programs from UCBLogo and MSWLogo. They're displayed when a variable has the same name as a proc or a primitive.

Supported platform is still only Windows 9x and newer, where OpenGL exists. Porting aUCBLogo to wxWindows and Linux is on my project list.

New are antialiased lines, polygon rendering and texture mapping (with bugs, but nice though).

I have added a reset.lg to reset the interpreter to a known state.

Also new is the (now working) dir (because shell does work better) and dirlg.lg. dirlg is particularly useful for browsing the examples in combination with reset.

I have tried to make all old demos work with the new graphics engine, but the projection matrix functions are still not working (spelltest.lg).

On some graphics cards (i.e. Intel Extreme Graphics) there's strange OpenGL SwapBuffers() behavior, therefore I have added the command singleBuffer (and to reset: doubleBuffer).

Also, to use the free Crimson Editor, I changed the startup behavior. Now if you call Logo.exe with a xxx.lg file name parameter (logo.exe xxx.lg), then the file will be loaded and the procedure xxx will be executed.

This is cool because so you can write your Logo programs in an editor like Crimson and then execute your program with just one hot key (if you assigning one, which I intensely recommend). The Crimson syntax file in the Crimson subdirectory is of course updated, too. You should copy the /Crimson/ directory to your Crimson Programs directory to enable syntax highlighting.

A MS HTMLHelp file (aUCBLogo.chm) is new in this package. So you can call in Logo the help with F1, and even better, include the .chm in Crimson, so you have the F1 help there, too (using Crimson's Tools/Conf. User Tools menu to configure).

The comparison operators now should work, even with strings. If one of the arguments is a number, then the other argument will be converted to a number if possible. If not possible, then a string compare is done.

On converting from MSWLogo or UCBLogo you should replace all dots . to _ because the dot . is

the item operator in aUCBLogo. Also, = should be changed to ==, because = is the assignment operator, == is the comparison operator. But the biggest hurdle are variables which have the same name as a procedure or a primitive. In such a case I mostly rename i.e. a variable "list" to "list_".

Also, I've ported some of the cool CSLS demos to work with aUCBLogo (basic, pascal, diff).

Changed and new primitives are:

< <= == > >=

ArcSin ArcCos radArcSin radArcCos

ignore replace

cd changeDir

doubleBuffer singleBuffer redraw saveSize setSaveSize

BitCopy BitPaste BitLoad BitMakeTransparent bitTrans BitSave

enableLineSmooth enLS disableLineSmooth disLS enablePolySmooth enPS
disablePolySmooth disPS enableRoundLineEnds enRLE disableRoundLineEnds disRLE

perspective unperspective setEye setPS

Ellipse fillEllipse fillPie fillRect

reRGBA RGBA HSBA addColors

setXYZ setZ PosXYZ setPosXYZ

DistanceXYZ towardsXYZ

setSpherePos setCylinderPos

Roll leftRoll rightRoll

Orientation setOrientation

PolyEnd PolyStart

Release Notes / Release Notes for Version 4.65

Sphere Ellipsoid

setLightPos setLightAmbient setLightDiffuse setLightSpecular

Texture enableTexture enTex disableTexture disTex

new library procedures are:

dir dirlg reset check

axes rotatescene

displaymatrix

BitPasteT has been deleted because now there is BitMakeTransparent, and OpenGL provides Alpha channel support (transparence).

PrintSize, SetPrintSize are deleted, because printing chooses the sizes automatically. And saving is controlled via setSaveSize, saveSize.

movePixel and moveLine are deleted, because there is double buffering.

For questions concerning this special version of good old UCBLLogo you may ask Andreas Micheler.

Buglist for version 4.65

I don't love to tell you those bugs, but I think it's better than leaving you hanging.

1) ;~+ Comment bug:

```

;~comment
; I don't know how to ~comment

```

but even worse:

```

;~+
No way! This is bogus!; I don't know how to ~
continue ; Pausing...

```

2) Procedure redefinition bug (wrong error message):

```
circle2
; []
doesn't like Unbound as input
```

circle2.lg is:

```
to circle r
  arc 360 r
end
```

3) Arcbug (arcbug.lg): on resizeing the screen some rubbish is drawn.

```
to arcbug
  cs
  fd 400
  lt 180
  arc 15 100
end
```

4) "=" bug (wrong error message):

```
show a=1
; = didn't output to []
```

5) another "=" bug:

```
i1=0
show i1
Unbound
```

Workaround is adding a space after a symbol with numeric end:

```
j1 =0
show j1
0
```

6) "+=" bug: the number input is not copied but changed.

```
a=0
b=a
a+=1
b
1 ; -)
```

Workaround: use a=int for initialising when using "+="!

Release Notes / Buglist for version 4.65

7) empty line "end" bug:

```
end

end
t defined
t
1234
; I don't know how to end in t
pr 1234 end
```

Workaround: don't use empty lines before "end".

9) setItem operator "a.b=c" bug:

```
a=[ ]
a.1=0
```

crashes the interpreter very badly.

10) ".="-bug:

```
a.=1
a ; -)
```

prints a bad error message.

11) "Shell"-bug: the output is cluttered with empty lines.

12) "a -1 -bug:

```
"a -1
a ; -)
```

should output like this:

```
"a - 1
[a - 1]
; -)
```

13) "a=1e-6"-bug:

```
a=1e-6
; I don't know how to 1e-6
```

Workaround is adding a space:

```
a= 1e-6
```



```
a
1e-006  ; -)
```

14) "=="-bug:

```
(1+1)==2
; too much inside ()'s
```

Workaround is adding a space:

```
(1+1) == 2
true  ; -)
```

15) "1-.5"-bug:

```
1-.5
; not enough inputs to .
```

^Workaround is adding a space:

```
1- .5
0.5  ; -)
```

16) case ignored bug:

```
pr "M  ;should print "M , not "m!
```

Release Notes for Version 4.64

This is a major rewrite of the core of UCBLLogo including the garbage collector, now with typesafe C++ classes, many new enums, templates for GC debugging and with a speed gain of about 20-40 times :-))) Top speed is better than 310 CPU clock cycles per logo instruction, which is not too bad for an interpreter. This was possible by the introduction of stacks and so the avoidance of new constructors during the runtime, which charged the GC too much.

The command line is now a RichEdit control with all its advantages. CTRL-ENTER inserts a new line, ENTER executes the line. There is no prompt any more (except some errors when pausing), because every line should be executable. The error "You don't say what to do with" now is removed in favour of the result with the comment ;-) so you can make fast computations by hand if you need them. Line continuations ~ are not needed any more.

e calls the editor (write the following settings to your autoexec.bat or into your system environment on Windows XP:

Release Notes / Release Notes for Version 4.64

```
set AEDITOR=C:\Windows\Notepad
set ALOGOLIB=C:\aUCBLogo\Lib
set ALOGOHELP=C:\aUCBLogo\Help).
```

h [command] opens a web browser with the help page for that command. You will have to change the directories in h.lg for correct operation.

A bit of new syntax has been introduced to make unquoted variable assignments and colonless variables possible, like

```
a_variable=some_other_var
b=[1 2 3]
c=3.14
d+=pi
```

Equality can be checked with ==, as in

```
show true==false
false
```

A variable now has higher priority than a procedure of the same name.

Also some new primitives are included:

repeatCount, ShellSpawn, traced, setCaseIgnored, CaseIgnoredP,

Matrix, setMatrix, TurtleMatrix, TM, setTurtleMatrix, setTM, IdentityMatrix, IDM,
setIdentityMatrix, setIDM, Pixel, setPixel, Line, movePixel, moveLine, fillRect, fill, fillEllipse,
setFC, setFloodColor, FloodColor, RGB, reRGB, HSB, addColors, savePicture, savePic,
loadPicture, loadPic, savePostScript, savePS,

CharP, intP, floatP, complexP, readCharExt, rCE, rotate, ArrayToList, ListToArray, merge,
mergePairs, toListList, mergeSort, _setButFirst, PropertyList, getProperty, putProperty,
removeProperty,

Faculty, primeP, factorize, =, lessEqualP, greaterEqualP, <=, >=, min, max, Norm, maxNorm, resize,
rnd, cross, invertMatrix, trunc, Tan, mod, ^, radTan, abs, xCopy, xAdd, xSub, xMul, xDiv, xMod,
+=, -=, *=, /=,

Arity, Primitives, PropertyLists, printOut, printOutTitles, eraseAll, eraseProcedures, eraseNames,

erasePropertyLists,

timeFine, timeMilli, TimeU, TimeURes, TimerFreq, time, MIPS, Tone, playWave, waitMS,

setTextFont, insertMode, overwriteMode, setPrintPrecision

LabelSize, setLabelSize, setLabelFont, setLabelWeight, setLabelAlign.

PrintSize, SetPrintSize, saveScreenBMP, saveBMP, setUpdateGraph, updateGraph, updateVars, Calls, updateCalls, refreshP, dispatchMessages, BitCopy, BitPaste, BitPasteT, scroll, scrollCalibrate, scrollCal

Multiple turtles: Turtle, newTurtle, setTurtle

3D: perspective, unperspective, setEye, leftRoll,lR, rightRoll,rr, upPitch,uP, downPitch,down, setZ, setXYZ, setPosXYZ, _setPosXYZ, setSpherePos, setCylinderPos, xCor, yCor, zCor, PosXYZ, setRoll, setPitch, setOrientation, Roll, Pitch, Orientation, towards, towardsXYZ, Distance, DistanceXYZ.

And some functions have been extended, especially the math functions, which can now take nested lists and arrays and complex numbers like

```
{[1 2] 3.14 {4 5 [6 7]}}+{[1 2] 3.14 {4 5 [6 7]}}
{[2 4] 6.28      {8 10 [12 14]}}
}  ; -)
```

```
(1+2i)*(2+4i)
-6+8i    ; -)
```

```
1+2e3i
1+2000i    ; -)
```

```
exp pi*2i
-1+1.22460635382238e-016i    ; -)
```

unary functions: BitNot, random, Int, round, truncate, Sin, Cos, Tan, radSin, radCos, radTan, Sqr, Sqrt, exp, LN, Log10, real, imag, conjugate

binary functions: +, -, *, /, Remainder, aShift, lShift, BitOr, BitAnd, BitXOr, Power, ArcTan, radArcTan

Release Notes / Release Notes for Version 4.64

Shell acts now more like in unix, it gives back the output in form of a list, but it has still some bugs.

A syntax highlighting scheme for the free Proton Editor is included (aUCBLogo.sch), another editor scheme is for the Crimson editor (crimson.zip).

For questions concerning this special version of good old UCBLogo you may ask Andreas Micheler.

Berkeley Logo User Manual

Copyright (C) 1993 by the Regents of the University of California

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS for A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

This is a program that is still being written. Many things are missing, including adequate documentation. This manual assumes that you already know how to program in Logo, and merely presents the details of this new implementation. Read

Computer Science Logo Style, Volume 1:

Symbolic Computing by

<a href="http://www.cs.berkeley.edu/~bh/"

target="_parent">Brian Harvey (MIT Press, 1997)

for a tutorial on Logo programming with emphasis on symbolic computation. Or you can look at the Logo Forum

[comp.lang.logo](news:comp.lang.logo)
for interesting information and intensive discussions.

Here are the special features of this dialect of Logo:

Source file compatible among Unix, DOS, and Mac platforms.
--

Random-access ARRAYS.

Variable number of inputs to user-defined procedures.

MUTATORS for LIST structure (dangerous).
--

PAUSE on ERROR, and other improvements to ERROR PROCESSING.

Comments and continuation lines; formatting is preserved when PROCEDURE DEFINITION(s) are saved or edited.

Terrapin-style TOKENIZATION (e.g., [2+3] is a LIST with one member)
--

but LCSI-style syntax (no special forms except TO). The best of both worlds.

First-class instruction and expression templates (see APPLY).

MACROS.

Features **not** found in Berkeley Logo include robotics, music, GUIs, animation, parallelism, and multimedia. For those, buy a commercial version.

Entering and leaving Logo

The process to start Logo depends on your operating system:

Entering and leaving Logo

Unix: Type the word `aucblogo` to the shell.
(The directory in which you've installed Logo must be in your path.)

Windows: Double-click on `aucblogo.exe` in the `aucblogo` folder.

To leave Logo, enter the command `"bye"`.

Under Unix or DOS, if you include one or more filenames on the command line when starting Logo, those files will be loaded before the interpreter starts reading commands from your terminal. If you load a file that executes some program that includes a `"bye"` command, Logo will run that program and exit. You can therefore write standalone programs in Logo and run them with shell/batch scripts. To support this technique, Logo does not print its usual welcoming and parting messages if you give file arguments to the `logo` command.

If you type your interrupt character (see table below) Logo will stop what it's doing and return to `toplevel`, as if you did `THROW "TOPLEVEL`. If you type your quit character Logo will pause as if you did `PAUSE`.

<code>toplevel</code>	[Ctrl-Pause]
<code>pause</code>	[Pause]

If you have an environment variable called `ALOGOLIB` whose value is the name of a directory, then Logo will use that directory instead of the default library. If you invoke a procedure that has not been defined, Logo first looks for a file in the current directory named `proc.lg` where `"proc"` is the procedure name in lower case letters. If such a file exists, Logo loads that file. If the missing procedure is still undefined, or if there is no such file, Logo then looks in the library directory for a file named `proc.lg` and, if it exists, loads it. If neither file contains a definition for the procedure, then Logo signals an error. Several procedures that are primitive in most versions of Logo are included in the default library, so if you use a different library you may want to include some or all of the default library in it.

Tokenization

Names of procedures, variables, and property lists are case-insensitive. So are the special words `END`, `TRUE`, and `FALSE`. Case of letters is preserved in everything you type, however.

Within square brackets, words are delimited only by spaces and square brackets.

Tokenization

`[2+3]`

is a list containing one word. Note, however, that the Logo primitives that interpret such a list as a Logo instruction or expression (RUN, IF, etc.) reparse the list as if it had not been typed inside brackets.

After a quotation mark outside square brackets, a word is delimited by a space, a square bracket, or a parenthesis.

A word not after a quotation mark or inside square brackets is delimited by a

space, a bracket, a parenthesis,
or an infix operator `+ - * / ^ = < > += -= *= /= == != <= >=.`

Note that words following colons are in this category. Note that quote and colon are not delimiters.

A word consisting of a question mark followed by a number

`?37`

when unparsed (i.e., where a procedure name is expected), is treated as if it were the sequence

`(? 37)`

making the number an input to the `? procedure`. (See the discussion of templates, below.) This special treatment does not apply to words read as data, to words with a non-number following the question mark, or if the question mark is backslashed.

A line (an instruction line or one read by READLIST or READWORD) can be continued onto the following line if its last character is a

tilde `~`.

READWORD preserves the tilde and the newline; READLIST does not.

Line Continuation: An instruction line or a line read by READLIST (but not by READWORD) is automatically continued to the next line, as if ended with a tilde, if there are unmatched

brackets, parentheses, braces, or vertical bars

pending. However, it's an error if the continuation line contains only the word END; this is to prevent runaway procedure definitions. Lines explicitly continued with a tilde avoid this restriction.

If a line being typed interactively on the keyboard is continued, either with a tilde or automatically, Logo will display a tilde as a prompt character for the continuation line.

A semicolon begins a comment in an instruction line. Logo ignores characters from the semicolon to the end of the line. A tilde as the last character still indicates a continuation line, but not a continuation of the comment.

Tokenization

For example, typing the instruction

```
print "abc;comment ~
def
```

will print the word abcdef. Semicolon has no special meaning in data lines read by READWORD or READLIST, but such a line can later be reparsed using RUNPARSE and then comments will be recognized.

To include an otherwise delimiting character (including semicolon or tilde) in a word, precede it with

```
backslash \ .
```

If the last character of a line is a backslash, then the newline character following the backslash will be part of the last word on the line, and the line continues onto the following line.

To include a backslash in a word, use \\.

If the combination backslash-newline is entered at the terminal, Logo will issue a backslash as a prompt character for the continuation line. All of this applies to data lines read with READWORD or READLIST as well as to instruction lines. A character entered with backslash is EQUALP to the same character without the backslash, but can be distinguished by the BACKSLASHEDP predicate. (However, BACKSLASHEDP recognizes backslashedness only on characters for which it is necessary:

```
whitespace, parentheses, brackets, infix operators,
backslash, vertical bar, tilde, quote,
question mark, colon, and semicolon.)
```

An alternative notation to include otherwise delimiting characters in words is to enclose a group of characters in vertical bars. All characters between vertical bars are treated as if they were letters. In data read with READWORD the vertical bars are preserved in the resulting word. In data read with READLIST (or resulting from a PARSE or RUNPARSE of a word) the vertical bars do not appear explicitly; all potentially delimiting characters (including spaces, brackets, parentheses, and infix operators) appear as though entered with a backslash. Within vertical bars, backslash may still be used; the only characters that must be backslashed in this context are backslash and vertical bar themselves.

Characters entered between vertical bars are forever special, even if the word or list containing them is later reparsed with PARSE or RUNPARSE. Characters typed after a backslash are treated somewhat differently: When a quoted word containing a backslashed character is runparsed, the backslashed character loses its special quality and acts thereafter as if typed normally. This distinction is important only if you are building a Logo expression out of parts, to be RUN later, and want to use parentheses.

For example,

```
PRINT RUN (SE "\ ( 2 " + 3 "\ ) )
```

will print 5, but

```
RUN (SE "MAKE " | ( | 2 )
```

will create a variable whose name is open-parenthesis. (Each example would fail if vertical bars and backslashes were interchanged.)

Data Structure Primitives

...are functions to construct data out of one or more inputs (constructors),

functions which select subranges of the input data (selectors),

functions which change the data itself (mutators),

functions asking for boolean properties of the input data (predicates),

and functions asking for non-boolean properties of the input data (queries).

Data Structure Primitives

- Constructors 34
 - Selectors 49
 - Mutators 57
 - Predicates 64
 - Queries 81
-

Constructors

...are functions outputting new data formed by their inputs.

Constructors

- Word 34
- StringBuffer 35
- StringBufferToWord 35
- List 35
- Sentence 36, Se 36
- fPut 36
- lPut 37
- Array 37
- IntArray 38
- Int16Array 39
- FloatArray 40
- mdarray 40
- Table 41
- Struct 42
- toList 43
- combine 44
- reverse 44
- rotate 45
- shuffle 45
- merge 46
- mergePairs 46
- toListList 46
- mergeSort 47
- genSym 47
- replace 47
- deepCopy 48

Word *word1 word2*
(**Word** *word1 word2 word3 ...*)

outputs a word formed by concatenating its inputs.

Examples:

```
show word "hal "lo      ;hallo
show (word [wor ld])    ;wor ld
show (word {a b c}) ;{a b c}
show word "12 "34      ;1234
```

StringBuffer *length*

outputs a word of *length length*, containing only null characters. This may be necessary if one uses DLCall and needs a string buffer.

Example:

```
to GetCurrentDirectory
  local [buf status w]
  buf=StringBuffer 256
  status=DLCall kernel32 [GetCurrentDirectoryA] (list "Int
    "bufferLength "Int 4*count buf
    "buffer "Word buf)
  output StringBufferToWord buf
end
kernel32=DynamicLibrary "kernel32
show GetCurrentDirectory
```

StringBufferToWord *buf*

outputs a new word which contains the first characters of the StringBuffer *buf* up to the first null character. This function only makes sense with DLCall, else you'll probably never use it. For an example see StringBuffer.

List *thing1 thing2*

(List *thing1 thing2 thing3 ...*)

outputs a list whose members are its inputs, which can be any Logo datum (word, list, or array).

Examples:

```
show list "hallo "world    ;[hallo world]
show list [] []           ;[[] []]
show (list "this "is "a "longer "List) ;[this is a longer list]
show list {arrays in}{a List}    ;[{arrays in} {a List}]
```

Sentence *thing1 thing2*

Se *thing1 thing2*

(Sentence *thing1 thing2 thing3 ...*)

(Se *thing1 thing2 thing3 ...*)

outputs a list whose members are its inputs, if those inputs are not lists, or the members of its inputs, if those inputs are lists.

Examples:

```
show se "Hallo "World ;[hallo world]
show se [Hallo][World] ;[Hallo World]
show (se [This is] "a "mixed [longer] [Sentence])
;[This is a mixed longer Sentence]
show (se "word [list] {array}) ;[word list {array}]
```

fPut *thing list*

fPut *thing array*

outputs a *list* equal to its second input with one extra member, the first input, at the beginning.

fPut is faster than lPut, because lists are in Logo built out of first-tail pairs.

Data Structure Primitives / Constructors / fPut

Array types are now supported (4.689).

Examples:

```
show fput "Hallo [World]    ;[hallo World]
show fput [Hallo] [World]   ;[[Hallo] World]
show fput "12 "34
; fput doesn't like 34 as input
show fput "12 [34] ;[12 34]
show fput "12 {34}      ;{12 34}
show fput 12 FloatArray {34}          ;{12 34}
```

lPut *thing list*

outputs a *list* equal to its second input with one extra member, the first input, at the end.

lPut is slow because internally the *list* must be gone through till its end, then the *thing* can be appended.

Array types are now supported (4.689).

Examples:

```
show lput "World [Hallo]    ;[Hallo world]
show lput [World] [Hallo]   ;[Hallo [World]]
show lput 34 12
; lput doesn't like 12 as input
show lput 34 [12]          ;[12 34]
show lput 34 {12}          ;{12 34}
```

Array *size*

(Array *size origin*)

Array *alist***Array *aintarray*****Array *aint16array***

Array *anarray*

outputs an array of " *size* " members (must be a positive integer), each of which initially is an empty list. Array members can be selected with `Item` and changed with `setItem`. The first member of the array is member number 1 unless an " *origin* " input (must be an integer) is given, in which case the first member of the array has that number as its index. (Typically 0 is used as the *origin* if anything.) Arrays are printed by `PRINT` and friends, and can be typed in, inside curly braces. `@` indicates an *origin* . This primitive also can be used to convert *alist* , *aint16array* or *aintarray* to an array, and to clone the array *anarray* .

Examples:

```
show array 3          ;{[] [] []}
show (array 3 0)      ;{[] [] []}@0
show {h a l l o}      ;{h a l l o}
show {a b c}@2        ;{a b c}@2
show Item 3 {a b c}@2 ;b
```

```
a={H a l l o}
setItem 2 a "e
show a      ;{H e l l o}
Array [1 2 3] ;{1 2 3}  ;-)
Array intarray [1 2 3] ;{1 2 3}  ;-)
Array int16array [1 2 3] ;{1 2 3}  ;-)
```

IntArray *size*

(IntArray *size origin*)

IntArray *anarray*

IntArray *alist*

IntArray *aint16array*

IntArray *aintarray*

outputs an IntArray of " *size* " members (must be a positive integer), each of which is an `Int`. Array members can be selected with `Item` and changed with `setItem`. The first member of the IntArray is member number 1 unless an " *origin* " input (must be an integer) is given, in which case the first member of the IntArray has that number as its index. (Typically 0 is used as the *origin* if

Data Structure Primitives / Constructors / IntArray

anything.) Arrays are printed by PRINT and friends. This primitive also can be used to convert *alist* , *anarray* or *anint16array* to IntArray, while clipping the values to the max and min integer. It can also be used to clone *anintarray* .

Examples:

```
show IntArray 3 ;{0 0 0}
show (IntArray 3 0) ;{0 0 0}@0
show IntArray {1 2 3} ;{1 2 3}
show IntArray [1 2 3] ;{1 2 3}
show IntArray int16array {1 2 3} ;{1 2 3}
show intarray {a b c} ; intarray doesn't like {a b c} as input
show Item 3 intarray [1 2 3] ;3
```

Int16Array *size*
(Int16Array *size origin*)
Int16Array *anarray*
Int16Array *alist*
Int16Array *anintarray*
Int16Array *anint16array*

outputs an Int16Array of " *size* " members (must be a positive integer), each of which is an Int16. Array members can be selected with Item and changed with setItem. The first member of the Int16Array is member number 1 unless an " *origin* " input (must be an integer) is given, in which case the first member of the Int16Array has that number as its index. (Typically 0 is used as the *origin* if anything.) Arrays are printed by PRINT and friends. This primitive also can be used to convert *alist* , *anarray* or *anintarray* to Int16Array, while clipping the values to the max and min 16 bit integer. It can also be used to clone *anint16array* .

Examples:

```

show Int16Array 3      ;Int16Array {0 0 0}
show (Int16Array 3 0)   ;Int16Array {0 0 0}@0
show Int16Array {1 2 3} ;Int16Array {1 2 3}
show Int16Array [1 2 3] ;Int16Array {1 2 3}
show Int16Array intarray {1 2 3}      ;Int16Array {1 2 3}
show int16array {a b c} ;  int16array doesn't like {a b c} as
input
show Item 3 int16array [1 2 3]  ;3

```

FloatArray *size***(FloatArray *size origin*)****FloatArray *anarray*****FloatArray *alist*****FloatArray *anintarray*****FloatArray *anint16array***

outputs a new FloatArray of " *size* " members (must be a positive integer), each of which is an Float. Array members can be selected with Item and changed with setItem. The first member of the FloatArray is member number 1 unless an " *origin* " input (must be an integer) is given, in which case the first member of the FloatArray has that number as its index. (Typically 0 is used as the *origin* if anything.) FloatArrays are printed by PRINT and friends. This primitive also can be used to convert *alist* , *anarray* , *anintarray* or *anint16array* to FloatArray. It can also be used to clone a floatarray.

Examples:

```

show FloatArray 3      ;{0 0 0}
show (FloatArray 3 0)   ;{0 0 0}@0
show FloatArray {1 2 3} ;{1 2 3}
show FloatArray [1 2 3] ;{1 2 3}
show FloatArray int16array {1 2 3}  ;{1 2 3}
show FloatArray {a b c} ;  floatarray doesn't like {a b c} as
input
show Item 3 FloatArray [1 2 3]      ;3

```

mdarray *sizelist* (library procedure)
(mdarray *sizelist* *origin*)

outputs a multi-dimensional array. The first input must be a list of one or more positive integers. The second input, if present, must be a single integer that applies to every dimension of the array.

Example:

```
(MDARRAY [3 5] 0)
```

outputs a two-dimensional array whose members range from [0 0] to [2 4].

Examples:

```
mdarray [2]      ;{[] []}  ;-)
mdarray [2 3]    ;{{[] [] []}{[] [] []}}  ;-)
mdarray [2 3 4]
{
  {
    {[] [] [] []}
    {[] [] [] []}
    {[] [] [] []}
  }
  {
    {[] [] [] []}
    {[] [] [] []}
    {[] [] [] []}
  }
}  ;-)
```

Table *size*
Table *alist*

outputs a new hash table of *size size* which has initially no members if using the first form, else the members are filled with *alist* . You can add new members by using `setItem` or the "t's=x" operator.

Hash tables are very fast for word lookup, much faster than searching in a big list or plist (constant time).

Example:

```
t=Table 5
setItem "a t 1234
show Item "a t      ;1234
t'b=5678
show t'b           ;5678
table [[a 1234][b 1.2]] ;[a 1234][b 1.2]  ;-)
```

Struct *typelist*

outputs a new Struct, the members are filled with *typelist*. *typelist* is a list of lists, which contain a field name, a field type and optionally a field item. The field type may be any of Int, Int16, Int8, UInt8, Word, IntArray, Int16Array, FloatArray. The types Word, IntArray, Int16Array and FloatArray require a third list item, the size. Having Word, size is the number of characters of the Word. You can change members by using setItem or the "t's=x" operator.

Structs are, like Tables, very fast for word lookup, much faster than searching in a big list or plist (constant time).

Example:

```

wavHeaderType=(list
  [ChunkID Word RIFF]
  [wavfilesize Int]
  [RIFFtype Word 4]    ;the last item is the string length

  [formatChunkID Word fmt\ ]
  [formatChunkSize Int 16]
  [compressionCode Int16 1]
  [NumberOfChannels Int16 1]
  (list "SampleRate "Int rate)
  (list "BytesPerSecond "Int rate*2)
  [BlockAlign Int16 2]
  [BitsPerSample Int16 16]

  [DataChunkID word data]
  (list "DataChunkSize "Int size*2)
)
wavHeader=struct wavHeaderType
wavsize=(SizeOf wavHeader)+size*2
wavHeader'RIFFtype=[WAVE]    ;example for setting a string
wavHeader'wavfilesize=wavsize
pr wavHeader

```

toList *atable*

toList *anArray*

toList *anIntArray*

toList *anInt16Array*

toList *aWord*

tolist *aNumber*

When *atable* is the input, outputs a new list with sublists of the form [name value]. The names are the names of the members, the values are the member's values. Else toList just converts the input type to List. *aWord* is converted to a list of characters, the same with *aNumber* .

Example:

```
t=Table 5
t'a=1234
show t'a           ;1234
t'b=[Hallo World!]
show t             ;[a 1234][b [Hallo World!]]
l=TableToList t
show l             ;[[a 1234][b [Hallo World!]]]
```

```
show toList {a b c} ;[a b c]
show toList intArray {1 2 3} ;[1 2 3]
show toList int16Array {1 2 3} ;[1 2 3]
```

```
show toList "Hallo" ;[h a l l o]
show toList 1234     ;[1 2 3 4]
show toList 12.345   ;[1 2 . 3 4 5]
```

combine *thing1 thing2*
(combine *thing1 thing2 ...*)

if *thing2* is a word, outputs **WORD** *thing1 thing2* . If *thing2* is a list, outputs **FPUT** *thing1 thing2* .

If the things are of the same array type, a new array of the same type but with all elements of the input arrays is the output.

Examples:

```
show combine "ha "llo      ;hallo
show combine "Hallo [World] ;[hallo World]
show combine {1 2 3}{4 5 6} ;{1 2 3 4 5 6}
show (combine {1 2 3}{4 5 6}{7 8}{9}) ;{1 2 3 4 5 6 7 8 9}
show combine FloatArray {1 2 3} FloatArray {4 5 6} ;{1 2 3 4 5 6}
show combine IntArray {1 2 3} IntArray {4 5 6} ;{1 2 3 4 5 6}
show combine Int16Array {1 2 3} Int16Array {4 5 6} ;{1 2 3 4 5 6}
```


Data Structure Primitives / Constructors / reverse

reverse *thing*

outputs a *thing* whose members are the members of the input, in reverse order.

Examples:

```
show reverse "HalloWorld"      ;dlrowollah  ;-)
show reverse [Hallo World]     ;[World Hallo]
show reverse [[a b][c d]]      ;[  [c d]    [a b]  ]
show reverse {an array}        ;{array an}
```

rotate *array places*

outputs a new *array* that is rotated against the *array* by *places places* .

Examples:

```
show rotate {h a l l o} 1      ;{o h a l l}
show rotate {h a l l o} -1     ;{a l l o h}
show rotate {1 2 3} 2          ;{2 3 1}
```

shuffle *array*

outputs a new *array* of the same type and length as the input, but with randomly shuffled elements.

Examples:

```
show shuffle {1 2 3 4 5 6 7 8 9}           ;{7 2 9 4 5 1 6 8 3}
show shuffle FloatArray {1 2 3 4 5 6 7 8 9} ;{1 5 4 7 2 6 8 3
9}
show shuffle IntArray {1 2 3 4 5 6 7 8 9}   ;{5 1 3 6 9 8 4 7 2}
show shuffle Int16Array {1 2 3 4 5 6 7 8 9} ;{8 5 7 2 6 3 1 4
9}
```

merge *list1 list2*

outputs a list consisting of the two lists elements merged in order.

Example:

```
merge [1 2 3][2 3 4]
[1 2 2 3 3 4]
```

mergePairs *listOfTwoLists*

outputs a list which contains the elements of the two lists elementwise sorted.

Example:

```
mergePairs [[2 4 1][3 6 5]]
[  [2 3 4 1 6 5]
]
; - )
```

toListList *list*

Data Structure Primitives / Constructors / toListList

outputs a new *list* containing lists of one element of the *list* .

Example:

```
toListList [2 4 1 3]
[  [2]
   [4]
   [1]
   [3]
]
```

mergeSort *list1*

outputs a new list containing the elements of *list1* in alphabetical order.

Example:

```
show mergeSort [5 3 2 4 1 12]      ;[1 12 2 3 4 5]
show mergeSort [d b c a aa]        ;[a aa b c d]
```

genSym

outputs a unique word each time it's invoked. The words are of the form g1, g2, g3, etc.

Example:

```
show genSym      ;g1, g2, g3...
```

replace *aword withaword inathing*

outputs a new word with the first occurrence of *aword* in *athing* replaced with *aword*.

Examples:

```
show replace "a "e "hallo      ;"hello
show replace "abba "elle "jabbai ;"jellei
show replace "a "b [aaa]      ;"abb
show replace "b "x {a b c}      ;{a x c}
```

deepCopy *athing*

outputs a copy of *athing*. Lists and Arrays are iterated into, so if one mutates this copy, the original *athing* stays the same.

Examples:

```
a={1 2 3}
b=deepCopy a
a.2="hallo
b      ;{1 2 3}  ;-)
```

```
l=[1 2 3]
b=deepCopy l
l.2="hallo
b      ;[1 2 3] ;-)
```

Selectors

...output selected part(s) of the input data.

Selectors

- first 49
- firsts 50
- last 50
- butFirst 51, bF 51
- butFirsts 51, bFs 51
- butLast 52, bL 52
- Item 52
- mdItem 53
- BitItem 54
- pick 54
- items 54
- remove 55
- remDup 55
- quoted 55
- real 55
- imag 56
- conjugate 56

first *thing*

if the input is a word, outputs the first character of the word.

If the input is a list, outputs the first member of the list.

If the input is an Array, FloatArray, IntArray or Int16Array, outputs the first item of the array.

Examples:

```
show first "Hallo" ;h
show first [H a l l o] ;H
show first {H a l l o} ;H
```

firsts *list*

outputs a *list* containing the FIRST of each member of the input *list* . It is an error if any member of the input *list* is empty. (The input itself may be empty, in which case the output is also empty.) This could be written as

```
to firsts : list      output map "first : list"
end
```

but is provided as a primitive in order to speed up the iteration tools MAP, MAP_SE, and FOREACH.

New in 4.689 is that Array arguments are now allowed.

Examples:

```
show firsts [[H a l l o] [W o r l d]] ;[H W]
to transpose :matrix
  if empty? first :matrix [op []]
  op fput firsts :matrix transpose bfs :matrix
end
```

```
show firsts {{1 2 3}[a b c]} ;{1 a}
```

last *thing*

If the input is a word, outputs the last character of the word. If the input is a list, outputs the last member of the list. If the input is an Array, FloatArray, IntArray or Int16Array, outputs the last item of the array.

Examples:

```
show last "Hallo" ;o
show last [H a l l o] ;o
show last {H a l l o} ;o
```

butFirst *wordorlist*

bF *wordorlist*

if the input is a word, outputs a word containing all but the first character of the input. If the input is a list, outputs a list containing all but the first member of the input. If the input is an Array, FloatArray, IntArray or Int16Array, outputs an array of the same type as the input containing all but the first member of the input.

Examples:

```
show bf "Hallo" ;allo
show bf [Ha llo] ;[llo]
show bf {Ha llo} ;{llo}
```

butFirsts *list*

bFs *list*

outputs a *list* containing the BUTFIRST of each member of the input *list* . It is an error if any member of the input *list* is empty. (The input itself may be empty, in which case the output is also empty.) Array support added in 4.689. This could be written as

```
to butfirsts : list      output map "butfirst : list
end
```

but is provided as a primitive in order to speed up the iteration tools MAP, MAP_SE, and FOREACH.

Example:

```
show bfs [[Ha llo][Wor ld]] ;[[llo][ld]]
show bfs {{Ha llo}[Wor ld]} ;{{llo}[ld]}
```

butLast wordorlist**bL wordorlist**

if the input is a word, outputs a word containing all but the last character of the input. If the input is a list, outputs a list containing all but the last member of the input. If the input is an Array, FloatArray, IntArray or Int16Array, outputs an array of the same type as the input containing all but the last member of the input.

Example:

```
show bL [Hallo World] ;[Hallo]
show bL "Hallo      ;hall
show bL {Hallo World} ;{Hallo}
```

outputitem Item index athing

outputitem thingname. *index* outputitem tablename' *index*

if *athing* is a word, outputs the "*index*"th character of the word. If *athing* is a list, outputs the "*index*"th member of the list. If *athing* is an array, outputs the "*index*"th item of the array. Index starts at 1 for words and lists; the starting *index* of an array is specified when the array is created.

If *index* is a list and *athing* is a list, or if *index* is an Array and *athing* is an Array, or if *index* is an IntArray and *athing* is a FloatArray, then a list of the items indexed by the *index* list elements is the result of Item. This is nice for permutations, as seen in the example.

The dot and singlequote notation is experimental, but very comfortable.

Examples:

```
w="hallo
show Item 2 w ;a
w.2 ;a
```



```
l=[a b c]
show l.3          ;c
```

```
a=array 4
show a.1          ;[]
```

```
l=[[a b][c d][e f]]
show (l.3).1      ;e
```

```
t=table 5
t'a=1234
show item "a t"    ;1234
show t'a          ;1234
```

```
Item [1 3 6][10 20 30 40 50 60]      ;[10 30 60]
Item IntArray [1 3 6] FloatArray[10 20 30 40 50 60] ;{10 30 60}
;-)
Item {1 3 6}{10 20 30 40 50 60}      ;{10 30 60}  ;-)
```

```
a={1 2 3 4 5 6}
p={6 3 4 5 1 2}
repeat 6 [a=Item p a pr a]
{6 3 4 5 1 2}
{2 4 5 1 6 3}
{3 5 1 6 2 4}
{4 1 6 2 3 5}
{5 6 2 3 4 1}
{1 2 3 4 5 6}
```

mdItem *indexlist anarray*

outputs the member of the multidimensional *anarray* selected by the list of numbers *indexlist* .

Example:

```
a={{ {1 2} {3 4} } {3 4} }
mdItem [1 2 1] a      ;3
```

```
a=[[[1 2][3 4]][5 6]]
mdItem [1 2 1] a      ;3
```

```
a=[Hallo World]
mdItem [2 3] a      ;r
```

BitItem *index* *athing*

outputs either true or false, depending on the value of the "*index*"th bit of *athing* .

Examples:

```
show BitItem 0 5      ;true
show BitItem 1 5      ;false
show BitItem 2 5      ;true
show BitItem 3 5      ;false
show BitItem 7 int8 -1 ;true
show BitItem 0 int8 -1 ;true
```

pick *list*

outputs a randomly chosen member of the input *list* .

Example:

```
show pick [a b c]      ;a, b or c randomly
show pick "abc"        ;the same output
```

items *startindex* *endindex* *athing*

Data Structure Primitives / Selectors / items

outputs the items from *startindex* to *endindex* of *athing* . At the moment *athing* can only be an array, intarray or int16array.

Examples:

```
show items 1 3 {a b c d e f}      ;{a b c}
show items 3 5 {a b c d e f}      ;{c d e}
```

remove *thing* *listorword*

outputs a copy of "*listorword*" with every member equal to "*thing*" removed.

Examples:

```
show remove "lo [Hal lo Wor ld]      ;[Hal Wor ld]
show remove "l "HalloWorld           ;haoword
show remove [] [[] a [] b [] c []]    ;[a b c]
```

remDup *listorword*

outputs a copy of "*listorword*" with duplicate members removed. If two or more members of the input are equal, the rightmost of those members is the one that remains in the output.

Examples:

```
show remDup [H a l l o W o r l d]    ;[H a W o r l d]
show remDup "HalloWorld              ;haworld
```

quoted *thing* (library procedure)

outputs its input, if a list; outputs its input with a quotation mark prepended, if a word.

realPart **real** *complexNumber*

outputs the realPart of a *complexNumber* as a real floating point number.

Example:

```
show real 3+4i      ;3
show real list 1+2i 3+4i      ;[1 3]
```

imaginaryPart **imag** *complexNumber*

outputs the imaginaryPart of the *complexNumber* as a real floating point number.

Example:

```
show imag 3+4i      ;4
show imag list 1+2i 3+4i      ;[2 4]
```

conjugated **conjugate** *complexNumber*

outputs the complex conjugated to the *complexNumber* . This is like $(\text{real } z) - 1i * (\text{imag } z)$.

Example:

```
show conjugate 1+2i      ;1-2i
show conjugate list 1+2i 3+4i      ;[1-2i 3-4i]
```

Mutators

...change the input data.

Mutators

- setItem 57
- _setItem 58
- mdSetItem 58
- setItems 59
- removeItem 59
- _setFirst 60
- _setButFirst 60
- _setBF 60
- push 61
- pop 61
- queue 62
- dequeue 62

setItem *index thing value*

command. Replaces the " *index* "th member of "array" with the new " *value* ". SetItem Ensures that the resulting array is not circular, i.e., " *value* " may not be a list or array that contains "array".

Examples:

```
a=[H a l l o]
setItem 2 a "e
show a      ;[H e l l o]
```

```
a={W o r l d}
setItem 3 a "a
show a      ;{W o a l d}
```

```
a="hallo
setItem 2 a "e
show a      ;hello
```

setItem *index array value*

thing. *index* = *value* table' *index* = *value*

command. Changes the "*index*"th member of "*array*" to be "*value*", like SETITEM, but without checking for circularity. WARNING: Primitives whose names start with a underbar are DANGEROUS. Their use by non-experts is not recommended. The use of setItem can lead to circular arrays, which will get some Logo primitives into infinite loops; and the loss of memory if a circular structure is released.

The experimental dot notation is still in test phase; *index* cannot be an expression yet.

The singlequote ' notation is also experimental.

Examples:

```
l=[a b c]
l.2="hallo
l
  [a hallo c]
  ;-)
_setItem 3 l 333
l
  [a hallo 333]
  ;-)
t=table 5
setItem "a t 1234
t      ;[a 1234] ;-)
t'b=5678
t      ;[a 1234][b 5678] ;-)
```

mdSetItem *indexlist array value*

Data Structure Primitives / Mutators / mdSetItem

command. Replaces the member of "*array*" chosen by "*indexlist*" with the new "*value*".

Examples:

```
a={{1 2}{3 4 5}}
mdSetItem [2 3] a "so"
show a      ;{{1 2}{3 4 so}}
```

```
a=[Hallo World]
mdSetItem [2 3] a "a"
show a      ;[Hallo Woald]
```

setItems *startindex* *athing* *newitems*

command. It sets the elements starting from *startindex* of *athing* to the elements of *newitems*. *athing* and *newitems* need not to be of the same type, but only Array, IntArray and Int16Array are supported so far.

Example:

```
a={1 2 3 4}
setitems 2 a {7 8}
show a
{1 7 8 4}
```

removeItem *index* *thing*

removes the *index*th member of the input *thing*, which may be a word, list or table.

In case of *thing* being a list with only one member, the list will contain an empty list after removing the first member. This looks strange, but it's not possible to avoid this, except making *removeItem* a selector instead of a mutator, which would be much more slow.

Examples:

```
w="aword
removeItem 2 w
w      ;aord  ;-)
l=[a b c]
removeItem 2 l
l      ;[a c] ;-)
removeItem 1 l
l      ;[c]  ;-)
removeItem 1 l
l      ;[[ ] ;-)
t=table 3
t'a=1234
t'b=5678
t'c=9012
removeItem "b t
t      ;[a 1234][c 9012] ;-)
```

_setFirst *list value*

command. Changes the first member of " *list* " to be " *value* ". WARNING: Primitives whose names start with a underbar are DANGEROUS. Their use by non-experts is not recommended. The use of _setFirst can lead to circular *list* structures, which will get some Logo primitives into infinite loops; unexpected changes to other data structures that share storage with the *list* being modified; and the loss of memory if a circular structure is released.

Example:

```
a=[1 2]
_setfirst a -first a
show a      ;[-1 2]
```

_setButFirst *list value*

`_setBF` *list value*

command. Changes the butfirst of "*list*" to be "*value*". WARNING: Primitives whose names start with a underbar are DANGEROUS. Their use by non-experts is not recommended. The use of `_setbf` can lead to circular *list* structures, which will get some Logo primitives into infinite loops; unexpected changes to other data structures that share storage with the *list* being modified; Logo crashes and coredumps if the butfirst of a *list* is not itself a *list*; and the loss of memory if a circular structure is released.

Example:

```
a=[1 2]
_setbf a [-2]
show a      ;[1 -2]
```

`push` *stackname thing*

command. Adds the "*thing*" to the stack that is the value of the variable whose name is "*stackname*". This variable must have a list as its value; the initial value should be the empty list. New members are added at the front of the list.

Examples:

```
s=[]
push "s "Hallo
show s      ;[hallo]
push "s "World
show s      ;[world hallo]
```

`pop` *stackname*

outputs the most recently PUSHed member of the stack that is the value of the variable whose name is "*stackname*" and removes that member from the stack.

Examples:

```
s=[world hallo]
show pop "s      ;world
show s      ;[hallo]
show pop "s      ;hallo
show s      ;[]
```

queue *queuename thing*

command. Adds the "*thing*" to the queue that is the value of the variable whose name is "*queuename*". This variable must have a list as its value; the initial value should be the empty list. New members are added at the back of the list, so this is a bit slow because the whole list must be traversed till its end.

Examples:

```
q=[]
queue "q "Hallo
show q      ;[hallo]
queue "q "World
show q      ;[hallo world]
```

dequeue *queuename*

outputs the least recently QUEUED member of the queue that is the value of the variable whose name is "*queuename*" and removes that member from the queue.

dequeue is an alias for pop.

Examples:

Data Structure Primitives / Mutators / dequeue

```
q=[Hallo World]
show dequeue "q"      ;Hallo
show q      ;[World]
show dequeue "q"      ;World
show q      ;[]
```

Predicates

...ask for boolean (true or false) properties of the input data.

Predicates

- WordP 64
 - ListP 65
 - ArrayP 65
 - emptyP 66
 - equalP 66
 - beforeP 67
 - _eq 67
 - MemberP 68
 - subStringP 69
 - NameInTableP 69
 - NumberP 69
 - CharP 70
 - IntP 70
 - Int16P 71
 - Int8P 71
 - UInt8P 72
 - floatP 72
 - complexP 73
 - backslashedP 73
 - circularP 74
 - AINumP 74
 - AlphaP 74
 - ASCIIIP 75
 - CntrlP 75
 - CSymP 76
 - DigitP 76
 - GraphP 77
 - LowerP 77
 - PrintP 78
 - PunctP 78
 - SpaceP 79
 - UpperP 79
 - xDigitP 80
-

WordP *thing***Word?** *thing*

outputs TRUE if the input is a word, FALSE otherwise.

Examples:

```
Word? "           ;true  ;-)
Word? []          ;false ;-)
Word? "aWord      ;true  ;-)
Word? [a list]    ;false ;-)
Word? {an array}   ;false ;-)
Word? (Word [a list]) ;true ;-)
```

ListP *thing***List?** *thing*

outputs TRUE if the input is a list, FALSE otherwise.

Examples:

```
List? [Hallo World] ;true  ;-)
List? []            ;true  ;-)
List? "             ;false ;-)
List? "Hallo        ;false ;-)
List? {an array}     ;false ;-)
List? (list "Hallo)  ;true  ;-)
List? (list {an array}) ;true ;-)
```

ArrayP *thing***Array?** *thing*

outputs TRUE if the input is an array, FALSE otherwise.

Examples:

```
Array? {Hallo World}      ;true  ;-)
Array? {}                 ;true  ;-)
Array? {1 2 3}            ;true  ;-)
Array? []                 ;false  ;-)
Array? [a list]           ;false  ;-)
Array? "aWord             ;false  ;-)
Array? ListToArray [Hallo World]      ;true  ;-)
```

emptyP *thing*

empty? *thing*

outputs TRUE if the input is the empty word or the empty list, FALSE otherwise.

Examples:

```
empty? "      ;true  ;-)
empty? []     ;true  ;-)
empty? "Hallo ;false  ;-)
empty? [Hallo] ;false  ;-)
empty? {}     ;false  ;-)
```

equalP *thing1 thing2*

equal? *thing1 thing2*

thing1 == thing2

outputs TRUE if the inputs are equal, FALSE otherwise.

Two numbers are equal if they have the same numeric value.

Two non-numeric words are equal if they contain the same characters in the same order.

If CaseIgnored? is true set by "setCaseIgnored true", then an upper case letter is considered the

Data Structure Primitives / Predicates / equalP

same as the corresponding lower case letter (This is the case by default.).

Two lists or arrays are equal if their members are equal.

Examples:

```
equal? true false      ;false  ;-)
equal? true true        ;true   ;-)
equal? false false      ;true   ;-)
equal? 1234 2345        ;false  ;-)
equal? pi 3.14          ;false  ;-)
equal? "A "a            ;true   ;-)
equal? "Hallo "World    ;false  ;-)
equal? [Hallo] [World]  ;false  ;-)
equal? [Hallo] [Hallo]  ;true   ;-)
equal? {1 2} {1 2}      ;true   ;-)
equal? {1 2} {3 4}      ;false  ;-)
```

beforeP *word1 word2*

before? *word1 word2*

outputs TRUE if *word1* comes before *word2* in ASCII collating sequence (for words of letters, in alphabetical order). Case-sensitivity is determined by the value of CaseIgnoredP. Note that if the inputs are numbers, the result may not be the same as with LESSP;

for example,

```
BEFOREP 3 12
```

is false because 3 collates after 1.

Examples:

```
before? 3 12            ;false  ;-)
less? 3 12              ;true   ;-)
before? "A "B           ;true   ;-)
before? [A] [B]         ;true   ;-)
before? [World] [Hallo] ;false  ;-)
```

`_eq` *thing1 thing2*

outputs TRUE if its two inputs are the same datum, so that applying a mutator to one will change the other as well.

Outputs FALSE otherwise, even if the inputs are equal in value.

WARNING: Primitives whose names start with a underbar are DANGEROUS. Their use by non-experts is not recommended. The use of mutators can lead to circular data structures, infinite loops, or Logo crashes.

Examples:

```
_eq 1 1      ;false  ;-)
a=1
_eq a a      ;true   ;-)
_eq a 1      ;false  ;-)
```

```
_eq [a] [a]   ;false  ;-)
a=[a]
_eq a [a]     ;false  ;-)
_eq a a       ;true   ;-)
```

MemberP *thing1 thing2***Member? *thing1 thing2***

if "*thing2*" is a list or an array, outputs TRUE if "*thing1*" is EQUALP to a member of "*thing2*", FALSE otherwise.

If "*thing2*" is a word, outputs TRUE if "*thing1*" is a one-character word EQUALP to a character of "*thing2*", FALSE otherwise.

Examples:

Data Structure Primitives / Predicates / MemberP

```
Member? "a "Hallo      ;true  ;-)
Member? "Hallo [Hallo World]      ;true  ;-)
Member? "Hallo {Hallo World}      ;true  ;-)
Member? " []      ;false  ;-)
```

subStringP *thing1 thing2*

subString? *thing1 thing2*

if " *thing1* " or " *thing2* " is a list or an array, outputs FALSE. If " *thing2* " is a word, outputs TRUE if " *thing1* " is EQUALP to a substring of " *thing2* ", FALSE otherwise.

Examples:

```
subString? "Hal "Hallo      ;true  ;-)
subString? "Hal [Hallo]      ;false  ;-)
```

NameInTableP *name table*

NameInTable? *name table*

outputs true if there's an item with *name name* in the *table* .

Examples:

```
t= table [[a 1234][b 5678]]
NameInTable? "a t      ;true  ;-)
NameInTable? "c t      ;false  ;-)
```

NumberP *thing*

Number? *thing*

outputs TRUE if the input is a number, FALSE otherwise.

Examples:

```
Number? 1234      ;true  ;-)
Number? "1234     ;false ;-)
Number? 2i        ;true  ;-)
Number? 2.2       ;true  ;-)
Number? [1234]    ;false ;-)
Number? {1}       ;false ;-)
```

CharP *thing*

Char? *thing*

outputs TRUE if the input is a character (a string with length 1), FALSE otherwise.

Examples:

```
Char? "A         ;true  ;-)
Char? Char 27    ;true  ;-)
Char? "Hallo     ;false ;-)
Char? [A]        ;false ;-)
Char? 1          ;false ;-)
Char? "1         ;true  ;-)
Char? "          ;false ;-)
Char? []         ;false ;-)
```

IntP *thing*

Int? *thing*

outputs TRUE if the input is a integer (32 bit), FALSE otherwise.

Examples:

Data Structure Primitives / Predicates / IntP

```

int? 1234      ;true  ;-)
int? 1.2        ;false ;-)
int? -IntMax    ;true  ;-)
int? int 1.2    ;true  ;-)
int? [1]        ;false ;-)
int? {1}        ;false ;-)
int? "1         ;false ;-)
int? int "1     ;true  ;-)

```

Int16P *thing***Int16? *thing***

outputs TRUE if the input is a 16 bit integer, FALSE otherwise.

Examples:

```

int16? int16 1234  ;true  ;-)
int16? 1234 ;false  ;-)
int16? 1.2  ;false  ;-)
int16? Int16Max ;true  ;-)
int16? Int16 1.2   ;true  ;-)
int16? [1]  ;false  ;-)
int16? {1}  ;false  ;-)
int16? "1   ;false  ;-)
int16? Int16 "1   ;true  ;-)

```

Int8P *thing***Int8? *thing***

outputs TRUE if the input is a 8 bit integer, FALSE otherwise.

Examples:

```
int8? int8 127 ;true  ;-)
int8? 1234 ;false  ;-)
int8? Int8Max ;true  ;-)
int8? 1.2 ;false  ;-)
int8? [1] ;false  ;-)
int8? "1 ;false  ;-)
int8? int8 "1 ;true  ;-)
```

UInt8P *thing*

UInt8? *thing*

outputs TRUE if the input is a 8 bit unsigned integer, FALSE otherwise.

Examples:

```
uint8? uint8 255 ;true  ;-)
uint8? 1234 ;false  ;-)
uint8? UInt8Max ;true  ;-)
uint8? 1.2 ;false  ;-)
```

floatP *thing*

float? *thing*

outputs TRUE if the input is a floating point number, FALSE otherwise.

Examples:

Data Structure Primitives / Predicates / floatP

```
float? 1.2      ;true  i-)
float? 1e-10     ;true  i-)
float? pi       ;true  i-)
float? FloatMax  ;true  i-)
float? "1       ;false i-)
float? "1.2     ;false i-)
float? float "1.2 ;true  i-)
```

complexP *thing***complex? *thing***

outputs TRUE if the input is a complex number, FALSE otherwise.

Examples:

```
complex? 1i      ;true  i-)
complex? 1.2+2.3e6i ;true  i-)
complex? 1.2     ;false i-)
complex? "1i     ;false i-)
```

backslashedP *char***backslashed? *char***

outputs TRUE if the input character was originally entered into Logo with a backslash (\) before it or within vertical bars (|) to prevent its usual special syntactic meaning, FALSE otherwise.

(Outputs TRUE only if the character is a

```
backslashed space, tab, newline, carriage return or one of
( ) [ ] { } < > ; | ~ " \ + - * / ^ = : ? .
```

Examples:

```
backslashed? "\\      ;true  ;-)
backslashed? "\[      ;true  ;-)
backslashed? "\=      ;true  ;-)
backslashed? "\a      ;false ;-)
```

circularP *thing*

circular? *thing*

outputs true if the input is a circular array or list structure. This helps finding circularity bugs if you use underbar _functions.

Examples:

```
a=[1]
_setFirst a a
circular? a      ;true  ;-)
```

```
a=[1]
_setbf a a
circular? a      ;true  ;-)
```

AlNumP *achar*

AlNum? *achar*

outputs true if the input is a character in the subset of alphanumeric characters.

Examples:

```
alnum? "a  ;true  ;-)
alnum? "A  ;true  ;-)
alnum? "1  ;true  ;-)
alnum? 1    ;false ;-)
alnum? "ü ;true  ;-)
alnum? "!  ;false ;-)
```

AlphaP *achar***Alpha?** *achar*

outputs true if the input is a character in the subset of alphabetic characters.

Examples:

```
alpha? "a  ;true  ;-)
alpha? "A  ;true  ;-)
alpha? "1  ;false ;-)
alpha? 1    ;false ;-)
alpha? "ü ;true  ;-)
alpha? "!" ;false ;-)
```

ASCIIP *achar***ASCII?** *achar*

outputs true if the input is a character in the subset of ASCII characters.

Examples:

```
ASCII? "a  ;true  ;-)
ASCII? "A  ;true  ;-)
ASCII? "1  ;true  ;-)
ASCII? 1    ;false ;-)
ASCII? "ü ;false ;-)
ASCII? "!" ;true  ;-)
```

CntrlP *achar***Cntrl?** *achar*

outputs true if the input is a character in the subset of control characters.

Examples:

```
cntrl? "a  ;false  ;-)
cntrl? "A  ;false  ;-)
cntrl? "1  ;false  ;-)
cntrl? 1    ;false  ;-)
cntrl? "ü ;true   ;-)
cntrl? "!" ;false  ;-)
cntrl? char 13 ;true ;-)
```

CSymP *achar*

CSym? *achar*

outputs true if the input is a character in the subset of C symbol characters.

Examples:

```
csym? "a  ;true   ;-)
csym? "A  ;true   ;-)
csym? "1  ;true   ;-)
csym? 1 ;false  ;-)
csym? "ü ;true   ;-)
csym? "!" ;false  ;-)
csym? char 13 ;false ;-)
```

DigitP *achar*

Digit? *achar*

outputs true if the input is a character in the subset of numerical digit characters.

Examples:

Data Structure Primitives / Predicates / DigitP

```
digit? "a  ;false  ;-)
digit? "A  ;false  ;-)
digit? "1  ;true   ;-)
digit? 1    ;false  ;-)
digit? "ü ;true   ;-) <= this seems to be a bug!!!
digit? "!" ;false  ;-)
digit? char 13 ;false ;-)
```

GraphP *achar***Graph?** *achar*

outputs true if the input is a character in the subset of graphical characters.

Examples:

```
graph? "a  ;true   ;-)
graph? "A  ;true   ;-)
graph? "1  ;true   ;-)
graph? 1    ;false  ;-)
graph? "ü ;true   ;-)
graph? "!" ;true   ;-)
graph? char 13 ;false ;-)
```

LowerP *achar***Lower?** *achar*

outputs true if the input is a character in the subset of lowercase characters.

Examples:

```
lower? "a  ;true  ;-)  
lower? "A  ;true  ;-)  <= in case-insensitive mode!!!  
lower? "1  ;false  ;-)  
lower? 1    ;false  ;-)  
lower? "ü ;false  ;-)  <= bug?  
lower? "!" ;false  ;-)  
lower? char 13 ;false  ;-)
```

PrintP *achar*

Print? *achar*

outputs true if the input is a character in the subset of printable characters.

Examples:

```
print? "a  ;true  ;-)  
print? "A  ;true  ;-)  
print? "1  ;true  ;-)  
print? 1    ;false  ;-)  
print? "ü ;true  ;-)  
print? "!" ;true  ;-)  
print? char 13 ;false  ;-)
```

PunctP *achar*

Punct? *achar*

outputs true if the input is a character in the subset of punctuation characters.

Examples:

```
punct? "a  ;false  ;-)
punct? "A  ;false  ;-)
punct? "1  ;false  ;-)
punct? 1    ;false  ;-)
punct? "ü ;true   ;-) <= bug?
punct? "!" ;true   ;-)
punct? char 13 ;false ;-)
```

SpaceP *achar*

Space? *achar*

outputs true if the input is a character in the subset of space characters.

Examples:

```
space? "a  ;false  ;-)
space? "A  ;false  ;-)
space? "1  ;false  ;-)
space? 1    ;false  ;-)
space? "ü ;false   ;-)
space? "!" ;false   ;-)
space? char 13 ;true   ;-)
space? "\ " ;true   ;-)
```

UpperP *achar*

Upper? *achar*

outputs true if the input is a character in the subset of uppercase characters.

Examples:

```
upper? "a  ;false  ;-)
upper? "A  ;false  ;-)    <= in case-insensitive mode!!!
upper? "1  ;false  ;-)
upper? 1    ;false  ;-)
upper? "ü ;true   ;-) <= bug!
upper? "!" ;false  ;-)
upper? char 13 ;false ;-)
upper? "\   ;false  ;-)
```

xDigitP *achar*

xDigit? *achar*

outputs true if the input is a character in the subset of hexadecimal digit characters.

Examples:

```
xdigit? "a ;true   ;-)
xdigit? "A ;true   ;-)
xdigit? "1 ;true   ;-)
xdigit? 1   ;false  ;-)
xdigit? "ü   ;false ;-)
xdigit? "!" ;false  ;-)
xdigit? char 13 ;false ;-)
xdigit? "\     ;false ;-)
```

Queries

...ask for properties of the input data.

Queries

- count 81
- SizeOf 82
- TypeOf 82
- ASCII 83
- rawASCII 83
- Char 84
- Member 84
- lowerCase 84
- upperCase 85
- standout 85
- parse 86
- runParse 86
- BackslashEncode 86

count *thing*

outputs the number of characters in the input, if the input is a word. outputs the number of members in the input, if it is a list, array or table (For an array, this may or may not be the index of the last member, depending on the array's origin.). For a table, count outputs not the table size, but the number of items set.

Examples:

```
count "Hallo" ;5 ;-)
count 1234 ;4 ;-)
count [Hallo World] ;2 ;-)
count {a b c} ;3 ;-)
count [[][]] ;2 ;-)
```

```
t=table 3
count t      ;0  ;-)
t'a=1234
count t      ;1  ;-)
t'b=567
count t      ;2  ;-)
t'c=890
count t      ;3  ;-)
removeItem "b t
count t      ;2  ;-)
t      ;[a 1234][c 890] ;-)
```

SizeOf x

outputs the size in bytes which will be needed on typebin x .

Examples:

```
sizeof int 1      ;4  ;-)
sizeof int16 1    ;2  ;-)
sizeof int8 1     ;1  ;-)
sizeof UInt8 1    ;1  ;-)
sizeof l.1      ;8  ;-)
sizeof li       ;16  ;-)
sizeof "abc     ;3  ;-)
sizeof intarray 3 ;12  ;-)
sizeof int16array 5 ;10 ;-)
sizeof [123456 a] ;7  ;-) (counting the characters)
sizeof {1 2 3}   ;3  ;-) (counting the characters)
sizeof struct [[a int][b int16][c int8]] ;7  ;-)
```

TypeOf x

outputs a word representing the type of x .

Data Structure Primitives / Queries / TypeOf

Examples:

```
typeof int 1      ;Int    ;-)
typeof int16 1    ;Int16  ;-)
typeof int8 1     ;Int8   ;-)
typeof uint8 1    ;UInt8  ;-)
typeof 1.1        ;Float  ;-)
typeof 1i         ;Complex ;-)
typeof "abc       ;Word   ;-)
typeof []         ;List   ;-)
typeof [1 2 3]    ;List   ;-)
typeof {1 2 3}    ;Array  ;-)
typeof intarray 3  ;IntArray ;-)
typeof struct [[a int][b int8]] ;Struct ;-)
```

ASCII *char*

outputs the integer (between 0 and 255) that represents the input character in the ASCII code. Interprets control characters as representing backslashed punctuation, and returns the character code for the corresponding punctuation character without backslash. (Compare RAWASCII.)

Examples:

```
ASCII "a      ;97    ;-)
ASCII 0       ;48    ;-)
ASCII "\[     ;91    ;-)
ASCII Char 1   ;40    ;-)
Char 40       ;(     ;-)
```

rawASCII *char*

outputs the integer (between 0 and 255) that represents the input character in the ASCII code. Interprets control characters as representing themselves. To find out the ASCII code of an arbitrary keystroke, use RAWASCII RC.

Example:

```
forever [show rawASCII rc]
```

Char *aint*

outputs the character represented in the ASCII code by the input, which must be an integer between 0 and 255.

Examples:

```
Char 65      ;A      ;-)
Char 66      ;B      ;-)
Char 97      ;a      ;-)
Char 98      ;b      ;-)
Char 48      ;0      ;-)
Char 49      ;1      ;-)
```

Member *thing1 thing2*

if "*thing2*" is a word or list and if MEMBERP with these inputs would output TRUE, outputs the portion of "*thing2*" from the first instance of "*thing1*" to the end. If MEMBERP would output FALSE, outputs the empty word or list according to the type of "*thing2*". It is an error for "*thing2*" to be an array (because butFirst does not work for arrays).

Examples:

```
Member "a "Hallo      ;allo      ;-)
Member "Hal "Hallo      ;      ;-)
Member? "Hal "Hallo      ;false      ;-)
Member "World [Hallo World]      ;[World] ;-)
Member "b {a b c}      ; member doesn't like {a b c} as input
```

Data Structure Primitives / Queries / lowerCase

lowerCase *word*

outputs a copy of the input *word* , but with all uppercase letters changed to the corresponding lowercase letter.

Examples:

```
lowerCase "HALLO      ;hallo    ;-)
show lowerCase [H A L L O]      ;h a l l o
show lowerCase {H A L L O}      ;{h a l l o}
```

upperCase *word*

outputs a copy of the input *word* , but with all lowercase letters changed to the corresponding uppercase letter.

Examples:

```
upperCase "hallo      ;HALLO    ;-)
show upperCase [h a l l o]      ;H A L L O
show upperCase {h a l l o}      ;{H A L L O}
```

standout *thing*

does not work yet. (forget the rest of this description!)

outputs a word that, when printed, will appear like the input but displayed in standout mode (boldface, reverse video, or whatever your terminal does for standout). The word contains terminal-specific magic characters at the beginning and end; in between is the printed form (as if displayed using TYPE) of the input. The output is always a word, even if the input is of some other type, but it may include spaces and other formatting characters. Note: a word output by STANDOUT while Logo is running on one terminal will probably not have the desired effect if printed on another type of terminal.

On the Macintosh, the way that standout works is incompatible with the use of characters whose ASCII code is greater than 127. Therefore, you have a choice to make: The instruction

```
CANINVERSE 0
```

disables standout, but enables the display of ASCII codes above 127, and the instruction

```
CANINVERSE 1
```

restores the default situation in which standout is enabled and the extra graphic characters cannot be printed.

parse *word*

outputs the list that would result if the input *word* were entered in response to a READLIST operation. That is, PARSE READWORD has the same value as READLIST for the same characters read.

Example:

```
parse "fd\ 100      ;[fd 100] ;-)
```

runParse *wordorlist*

outputs the list that would result if the input word or list were entered as an instruction line; characters such as infix operators and parentheses are separate members of the output. Note that sublists of a runparsed list are not themselves runparsed.

Examples:

```
runParse "1\+2      ;[1 + 2] ;-)
runParse [(1+2)*3]   ;[( 1 + 2 ) * 3] ;-)
```

BackslashEncode *achar*

outputs a new vbarstring with encoded backslashes of *achar* .

Data Structure Primitives / Queries / BackslashEncode

```
BackslashEncode char 9
|   |   i-)
BackslashEncode char 13
|
|   i-)
```

Communication

Here are functions concerning output and input from and to Logo.

Communication

- Transmitters 89
 - Receivers 92
 - FileAccess 100
 - Environment 115
 - Terminal Access 119
 - Port Input and Output 124
 - Timing 127
 - Dynamic Libraries 129
-

Transmitters

...are commands which write to the current writer (using the console as default).

Note: If there is a variable named `PRINTDEPTHLIMIT` with a nonnegative integer value, then complex list and array structures will be printed only to the allowed depth. That is, members of members of... of members will be allowed only so far. The members omitted because they are just past the depth limit are indicated by an ellipsis for each one, so a too-deep list of two members will print as [... ...].

If there is a variable named `PRINTWIDTHLIMIT` with a nonnegative integer value, then only the first so many members of any array or list will be printed. A single ellipsis replaces all missing data within the structure. The width limit also applies to the number of characters printed in a word, except that a `PRINTWIDTHLIMIT` between 0 and 9 will be treated as if it were 10 when applied to words. This limit applies not only to the top-level printed datum but to any substructures within it.

Transmitters

- `print 89, pr 89`
- `type 90`
- `show 90`
- `dir 91`
- `dirlg 91`
- `displaymatrix 91`

`print thing`

`pr thing`

`(print thing1 thing2 ...)`

`(pr thing1 thing2 ...)`

command. Prints the input or inputs to the current write stream (initially the terminal). All the inputs are printed on a single line, separated by spaces, ending with a newline. If an input is a list, square brackets are not printed around it, but brackets are printed around sublists. Braces are always printed around arrays.

Examples:

```
pr 1234      ;1234
pr pi        ;3.14159
pr "Hallo    ;hallo
pr [Hallo World!] ;Hallo World!
pr {an array}  ;{an array}
pr [[sublist][structure]] ;[[sublist][structure]]
(pr)          ;prints an empty line
(pr "one "two "three) ;one two three
(pr 1 "one [One] {one}) ;1 one One {one}
```

type *thing*
(type *thing1 thing2 ...*)

command. Prints the input or inputs like PRINT, except that no newline character is printed at the end and multiple inputs are not separated by spaces. Note: printing to the terminal is ordinarily "line buffered"; that is, the characters you print using TYPE will not actually appear on the screen until either a newline character is printed (for example, by PRINT or SHOW) or Logo tries to read from the keyboard (either at the request of your program or after an instruction prompt). This buffering makes the program much faster than it would be if each character appeared immediately, and in most cases the effect is not disconcerting. To accommodate programs that do a lot of positioned text display using TYPE, Logo will force printing whenever SETCURSOR is invoked. This solves most buffering problems. Still, on occasion you may find it necessary to force the buffered characters to be printed explicitly; this can be done using the WAIT command. WAIT 0 will force printing without actually waiting.

Examples:

```
(type "Hallo "World)
(pr)          ;halloworld
(type [Hallo] [World])
(pr)          ;HalloWorld
(type [Hallo][World!])
wait 0        ;HalloWorld!
```

show *thing*

Communication / Transmitters / show

(show *thing1 thing2 ...*)

command. Prints the input or inputs like PRINT, except that if an input is a list it is printed inside square brackets.

Example:

```
show [this is a list]      ;[this is a list]
```

dir (library *procedure*)

command printing the file list of the current working directory using the output of shell.

dirlg (library *procedure*)

command printing the logo (*.lg) file list of the current working directory using the output of shell. This is a very useful command to browse the *.lg examples.

displaymatrix (library *procedure*)

displays a matrix in the console.

Examples:

```
displaymatrix [[1 2][3 4]]
  1.000    3.000
  2.000    4.000
```

```
displaymatrix {{1 2}{3 4}}
  1.000    3.000
  2.000    4.000
```

Receivers

...receive one or more data form either the keyboard, a file or from the operating system.

Receivers

- readList 92, rL 92
- readWord 93, rW 93
- readChar 93, rC 93
- readCharExt 94, rCE 94
- readChars 95, rCs 95
- readIntBin 95
- readInt16Bin 95
- readInt8Bin 95
- readUInt8Bin 96
- readFloatBin 96
- readComplexBin 96
- readIntArrayBin 96
- readInt16ArrayBin 96
- readFloatArrayBin 96
- readStructBin 97
- Shell 97
- ShellSpawn 97
- LogoVersion 98
- OSScreenSize 98

readList

rL

reads a line from the read stream (initially the terminal) and outputs that line as a list. The line is separated into members as though it were typed in square brackets in an instruction. If the read stream is a file, and the end of file is reached, READLIST outputs the empty word (not the empty list). READLIST processes backslash, vertical bar, and tilde characters in the read stream; the output list will not contain these characters but they will have had their usual effect. READLIST does not, however, treat semicolon as a comment character.

Example:


```
readList
a b c (1+2)*3      ;[a b c (1+2)*3] ;-)
```

readWord**rW**

reads a line from the read stream and outputs that line as a word. The output is a single word even if the line contains spaces, brackets, etc. If the read stream is a file, and the end of file is reached, READWORD outputs the empty list (not the empty word). READWORD processes backslash, vertical bar, and tilde characters in the read stream. In the case of a tilde used for line continuation, the output word DOES include the tilde and the newline characters, so that the user program can tell exactly what the user entered. Vertical bars in the line are also preserved in the output. Backslash characters are not preserved in the output, but the character following the backslash has 128 added to its representation. Programs can use BACKSLASHEDP to check for this code. (Backslashedness is preserved only for certain characters. See BACKSLASHEDP.)

Example:

```
readWord
a b c (1+2)*3      ;a\ b\ c\ \ (1\+2\)\*3      ;-)
```

readChar**rC**

reads a single character from the read stream and outputs that character as a word. If the read stream is a file, and the end of file is reached, READCHAR outputs the empty list (not the empty word). If the read stream is a terminal, echoing is turned off when READCHAR is invoked, and remains off until READLIST or READWORD is invoked or a Logo prompt is printed. Backslash, vertical bar, and tilde characters have no special meaning in this context.

Extensions: If the read stream is the keyboard and special keys like the arrow keys or numpad keys are typed, then rc returns char 255 and a consecutive call to readCharExt returns the extended char value.

Example:

```
to keyptest
  forever
    [ if key?
      [ c=readchar
        ifelse c == char 255
          [ c=readCharExt
            (pr "ext c)
          ]
          [ (pr rawASCII c c)
            ]
        ]
      ]
    ]
  ]
end
```

readCharExt **rCE**

If the read stream is the keyboard and special keys like the arrow keys are typed, then rc returns char 255 and a consecutive call to readCharExt returns the extended char value.

Output is the extended char value as an integer number.

Example:

```
to keyptest
  forever
    [ if key?
      [ c=readchar
        ifelse c == char 255
          [ c=readCharExt
            (pr "ext c)
          ]
        [ (pr rawASCII c c)
          ]
      ]
    ]
  ]
end
```

readChars *num***rCs *num***

reads "*num*" characters from the read stream and outputs those characters as a word. If the read stream is a file, and the end of file is reached, READCHARS outputs the empty list (not the empty word). If the read stream is a terminal, echoing is turned off when READCHARS is invoked, and remains off until READLIST or READWORD is invoked or a Logo prompt is printed. Backslash, vertical bar, and tilde characters have no special meaning in this context.

readIntBin

reads an integer (32 bit) from the read stream and outputs it as a new integer.

readInt16Bin

reads an integer (16 bit) from the read stream and outputs it as a new Int16.

readInt8Bin

reads an integer (8 bit) from the read stream and outputs it as a new Int8.

readUInt8Bin

reads an unsigned integer (8 bit) from the read stream and outputs it as a new UInt8.

readFloatBin

reads a float (64 bit) from the read stream and outputs it as a new float.

readComplexBin

reads a complex float (2*64 bit) from the read stream and outputs it as a new complex.

readIntArrayBin *size*

reads an IntArray (*size* *32 bit) from the read stream and outputs it as a new IntArray.

readInt16ArrayBin *size*

reads an Int16Array (*size* *16 bit) from the read stream and outputs it as a new Int16Array.

readFloatArrayBin *size*

reads an FloatArray (*size* *64 bit) from the read stream and outputs it as a new FloatArray.

readStructBin *astruct*

reads an Struct from the read stream and outputs it as a new Struct.

Shell *command*
 (Shell *command wordflag*)

Outputs the result of running "*command*" as a shell *command* . (The *command* is sent to /bin/sh, not csh or other alternatives.) If the *command* is a literal list in the instruction line, and if you want a backslash character sent to the shell, you must use \\ to get the backslash through Logo's reader intact. The output is a list containing one member for each line generated by the shell *command* . Ordinarily each such line is represented by a list in the output, as though the line were read using READLIST. If a second input is given, regardless of the value of the input, each line is represented by a word in the output as though it were read with READWORD.

Example:

```
to dir
  foreach bf bf bf shell [dir]
  [   if not empty? ?
      [   (pr last ? "\; butlast bf bf ?)
        ]
    ]
end
```

The Macintosh, of course, is not programmable.

ShellSpawn *command*

runs "*command*" as a shell *command* but unlike shell does not collect the output and does not wait for the operation to complete.

Example:

```
make "WebBrowser" "C:\\Progra\\~1\\Intern\\~1\\iexplore|  
; make "WebBrowser  
" "C:\\bin\\netscape\\program\\netscape.exe| ;as another example  
make "HTMLHelpDir" "C:\\aUCBLogo\\help
```

```
if not shellspawn (list (word :WebBrowser " | | :HTMLHelpDir "\\  
"index.frame.html))  
[ pr [File Not Found]  
]
```

LogoVersion

outputs a list with four elements: 1) the name of the Logo translator, 2) the version number of the Logo translator (float), 3) the operating system name, 4) the version number of the operating system.

Example:

```
show LogoVersion ;[aUCBLogo 4.689 Windows NT 5.1]
```

OSScreenSize

outputs a list of two integer numbers, width and height of the computer's current screen resolution.

Example:

Communication / Receivers / OSScreenSize

```
show OSScreenSize      ;[1280 1024]
```

FileAccess

...are functions for opening, positioning and closing files.

FileAccess

- openRead 100
 - openReadBin 102
 - openWrite 103
 - openWriteBin 103
 - openAppend 104
 - openAppendBin 104
 - openUpdate 105
 - openUpdateBin 105
 - close 105
 - allOpen 106
 - closeall 106
 - eraseFile 106, erF 106
 - dribble 107
 - noDribble 107
 - setReader 107
 - setWriter 108
 - Reader 109
 - Writer 109
 - setReaderPos 109
 - setWriterPos 110
 - ReaderPos 110
 - WriterPos 110
 - EofP 110
 - FileSize 111
 - FileTime 111
 - FileP 111
 - DirectoryP 111
 - getWorkingDirectory 112
 - changeDir 112, cd 112
 - makeDirectory 112
 - Files 113
 - loadpalette 113
-

Communication / FileAccess / openRead

openRead *filename*

command. Opens the named file for reading. The read position is initially at the beginning of the file.

Example:

```
openRead "tree.lg
setReader "tree.lg
while [not eof?] [pr readWord]
```

Whose output is:

```
to tree [level 7][size 100]
  if level==0 [stop]
  fd size
  lt 80
  tree2 level-1 size/2
  rt 70
  (tree level-1 size*2/3)
  rt 90
  tree2 level-1 size/2
  lt 80
  bk size
end size
```

```
to tree2 level size
  if level==0 [stop]
  fd size
  rt 80
  (tree level-1 size/2)
  lt 70
  tree2 level-1 size*2/3
  lt 90
  (tree level-1 size/2)
  rt 40
  (tree level-1 size*4/5)
  rt 40
  bk size
end size
```

Then you can ask:

```
ReaderPos      ;387  ;-)
```

openReadBin *filename*

command. Opens the named file for binary reading. The read position is initially at the beginning of the file.

Example:

```
to loadpalette file
  local [mypal]
  mypal=array 2^8
  openReadBin file
  setReader file
  while [not eof?]
    [   mypal.repcount=(rgb
      (rawASCII readChar)/2^8
      (rawASCII readChar)/2^8
      (rawASCII readChar)/2^8)
    ]
  close file
  setReader []
  output mypal
end
loadpalette "topograf.pal"
```

openWrite *filename*

command. Opens the named file for writing. If the file already existed, the old version is deleted and a new, empty file created.

Example:

```
openWrite "temp.txt"
setWriter "temp.txt"
pr [This is a test file!]
setWriter []
close "temp.txt"
```

And the file "temp.txt" then contains the line:

```
This is a test file!
```

openWriteBin *filename*

command. Opens the named file for binary writing. If the file already existed, the old version is deleted and a new, empty file created.

Example:

```
openWrite "temp.txt"
setWriter "temp.txt"
repeat 256 [type char repcount]
setWriter []
close "temp.txt"
```

And the file "temp.txt" then should contain all the 256 chars.

openAppend *filename*

command. Opens the named file for writing. If the file already exists, the write position is initially set to the end of the old file, so that newly written data will be appended to it.

Example:

```
openAppend "temp.txt"
setWriter "temp.txt"
pr [and a line more]
close "temp.txt"
writer    ;[temp.txt] ;-)
setWriter []
```

And "temp.txt" contains maybe this:

```
This is a test file!
and a line more
```

openAppendBin *filename*

Communication / FileAccess / openAppendBin

command. Opens the named file for binary writing. If the file already exists, the write position is initially set to the end of the old file, so that newly written data will be appended to it.

See openAppend!

openUpdate *filename*

command. Opens the named file for reading and writing. The read and write position is initially set to the end of the old file, if any. Note: each open file has only one position, for both reading and writing. If a file opened for update is both READER and WRITER at the same time, then SETREADERPOS will also affect WRITERPOS and vice versa. Also, if you alternate reading and writing the same file, you must SETREADERPOS between a write and a read, and SETWRITERPOS between a read and a write.

Example:

```
openUpdate "temp.txt
setReader "temp.txt
pr readlist      ;This is a test file!
pr readlist      ;and a line more
setReader []
close "temp.txt
```

openUpdateBin *filename*

command. Opens the named file for binary reading and writing. The read and write position is initially set to the end of the old file, if any. Note: each open file has only one position, for both reading and writing. If a file opened for update is both READER and WRITER at the same time, then SETREADERPOS will also affect WRITERPOS and vice versa. Also, if you alternate reading and writing the same file, you must SETREADERPOS between a write and a read, and SETWRITERPOS between a read and a write.

See openUpdate!

close *filename*

command. Closes the named file.

Example:

```
openRead "temp.txt"
close "temp.txt"
```

allOpen

outputs a list whose members are the names of all files currently open. This list does not include the dribble file, if any.

Example:

```
openRead "temp.txt"
show allOpen      ; [temp.txt]
close "temp.txt"
```

closeall (library *procedure*)

command. Closes all open files. Abbreviates FOREACH ALLOPEN [CLOSE ?]

eraseFile *filename***erF *filename***

command. Erases (deletes, removes) the named file, which should not currently be open.

Example:

```
eraseFile "temp.txt"
```

dribble *filename*

command. Creates a new file whose name is the input, like OPENWRITE, and begins recording in that file everything that is read from the keyboard or written to the terminal. That is, this writing is in addition to the writing to WRITER. The intent is to create a transcript of a Logo session, including things like prompt characters and interactions.

Example:

```
dribble "temp.txt  
callttt
```

Then play Tic-Tac-Toe and execute:

```
noDribble
```

After all this a transcript of the game should be in the file "temp.txt".

noDribble

command. Stops copying information into the dribble file, and closes the file.

setReader *filename***setReader *anintarray*****setReader *anint16array***

command. Makes the named file, *anintarray* or *anint16array* the read stream, used for READLIST, etc. The file must already be open with OPENREAD or OPENUPDATE. If the input is the empty list, then the read stream becomes the terminal, as usual. Changing the read stream does not close the file that was previously the read stream, so it is possible to alternate between files.

See also: openRead, setWriter, readIntBin

setWriter *filename*
setWriter *anintarray*
setWriter *anint16array*

command. Makes the named file, *anintarray* or *anint16array* the write stream, used for PRINT, etc. The file must already be open with OPENWRITE, OPENAPPEND, or OPENUPDATE. If the input is the empty list, then the write stream becomes the terminal, as usual. Changing the write stream does not close the file that was previously the write stream, so it is possible to alternate between files. On writing to *anintarray* or *anint16array* it need not to be opened before this command.

Examples:

```
[DataChunkID word data]
(list "DataChunkSize "Int size*2)
)
wavHeader=struct wavHeaderType
wavsize=(SizeOf wavHeader)+size*2
wavHeader'RIFFtype=[WAVE] ;example for setting a string
wavHeader'wavfilesize=wavsize
pr wavHeader
wav=Int16Array int wavsize/2
setWriter wav
typeBin wavHeader
repeat size
[ phi=360*repcount/rate
  typebin Int16 16383*( (sin 40*phi)
                      - (sin 41*phi) )
]
setWriter []
setReader wav
wh=readStructBin wavHeaderType
pr wh
setReader []
playWave wav 1+8
ignore readChar
playWave [] 0
```


end

Reader

outputs the name of the current read stream file, or the empty list if the read stream is the terminal.

Writer

outputs the name of the current write stream file, or the empty list if the write stream is the terminal.

setReaderPos *charpos*

command. Sets the file pointer of the read stream file so that the next READLIST, etc., will begin reading at the " *charpos* "th character in the file, counting from 0. (That is, SETREADERPOS 0 will start reading from the beginning of the file.) Meaningless if the read stream is the terminal.

Example:

```
openWrite "temp.txt
setWriter "temp.txt
pr [This is a temp file]
setWriter []
close "temp.txt
```

```
openRead "temp.txt
setReader "temp.txt
pr readList      ;This is a temp file
setReaderPos 6
pr readList      ;s a temp file
ReaderPos      ;21  ;-)
close "temp.txt
setReader []
```

setWriterPos *charpos*

command. Sets the file pointer of the write stream file so that the next PRINT, etc., will begin writing at the " *charpos* "th character in the file, counting from 0. (That is, SETWRITERPOS 0 will start writing from the beginning of the file.) Meaningless if the write stream is the terminal.

Works like setReaderPos, but for the Writer.

ReaderPos

outputs the file position of the current read stream file.

WriterPos

outputs the file position of the current write stream file.

EofP

Eof?

predicate, outputs TRUE if there are no more characters to be read in the read stream file, FALSE otherwise.

FileSize *filename*

outputs the size of the file named *filename* . The file must be open.

Example:

```
openread "bounce3.lg
filesize "bounce3.lg    ;3784    ;-)
close "bounce3.lg
```

FileTime *filename*

outputs the creation time and date of the file named *filename* in the following list: [Hour Minute Second MilliSecond DayOfWeek Day Month Year]

Example:

```
FileTime "bounce3.lg
```

FileP *filename***File? *filename***

predicate, outputs TRUE if a file of the specified name exists and can be read, FALSE otherwise.

Examples:

```
FileP "bounce.lg    ;true    ;-)
FileP "csls         ;false   ;-)
```

DirectoryP *dirname***Directory?** *dirname*

outputs true if the directory named *dirname* exists.

Examples:

```
Directory? "csls      ;true  ;-)
Directory? "bounce.lg ;false ;-)
```

getWorkingDirectory

outputs the current working directory.

Example:

```
getWorkingDirectory    ;C:\auchblogo  ;-)
```

changeDir *path***cd** *path*

command which changes the current working directory to *path* .

Examples:

```
cd "csls
ttt
cd "..
bounce
```

makeDirectory *dirname*

Communication / FileAccess / makeDirectory

command to create a directory named *dirname* in the current working directory.

Examples:

```
makeDirectory [TestDir]  
makeDirectory "testdir2"
```

Files

(Files *filespec flags*)

outputs the files matching the *filespec* and *flags* in the current working directory.

filespec is a string containing some wildcards like * and ?.

flags can be a combination of:

wxDIR_FILES include files

wxDIR_DIRS include directories

wxDIR_HIDDEN include hidden files

wxDIR_DOTDOT include '.' and '..'

Examples:

```
show Files ;...  
show (Files [a*.lg] wxDIR_FILES)  
;[am.lg am2.lg ant.lg ant2.lg arc2test.lg axes.lg]
```

loadpalette *palname*

operation which outputs an array of 256 integers representing the rgb-colors of the palette file *palname* .

Example:

```
topograf=loadpalette "topograf.pal  
setPenSize [10 10]  
repeat 256 [  
    setPenColor topograf.repcount  
    fd 1  
]
```

Environment

For customizing the Logo environment, file and dir locations, the most important can now be read and changed:

Environment

- LogoComspec 115
- setLogoComspec 115
- LogoEditor 116
- setLogoEditor 116
- LogoHelpDir 116
- setLogoHelpDir 116
- LogoLibDir 117
- setLogoLibDir 117
- LogoTempDir 117
- setLogoTempDir 117

LogoComspec

outputs the OS environment variable COMSPEC or the path set by setLogoComspec, which should point to the command shell of the OS.

Example:

```
show LogoComspec ; C:\WINDOWS\system32\cmd.exe
```

setLogoComspec *path*

Command to change the internal environment variable COMSPEC to the *path path* . This variable is used to localize the command shell, called in Shell and ShellSpawn.

Example:

```
setLogoComspec " | D:\WINXP\system32\cmd.exe |
```

LogoEditor

outputs the environment variable AEDITOR or the path set by setLogoEditor, which is either an empty word or the path to the external editor that Logo should call.

Example:

```
show LogoEditor ;c:\programme\crimson editor\cedt.exe
```

setLogoEditor *path*

Command to change the internal environment variable AEDITOR to the *path path* . The edit primitive will call that program pointed to by *path* .

Examples:

```
setLogoEditor [C:\\Programme\\Crimson Editor\\cedt.exe]  
setLogoEditor "
```

LogoHelpDir

outputs the environment variable ALOGOHELP or the path set by setLogoEditor, which is assumed as the path to the help files auchlogo/help/*.txt.

Example:

```
show LogoHelpDir ;help
```

setLogoHelpDir

Communication / Environment / setLogoHelpDir

Command to change the internal environment variable ALOGOHELP to the path *path*. The help primitive will read the help files from this location.

Example:

```
setLogoHelpDir [C:\\aucblogo\\help] ;for an absolut path
```

LogoLibDir

outputs the environment variable ALOGOLIB or the path set by setLogoLibDir, which will be taken as the location of the library procedures, normally found in aucblogo/lib/*.lg.

Example:

```
show LogoLibDir ;C:\\aUCBLogo\\lib
```

setLogoLibDir *path*

Command to change the internal environment variable ALOGOLIB to the *path path*. The Logo Reader will load library procedures from this location.

Example:

```
setLogoLibDir [C:\\aucblogo\\lib]
```

LogoTempDir

outputs the environment variable ATEMP or the path set by setLogoTempDir, where temporary edit session files are saved. This is normally aucblogo/temp.

Example:

```
show LogoTempDir ;C:\\aUCBLogo\\TEMP
```

setLogoTempDir

Command to change the internal environment variable ATEMP to the path path. That's where temporary edit session files are saved.

Example:

```
setLogoTempDir [C:\\aucblogo\\temp]
```

Terminal Access

These are functions controlling the text console.

Terminal Access

- KeyP 119
- clearText 119, cT 119
- setCursor 120
- Cursor 120
- CharUnderCursor 120
- WordUnderCursor 120
- setMargins 121
- setTextColor 121, setTC 121
- setTextFont 121
- setTextSize 122
- boldTextMode 122
- plainTextMode 122
- insertMode 122
- overwriteMode 122
- setTextSelection 123
- enableTextMouseEvents 123
- disableTextMouseEvents 123

KeyP

Key?

predicate, outputs TRUE if there are characters waiting to be read from the read stream. If the read stream is a file, this is equivalent to NOT EOF. If the read stream is the terminal, then echoing is turned off and the terminal is set to CBREAK (character at a time instead of line at a time) mode. It remains in this mode until some line-mode reading is requested (e.g., READLIST). The Unix operating system forgets about any pending characters when it switches modes, so the first KEYP invocation will always output FALSE.

clearText

cT

command. Clears the text screen of the terminal.

setCursor *vector*

command. The input is a list of two numbers, the x and y coordinates of a screen position (origin in the upper left corner, positive direction is southeast). The screen cursor is moved to the requested position. This command also forces the immediate printing of any buffered characters.

Examples:

```
setCursor [10 3]  
setCursor [0 0]
```

Cursor

outputs a list containing the current x and y coordinates of the screen cursor. Logo may get confused about the current cursor position if, e.g., you type in a long line that wraps around or your program prints escape codes that affect the terminal strangely.

CharUnderCursor

outputs the char under the cursor in the text console. This is very nice for cursorgames. See cursorgame.lg!

WordUnderCursor

outputs the word under the cursor in the text console. This is very nice for textmode menus. See the linked help [helpwalk.lg](..\helpwalk.lg)!

setMargins *vector*

does not work yet.

command. The input must be a list of two numbers, as for SETCURSOR. The effect is to clear the screen and then arrange for all further printing to be shifted down and to the right according to the indicated margins. Specifically, every time a newline character is printed (explicitly or implicitly) Logo will type *x_margin* spaces, and on every invocation of SETCURSOR the margins will be added to the input *x* and *y* coordinates. (CURSOR will report the cursor position relative to the margins, so that this shift will be invisible to Logo programs.) The purpose of this command is to accommodate the display of terminal screens in lecture halls with inadequate TV monitors that miss the top and left edges of the screen.

setTextColor *foreground background*

setTC *foreground background*

command (Windows and DOS extended only). The inputs are color numbers, as for turtle graphics. Future printing to the text window will use the specified colors for *foreground* (the characters printed) and *background* (the space under those characters). Using STANDOUT will revert to the default text window colors.

See also setPC and the color database.

setTextFont *fontName*

command which selects the *fontName* font into the console window. *fontName* is a string or a list of strings.

Example:

```
setTextFont [Courier New]  
pr [Hallo]
```

setTextSize *size*

command which sets the *size* of new text in the console window. *size* should be in moderate range (0..600).

Example:

```
setTextSize 20
```

boldTextMode

switches the console to bold text mode.

plainTextMode

turns bold text mode in the console off.

insertMode

command. Switches the console to insert mode. See [cursorgame.lg](..\cursorgame.lg)!

overwriteMode

Communication / Terminal Access / overwriteMode

command. Switches the console to overwrite mode. See [---](..\cursorgame.lg)

setTextSelection *startposlist endposlist*

command. Sets the current selection in the text console to the range from *startposlist* to *endposlist*. The poslists should be valid cursor locations.

Example:

```
setTextSelection [1 1][10 1]
setTextSelection [0 0][0 0]
```

enableTextMouseEvents

enables automatic processing of mouse events in the text console. This is the default status, so you can set the cursor clicking with the mouse, and select text with the mouse.

disableTextMouseEvents

disables automatic processing of mouse events in the text console. This can be very useful if you need full control over the cursor position and over text selections, like in [---](..\textmodevalues.lg)

Port Input and Output

Here is a group of functions to set, get and manipulate IO ports via io.dll.

Port Input and Output

- PortOut 124
 - PortIn 124
 - setPortBit 124
 - clearPortBit 125
 - getPortBit 125
 - notPortBit 125
 - leftPortShift 125
 - rightPortShift 125
-

PortOut *adress num*

command which sets the 8 bit unsigned integer *num* to the port *adress* .

Example:

```
PortOut 888 255  
;...sets all bits on the standard parallel port
```

PortIn *adress*

outputs the value of the bits of the port at *adress* .

Example:

```
show PortIn 888
```

Communication / Port Input and Output / setPortBit

setPortBit *address bitnr*

command that sets the bit number *bitnr* of the port *address* to logic high.

Example:

```
setPortBit 888 3
```

clearPortBit

command that clears the bit number *bitnr* of the port *address* to logic low.

Example:

```
clearPortBit 888 3
```

getPortBit *address bitnr*

outputs the boolean value of the bit *bitnr* of the port *address* .

notPortBit *address bitnr*

command which toggles the bit *bitnr* in the port *address* .

leftPortShift *address bitnr*

command that shifts the bits of the port *address* by *bitnr* to the left.

rightPortShift *adress bitnr*

command that shifts the bits of the port *adress* by *bitnr* to the right.

Timing

...functions are good to write machine independent timing,
and to read the system time.

Timing

- TimeFine 127
 - TimeMilli 127
 - TimeU 127
 - TimeURes 127
 - TimerFreq 128
 - Time 128
 - MIPS 128
-

TimeFine

outputs the system time in seconds as a floating point number.

TimeMilli

outputs the system time in milliseconds as a floating point number.

TimeU

outputs the system time in microseconds as a floating point number.

TimeURes

outputs the system time resolution in microseconds as a floating point number.

TimerFreq

outputs the system time frequency in megahertz as a floating point number.

`datetime`list **Time**

outputs a list containing the current real time clock time in a list: [Hour Minute Second MilliSecond DayOfWeek Day Month Year]

`mipsvalue` **MIPS**

outputs the million instructions per second of the current machine, which is very useful for writing machine-independent timing.

Dynamic Libraries

Here are a constructor for a dynamic library node which can be assigned to a variable and a dynamic library call function.

Dynamic Libraries

- DynamicLibrary 129
 - DynamicLibraryCall 130, DLCall 130
-

DynamicLibrary *dlname*

outputs a new dynamic library node with the name *dlname* which can be assigned to a variable for usage with DLCall. The dynamic library is being loaded and stays in memory as long as the variable, to which it is assigned, exists.

Examples:

```
to FindWindowA title
  output DLCall user32 [FindWindowA] (list "Int
    "Class "Int 0
    "Title "Word title)
end
to SetFocus hWnd
  ignore DLCall user32 [SetFocus] (list "Int
    "hWnd "Int hWnd)
end
user32=DynamicLibrary "user32
hWnd=FindWindowA [aUCBLogo-4.67]
SetFocus hWnd
```

```
to GetCurrentDirectory
  local [buf status w]
  buf=StringBuffer 256
  status=DLCall kernel32 [GetCurrentDirectoryA] (list "Int
    "bufferLength "Int 4*count buf
    "buffer "Word buf)
  output StringBufferToWord buf
end
kernel32=DynamicLibrary "kernel32
show GetCurrentDirectory
```

DynamicLibraryCall *aDynamicLibrary dfunctionname paramlist*

DLCall *aDynamicLibrary dfunctionname paramlist*

command or operation, depending on first of *paramlist*. The function with name *dfunctionname* in the dynamic library *aDynamicLibrary* is being called, if *aDynamicLibrary* has been set to a loaded DynamicLibrary. *dfunctionname* can also be a list with one element because so you can use case sensitive names without having to setCaseIgnored false.

The first item of *paramlist* is the return type of the function. It can be one of the following words:

```
"Int
"Int16
"Int8
"UInt8
"Float
"Word
"Void
```

Then the function's parameters follow in groups of three:

- 1.) The name of the parameter, which is only for documentation,
- 2.) The type of the parameter, which can be one of the following words:

```
"Int  
"IntPtr  
"Int16  
"IntPtr16  
"Int8  
"IntPtr8  
"UInt8  
"UInt8Ptr  
"Float  
"FloatPtr  
"Word  
"IntArray  
"Int16Array
```

The DLCall primitive supports now argument-by-pointer when "IntPtr, "IntPtr16, "IntPtr8, "UInt8Ptr or "FloatPtr are specified as argument type.

3.) The value of the parameter, which must match the type.

Examples:

```
to FindWindowA title
  output DLError user32 [FindWindowA] (list "Int
    "Class "Int 0
    "Title "Word title)
end
to GetWindowRect hWnd
  local [status iarr]
  iarr=IntArray 4
  status=DLError user32 [GetWindowRect] (list "Int
    "hWnd "Int hWnd
    "rect "IntArray iarr)
  output iarr
end
to MoveWindow hWnd x y
  local [xy status]
  rect=GetWindowRect hWnd
  status=DLError user32 [MoveWindow] (list "Int
    "hWnd "Int hWnd
    "X "Int x
    "Y "Int y
    "nWidth "Int rect.3-rect.1
    "nHeight "Int rect.4-rect.2
    "bRepaint "Int 1)
end
user32=DynamicLibrary "user32
hWnd=FindWindowA [aUCBLogo-4.67]
MoveWindow hWnd 200 100
```

Arithmetic

Arithmetic

In this group are numeric operations for computing, arithmetic mutators which change data itself mathematically, arithmetic predicates to ask boolean questions on numbers, functions for random numbers and rational numbers (fractions), print formatting and bitwise operations.

Arithmetic

- Numeric Operations 134
 - Arithmetic Mutators 160
 - Arithmetic Predicates 162
 - Random Numbers 165
 - Rational numbers 167
 - Print formatting 170
 - Bitwise Operations 172
-

Numeric Operations

...are needed for numeric computations.

Most of them also work on list and array structures containing numbers (if they work on lists, then they also work on arrays). In such a case, for a binary function, the first input sets the output structure, so if the first input is a number, then the result is a number, too.

The numeric operations which take two arguments (binary math functions) are non-commutative primitives, so the order of arguments does matter, if one arg is a list and the other a scalar. It was much easier to make **all** binary math functions non-commutative than only just some of them.

You can also use numeric infix operators `+` `-` `*` `/` for computations.

Numeric Operations

- Sum 135
- Difference 136
- Minus137
- Product 137
- Quotient 138
- Remainder 138
- Modulo 139, mod 139
- Float 139
- BigFloat 140
- BigFloatSetPrecision 140
- Int 140
- Int16 141
- Int8 141
- UInt8 142
- round 142
- truncate 143, trunc 143
- abs 143
- Signum 143
- Sqr 144
- Sqrt 144
- Power 145
- exp 145

Arithmetic / Numeric Operations

- Log10 145
- LN146
- Sin 146
- radSin 147
- Cos 147
- radCos 148
- Tan 148
- radTan 148
- ArcSin 149
- radArcSin 149
- ArcCos 150
- radArcCos 150
- ArcTan 150
- radArcTan 151
- Faculty 151
- factorize 152
- min 152
- max 153
- Norm 153
- maxNorm 153
- iSeq 154
- rSeq 154
- rSeqFloatArray 154, rSeqFA 154
- gcd 155
- lcm 155
- resize 155
- lowPassFilter 156
- saturateAbove 156
- saturateBelow 157
- cross 157
- invertMatrix 157
- transposematrix 158, transpose 158
- MandellIterate 158

Sum *num1 num2*

(Sum *num1 num2 num3 ...*)

num1 + num2

outputs the sum of its inputs.

It's a non-commutative primitive (like all binary math functions), so the order or arguments does matter, if one arg is a list and the other a scalar.

Example:

```
1+2      ;3      ;-)
1.2+2.4   ;3.6    ;-)
1i+2i     ;0+3i   ;-)
[1 2]+[3 4] ;[4 6] ;-)
[1 2]+3    ;[4 5] ;-)
1+[2 3]    ;6      ;-)
```

```
sum 1 2      ;3      ;-)
sum 1.2 2.4   ;3.6    ;-)
sum 1i 2i     ;0+3i   ;-)
sum [1 2][3 4] ;[4 6] ;-)
sum [1 2] 3    ;[4 5] ;-)
sum 1 [2 3]    ;6      ;-)
(sum 1 2 3 4 5) ;15    ;-)
```

Difference *num1 num2*

num1 - num2

outputs the difference of its inputs. Minus sign means infix difference in ambiguous contexts (when preceded by a complete expression), unless it is preceded by a space and followed by a nonspace.

Examples:

```
1-2      ;-1      ;-)
1.2-2.4   ;-1.2    ;-)
1i-2i     ;0-1i   ;-)
[1 2]-[3 4] ;[-2 -2] ;-)
[1 2]- 3    ;[-2 -1] ;-)
1-[2 3]    ;-4      ;-)
```

```

Difference 1 2      ; -1  ; -)
Difference 1.2 2.4   ; -1.2 ; -)
Difference 1i 2i     ; 0-1i  ; -)
Difference [1 2][3 4] ; [-2 -2] ; -)
Difference [1 2] 3    ; [-2 -1] ; -)
Difference 1 [2 3]    ; -4  ; -)

```

Minus *num*

- *num*

outputs the negative of its input. Minus sign means unary minus if it is immediately preceded by something requiring an input, or preceded by a space and followed by a nonspace. There is a difference in binding strength between the two forms:

```

MINUS 3 + 4 means -(3+4)
- 3 + 4      means (-3)+4

```

Product *num1 num2*

(Product *num1 num2 num3 ...*)

num1 * *num2*

outputs the product of its inputs.

It's a non-commutative primitive (like all binary math functions), so the order or arguments does matter, if one arg is a list and the other a scalar.

Examples:

```

2*3      ; 6  ; -)
2*pi     ; 6.28319  ; -)
[1 2 3]*[4 5 6] ; [4 10 18] ; -)
0+[1 2 3]*[4 5 6] ; 32  ; -) ==scalar product!
1*[2 3 4] ; 24  ; -) multiply all numbers in a list
[2 3 4]*2   ; [4 6 8] ; -)

```

```
Product 2 3      ;6      ;-)
Product 2 pi      ;6.28319 ;-)
Product [1 2 3] [4 5 6] ;[4 10 18] ;-)
0+Product [1 2 3] [4 5 6] ;32 ;-) ==scalar product!
```

Quotient *num1 num2***(Quotient *num*)***num1 / num2*

outputs the quotient of its inputs. The quotient of two integers is an integer if and only if the dividend is a multiple of the divisor. (In other words, QUOTIENT 5 2 is 2.5, not 2, but QUOTIENT 4 2 is 2, not 2.0 -- it does the right thing.) With a single input, QUOTIENT outputs the reciprocal of the input.

Examples:

```
2/3      ;0.666667 ;-)
pi/2      ;1.5708  ;-)
[1 2 3]/[4 5 6] ;[0.25 0.4 0.5] ;-)
```

```
Quotient 2 3      ;0.666667 ;-)
Quotient pi 2      ;1.5708  ;-)
Quotient [1 2 3] [4 5 6] ;[0.25 0.4 0.5] ;-)
(Quotient 4)      ;0.25    ;-)
```

Remainder *num1 num2*

outputs the remainder on dividing " *num1* " by " *num2* "; both must be integers and the result is an integer with the same sign as *num1* .

Examples:

Arithmetic / Numeric Operations / Remainder

```
Remainder 4 3      ;1  ;-)
Remainder 5 3      ;2  ;-)
Remainder 6 3      ;0  ;-)
Remainder 7 3      ;1  ;-)
Remainder -7 3     ;-1  ;-)
Modulo -7 3       ;2  ;-)
Remainder [9 10 11][9 9 9] ;[0 1 2] ;-)
```

Modulo *num1 num2***mod *num1 num2***

outputs the remainder on dividing " *num1* " by " *num2* "; both must be integers and the result is an integer with the same sign as *num2* .

Examples:

```
mod 4 3      ;1  ;-)
mod 5 3      ;2  ;-)
mod 6 3      ;0  ;-)
mod 7 3      ;1  ;-)
mod -7 3     ;2  ;-)
Remainder -7 3 ;-1  ;-)
mod [9 10 11][9 9 9] ;[0 1 2] ;-)
```

Float *num*

outputs its input *num* converted to float numbers. *num* can be any number or list or array of numbers.

Exampels:

```
Float 1 ;1 ;-)
TypeOf 1 ;Int ;-)
TypeOf Float 1 ;Float ;-)
Float [1 2 3] ;[1 2 3] ;-)
Float {1 2 3} ;{1 2 3} ;-)
```

BigFloat *num*

outputs its input *num* converted to BigFloat numbers. *num* can be any number or list or array of numbers.

Exampel:

```
show 2*radArcCos BigFloat 0
```

BigFloatSetPrecision *precision*

Command to set the internal and print *precision* of BigFloat numbers to *precision* .

Example:

```
BigFloatSetPrecision 128
show 2*radArcCos BigFloat 0 ;gives Pi
```

Int *num*

outputs its input with fractional part removed, i.e., an integer with the same sign as the input, whose absolute value is the largest integer less than or equal to the absolute value of the input.

Note: Inside the computer numbers are represented in two different forms, one for integers and one for numbers with fractional parts. However, on most computers the largest number that can be represented in integer format is smaller than the largest integer that can be represented (even with

Arithmetic / Numeric Operations / Int

exact precision) in floating-point (fraction) format. The INT operation will always output a number whose value is mathematically an integer, but if its input is very large the output may not be in integer format. In that case, operations like REMAINDER that require an integer input will not accept this number.

Examples:

```
Int 1.2      ;1      ;-)
Int -1.2     ;-1     ;-)
Int 1.9      ;1      ;-)
Int -1.9     ;-1     ;-)
Int [1.2 -1.2 1.9 -1.9] ;[1 -1 1 -1] ;-)
```

Int16 *num*

outputs a 16 bit integer whose value is *num* with its fractional part truncated. This is mainly interesting when writing to binary files or defining a Struct and then writing that to a binary file.

Examples:

```
show Int16 32767      ;32767
show Int16 32768      ; int16 doesn't like 32768 as input
show Int16 -32768     ;-32768
show Int16 -32769     ; int16 doesn't like -32769 as input
show Int16 [1.2 -1.2 1.9 -1.9] ;[1 -1 1 -1]
show Int16 {1.2 -1.2 1.9 -1.9} ;{1 -1 1 -1}
```

Int8 *num*

outputs a 8 bit integer whose value is *num* with its fractional part truncated. This is mainly interesting when writing to binary files or defining a Struct and then writing that to a binary file.

Examples:

```
show Int8 127      ;127
show Int8 128      ; int8 doesn't like 128 as input
show Int8 -128     ; -128
show Int8 -129     ; int8 doesn't like -129 as input
show Int8 [1.2 -1.2 1.9 -1.9] ;[1 -1 1 -1]
show Int8 {1.2 -1.2 1.9 -1.9} ;{1 -1 1 -1}
```

UInt8 *num*

outputs a 8 bit unsigned integer whose value is *num* with its fractional part truncated. This is mainly interesting when writing to binary files or defining a Struct and then writing that to a binary file.

Examples:

```
show UInt8 255     ;255
show UInt8 256     ; uint8 doesn't like 256 as input
show UInt8 0       ;0
show UInt8 -1      ; uint8 doesn't like -1 as input
show UInt8 [1.2 -1.2 1.9 -1.9] ; uint8 doesn't like -1.2 as
input
show UInt8 {1.2 -1.2 1.9 -1.9} ; uint8 doesn't like -1.2 as
input
```

round *num*

outputs the nearest integer to the input.

Examples:

Arithmetic / Numeric Operations / round

```
round 1.2      ;1  ;-)
round -1.2     ;-1 ;-)
round 1.9      ;2  ;-)
round -1.9     ;-2 ;-)
round [1.2 -1.2 1.9 -1.9] ;[1 -1 2 -2] ;-)
```

truncate *num***trunc *num***

outputs the biggest integer absolutely smaller than *num* .

Examples:

```
trunc 1.2      ;1  ;-)
trunc -1.2     ;-1 ;-)
trunc 1.9      ;1  ;-)
trunc -1.9     ;-1 ;-)
trunc [1.2 -1.2 1.9 -1.9] ;[1 -1 1 -1] ;-)
```

abs *num*

outputs the absolute value of *num* , that is *num* with its sign made positive.

Examples:

```
abs -1      ;1  ;-)
abs -1.2    ;1.2 ;-)
abs [-1 -2.2] ;[1 2.2] ;-)
```

Signum *num*

outputs 1 if *num* is greater zero. outputs -1 if *num* is below zero, else outputs 0.

Examples:

```
Signum 0          ;0   ;-)
Signum 1234       ;1   ;-)
Signum -4321      ;-1  ;-)
Signum pi         ;1   ;-)
Signum -pi        ;-1  ;-)
Signum 1.2+0i     ;1   ;-)
Signum -1.2+0i    ;-1  ;-)
Signum 1i         ; Signum doesn't like 0+1i as input
Signum (list 1234 -4321 0 1.2 -1.2 pi+0i -pi+0i)
;[1 -1 0 1 -1 1 -1] ;-)

```

Sqr *num*

outputs the square of the input.

Examples:

```
Sqr 2          ;4   ;-)
Sqr 0.25       ;0.0625 ;-)
Sqr 1i         ;-1+0i ;-)
Sqr [2 0.25 1i] ;[4 0.0625 -1+0i] ;-)

```

Sqrt *num*

outputs the square root of the input.

Examples:

Arithmetic / Numeric Operations / Sqrt

```
Sqrt 4      ;2      ;-)
Sqrt 2      ;1.41421 ;-)
Sqrt -1     ;0+1i    ;-)
Sqrt [4 2 -1] ;[2 1.41421 0+1i] ;-)
```

Power *num1 num2*

num1 ^ *num2*

outputs "*num1*" to the "*num2*" power. If *num1* is negative, then *num2* must be an integer.

Examples:

```
2^0      ;1      ;-)
2^1      ;2      ;-)
2^2      ;4      ;-)
2^3      ;8      ;-)
2^(1/12) ;1.05946 ;-)
4^2      ;16     ;-)
10^300   ;1e+300 ;-)
Power [2 2 2 2 2 4 10] (list 0 1 2 3 1/12 2 300)
;[1 2 4 8 1.05946 16 1e+300] ;-)
```

exp *num*

outputs e (2.718281828+) to the input power.

Examples:

```
exp 0      ;1      ;-)
exp 1      ;2.71828 ;-)
exp 2      ;7.38906 ;-)
exp 1i*pi   ;-1+1.22461e-016i ;-) nearly -1, so nearly correct
exp 1i*2*pi ;1-2.44921e-016i  ;-) nearly 1, so nearly correct
exp (list 0 1 2 1i*pi 2i*pi)
;[1 2.71828 7.38906 -1+1.22461e-016i 1-2.44921e-016i] ;-)
```

Log10 *num*

outputs the common (decimal) logarithm of the input.

Examples:

```
Log10 10          ;1   ;-)
Log10 100         ;2   ;-)
Log10 1000        ;3   ;-)
Log10 1           ;0   ;-)
Log10 1/10        ;-1  ;-)
Log10 (list 1/10 1 10 100 1000) ;[-1 0 1 2 3] ;-)
```

LN *num*

outputs the natural logarithm (Logarithmus Naturalis) of the input.

Examples:

```
LN exp 1          ;1   ;-)
LN (exp 1)^2       ;2   ;-)
LN (list (exp 1)^-1 1 exp 1 (exp 1)^2) ;[-1 0 1 2] ;-)
```

Sin *degrees*

outputs the sine of its input, which is taken in *degrees* .

Examples:

```

Sin 0      ;0      ;-)
Sin 90     ;1      ;-)
Sin 180    ;5.30717e-017  ;-)
Sin 270    ;-1     ;-)
Sin 360    ;0      ;-)
Sin 45     ;0.707107  ;-)
Sin [0 45 90 135 180 225 270 315 360]
;[0 0.707107 1 0.707107 5.30717e-017 -0.707107 -1 -0.707107 0] ;-)
Sin {0 45 90 135 180 225 270 315 360}
;{0 0.707107 1 0.707107 5.30717e-017 -0.707107 -1 -0.707107 0}
;-)

```

radSin *radians*

outputs the sine of its input, which is taken in *radians* .

Examples:

```

radSin 0      ;0      ;-)
radSin pi/2   ;1      ;-)
radSin pi     ;1.22461e-016  ;-)
radSin 2*pi   ;-2.44921e-016  ;-)
radSin (list 0 pi/2 pi 3/2*pi 2*pi)
;[0 1 1.22461e-016 -1 -2.44921e-016] ;-)

```

Cos *degrees*

outputs the cosine of its input, which is taken in *degrees* .

Examples:

```

Cos 0      ;1      ;-)
Cos 90     ;2.65358e-017  ;-)
Cos 180    ;-1     ;-)
Cos 270    ;-7.96075e-017  ;-)
Cos 360    ;1      ;-)
Cos [0 45 90 135 180 225 270 315 360]
;[1 0.707107 2.65358e-017 -0.707107 -1 -0.707107 -7.96075e-017
0.707107 1] ;-)

```

radCos *radians*

outputs the cosine of its input, which is taken in *radians* .

Examples:

```

radCos 0      ;1      ;-)
radCos pi/2   ;6.12303e-017  ;-)
radCos pi     ;-1     ;-)
radCos 2*pi   ;1      ;-)
radCos (list 0 pi/2 pi 3/2*pi 2*pi)
;[1 6.12303e-017 -1 -1.83691e-016 1] ;-)

```

Tan *degrees*

outputs the tangens of its input, which is taken in *degrees* .

Examples:

```

Tan 0      ;0      ;-)
Tan 45     ;1      ;-)
Tan 60     ;1.73205  ;-)
Tan -45    ;-1     ;-)
Tan [0 45 60] ;[0 1 1.73205] ;-)

```

radTan *radians*

outputs the tangens of its input, which is taken in *radians* .

Examples:

```
radTan 0          ;0    ;-)
radTan pi/4       ;1    ;-)
radTan pi/3       ;1.73205 ;-)
radTan (list 0 pi/4 pi/3) ;[0 1 1.73205] ;-)
```

ArcSin *x*

outputs the arcus sine of its input. The result is in degrees.

Examples:

```
ArcSin 0          ;0    ;-)
ArcSin 1/2        ;30   ;-)
ArcSin (sqrt 3)/2 ;60   ;-)
ArcSin 1          ;90   ;-)
ArcSin -1         ;-90  ;-)
ArcSin (list 0 1/2 (Sqrt 3)/2 1) ;[0 30 60 90] ;-)
```

radArcSin *x*

outputs the arcus sine of its input. The result is in radians.

Examples:

```
radArcSin 0      ;0  ;-)
pi/(radArcSin 1/2)      ;6  ;-)
pi/(radArcSin (Sqrt 3)/2)      ;3  ;-)
pi/radArcSin 1      ;2  ;-)
as=radArcSin (list 1/2 (Sqrt 3)/2 1)
(rseq pi pi 3)/as      ;[6 3 2] ;-)
```

ArcCos x

outputs the arcus cosine of its input. The result is in degrees.

Examples:

```
ArcCos 1      ;0  ;-)
ArcCos (Sqrt 3)/2      ;30  ;-)
ArcCos 1/Sqrt 2      ;45  ;-)
ArcCos 1/2      ;60  ;-)
ArcCos 0      ;90  ;-)
ArcCos (list 1 (Sqrt 3)/2 1/Sqrt 2 1/2 0)      ;[0 30 45 60 90] ;-)
```

radArcCos x

outputs the arcus cosine of its input. The result is in radians. See ArcCos!

Examples:

```
radArcCos 1      ;0  ;-)
pi/radArcCos (Sqrt 3)/2      ;6  ;-)
pi/radArcCos 1/Sqrt 2      ;4  ;-)
pi/radArcCos 1/2      ;3  ;-)
pi/radArcCos 0      ;2  ;-)
pi/radArcCos -1      ;1  ;-)
as=radArcCos (list (Sqrt 3)/2 1/Sqrt 2 1/2 0)
(rseq pi pi 4)/as      ;[6 4 3 2] ;-)
```

ArcTan x y

outputs the arctangent of y/x , in degrees, of its input, if x is nonzero, or 90 or -90 depending on the sign of y , if x is zero.

Examples:

```
ArcTan 1 0      ;0      ;-)
ArcTan 1 1      ;45     ;-)
ArcTan 1 Sqrt 3  ;60     ;-)
ArcTan 1 1e300   ;90     ;-)
ArcTan -1 1      ;135    ;-)
ArcTan -1 -1     ;-135   ;-)
ArcTan 1 -1      ;-45    ;-)
ArcTan rseq 1 1 5 (list 0 1 Sqrt 3 1e300 -1)
;[0 45 60 90 -45] ;-)
```

radArcTan x y

outputs the arctangent of y/x , in radians, of its input, if x is nonzero, or $\pi/2$ or $-\pi/2$ depending on the sign of y , if x is zero.

The expression

```
2*(RADARCTAN 0 1)
```

can be used to get the value of π .

Examples:

```
radArcTan 1 0      ;0      ;-)
pi/(radArcTan 1 1)  ;4      ;-)
pi/(radArcTan 1 Sqrt 3) ;3      ;-)
pi/(radArcTan 1 1e300) ;2      ;-)
```

Faculty *integer*

outputs the faculty of the *integer* . This primitive is exact if possible.

Examples:

```
Faculty 0      ;1      ;-)
Faculty 1      ;1      ;-)
Faculty 2      ;2      ;-)
Faculty 3      ;6      ;-)
Faculty 4      ;24     ;-)
Faculty 5      ;120    ;-)
```

factorize *integer*

outputs a list containing the prime factors of the *integer* .

Examples:

```
factorize 12      ;[3 2 2] ;-)
factorize 120     ;[5 3 2 2 2] ;-)
bigFloatSetPrecision 800
show factorize Faculty 333 ;wait a few seconds and watch!
```

min *num1 num2*
(min *num1 num2 ...*)

outputs the minimum of all the number parameters.

Examples:

```
min 1 2      ;1  ;-)
(min 5 2 2.3 5.0 2.9 3.0)      ;2  ;-)
(min [1 2 3 4]) ;1  ;-)
(min {4 3 2 1}) ;1  ;-)
```

max *num1 num2*
(max *num1 num2 ...*)

outputs the maximum of all the number parameters.

Examples:

```
max 1 2      ;2  ;-)
(max 5 2 2.3 5.0 2.9 3.0)      ;5  ;-)
(max [1 2 3 4]) ;4  ;-)
(max {4 3 2 1}) ;4  ;-)
```

Norm *num*
(Norm *num1 num2 ...*)
Norm *vectorlistOrArray*

outputs the norm of the numbers, meaning $\sqrt{\text{num1}^2 + \text{num2}^2 + \dots}$

Examples:

```
(Norm 3 4)      ;5  ;-)
Norm [3 4]      ;5  ;-)
```

maxNorm *num*
(maxNorm *num1 num2 ...*)

outputs the maximum norm of the numbers, meaning $\max(\text{num1}, \text{num2}, \dots)$

Examples:

```
(maxNorm 3 4)      ; 4      ; -)
maxNorm [3 4]      ; 4      ; -)
```

iSeq *from to*

outputs a list of the integers *from* FROM *to* TO, inclusive.

Examples:

```
iSeq 3 7           ; [3 4 5 6 7] ; -)
iSeq 7 3           ; [7 6 5 4 3] ; -)
```

rSeq *from to count*

outputs a list of COUNT equally spaced rational or complex numbers between FROM and TO, inclusive. rSeq is also useful *to* generate a list of equal numbers, like in the last of the examples.

Examples:

```
rSeq 3 5 9         ; [3 3.25 3.5 3.75 4 4.25 4.5 4.75 5] ; -)
rSeq 3 5 5         ; [3 3.5 4 4.5 5] ; -)
rSeq pi pi 4       ; [3.14159 3.14159 3.14159 3.14159] ; -)
```

rSeqFloatArray *from to count*

rSeqFA *from to count*

outputs a FloatArray of COUNT equally spaced rational or complex numbers between FROM and

Arithmetic / Numeric Operations / rSeqFloatArray

TO, inclusive. rSeqFA is also useful *to* generate a FloatArray of equal numbers, like in the last of the examples.

Examples:

```
rSeqFA 3 5 9           ;[3 3.25 3.5 3.75 4 4.25 4.5 4.75 5] ;-)
rSeqFA 3 5 5           ;[3 3.5 4 4.5 5] ;-)
rSeqFA pi pi 4         ;[3.14159 3.14159 3.14159 3.14159] ;-)
```

gcd *number1 number2*

outputs the greatest common divisor of the two input numbers.

Examples:

```
gcd 8 12           ;4 ;-)
gcd 10 20          ;10 ;-)
```

lcm *number1 number2* (library procedure)

outputs the least common multiple of the two input numbers.

Example:

```
lcm 8 12
24 ;-)
```

resize *array newsiz*

outputs an interpolated *array* of size *newsiz* .

Examples:

```
resize {1 1} 5      ;{1 1 1 1 1}  ;-)
resize {1 2} 5      ;{1 1.25 1.5 1.75 2}  ;-)
resize {1 2 3} 5    ;{1 1.5 2 2.5 3}  ;-)
resize {1 2 3} 6    ;{1 1.5 2 2.33333 2.66667 3}  ;-)
```

lowPassFilter *in weight*

outputs a new FloatArray, IntArray or Int16Array, depending on the type of the input, of the low pass filtered FloatArray, IntArray or Int16Array *in* . *weight* is the weighting of the average used for the computation. The applied simple algorithm is

```
m.(1)= in .(1)
for [n 1 [count in ]]
[   out.n=(m.n* weight + in .n)/( weight +1)
    m.(n+1)=out.n
]
```

Example:

```
x=rSeqFA -1 1 1000
y=(sin x*x*20*360)*300
x=x*400
setPC 0
setXY x y
setPC 4
setXY x FloatArray lowPassFilter y 10
cs
setXY x FloatArray lowPassFilter IntArray y 10
cs
setXY x FloatArray lowPassFilter Int16Array y 10
```

saturateAbove *limit in*

Arithmetic / Numeric Operations / saturateAbove

outputs a new FloatArray, IntArray or Int16Array, depending on the type of the input, of the saturated FloatArray, IntArray or Int16Array *in* . Every item of *in* is checked if it goes above the *limit* . If it does, the *limit* is output.

Example:

```
saturateAbove 50 rseqFA 1 100 10
;{1 12 23 34 45 50 50 50 50 50} ; -)
```

saturateBelow *limit in*

outputs a new FloatArray, IntArray or Int16Array, depending on the type of the input, of the saturated FloatArray, IntArray or Int16Array *in* . Every item of *in* is checked if it goes above below *limit* . If it does, the *limit* is output.

Example:

```
saturateBelow 50 rseqFA 1 100 10
;{50 50 50 50 50 56 67 78 89 100} ; -)
```

z cross x y

outputs the cross product of the 3d arrays *x* and *y* .

Example:

```
cross {1 0 0}{0 1 0} ;{0 0 1} ; -)
```

invertedMatrix invertMatrix *inputMatrix*

outputs the invertedMatrix of the *inputMatrix* . If the determinant is zero an error is thrown. There are internal special cases for d=1x1,2x2,3x3 so then the runtime is optimal.

inputMatrix : array n of array n. The inner arrays are the column vectors.

Example:

```
invertMatrix {{1 0 1}{2 1 2}{3 2 1}}
{
  {1.5 -1 0.5}
  {-2 1 0}
  {-0.5 1 -0.5}
} ; -)
```

transposematrix *amatrix*

transpose *amatrix*

outputs the transposed *amatrix* .

Example:

```
transpose [[1 2][3 4][5 6]]
[
  [1 3 5]
  [2 4 6]
]
; -)
```

MandelIterate *z c maxiter*

is a nice example for a user-defined primitive. It helps computing the Mandelbrot set faster. Like in:

```
repeat maxiter [
  z *= z
  z += c
  if z > 4 [output repcount]
]
output maxiter
```

Additionally z and c can be of type Array of complex numbers, so one can compute several pixels at once. For examples see `mandel.lg...mandel5.lg`!

Arithmetic Mutators

...change the input data mathematically. For a nice example see [pixtest.lg!](#)

But be careful: the xCommands are very picky about the argument types, they must be exactly the same. So if you have a i.e. list of numbers, be sure to have either only Int's or only Float's inside. A mixture will probably produce an error.

Arithmetic Mutators

- xCopy 160
- xAdd 160
- xSub 160
- xMul 161
- xDiv 161
- xMod 161

xCopy *var value*

assigns *value* to the variable *var* without creating a new object, which is faster than `=`. But this is a dangerous function, especially with arrays.

xAdd *var value*

var += *value*

adds *value* to the variable *var* without creating a new number, which is faster than `+`.

Examples:

```
a=int 0
repeat 1000000 [a+=1]
a ;1000000 ;-)
```

Arithmetic / Arithmetic Mutators / xSub

xSub *var value**var -= value*

subtracts *value* from the variable *var* without creating a new number, which is faster than `-`.

Example:

```
a=int 1000000
repeat 1000000 [a-=1]
a      ;0      ;-)
```

xMul *var value**var *= value*

multiplies the variable *var* by *value* without creating a new number, which is faster than `*`.

xDiv *var value**var /= value*

divides the variable *var* by *value* without creating a new number, which is faster than `/`.

xMod *var value*

changes the variable *var* to its modulo of *value* without creating a new number, which is faster than `mod`.

Arithmetic Predicates

...are arithmetic questions with boolean answers. For many examples see `check\checkcompare.lg!`

Arithmetic Predicates

- `lessP` 162
- `greaterP` 162
- `lessEqualP` 163
- `greaterEqualP` 163
- `primeP` 164

`lessP thing1 thing2`

`less? thing1 thing2`

thing1 < thing2

outputs TRUE if its first input is strictly less than its second.

If the args are numbers they are compared by their numerical value. If they are strings they are string-compared. if they are lists or arrays, they are elementwise compared. Otherwise the node pointers are compared.

Examples:

```
1 < 2           ;true  ;-)
1 < 1           ;false ;-)
"a < "b         ;true  ;-)
[1 0] < [0 1]   ;false ;-)
[1 0] < [1 1]   ;true  ;-)
```

`greaterP thing1 thing2`

`greater? thing1 thing2`

thing1 > thing2

outputs TRUE if its first input is strictly greater than its second.

Arithmetic / Arithmetic Predicates / greaterP

If the args are numbers they are compared by their numerical value. If they are strings they are string-compared. if they are lists or arrays, they are elementwise compared. Otherwise the node pointers are compared.

Examples:

```
1 > 1      ;false  ;-)
2 > 1      ;true   ;-)
"b > "a    ;true   ;-)
[1 1] > [1 0] ;true   ;-)
```

lessEqualP *thing1 thing2*

lessEqual? *thing1 thing2*

thing1 <= thing2

outputs TRUE if its first input is less or equal than its second.

If the args are numbers they are compared by their numerical value. If they are strings they are string-compared. if they are lists or arrays, they are elementwise compared. Otherwise the node pointers are compared.

Examples:

```
1 <= 1     ;true   ;-)
1 <= 2     ;true   ;-)
1 <= 0     ;false  ;-)
"a <= "b   ;true   ;-)
"a <= "a   ;true   ;-)
"b <= "a   ;false  ;-)
[1 0] <= [1 1] ;true   ;-)
```

greaterEqualP *thing1 thing2*

greaterEqual? *thing1 thing2*

thing1 >= thing2

outputs TRUE if its first input is greater or equal than its second.

If the args are numbers they are compared by their numerical value. If they are strings they are string-compared. if they are lists or arrays, they are elementwise compared. Otherwise the node pointers are compared.

Examples:

```
1>=1      ;true  ;-)
2>=1      ;true  ;-)
1>=2      ;false ;-)
"a >= "b   ;false ;-)
[1 1] >= [1 0] ;true  ;-)
```

primeP *integer*

prime? *integer*

isprime *integer* (library procedure)

outputs true if the *integer* is a prime number, otherwise false.

Examples:

```
prime? 0      ;true  ;-)
prime? 1      ;true  ;-)
prime? 2      ;true  ;-)
prime? 3      ;true  ;-)
prime? 4      ;false ;-)
prime? 5      ;true  ;-)
prime? 6      ;false ;-)
prime? 7      ;true  ;-)
prime? 8      ;false ;-)
prime? 9      ;false ;-)
prime? 431    ;true  ;-)
prime? 2^32-5 ;true  ;-)
```

Random Numbers

...are created using `random` and `rnd`.

The random number generator can be set to a reproducible sequence using `reRandom`.

Random Numbers

- `random` 165
- `reRandom` 165
- `rnd` 165

`random` *num*

outputs a random nonnegative integer less than its input, which must be an integer, or a list or array of integers.

Examples:

```
random 10      ;4 ;-)
random [100 200 300] ;[18 113 208] ;-)
random {100 200 300} ;{78 100 135} ;-)
```

`reRandom` (**`reRandom`** *seed*)

command. Makes the results of `RANDOM` reproducible. Ordinarily the sequence of random numbers is different each time Logo is used. If you need the same sequence of pseudo-random numbers repeatedly, e.g. to debug a program, say `RERANDOM` before the first invocation of `RANDOM`. If you need more than one repeatable sequence, you can give `RERANDOM` an integer input; each possible input selects a unique sequence of numbers.

rnd

outputs a random floating point number in the range of 0 to 1.

Rational numbers

...are implemented as lists of two numbers, the nominator and the denominator. There's no internal Logo node type "Ratio" yet.

Rational numbers

- ratio 167
- float2ratio 167
- ratio2float 168
- radd 168
- rsub 168
- rmul 168
- rdiv 169

ratio *nominator denominator* (library procedure)

outputs a rational number as a list of two numbers, *nominator* and denominator, shortened.

Examples:

```
ratio 20 10    ;2    ;-)
ratio 10 20    ;[1 2] ;-)
ratio 3 4      ;[3 4] ;-)
ratio 12 8     ;[3 2] ;-)
```

float2ratio *number* (library procedure)

outputs a list of nominator and denominator equivalent to the floating point number.

Examples:

```
float2ratio 0.25      ;[1 4] ;-)
float2ratio 1.2       ;[6 5] ;-)
float2ratio 3.14      ;[157 50] ;-)
```

ratio2float *rationr* (library procedure)

outputs a floating point number equivalent to the rational number *rationr*.

Examples:

```
ratio2float [1 10]    ;0.1 ;-)
ratio2float [2 3]     ;0.666667 ;-)
```

radd *ratio1 ratio2* (library procedure)

outputs a rational number as a list of two numbers, nominator and denominator, which is the sum of the two input rationals.

Example:

```
radd [1 2][1 2]      ;1 ;-)
radd [1 2][3 4]      ; 1/2+3/4=5/4 ;[5 4] ;-)
```

rsub *ratio1 ratio2* (library procedure)

outputs a rational number as a list of two numbers, nominator and denominator, which is the difference of the two input rationals.

Example:

```
rsub [1 2][3 4] ; 1/2-3/4=-1/4 ;[-1 4] ;-)
```

Arithmetic / Rational numbers / `rmul`

`rmul ratio1 ratio2` (library procedure)

outputs a rational number as a list of two numbers, nominator and denominator, which is the product of the two input rationals.

Example:

```
rmul [1 2][3 4] ; (1/2)*(3/4)=3/8 ; [3 8] ; -)
```

`rdiv ratio1 ratio2` (library procedure)

outputs a rational number as a list of two numbers, nominator and denominator, which is the quotient of the two input rationals.

Example:

```
rdiv [1 2][3 4] ; (1/2)/(3/4)=2/3 ; [2 3] ; -)
```

Print formatting

...is sometimes a convenient way to display numbers.

Print formatting

- Form 170
- intForm 170
- setPrintPrecision 171
- hex 171

Form *num width precision*

outputs a word containing a printable representation of "*num*", possibly preceded by spaces (and therefore not a number for purposes of performing arithmetic operations), with at least "*width*" characters, including exactly "*precision*" digits after the decimal point. (If "*precision*" is 0 then there will be no decimal point in the output.)

As a debugging feature, (FORM *num* -1 format) will print the floating point "*num*" according to the C printf "format", to allow

```
to hex : num      op form : num  -1 " |%08X %08X|
end
```

to allow finding out the exact result of floating point operations. The precise format needed may be machine-dependent.

Examples:

```
Form pi 16 14      ;3.14159265358979  ;-)
Form exp 1 10 8     ;2.71828183      ;-)
```

intForm *num width base*

Arithmetic / Print formatting / intForm

outputs a word containing ciphers that represent the number in the *base base* . I.e. if *base ==10* then the number will be output decimal, if *base ==2* the number will be output binary, if *base ==16* the output will be hexadecimal, etc.

Negative integers will be converted to unsigned int, so -1 in *base ==16* will be FFFFFFFF.

Examples:

```
intForm 1234 0 10      ;1234  ;-)
intForm 1234 0 16      ;4D2   ;-)
intForm 1234 0 2        ;10011010010  ;-)
intForm -1 0 16         ;FFFFFFF  ;-)
intForm -2 0 16         ;FFFFFFFE  ;-)
```

setPrintPrecision *precision*

command to set the *precision* of printing float and complex numbers.

Example:

```
setPrintPrecision 15
pi-(exp 1)*1i
3.14159265358979-2.71828182845905i  ;-)
```

hex *number* (library procedure)

outputs an integer as a hexadecimal number. This is good for fast checking of color values.

Examples:

```
hex 16      ;10  ;-)
hex 256     ;100  ;-)
hex 255     ;FF   ;-)
hex PC      ;FF000000  ;-) (or whatever value PC has)
```

Bitwise Operations

...are supplied to make bitwise computations like in C.

Bitwise Operations

- BitAnd 172
- BitOr 172
- BitXOr 173
- BitNot 173
- aShift 173
- lShift 174

BitAnd *num1 num2*
(BitAnd *num1 num2 num3 ...*)

outputs the bitwise AND of its inputs, which must be integers or a list or array of integers.

Examples:

```
BitAnd 1 2      ;0  ;-)
BitAnd 3 2      ;2  ;-)
BitAnd [1 2 3][1 1 1] ;[1 0 1] ;-)
```

BitOr *num1 num2*
(BitOr *num1 num2 num3 ...*)

outputs the bitwise OR of its inputs, which must be integers or a list or array of integers.

Examples:

Arithmetic / Bitwise Operations / BitOr

```
BitOr 1 2      ;3   ;-)
BitOr 3 2      ;3   ;-)
BitOr [1 2 3][0 0 0] ;[1 2 3] ;-)
```

BitXOr *num1 num2***(BitXOr *num1 num2 num3 ...*)**

outputs the bitwise EXCLUSIVE OR of its inputs, which must be integers or a list or array of integers.

Examples:

```
BitXOr 1 1      ;0   ;-)
BitXOr 1 0      ;1   ;-)
BitXOr [1 2 3][1 1 1] ;[0 3 2] ;-)
```

BitNot *num*

outputs the bitwise NOT of its input *num* , which must be an integer, or a list or array of integers.

Examples:

```
BitNot 1          ; -2   ;-)
BitNot 0          ; -1   ;-)
BitNot [1 2 3]    ; [-2 -3 -4] ;-)
BitNot {1 2 3}    ; {-2 -3 -4} ;-)
```

aShift *num1 num2*

outputs "*num1* " arithmetic-shifted to the left by "*num2* " bits. If *num2* is negative, the shift is to the right with sign extension.

Examples:

```
aShift 1 1      ;2  ;-)
aShift 2 -1     ;1  ;-)
aShift -2 -1    ;-1 ;-)
aShift [1 2 3][1 2 3] ;[2 8 24] ;-)
```

lShift *num1 num2*

has a bug: right shift with zero fill does not work yet!

outputs "*num1*" logical-shifted to the left by "*num2*" bits. If *num2* is negative, the shift is to the right with zero fill. The inputs must be integers.

Examples:

```
lShift 1 1      ;2  ;-)
lShift 2 -1     ;1  ;-)
lShift [1 2 3][1 2 3] ;[2 8 24] ;-)
```

Logical Operations

Logical Operations

- and 175
- or 175
- and2 176
- or2 176
- not 177

and *tf1 tf2*
(and *tf1 tf2 tf3 ...*)

outputs TRUE if all inputs are TRUE, otherwise FALSE. All inputs must be TRUE or FALSE. (Comparison is case-insensitive regardless of the value of CASEIGNOREDP. That is, "true" or "True" or "TRUE" are all the same.)

Examples:

```
and true false      ;false  ;-)
and true true       ;true   ;-)
and false true      ;false  ;-)
and false false     ;false  ;-)
(and true true true)      ;true   ;-)
(and true true false)    ;false  ;-)
```

or *tf1 tf2*
(or *tf1 tf2 tf3 ...*)

outputs TRUE if any input is TRUE, otherwise FALSE. All inputs must be TRUE or FALSE. (Comparison is case-insensitive regardless of the value of CASEIGNOREDP. That is, "true" or

"True" or "TRUE" are all the same.)

Examples:

```
or false false      ;false  ;-)
or false true       ;true   ;-)
or true false       ;true   ;-)
or true true        ;true   ;-)
(or false false true) ;true   ;-)
```

tf1 and2 tf2

outputs true if both boolean inputs are true, otherwise false.

Note: This is an infix operator with the same functionality as and.

Examples:

```
false and2 false      ;false  ;-)
false and2 true       ;false   ;-)
true and2 false       ;false   ;-)
true and2 true        ;true    ;-)
```

tf1 or2 tf2

outputs true if one of the boolean inputs is true, otherwise false.

Note: This is an infix operator with the same functionality as or.

Examples:

Logical Operations / or2

```
false or2 false    ;false  ;-)
false or2 true     ;true   ;-)
true or2 false     ;true   ;-)
true or2 true      ;true   ;-)
show 1==0 or2 1<0   ;false
show 1==0 or2 1>0   ;true
```

not *tf*

outputs TRUE if the input is FALSE, and vice versa.

Examples:

```
not false    ;true  ;-)
not true     ;false ;-)
```

Graphics

aUCBLogo provides traditional Logo turtle graphics with one or more turtles. Collision detection is not supported. This is the most hardware-dependent part of Logo; some features may exist on some machines but not others. Nevertheless, the goal has been to make Logo programs as portable as possible, rather than to take fullest advantage of the capabilities of each machine. In particular, Logo attempts to scale the screen so that turtle coordinates $[-400 -300]$ and $[400 300]$ fit on the graphics window, and so that the aspect ratio is 1:1, although some PC screens have nonstandard aspect ratios.

The center of the graphics window (which may or may not be the entire screen, depending on the machine used) is turtle location $[0 0]$. Positive X is to the right; positive Y is up. Headings (angles) are measured in degrees clockwise from the positive Y axis. (This differs from the common mathematical convention of measuring angles counterclockwise from the positive X axis.) The turtle is represented as an isocles triangle; the actual turtle position is at the midpoint of the base (the short side).

Colors are, of course, hardware-dependent. However, Logo provides partial hardware independence by interpreting color numbers 0 through 7 uniformly on all computers:

0	black	1	blue	2	green	3	cyan
4	red	5	magenta	6	yellow	7	white

Where possible, Logo provides additional user-settable colors; how many are available depends on the hardware and operating system environment. If at least 16 colors are available, Logo tries to provide uniform initial settings for the colors 8-15:

8	brown	9	tan	10	forest	11	aqua
12	salmon	13	purple	14	orange	15	grey

Logo begins with a white background and black pen.

Graphics

- Color database 179
- Relative Turtle Motion 195

Graphics

- Absolute Turtle Motion 200
 - Turtle Motion Queries 212
 - Turtle and Window Control 218
 - Turtle and Window Queries 234
 - Multiple Turtles 237
 - Pen and Background Control 238
 - enable and disable flags 245
 - Pen Queries 251
 - Drawing Curves 254
 - Drawing filled shapes 258
 - Lighting 270
 - Fog 272
 - Pictures 275
 - Bitmaps 281
 - Direct Graphics 286
 - Projection Matrix 294
 - Texturing 298
 - Shadows 302
-

Color database

Colors can also set by name, but the color database of wxWidgets is limited to the following list (RGB8 is like RGB r/255 g/255 b/255), but be aware, there is a second list of even more colors after the end of the first colors list. The color values of the first color of the list with a given name will be taken. You can fetch the complete list of available colors using the primitive `getColorDatabase`.

```
"|AQUAMARINE| RGB8 112 219 147
"|BLACK| RGB8 0 0 0
"|BLUE| RGB8 0 0 255
"|BLUE VIOLET| RGB8 159 95 159
"|BROWN| RGB8 165 42 42
"|CADET BLUE| RGB8 95 159 159
"|CORAL| RGB8 255 127 0
"|CORNFLOWER BLUE| RGB8 66 66 111
"|CYAN| RGB8 0 255 255
"|DARK GREY| RGB8 47 47 47
"|DARK GREEN| RGB8 47 79 47
"|DARK OLIVE GREEN| RGB8 79 79 47
"|DARK ORCHID| RGB8 153 50 204
"|DARK SLATE BLUE| RGB8 107 35 142
"|DARK SLATE GREY| RGB8 47 79 79
"|DARK TURQUOISE| RGB8 112 147 219
"|DIM GREY| RGB8 84 84 84
"|FIREBRICK| RGB8 142 35 35
"|FOREST GREEN| RGB8 35 142 35
"|GOLD| RGB8 204 127 50
"|GOLDENROD| RGB8 219 219 112
"|GREY| RGB8 128 128 128
"|GREEN| RGB8 0 255 0
"|GREEN YELLOW| RGB8 147 219 112
"|INDIAN RED| RGB8 79 47 47
"|KHAKI| RGB8 159 159 95
"|LIGHT BLUE| RGB8 191 216 216
"|LIGHT GREY| RGB8 192 192 192
"|LIGHT STEEL BLUE| RGB8 143 143 188
"|LIME GREEN| RGB8 50 204 50
"|LIGHT MAGENTA| RGB8 255 0 255
"|MAGENTA| RGB8 255 0 255
"|MAROON| RGB8 142 35 107
```


Graphics / Color database

```

"|MEDIUM AQUAMARINE| RGB8  50 204 153
"|MEDIUM GREY| RGB8  100 100 100
"|MEDIUM BLUE| RGB8  50 50 204
"|MEDIUM FOREST GREEN| RGB8  107 142 35
"|MEDIUM GOLDENROD| RGB8  234 234 173
"|MEDIUM ORCHID| RGB8  147 112 219
"|MEDIUM SEA GREEN| RGB8  66 111 66
"|MEDIUM SLATE BLUE| RGB8  127 0 255
"|MEDIUM SPRING GREEN| RGB8  127 255 0
"|MEDIUM TURQUOISE| RGB8  112 219 219
"|MEDIUM VIOLET RED| RGB8  219 112 147
"|MIDNIGHT BLUE| RGB8  47 47 79
"|NAVY| RGB8  35 35 142
"|ORANGE| RGB8  204 50 50
"|ORANGE RED| RGB8  255 0 127
"|ORCHID| RGB8  219 112 219
"|PALE GREEN| RGB8  143 188 143
"|PINK| RGB8  188 143 234
"|PLUM| RGB8  234 173 234
"|PURPLE| RGB8  176 0 255
"|RED| RGB8  255 0 0
"|SALMON| RGB8  111 66 66
"|SEA GREEN| RGB8  35 142 107
"|SIENNA| RGB8  142 107 35
"|SKY BLUE| RGB8  50 153 204
"|SLATE BLUE| RGB8  0 127 255
"|SPRING GREEN| RGB8  0 255 127
"|STEEL BLUE| RGB8  35 107 142
"|TAN| RGB8  219 147 112
"|THISTLE| RGB8  216 191 216
"|TURQUOISE| RGB8  173 234 234
"|VIOLET| RGB8  79 47 79
"|VIOLET RED| RGB8  204 50 153
"|WHEAT| RGB8  216 216 191
"|WHITE| RGB8  255 255 255
"|YELLOW| RGB8  255 255 0
"|YELLOW GREEN| RGB8  153 204 50

```

Here's a long list of additional colors, which I inserted as `farben.h` into the `wxWidgets` library file `gdicmn.cpp`, so they are now all available in `aUCBLogo`.

```
"|ALICEBLUE| RGB8 240 248 255
"|ANTIQUWHITE| RGB8 250 235 215
"|ANTIQUWHITE1| RGB8 255 239 219
"|ANTIQUWHITE2| RGB8 238 223 204
"|ANTIQUWHITE3| RGB8 205 192 176
"|ANTIQUWHITE4| RGB8 139 131 120
"|AQUAMARINE| RGB8 127 255 212
"|AQUAMARINE1| RGB8 127 255 212
"|AQUAMARINE2| RGB8 118 238 198
"|AQUAMARINE3| RGB8 102 205 170
"|AQUAMARINE4| RGB8 69 139 116
"|AZURE| RGB8 240 255 255
"|AZURE1| RGB8 240 255 255
"|AZURE2| RGB8 224 238 238
"|AZURE3| RGB8 193 205 205
"|AZURE4| RGB8 131 139 139
"|BEIGE| RGB8 245 245 220
"|BISQUE| RGB8 255 228 196
"|BISQUE1| RGB8 255 228 196
"|BISQUE2| RGB8 238 213 183
"|BISQUE3| RGB8 205 183 158
"|BISQUE4| RGB8 139 125 107
"|BLACK| RGB8 0 0 0
"|BLANCHEDALMOND| RGB8 255 235 205
"|BLUE| RGB8 0 0 255
"|BLUE1| RGB8 0 0 255
"|BLUE2| RGB8 0 0 238
"|BLUE3| RGB8 0 0 205
"|BLUE4| RGB8 0 0 139
"|BLUEVIOLET| RGB8 138 43 226
"|BROWN| RGB8 165 42 42
"|BROWN1| RGB8 255 64 64
"|BROWN2| RGB8 238 59 59
"|BROWN3| RGB8 205 51 51
"|BROWN4| RGB8 139 35 35
```

Graphics / Color database

```

" |BURLYWOOD| RGB8 222 184 135
" |BURLYWOOD1| RGB8 255 211 155
" |BURLYWOOD2| RGB8 238 197 145
" |BURLYWOOD3| RGB8 205 170 125
" |BURLYWOOD4| RGB8 139 115 85
" |CADETBBLUE| RGB8 95 158 160
" |CADETBBLUE1| RGB8 152 245 255
" |CADETBBLUE2| RGB8 142 229 238
" |CADETBBLUE3| RGB8 122 197 205
" |CADETBBLUE4| RGB8 83 134 139
" |CHARTREUSE| RGB8 127 255 0
" |CHARTREUSE1| RGB8 127 255 0
" |CHARTREUSE2| RGB8 118 238 0
" |CHARTREUSE3| RGB8 102 205 0
" |CHARTREUSE4| RGB8 69 139 0
" |CHOCOLATE| RGB8 210 105 30
" |CHOCOLATE1| RGB8 255 127 36
" |CHOCOLATE2| RGB8 238 118 33
" |CHOCOLATE3| RGB8 205 102 29
" |CHOCOLATE4| RGB8 139 69 19
" |CORAL| RGB8 255 127 80
" |CORAL1| RGB8 255 114 86
" |CORAL2| RGB8 238 106 80
" |CORAL3| RGB8 205 91 69
" |CORAL4| RGB8 139 62 47
" |CORNFLOWERBLUE| RGB8 100 149 237
" |CORNSILK| RGB8 255 248 220
" |CORNSILK1| RGB8 255 248 220
" |CORNSILK2| RGB8 238 232 205
" |CORNSILK3| RGB8 205 200 177
" |CORNSILK4| RGB8 139 136 120
" |CYAN| RGB8 0 255 255
" |CYAN1| RGB8 0 255 255
" |CYAN2| RGB8 0 238 238
" |CYAN3| RGB8 0 205 205
" |CYAN4| RGB8 0 139 139

```

```

" | DARKBLUE | RGB8 0 0 139
" | DARKCYAN | RGB8 0 139 139
" | DARKGOLDENROD | RGB8 184 134 11
" | DARKGOLDENROD1 | RGB8 255 185 15
" | DARKGOLDENROD2 | RGB8 238 173 14
" | DARKGOLDENROD3 | RGB8 205 149 12
" | DARKGOLDENROD4 | RGB8 139 103 11
" | DARKGREEN | RGB8 0 100 0
" | DARKGREY | RGB8 169 169 169
" | DARKKHAKI | RGB8 189 183 107
" | DARKMAGENTA | RGB8 139 0 139
" | DARKOLIVEGREEN | RGB8 85 107 47
" | DARKOLIVEGREEN1 | RGB8 202 255 112
" | DARKOLIVEGREEN2 | RGB8 188 238 104
" | DARKOLIVEGREEN3 | RGB8 162 205 90
" | DARKOLIVEGREEN4 | RGB8 110 139 61
" | DARKORANGE | RGB8 255 140 0
" | DARKORANGE1 | RGB8 255 127 0
" | DARKORANGE2 | RGB8 238 118 0
" | DARKORANGE3 | RGB8 205 102 0
" | DARKORANGE4 | RGB8 139 69 0
" | DARKORCHID | RGB8 153 50 204
" | DARKORCHID1 | RGB8 191 62 255
" | DARKORCHID2 | RGB8 178 58 238
" | DARKORCHID3 | RGB8 154 50 205
" | DARKORCHID4 | RGB8 104 34 139
" | DARKRED | RGB8 139 0 0
" | DARKSALMON | RGB8 233 150 122
" | DARKSEAGREEN | RGB8 143 188 143
" | DARKSEAGREEN1 | RGB8 193 255 193
" | DARKSEAGREEN2 | RGB8 180 238 180
" | DARKSEAGREEN3 | RGB8 155 205 155
" | DARKSEAGREEN4 | RGB8 105 139 105
" | DARKSLATEBLUE | RGB8 72 61 139
" | DARKSLATEGRAY | RGB8 47 79 79
" | DARKSLATEGRAY1 | RGB8 151 255 255
" | DARKSLATEGRAY2 | RGB8 141 238 238
" | DARKSLATEGRAY3 | RGB8 121 205 205
" | DARKSLATEGRAY4 | RGB8 82 139 139
" | DARKTURQUOISE | RGB8 0 206 209

```

Graphics / Color database

```
"|DARKVIOLET| RGB8 148 0 211
"|DEEPPINK| RGB8 255 20 147
"|DEEPPINK1| RGB8 255 20 147
"|DEEPPINK2| RGB8 238 18 137
"|DEEPPINK3| RGB8 205 16 118
"|DEEPPINK4| RGB8 139 10 80
"|DEEPSKYBLUE| RGB8 0 191 255
"|DEEPSKYBLUE1| RGB8 0 191 255
"|DEEPSKYBLUE2| RGB8 0 178 238
"|DEEPSKYBLUE3| RGB8 0 154 205
"|DEEPSKYBLUE4| RGB8 0 104 139
"|DIMGREY| RGB8 105 105 105
"|DODGERBLUE| RGB8 30 144 255
"|DODGERBLUE1| RGB8 30 144 255
"|DODGERBLUE2| RGB8 28 134 238
"|DODGERBLUE3| RGB8 24 116 205
"|DODGERBLUE4| RGB8 16 78 139
"|FIREBRICK| RGB8 178 34 34
"|FIREBRICK1| RGB8 255 48 48
"|FIREBRICK2| RGB8 238 44 44
"|FIREBRICK3| RGB8 205 38 38
"|FIREBRICK4| RGB8 139 26 26
"|FLORALWHITE| RGB8 255 250 240
"|FORESTGREEN| RGB8 34 139 34
"|GAINSBORO| RGB8 220 220 220
"|GHOSTWHITE| RGB8 248 248 255
"|GOLD| RGB8 255 215 0
"|GOLD1| RGB8 255 215 0
"|GOLD2| RGB8 238 201 0
"|GOLD3| RGB8 205 173 0
"|GOLD4| RGB8 139 117 0
"|GOLDENROD| RGB8 218 165 32
"|GOLDENROD1| RGB8 255 193 37
"|GOLDENROD2| RGB8 238 180 34
"|GOLDENROD3| RGB8 205 155 29
"|GOLDENROD4| RGB8 139 105 20
"|GRAY81| RGB8 207 207 207
"|GRAY91| RGB8 232 232 232
```

```
"|GREEN| RGB8 0 255 0
"|GREEN1| RGB8 0 255 0
"|GREEN2| RGB8 0 238 0
"|GREEN3| RGB8 0 205 0
"|GREEN4| RGB8 0 139 0
"|GREENYELLOW| RGB8 173 255 47
"|GREY| RGB8 190 190 190
"|GREY11| RGB8 28 28 28
"|GREY21| RGB8 54 54 54
"|GREY31| RGB8 79 79 79
"|GREY41| RGB8 105 105 105
"|GREY51| RGB8 130 130 130
"|GREY61| RGB8 156 156 156
"|GREY71| RGB8 181 181 181
"|HONEYDEW| RGB8 240 255 240
"|HONEYDEW1| RGB8 240 255 240
"|HONEYDEW2| RGB8 224 238 224
"|HONEYDEW3| RGB8 193 205 193
"|HONEYDEW4| RGB8 131 139 131
"|HOTPINK| RGB8 255 105 180
"|HOTPINK1| RGB8 255 110 180
"|HOTPINK2| RGB8 238 106 167
"|HOTPINK3| RGB8 205 96 144
"|HOTPINK4| RGB8 139 58 98
"|INDIANRED| RGB8 205 92 92
"|INDIANRED1| RGB8 255 106 106
"|INDIANRED2| RGB8 238 99 99
"|INDIANRED3| RGB8 205 85 85
"|INDIANRED4| RGB8 139 58 58
"|IVORY| RGB8 255 255 240
"|IVORY1| RGB8 255 255 240
"|IVORY2| RGB8 238 238 224
"|IVORY3| RGB8 205 205 193
"|IVORY4| RGB8 139 139 131
```

Graphics / Color database

```
"|KHAKI1| RGB8 255 246 143
"|KHAKI2| RGB8 238 230 133
"|KHAKI3| RGB8 205 198 115
"|KHAKI4| RGB8 139 134 78
"|LAVENDER| RGB8 230 230 250
"|LAVENDERBLUSH| RGB8 255 240 245
"|LAVENDERBLUSH1| RGB8 255 240 245
"|LAVENDERBLUSH2| RGB8 238 224 229
"|LAVENDERBLUSH3| RGB8 205 193 197
"|LAVENDERBLUSH4| RGB8 139 131 134
"|LAWNGREEN| RGB8 124 252 0
"|LEMONCHIFFON| RGB8 255 250 205
"|LEMONCHIFFON1| RGB8 255 250 205
"|LEMONCHIFFON2| RGB8 238 233 191
"|LEMONCHIFFON3| RGB8 205 201 165
"|LEMONCHIFFON4| RGB8 139 137 112
"|LIGHTBLUE| RGB8 173 216 230
"|LIGHTBLUE1| RGB8 191 239 255
"|LIGHTBLUE2| RGB8 178 223 238
"|LIGHTBLUE3| RGB8 154 192 205
"|LIGHTBLUE4| RGB8 104 131 139
"|LIGHTCORAL| RGB8 240 128 128
"|LIGHTCYAN| RGB8 224 255 255
"|LIGHTCYAN1| RGB8 224 255 255
"|LIGHTCYAN2| RGB8 209 238 238
"|LIGHTCYAN3| RGB8 180 205 205
"|LIGHTCYAN4| RGB8 122 139 139
"|LIGHTGOLDENROD| RGB8 238 221 130
"|LIGHTGOLDENROD1| RGB8 255 236 139
"|LIGHTGOLDENROD2| RGB8 238 220 130
"|LIGHTGOLDENROD3| RGB8 205 190 112
"|LIGHTGOLDENROD4| RGB8 139 129 76
```

```
"|LIGHTGRAY| RGB8 211 211 211
"|LIGHTGREEN| RGB8 144 238 144
"|LIGHTPINK| RGB8 255 182 193
"|LIGHTPINK1| RGB8 255 174 185
"|LIGHTPINK2| RGB8 238 162 173
"|LIGHTPINK3| RGB8 205 140 149
"|LIGHTPINK4| RGB8 139 95 101
"|LIGHTSALMON| RGB8 255 160 122
"|LIGHTSALMON1| RGB8 255 160 122
"|LIGHTSALMON2| RGB8 238 149 114
"|LIGHTSALMON3| RGB8 205 129 98
"|LIGHTSALMON4| RGB8 139 87 66
"|LIGHTSEAGREEN| RGB8 32 178 170
"|LIGHTSKYBLUE| RGB8 135 206 250
"|LIGHTSKYBLUE1| RGB8 176 226 255
"|LIGHTSKYBLUE2| RGB8 164 211 238
"|LIGHTSKYBLUE3| RGB8 141 182 205
"|LIGHTSKYBLUE4| RGB8 96 123 139
"|LIGHTSLATEBLUE| RGB8 132 112 255
"|LIGHTSLATEGRAY| RGB8 119 136 153
"|LIGHTSTEELBLUE| RGB8 176 196 222
"|LIGHTSTEELBLUE1| RGB8 202 225 255
"|LIGHTSTEELBLUE2| RGB8 188 210 238
"|LIGHTSTEELBLUE3| RGB8 162 181 205
"|LIGHTSTEELBLUE4| RGB8 110 123 139
"|LIGHTYELLOW| RGB8 255 255 224
"|LIGHTYELLOW1| RGB8 255 255 224
"|LIGHTYELLOW2| RGB8 238 238 209
"|LIGHTYELLOW3| RGB8 205 205 180
"|LIGHTYELLOW4| RGB8 139 139 122
"|LIMEGREEN| RGB8 50 205 50
"|LINEN| RGB8 250 240 230
"|LTGOLDENRODYELLO| RGB8 250 250 210
```


Graphics / Color database

```

"|MAGENTA| RGB8 255 0 255
"|MAGENTA1| RGB8 255 0 255
"|MAGENTA2| RGB8 238 0 238
"|MAGENTA3| RGB8 205 0 205
"|MAGENTA4| RGB8 139 0 139
"|MAROON| RGB8 176 48 96
"|MAROON1| RGB8 255 52 179
"|MAROON2| RGB8 238 48 167
"|MAROON3| RGB8 205 41 144
"|MAROON4| RGB8 139 28 98
"|MEDIUMAQUAMARINE| RGB8 102 205 170
"|MEDIUMBLUE| RGB8 0 0 205
"|MEDIUMORCHID| RGB8 186 85 211
"|MEDIUMORCHID1| RGB8 224 102 255
"|MEDIUMORCHID2| RGB8 209 95 238
"|MEDIUMORCHID3| RGB8 180 82 205
"|MEDIUMORCHID4| RGB8 122 55 139
"|MEDIUMPURPLE| RGB8 147 112 219
"|MEDIUMPURPLE1| RGB8 171 130 255
"|MEDIUMPURPLE2| RGB8 159 121 238
"|MEDIUMPURPLE3| RGB8 137 104 205
"|MEDIUMPURPLE4| RGB8 93 71 139
"|MEDIUMSEAGREEN| RGB8 60 179 113
"|MEDIUMSLATEBLUE| RGB8 123 104 238
"|MEDIUMTURQUOISE| RGB8 72 209 204
"|MEDIUMVIOLETRED| RGB8 199 21 133
"|MEDSPRINGGREEN| RGB8 0 250 154
"|MIDNIGHTBLUE| RGB8 25 25 112
"|MINTCREAM| RGB8 245 255 250
"|MISTYROSE| RGB8 255 228 225
"|MISTYROSE1| RGB8 255 228 225
"|MISTYROSE2| RGB8 238 213 210
"|MISTYROSE3| RGB8 205 183 181
"|MISTYROSE4| RGB8 139 125 123
"|MOCCASIN| RGB8 255 228 181

```

```
"|NAVAJOWHITE| RGB8 255 222 173
"|NAVAJOWHITE1| RGB8 255 222 173
"|NAVAJOWHITE2| RGB8 238 207 161
"|NAVAJOWHITE3| RGB8 205 179 139
"|NAVAJOWHITE4| RGB8 139 121 94
"|NAVYBLUE| RGB8 0 0 128
"|OLDLACE| RGB8 253 245 230
"|OLIVEDRAB| RGB8 107 142 35
"|OLIVEDRAB1| RGB8 192 255 62
"|OLIVEDRAB2| RGB8 179 238 58
"|OLIVEDRAB3| RGB8 154 205 50
"|OLIVEDRAB4| RGB8 105 139 34
"|ORANGE| RGB8 255 165 0
"|ORANGE1| RGB8 255 165 0
"|ORANGE2| RGB8 238 154 0
"|ORANGE3| RGB8 205 133 0
"|ORANGE4| RGB8 139 90 0
"|ORANGERED| RGB8 255 69 0
"|ORANGERED1| RGB8 255 69 0
"|ORANGERED2| RGB8 238 64 0
"|ORANGERED3| RGB8 205 55 0
"|ORANGERED4| RGB8 139 37 0
"|ORCHID| RGB8 218 112 214
"|ORCHID1| RGB8 255 131 250
"|ORCHID2| RGB8 238 122 233
"|ORCHID3| RGB8 205 105 201
"|ORCHID4| RGB8 139 71 137
"|PALEGOLDENROD| RGB8 238 232 170
"|PALEGREEN| RGB8 152 251 152
"|PALEGREEN1| RGB8 154 255 154
"|PALEGREEN2| RGB8 144 238 144
"|PALEGREEN3| RGB8 124 205 124
"|PALEGREEN4| RGB8 84 139 84
```

Graphics / Color database

```

"| PALETURQUOISE| RGB8 175 238 238
"| PALETURQUOISE1| RGB8 187 255 255
"| PALETURQUOISE2| RGB8 174 238 238
"| PALETURQUOISE3| RGB8 150 205 205
"| PALETURQUOISE4| RGB8 102 139 139
"| PALEVIOLETRED| RGB8 219 112 147
"| PALEVIOLETRED1| RGB8 255 130 171
"| PALEVIOLETRED2| RGB8 238 121 159
"| PALEVIOLETRED3| RGB8 205 104 137
"| PALEVIOLETRED4| RGB8 139 71 93
"| PAPAYAWHIP| RGB8 255 239 213
"| PEACHPUFF| RGB8 255 218 185
"| PEACHPUFF1| RGB8 255 218 185
"| PEACHPUFF2| RGB8 238 203 173
"| PEACHPUFF3| RGB8 205 175 149
"| PEACHPUFF4| RGB8 139 119 101
"| PERU| RGB8 205 133 63
"| PINK| RGB8 255 192 203
"| PINK1| RGB8 255 181 197
"| PINK2| RGB8 238 169 184
"| PINK3| RGB8 205 145 158
"| PINK4| RGB8 139 99 108
"| PLATINUM| RGB8 085 088 090
"| PLUM| RGB8 221 160 221
"| PLUM1| RGB8 255 187 255
"| PLUM2| RGB8 238 174 238
"| PLUM3| RGB8 205 150 205
"| PLUM4| RGB8 139 102 139
"| POWDERBLUE| RGB8 176 224 230
"| PURPLE| RGB8 160 32 240
"| PURPLE1| RGB8 155 48 255
"| PURPLE2| RGB8 145 44 238
"| PURPLE3| RGB8 125 38 205
"| PURPLE4| RGB8 85 26 139
"| RED| RGB8 255 0 0
"| RED1| RGB8 255 0 0
"| RED2| RGB8 238 0 0
"| RED3| RGB8 205 0 0
"| RED4| RGB8 139 0 0

```

```

" | ROSYBROWN | RGB8 188 143 143
" | ROSYBROWN1 | RGB8 255 193 193
" | ROSYBROWN2 | RGB8 238 180 180
" | ROSYBROWN3 | RGB8 205 155 155
" | ROSYBROWN4 | RGB8 139 105 105
" | ROYALBLUE | RGB8 65 105 225
" | ROYALBLUE1 | RGB8 72 118 255
" | ROYALBLUE2 | RGB8 67 110 238
" | ROYALBLUE3 | RGB8 58 95 205
" | ROYALBLUE4 | RGB8 39 64 139
" | SADDLEBROWN | RGB8 139 69 19
" | SALMON | RGB8 250 128 114
" | SALMON1 | RGB8 255 140 105
" | SALMON2 | RGB8 238 130 98
" | SALMON3 | RGB8 205 112 84
" | SALMON4 | RGB8 139 76 57
" | SANDYBROWN | RGB8 244 164 96
" | SEAGREEN | RGB8 46 139 87
" | SEAGREEN1 | RGB8 84 255 159
" | SEAGREEN2 | RGB8 78 238 148
" | SEAGREEN3 | RGB8 67 205 128
" | SEAGREEN4 | RGB8 46 139 87
" | SEASHELL | RGB8 255 245 238
" | SEASHELL1 | RGB8 255 245 238
" | SEASHELL2 | RGB8 238 229 222
" | SEASHELL3 | RGB8 205 197 191
" | SEASHELL4 | RGB8 139 134 130
" | SIENNA | RGB8 160 82 45
" | SIENNA1 | RGB8 255 130 71
" | SIENNA2 | RGB8 238 121 66
" | SIENNA3 | RGB8 205 104 57
" | SIENNA4 | RGB8 139 71 38
" | SKYBLUE | RGB8 135 206 235
" | SKYBLUE1 | RGB8 135 206 255
" | SKYBLUE2 | RGB8 126 192 238
" | SKYBLUE3 | RGB8 108 166 205
" | SKYBLUE4 | RGB8 74 112 139

```

Graphics / Color database

```

"| SLATEBLUE| RGB8 106 90 205
"| SLATEBLUE1| RGB8 131 111 255
"| SLATEBLUE2| RGB8 122 103 238
"| SLATEBLUE3| RGB8 105 89 205
"| SLATEBLUE4| RGB8 71 60 139
"| SLATEGRAY1| RGB8 198 226 255
"| SLATEGRAY2| RGB8 185 211 238
"| SLATEGRAY3| RGB8 159 182 205
"| SLATEGRAY4| RGB8 108 123 139
"| SLATEGREY| RGB8 112 128 144
"| SNOW| RGB8 255 250 250
"| SNOW1| RGB8 255 250 250
"| SNOW2| RGB8 238 233 233
"| SNOW3| RGB8 205 201 201
"| SNOW4| RGB8 139 137 137
"| SPRINGGREEN| RGB8 0 255 127
"| SPRINGGREEN1| RGB8 0 255 127
"| SPRINGGREEN2| RGB8 0 238 118
"| SPRINGGREEN3| RGB8 0 205 102
"| SPRINGGREEN4| RGB8 0 139 69
"| STEELBLUE| RGB8 70 130 180
"| STEELBLUE1| RGB8 99 184 255
"| STEELBLUE2| RGB8 92 172 238
"| STEELBLUE3| RGB8 79 148 205
"| STEELBLUE4| RGB8 54 100 139
"| TAN| RGB8 210 180 140
"| TAN1| RGB8 255 165 79
"| TAN2| RGB8 238 154 73
"| TAN3| RGB8 205 133 63
"| TAN4| RGB8 139 90 43
"| THISTLE| RGB8 216 191 216
"| THISTLE1| RGB8 255 225 255
"| THISTLE2| RGB8 238 210 238
"| THISTLE3| RGB8 205 181 205
"| THISTLE4| RGB8 139 123 139

```

```
"|TOMATO| RGB8 255 99 71
"|TOMATO1| RGB8 255 99 71
"|TOMATO2| RGB8 238 92 66
"|TOMATO3| RGB8 205 79 57
"|TOMATO4| RGB8 139 54 38
"|TURQUOISE| RGB8 64 224 208
"|TURQUOISE1| RGB8 0 245 255
"|TURQUOISE2| RGB8 0 229 238
"|TURQUOISE3| RGB8 0 197 205
"|TURQUOISE4| RGB8 0 134 139
"|VIOLET| RGB8 238 130 238
"|VIOLETRED| RGB8 208 32 144
"|VIOLETRED1| RGB8 255 62 150
"|VIOLETRED2| RGB8 238 58 140
"|VIOLETRED3| RGB8 205 50 120
"|VIOLETRED4| RGB8 139 34 82
"|WHEAT| RGB8 245 222 179
"|WHEAT1| RGB8 255 231 186
"|WHEAT2| RGB8 238 216 174
"|WHEAT3| RGB8 205 186 150
"|WHEAT4| RGB8 139 126 102
"|WHITE| RGB8 255 255 255
"|WHITESMOKE| RGB8 245 245 245
"|YELLOW| RGB8 255 255 0
"|YELLOW1| RGB8 255 255 0
"|YELLOW2| RGB8 238 238 0
"|YELLOW3| RGB8 205 205 0
"|YELLOW4| RGB8 139 139 0
```

Relative Turtle Motion

This section describes the basic set of commands to move the turtle relative to its position, including commands to change the turtle's course.

When the pen is set to down (with PenDown), then the turtle will draw a straight line along its path.

Relative movements are the strength of turtle graphics, especially when you repeat complete figures and iteratively call them, so you can for example create beautiful fractal drawings with ease.

Relative Turtle Motion

- forward 195, fd 195
- back 196, bk 196
- left 196, lt 196
- right 197, rt 197
- leftRoll 197, lr 197
- rightRoll 198, rr 198
- upPitch 198, uP 198
- downPitch 198, down 198

forward *dist*

fd *dist*

moves the turtle forward, in the direction that it's facing, by the specified distance (measured in turtle steps).

If penDown?==true then the turtle draws a line in her path. If cylinder lines are enabled then she draws a cylinder in her path.

Examples:

```
repeat 4 [forward 100 right 90]
```

```
cs
fd 100
rt 90
fd 100
lt 60
fd 50
```

```
cs
perspective
enableCylinderLines
setpenSize [50 50]
setPenColor 4
fd 100
rt 90
fd 100
```

back *dist***bk *dist***

moves the turtle backward, i.e., exactly opposite to the direction that it's facing, by the specified distance. (The heading of the turtle does not change.)

Example:

```
pu fd 100 pd
bk 200
```

left *degrees***lt *degrees***

turns the turtle counterclockwise by the specified angle, measured in *degrees* (1/360 of a circle).

Examples:

Graphics / Relative Turtle Motion / left

```
lt 90
fd 100
lt 180
fd 100
lt 45
fd 100
lt 135
fd 100
lt 45
fd 100
```

right *degrees*

rt *degrees*

turns the turtle clockwise by the specified angle, measured in *degrees* (1/360 of a circle).

Examples:

```
rt 90
fd 100
rt 180
fd 100
rt 45
fd 100
rt 135
fd 100
rt 45
fd 100
```

leftRoll *degrees*

lr *degrees*

Rolls the turtle (on to his left side) around the forward vector by the specified angle, measured in *degrees* (1/360 of a circle). This command is designed to run in perspective mode.

Example:

```
perspective
lr 45
repeat 3 [fd 100 lt 120]
```

rightRoll *angle*

rr *angle*

Rolls the turtle (on to his right side) around the forward vector by the specified *angle* , measured in degrees (1/360 of a circle). This command is designed to run in perspective mode.

Example:

```
perspective
rr 45
repeat 3 [fd 100 rt 120]
```

upPitch *angle*

uP *angle*

Pitches the turtle nose up around its base by the specified *angle* , measured in degrees (1/360 of a circle). This command is designed to run in perspective mode.

Example:

```
perspective
up 45
repeat 3 [fd 100 rt 120]
```

downPitch *angle*

down *angle*

Graphics / Relative Turtle Motion / downPitch

Pitches the turtles nose downward around its base by the specified *angle* , measured in degrees (1/360 of a circle). This command is designed to run in perspective mode.

Example:

```
perspective  
down 45  
repeat 3 [fd 100 rt 120]
```

Absolute Turtle Motion

These additional commands enable you to set the turtle's absolute position and course.

Also the home command is quite useful if you've lost your turtle in 3d space ;-)

When the pen is set to down (with PenDown), the turtle will draw a straight line from its last position to its new position.

Absolute Turtle Motion

- Home 200
- setX 201
- setY 201
- setZ 201
- setXY 202
- setXYZ 202
- setPos 204
- _setPos 205
- setPosXYZ 205
- _setPosXYZ 206
- setSpherePos 206
- setCylinderPos 207
- setHeading 208, setH 208
- setPitch 208
- setRoll 209
- setOrientation 209
- spinX 210
- spinY 210
- spinZ 210

Home

moves the turtle to the center of the screen. Equivalent to SETPOS [0 0].

Example:

```
setXY 100 100  
Home
```

setX *xcor*

Moves the turtle horizontally along the X axis from its current position to a new absolute X coordinate.

The argument is the new X coordinate.

Example:

```
setX 100  
setY 100  
setX 0  
setY 0
```

setY *ycor*

Moves the turtle vertically along the Y axis from its current position to a new absolute Y coordinate.

The argument is the new Y coordinate.

Example:

```
setX 100  
setY 100  
setX 0  
setY 0
```

setZ *zcor*

Moves the turtle along the Z axis from its current position to a new absolute Z coordinate.

The argument is the new Z coordinate.

This command is designed to run in perspective mode.

Example:

```
perspective
pu setX 100 pd
setZ 100
setY 100
setZ 0
setY 0
```

setXY *xcor ycor*

moves the turtle to an absolute screen position. The two inputs are numbers, Lists, Arrays or FloatArrays representing the X and Y coordinates.

Examples:

```
setXY 100 0
setXY 100 100
setXY 0 100
setXY 0 0
```

```
x=rSeq -1 1 1000
y=(sin x*x*20*360)*300
x=x*400
setPC 0
setXY x y
setPC 4
setXY x toList lowPassFilter Int16Array y 10
```

Graphics / Absolute Turtle Motion / setXYZ

setXYZ *xcor ycor zcor*

Moves the turtle to an absolute 3D position. The three arguments are numbers, Lists, Arrays or FloatArrays representing the X, Y and Z coordinates.

This command is designed to run in perspective mode. See also PosXYZ.

Example (draw a wireframe cube):

```
perspective
l=100
setXYZ 1 0 0
setXYZ 1 1 0
setXYZ 0 1 0
setXYZ 0 0 0
setXYZ 1 0 0
setXYZ 1 0 1
setXYZ 0 0 1
setXYZ 0 0 0
setXYZ 0 0 1
setXYZ 0 1 1
setXYZ 1 1 1
setXYZ 1 0 1
setXYZ 1 1 1
setXYZ 1 1 0
setXYZ 0 1 0
setXYZ 0 1 1
rotatescene
```

Example (draw some sine waves in 3D):

```

perspective
for [i 0 360 10] ~
[   for [j 0 360] ~
    [   setxyz j 200*(sin j)*sin i -i
      ]
    pu
    setxyz 0 0 -:i
    pd
  ]
rotatescene

```

setPos *pos*

Moves the turtle to an absolute X,Y coordinate. The argument is a list of two numbers, the X and Y coordinates. See also Pos.

Example 1 (draw a square):

```

cs
setpos [0 100]
setpos [100 100]
setpos [100 0]
setpos [0 0]

```

Example 2 (the most common logo question):

```

make "x 0
make "y 100

```

then

```

setpos [:x :y]

```

will fail, but


```
setpos (list :x :y)
```

will work.

Why?

Because the first case is a list that contains 2 words :x and :y. In the second case a list is BUILT containing the VALUE of :x and :y. You can see this more clearly by using the show command.

```
show [:x :y]           ;[:x :y]  
show (list :x :y)      ;[0 100]
```

_setPos pos

moves the turtle to an absolute screen position. The input is a list of two numbers, the X and Y coordinates. `_setPos` does, unlike `setPos`, not change the turtle heading and it does no wrapping, but it is faster than `setPos`.

Example:

```
_setPos [200 100]
```

setPosXYZ pos

Moves the turtle to an absolute X,Y,Z coordinate. The argument is a list of three numbers, the X, Y and Z coordinates. This command is designed to run in perspective mode. See also `PosXYZ`.

Example (draw a cube):

```
perspective
setposxyz [0 100 0]
setposxyz [100 100 0]
setposxyz [100 0 0]
setposxyz [0 0 0]
setposxyz [0 0 100]
setposxyz [100 0 100]
setposxyz [100 100 100]
setposxyz [0 100 100]
setposxyz [0 0 100]
setposxyz [0 100 100]
setposxyz [0 100 0]
setposxyz [100 100 0]
setposxyz [100 100 100]
setposxyz [100 0 100]
setposxyz [100 0 0]
```

`_setPosXYZ` *pos*

moves the turtle to an absolute screen position. The input is a list of three numbers, the X, Y and Z coordinates. `_setPosXYZ` does, unlike `setPosXYZ`, no wrapping, but it is faster than `setPosXYZ`.

Example:

```
_setPosXYZ [100 200 300]
```

`setSpherePos` *r phi theta*

moves the turtle to the spherical coordinate position described by radius *r*, XY-angle *phi* and the inclination *theta*.

Example (partial wire sphere in 3D):

```
pu
for [s 48 360 24]
[   for [t 45 180 16]
    [   setSpherePos  r  s t pd
      ]
    pu
  ]
pu
for [t 45 180 16]
[   for [s 48 360 24]
    [   setSpherePos  r  s t pd
      ]
    pu
  ]
]
```

setCylinderPos *r phi y*

moves the turtle to the cylinder coordinate position described by radius *r*, XY-angle *phi* and the Y position.

Example (wire cylinder in 3D):

```
pu
for [s 0 360 24]
[   for [h 0 180 20]
    [   setCylinderPos 100 s h
        pd
      ]
    pu
  ]
pu
for [h 0 180 20]
[   for [s 0 360 24]
    [   setCylinderPos 100 s h
        pd
      ]
    pu
  ]
]
```

setHeading *angle*

setH *angle*

Turns the turtle to a new absolute heading.

The argument is an *angle* , the heading in degrees clockwise from the positive Y axis.

See also HEADING .

If you are in perspective mode then, the heading in degrees which is positive from the positive X-Axis to the positive Y-Axis rotating about the Z-Axis.

Example:

```
setheading 45
show heading      ; 45
```

setPitch *angle*

Graphics / Absolute Turtle Motion / setPitch

Pitches the turtle to a new absolute pitch.

The argument is a *angle* , the pitch in degrees which is positive from the negative Z-Axis to the position Y-Axis rotating about the X-Axis.

It is important to Understand your ORIENTATION in 3D.

This command is designed to run in PERSPECTIVE mode. See also PITCH .

Example:

```
perspective
setpitch 45
show pitch      ;45
```

setRoll *angle*

Rolls the turtle to a new absolute roll.

The argument is a *angle* , the roll in degrees which is positive from the positive X-Axis to the negative Z-Axis rotating about the Y-Axis.

It is important to Understand your ORIENTATION in 3D.

This command is designed to run in PERSPECTIVE mode. See also ROLL .

Example:

```
perspective
setroll 45
show roll      ;45
```

setOrientation *list*

Orients the turtle to a new absolute orientation.

The argument is a *list*, the [roll pitch heading] in degrees.

It is important to Understand your Orientation in 3D.

This command is designed to run in perspective mode. See also ORIENTATION command.

Example:

```
setorientation [180 45 90]  
show orientation      ;[180 45 90]
```

spinX *angle*

rotates the turtle around the x-axis. This command is designed to run in PERSPECTIVE mode.

Example:

```
perspective  
cs  
spinX 30  
fd 100
```

spinY *angle*

rotates the turtle around the y-axis. This command is designed to run in PERSPECTIVE mode.

Example:

```
perspective  
cs  
spinY 30  
fd 100
```

Graphics / Absolute Turtle Motion / spinZ

spinZ *angle*

rotates the turtle around the z-axis. This command is designed to run in PERSPECTIVE mode.

Example:

```
perspective
cs
spinZ 30
fd 100
```

Turtle Motion Queries

...are asking the turtle questions, getting numbers or lists of numbers as answers.

Turtle Motion Queries

- xCor 212
- yCor 212
- zCor 213
- Pos 213
- PosXYZ 213
- Heading 214
- Pitch 214
- Roll 214
- Orientation 215
- towards 215
- towardsXYZ 215
- Distance 216
- DistanceXYZ 216
- Pixel 217
- Scrunch 217

xCor

Outputs a number, the turtle's X coordinate.

Example:

```
setx 100  
show xcor      ;100
```

yCor

Graphics / Turtle Motion Queries / yCor

Outputs a number, the turtle's Y coordinate.

Example:

```
sety 100  
show ycor      ;100
```

zCor

Outputs a number, the turtle's Z coordinate.

This command is designed to run in perspective mode.

Example:

```
perspective  
setz 100  
show zcor      ;100
```

Pos

Outputs the turtle's current position, as a list of two numbers, the X and Y coordinates.

Example:

```
setpos [100 100]  
show pos      ;[100 100]
```

turtlePos PosXYZ

Outputs the turtle's current position, as a list of three numbers, the X, Y and Z coordinates.

This command is designed to run in perspective mode.

Example:

```
perspective  
setposxyz [100 100 50]  
show posxyz      ;[100 100 50]
```

angle **Heading**

Outputs an angle, the turtle's heading in degrees.

Example:

```
setheading 90  
show heading      ;90
```

angle **Pitch**

Outputs a angle, the turtle's pitch in degrees.

It is important to Understand your Orientation in 3D. This command is designed to run in perspective mode.

Example:

```
setpitch 90  
show pitch      ;90
```

angle **Roll**

Graphics / Turtle Motion Queries / Roll

Outputs an angle, the turtle's roll in degrees.

It is important to Understand your Orientation in 3D. See also SETROLL.

This command is designed to run in perspective mode.

Example:

```
setroll 90  
show roll ;90
```

orientationList **Orientation**

Outputs a list, the turtle's [roll pitch heading] each in degrees.

It is important to Understand your Orientation in 3D. This command is designed to run in perspective mode.

Example:

```
setorientation [180 45 90]  
show orientation ;[180 45 90]
```

angle **towards** *pos*

Outputs an angle, the heading in degrees, at which the turtle should be headed so that it would point from its current position towards the position *pos* given as the argument.

Example:

```
show towards [100 100] ;45  
setheading towards [300 400] fd distance [300 400]
```

orientationList **towardsXYZ** *pos*

Outputs a list, containing [roll pitch heading] at which the turtle should be oriented so that it would point from its current position to the position *pos* given as the [x y z] argument.

It is important to Understand your Orientation in 3D.

This command is designed to run in perspective mode.

Example:

```
show towardsxyz [100 100 0]      ;[0 0 45]
setorientation towardsxyz [100 100 100]
fd distancexyz [100 100 100]
```

dist **Distance** *pos*

Outputs a number, the distance the turtle must travel along a straight line to reach the position *pos* given as the argument.

Example:

```
show distance [0 100]      ;100
show distance [300 400]    ;500
setheading towards [300 400]
fd distance [300 400]
```

dist **DistanceXYZ** *pos*

Outputs a number, the distance the turtle must travel along a straight line to reach the xyz position given as the argument.

This command is designed to run in perspective mode.

Graphics / Turtle Motion Queries / DistanceXYZ

Example:

```
show towardsxyz [0 100 0] ;100
show towardsxyz [100 100 100] ;173.205080756888
setorientation towardsxyz [100 100 100]
fd distancexyz [100 100 100]
```

color Pixel

outputs the color of the pixel under the turtle.

Example:

```
setPixel [0 0] RGB 1 0 0
reRGBA Pixel ;[1 0 0 1] ;-
```

Scrunch

outputs a list containing two numbers, the X and Y scrunch factors, as used by SETSCRUNCH. (But note that SETSCRUNCH takes two numbers as inputs, not one list of numbers.)

Turtle and Window Control

Here are miscellaneous commands and a few operations to control the graph window and the turtle.

Turtle and Window Control

- showTurtle 219, sT 219
- hideTurtle 219, hT 219
- clean 220
- clearScreen 220, cS 220
- wrap 220
- Window 221
- Fence 221
- perspective 221
- unperspective 222
- setEye 222
- fill 223
- Label 223
- LabelSize 224
- setLabelSize 224
- LabelFont 224
- setLabelFont 225
- LabelWeight 225
- setLabelWeight 225
- LabelAlign 226
- setLabelAlign 226
- TextScreen 226, TS 226
- fullScreen 227, fS 227
- splitScreen 227, sS 227
- setVarsSplitter 227
- setCallsSplitter 227
- allFullScreen 228
- notFullScreen 228
- setScrunch 228
- refresh 229
- noRefresh 229
- singleBuffer 229
- doubleBuffer 229
- refreshP 229

Graphics / Turtle and Window Control

- redraw 230
 - setUpDateGraph 230
 - updateGraph 230
 - updateVars 231
 - updateVarsOnStep 231
 - Calls 231
 - updateCalls 231
 - dispatchMessages 231
 - scroll 232
 - scrollCalibrate 232, scrollCal 232
 - axes 232
 - rotateScene 233
 - setScreenRange 233
-

showTurtle**sT**

makes the turtle visible.

Example:

```
ht  
tree  
st
```

hideTurtle**hT**

makes the turtle invisible. It's a good idea to do this while you're in the middle of a complicated drawing, because hiding the turtle speeds up the drawing substantially.

Example:

```
ht  
tree  
st
```

clean

erases all drawings off the graphics window. The turtle's state (position, heading, pen mode, etc.) is not changed.

Example:

```
setXY 200 100  
seth 30  
clean
```

clearScreen
cS

erases the graphics window and sends the turtle to its initial position and heading. Like HOME and CLEAN together.

wrap

tells the turtle to enter wrap mode: From now on, if the turtle is asked to move past the boundary of the graphics window, it will "wrap around" and reappear at the opposite edge of the window. The top edge wraps to the bottom edge, while the left edge wraps to the right edge. (So the window is topologically equivalent to a torus.) This is the turtle's initial mode. Compare WINDOW and FENCE.

Example:


```
Window  
fd 400  
wrap  
cs  
fd 400
```

Window

tells the turtle to enter window mode: From now on, if the turtle is asked to move past the boundary of the graphics window, it will move offscreen. The visible graphics window is considered as just part of an infinite graphics plane; the turtle can be anywhere on the plane. (If you lose the turtle, HOME will bring it back to the center of the window.) Compare WRAP and FENCE.

Example:

```
Window  
fd 400  
wrap  
cs  
fd 400
```

Fence

tells the turtle to enter fence mode: From now on, if the turtle is asked to move past the boundary of the graphics window, it will move as far as it can and then stop at the edge with an "out of bounds" error message. Compare WRAP and WINDOW.

Example:

```
Fence  
fd 400      ; turtle out of bounds
```

perspective

command that prepares the screen for 3D commands and internally calls `setEye {400 400 600} {0 0 0} {0 1 0}`.

Example:

```
perspective
axes
```

unperspective

command that prepares the screen for 2D commands after perspective mode.

Example:

```
perspective
axes
wait 1000
unperspective
axes
```

setEye eye center upvector

command that sets the 3D projection parameters to new values. All parameters must be 3D arrays, like `{400 400 600}`.

eye : the position of the observer *eye* relative to `{0 0 0}`, defaults to `{400 400 600}` in perspective mode.

center : the position of the *center* of the screen, defaults to `{0 0 0}`.

upvector : the direction which is up on the screen, defaults to `{0 1 0}`, so Y will be upwards, in

Graphics / Turtle and Window Control / setEye

perspective mode.

Example:

```
cS
perspective
axes
setEye {100 400 600}{0 0 0}{0 1 0}
redraw
```

fill

(**fill** *"true"*)

fills in a region of the graphics window containing the turtle and bounded by lines that have been drawn earlier. This is not portable; it doesn't work for all machines, and may not work exactly the same way on different machines.

The command (fill "true) fills the area around the turtle defined by the color specified by penColor. Filling continues outward in all directions as long as the color is encountered. This style is useful for filling areas with multicolored boundaries.

Example:

```
disLS
box
rt 45
pu fd 10
fill
```

Label *text*

takes a word or list as input, and prints the input on the graphics window, starting at the turtle's position. A word will always be in lowercase while the case of a list of words will be preserved.

Example:

```
label "Hallo" ; will print "hallo" in the graph window
label [Hallo World] ; will print "Hallo World"
```

size **LabelSize** *text*
 fontSize (**LabelSize**)

With one argument, this command will output the size of the given *text*. The input, which may be a word or a list is the same as what you would give to Label. You can use this information to build other forms of Label. Other forms might be CENTERLABEL or VERTICALLABEL. You can also use this information to "prepare" a site for *text* (i.e. frame it or set a background).

size:(List) List of 2 integers [width height] of the *text* in the current font.

text :(Thing) Any thing you wish to label with.

Example:

```
show labelsize "Hallo" ; [44 24]
```

Without an argument, LabelSize outputs the current label font size.

Example:

```
show (LabelSize) ; [20 20]
```

setLabelSize *xylist*

sets the size of the next text printed by Label. *xylist* is a list of width and height of a single char.

Example:

```
setLabelSize [10 20] ;set label size to default size
```

Graphics / Turtle and Window Control / LabelFont

fontFaceName **LabelFont**

outputs the current label font name.

Example:

```
LabelFont      ;Times      ; - )
```

setLabelFont *fontFaceName*

sets the default font for label to the word *fontFaceName* .

Examples:

```
setLabelFont "Times  
label "hallo  
fd 100  
setLabelFont "|Courier New|  
label "world
```

weight **LabelWeight**

outputs the currently set label weight.

Example:

```
LabelWeight      ;400      ; - )
```

setLabelWeight *weight*

does not work yet.

sets the *weight* of the label font. *weight* 400 is normal, 1000 is fat.

Example:

```
setLabelWeight 800
label "hallo"
```

alignmentlist **LabelAlign**

outputs the currently set label alignment as a list of two integers [xalign yalign].

There are three possible values for xalign and yalign: -1, 0, 1.

-1 stands for right and top, 0 centers the text, 1 aligns left and bottom. 1 is default.

Example:

```
LabelAlign      ;[0 0] ;-) x=center, y=center
```

setLabelAlign *xalign yalign*

sets the alignment of the label font. There are three possible values for *xalign* and *yalign* : -1, 0, 1.

-1 stands for right and top, 0 centers the text, 1 aligns left and bottom. 1 is default.

Example:

```
setLabelAlign -1 1 ;right-bottom
label "hallo"
```

Graphics / Turtle and Window Control / TextScreen

rearranges the size and position of windows to maximize the space available in the text window (the window used for interaction with Logo). See also SPLITSCREEN and FULLSCREEN.

fullScreen**fS**

rearranges the size and position of windows to maximize the space available in the graphics window. See also SPLITSCREEN and TEXTSCREEN.

Since there must be a text window to allow printing, Logo automatically switches from fullscreen to splitscreen whenever anything is printed.

splitScreen**sS****(splitScreen *ratio*)****(ss *ratio*)**

rearranges the sizes and positions of windows to allow some room for text interaction while also keeping the graphics window visible. The optional *ratio* must be a number in the range 0..1. 0 means the graph window is minimized, the text window is maximized. 1 means the opposite. 0.5 means they are equally tiled. See also TEXTSCREEN and FULLSCREEN.

setVarsSplitter *ratio*

sets the window splitter between the text console and the Vars and Calls windows to the *ratio* . The *ratio* must be a number in the range 0..1.

Example:

```
setVarsSplitter 0.6
```

setCallsSplitter *ratio*

sets the window splitter between the Vars window and the Calls windows to the *ratio* . The *ratio* must be a number in the range 0..1.

Example:

```
setCallsSplitter 0.6
```

allFullScreen

maximizes the aUCBLogo window to fill the screen completely. The menu is also not shown any more. To leave this mode run the command notFullScreen.

notFullScreen

leaves full screen mode entered previously by allFullScreen.

setScrunch *xscale* *yscale*

adjusts the aspect ratio and scaling of the graphics display. After this command is used, all further turtle motion will be adjusted by multiplying the horizontal and vertical extent of the motion by the two numbers given as inputs.

For example, after the instruction

```
setScrunch 2 1
```

motion at a heading of 45 degrees will move twice as far horizontally as vertically. If your squares don't come out square, try this. (Alternatively, you can deliberately misadjust the aspect ratio to draw an ellipse.)

For Unix machines and Macintoshes, both scale factors are initially 1. For DOS machines, the scale

Graphics / Turtle and Window Control / setScrunch

factors are initially set according to what the hardware claims the aspect ratio is, but the hardware sometimes lies. The values set by SETSCRUNCH are remembered in a file (called SCRUNCH.DAT) and are automatically put into effect when a Logo session begins.

refresh

tells Logo to record the turtle's motions so that they can be reconstructed with another setEye position.

noRefresh

tells Logo not to record the turtle's motions. This will make drawing faster and less memory requiring, but prevents redraw with another setEye setting.

singleBuffer

sets the drawing to single buffered mode. On some graphics cards the SwapBuffers() function behaves strangely, this is for those bad cases.

After calling singleBuffer all the drawing goes directly to the graph window.

Several demos might flicker in this mode.

doubleBuffer

sets the drawing to double buffered. This may be necessary when you have switched to singleBuffer first and now you want to switch back to default.

refreshP
refresh?

outputs if Logo is recording the graphics commands.

redraw

does redraw the graphics saved in the graphics recorder.

setUpdateGraph *isUpdating*

sets the auto update of the graphic window to either true or false. If false, then the turtle draws only hidden on the memory screen, which is good for animations.

Example:

```
to rotateTree
  ht
  setUpdateGraph false
  forever
  [ clean
    rt 1
    tree 6 100
    updategraph
    if key? [stop]
  ]
end
```

updateGraph

Graphics / Turtle and Window Control / updateGraph

immediately updates the graphics screen from the memory screen. This is good for animations in conjunction with `setUpdateGraph false`.

updateVars

updates the Vars window showing all variables and their values. This is good for debugging.

updateVarsOnStep *yesno*

sets the calling of `updateVars` during stepping to true or false. This might improve the performance a lot if you have many variables.

Example:

```
updateVarsOnStep false
```

Calls

outputs a list containing all nested user procedure calls with their parameters.

updateCalls

updates the Calls window showing all nested calls and their parameters. It's good to see the Calls when being in a big program and wanting to know where a stack overflow occurs.

dispatchMessages

dispatches all pending windows messages, i.e. mouse moves, mouseclicks, etc. This is necessary for instantaneous reactions on user input, esp. mouse actions. See example

Example:

```
forever
[ dispatchMessages
  m=MousePos
  updateVars
]
```

scroll *sizelist displacementlist*

command which scrolls the graph window of size *sizelist* right and down at the turtle by *displacementlist* pixels.

Example:

realDisplacementList **scrollCalibrate *displacementlist***
 realDisplacementList **scrollCal *displacementlist***

outputs the displacement which scroll will scroll by displacement pixels. This is necessary because logo coordinates are scaled to window coordinates and scroll can only scroll by whole pixels, which will probably be no integer in logo coordinates.

So if you want to scroll the screen i.e. by 2 pixels in x direction then you will need to do the following:

```
cs rbox fill
_setPos [-400 299]
scc=scrollCal [2 0]
scroll [800 600] scc
```

axes (library *procedure*)

command drawing axes as to have a coordinate system. This is very useful in perspective mode.

rotatescene (library *procedure*)

command which calls setEye and redraw in a loop with user interaction to rotate a drawn 3D scene. This is shown off in many examples.

setScreenRange

When you use Graph windows with a defined shape then setScreenRange might be useful to get away from the default [800,600] logical pixels.

Turtle and Window Queries

...are questions you can ask the turtle and the mouse.

Turtle and Window Queries

- shownP 234
- MousePos 234
- TextMousePos 234
- TextMouseX 235
- TextMouseY 235
- MouseButtons 235

shownP

shown?

outputs TRUE if the turtle is shown (visible), FALSE if the turtle is hidden. See SHOWTURTLE and HIDETURTLE.

poslist **MousePos**

outputs the position of the mouse in the graph window as a list of three floating point numbers.

poslist **TextMousePos**

outputs the position of the mouse in the console window as a list of two integers, meaning the x and y mouse location in characters.

Example:

```
forever [
  textpos=TextMousePos
  dispatchMessages
  updateVars
]
;look at the vars window while moving the mouse in the console!
```

x **TextMouseX**

outputs the column coordinate of the mouse in the console window, in units of characters.

Example:

```
forever [
  textxpos=TextMouseX
  dispatchMessages
  updateVars
]
;look at the vars window while moving the mouse in the console!
```

y **TextMouseY**

outputs the line coordinate of the mouse in the console window, in units of characters.

Example:

```
forever [
  textypos=TextMouseY
  dispatchMessages
  updateVars
]
;look at the vars window while moving the mouse in the console!
```

buttons **MouseButton**

outputs the mouse buttons value. 1=left button, 2=right button, 3=left&right button.

Multiple Turtles

Now multiple turtles can be used. But unlike in MSWLogo (where there is an internal array of turtles) those turtles are able to be assigned to variables.

Multiple Turtles

- Turtle 237
- newTurtle 237
- setTurtle 237

activeTurtle **Turtle**

outputs the activeTurtle for usage with a variable assignment.

Example:

```
oldt=turtle  
setTurtle newTurtle  
fd 100  
setTurtle oldt
```

aNewTurtle **newTurtle**

outputs a new Turtle which can be used with setTurtle or it can be stored in a variable.

setTurtle *aTurtle*

command which sets the active turtle to *aTurtle* . *aTurtle* can be obtained from turtle or newTurtle or indirectly from a variable. See also [turtlestest.lg](..\turtlestest.lg)!

Pen and Background Control

The turtle carries a pen that can draw pictures. At any time the pen can be UP (in which case moving the turtle does not change what's on the graphics screen) or DOWN (in which case the turtle leaves a trace). If the pen is down, it can operate in one of three modes: PAINT (so that it draws lines when the turtle moves), ERASE (so that it erases any lines that might have been drawn on or through that path earlier), or REVERSE (so that it inverts the status of each point along the turtle's path).

Pen and Background Control

- PenDown 238, PD 238
- PenUp 238, PU 238
- PenPaint 239, PPt 239
- PenErase 239, PE 239
- PenReverse 239, PX 239
- setPenColor 239, setPC 239
- setFloodColor 240, setFC 240
- setMaterialAmbient 241
- setMaterialDiffuse 241
- setMaterialSpecular 241
- setMaterialEmission 241
- setMaterialShininess 241
- setScreenColor 242, setSC 242
- setPenSize 242, setPS 242
- setPenPattern 242
- setpen 243
- setDepthFunc 243

PenDown

PD

sets the pen's position to DOWN, without changing its mode.

PenUp

PU

sets the pen's position to UP, without changing its mode.

PenPaint**PPt**

sets the pen's position to DOWN and mode to PAINT.

PenErase**PE**

sets the pen's position to DOWN and mode to ERASE.

PenReverse**PX**

sets the pen's position to DOWN and mode to REVERSE. (This may interact in hardware-dependent ways with use of color.)

setPenColor *colornumber*

setPC *colornumber*

setPenColor *rgblist*

setPC *rgblist*

setPenColor *colorname*

setPC *colorname*

sets the pen color to the given number, which must be a integer, or a *rgblist* (consisting of three

items, red, green, and blue, which must be in the range 0..1), or must be one of the color names in the wxWidgets color database. Color 0 is always black; color 7 is always white. Other color numbers may or may not be consistent between machines, but colors by name should be platform independent - as is wxWidgets.

Examples:

```
disls      ;is neccessary to make independent color drawings
setpc 1
box
setpc 2
box
setpc 3
box
setpc 4
box
setpc 7
box
setpc rgb 1 0 0
box
setpc rgb 0 1 0 box
setpc rgb 0 0 1 box
setpc rgba 0 0 0 .2 box
setpc rgba 0 0 0 .2 box
setpc rgba 0 0 0 .2 box
setpc rgba 0 0 0 .2 box
setpc rgba 0 0 0 .2 box
setpc [0 0.5 0] box
setpc "red box
setpc "yellow box
```

setFloodColor *acolor*

setFC *acolor*

sets the flood color to the given *acolor*, which must be a integer, or a rgblist (consisting of three items, red, green, and blue, which must be in the range 0..1), or must be one of the color names in the wxWidgets color database. Color 0 is always black; color 7 is always white. Other colors may

or may not be consistent between machines. See also setPC.

Example:

```
setfc rgb 1 0 1  
fbox
```

setMaterialAmbient *acolor*

sets the material ambient color to *acolor* . For allowed things in *acolor* see setPC!

setMaterialDiffuse *acolor*

sets the material diffuse color to *acolor* . For allowed things in *acolor* see setPC!

setMaterialSpecular *acolor*

sets the material specular color to *acolor* . For allowed things in *acolor* see setPC!

setMaterialEmission *acolor*

sets the material emission color to *acolor* . For allowed things in *acolor* see setPC!

setMaterialShininess *shininess*

sets the material *shininess* , a number between 0 and about 100.

Example:

```
perspective
setPC "red
setMaterialShininess 10
Sphere 200
```

setScreenColor *acolor*

setSC *acolor*

set the screen background color. See also setPC and the color database.

setPenSize *size*

setPS *size*

sets the pen *size* to the *size* list of two numbers. The second number is ignored at the moment.

In perspective mode, if cylinderLines are enabled, the first list entry is the radius of the cylinders.

Example:

```
setPenSize [3.5 4]
fd 100
cs
perspective
setpc 4
enCL
setPenSize [50 50]
fd 100
```

setPenPattern *pattern*

Does not work yet!

setpen *list* (library procedure)

sets the pen's position, mode, and hardware-dependent characteristics according to the information in the input list, which should be taken from an earlier invocation of PEN.

setDepthFunc *num*

sets the depth comparison function of OpenGL. This can be useful when you want to overwrite drawn graphics, which might be nearer to the eye. See bounce3.lg for an example!

0=GL_NEVER Never passes.

1=GL_LESS Passes if the incoming z value is less than the stored z value. This is the default value.

2=GL_EQUAL Passes if the incoming z value is equal to the stored z value.

3=GL_LEQUAL Passes if the incoming z value is less than or equal to the stored z value.

4=GL_GREATER Passes if the incoming z value is greater than the stored z value.

5=GL_NOTEQUAL Passes if the incoming z value is not equal to the stored z value.

6=GL_GEQUAL Passes if the incoming z value is greater than or equal to the stored z value.

7=GL_ALWAYS Always passes.

Especially 3 and 7 are most useful.

Examples:

```
perspective
cs
setPC "red
setDepthFunc 7
Sphere 100
setDepthFunc 3
Sphere 100
```

enable and disable flags

enable and disable flags

- `enableLineSmooth` 245, `enLS` 245
- `disableLineSmooth` 245, `disLS` 245
- `enablePolySmooth` 246, `enPS` 246
- `disablePolySmooth` 246, `disPS` 246
- `enableRoundLineEnds` 246, `enRLE` 246
- `disableRoundLineEnds` 247, `disRLE` 247
- `enableCylinderLines` 247, `enCL` 247
- `disableCylinderLines` 248, `disCL` 248
- `enableDepthTest` 248, `enDT` 248
- `disableDepthTest` 248, `disDT` 248
- `enableLighting` 248
- `disableLighting` 249
- `enableDither` 249
- `disableDither` 249
- `enablePointSmooth` 249
- `disablePointSmooth` 249
- `enableFog` 249
- `disableFog` 250

`enableLineSmooth`

`enLS`

enables antialiasing of lines.

Example:

```
enLS
circle 300
```

disableLineSmooth
disLS

disables antialiasing of lines.

Example:

```
disLS  
circle 300
```

enablePolySmooth
enPS

enables antialiasing of polygons.

Be careful: This can lower the drawing speed by a large amount!

Example:

```
enPS  
fillCircle 300
```

disablePolySmooth
disPS

disables antialiasing of polygons.

Example:

```
disPS  
fillCircle 300
```

enableRoundLineEnds
enRLE

enables drawing of round line ends at every line ending.

This can lower the drawing speed, esp. in 3D, because then the line ends are spheres! But it's very nice!

Example:

```
enRLE
setPS [50 50]
fd 100
```

disableRoundLineEnds
disRLE

disables drawing of round line ends at every line ending.

Example:

```
disRLE
setPS [50 50]
fd 100
```

enableCylinderLines
enCL

enables drawing of automatic cylinder lines when drawing with the turtle.

This command is designed to run in perspective mode.

Example:

```
perspective
enCL
setps [50 50]
setpc 4
fd 100
rt 90
fd 100
```

disableCylinderLines
disCL

disables drawing of automatic cylinder lines when drawing with the turtle. Instead flat lines are drawn.

enableDepthTest
enDT

enables OpenGL's depth test of graphics. perspective automatically enables depthtest.

disableDepthTest
disDT

disables OpenGL's depth test of graphics. perspective automatically enables depthtest.

Example:

```
perspective
clockwork      ;this looks very unpretty
disDT
clockwork      ;now you should see the difference
```

enableLighting

enables the OpenGL lighting. It is enabled by default after invoking perspective.

disableLighting

disables OpenGL's lighting. This may sometimes work better with alpha blending (transparency).

enableDither

enables OpenGL's dithering of colors. This might improve the quality of graphics on displays having only 8 bit or 16 bit colors, but probably slows down the rendering.

disableDither

disables OpenGL's dithering. This is the default.

enablePointSmooth

enables OpenGL's point smoothing. This is nice if you want to have antialiased points, which flicker less on moving.

disablePointSmooth

disables OpenGL's point smoothing. This is the default.

enableFog

enables OpenGL's fog effect. This is especially useful if you draw lots of pixels, because they aren't lighted and depth might be hard to see. With fog this can be overcome, as used in IFS3D.lg.

disableFog

disables OpenGL's fog effect. This is the default.

Pen Queries

...ask teh pen questions on its properties.

Pen Queries

- PenDownP 251
 - PenMode 251
 - PenColor 251, PC 251
 - FloodColor 252, FC 252
 - Palette 252
 - PenSize 252, PenPattern 252
 - pen 252
 - ScreenColor 253, SC 253
-

PenDownP

PenDown?

outputs TRUE if the pen is down, FALSE if it's up.

PenMode

outputs one of the words PAINT, ERASE, or REVERSE according to the current pen mode.

PenColor

PC

outputs a color number, a nonnegative integer that is associated with a particular color by the hardware and operating system.

Examples:

```
setpc 0
pc      ; -16777216  ; -)
hex pc   ; FF000000  ; -)
reRGBA pc ; [0 0 0 1] ; -)
setpc 4
pc      ; -16776961  ; -)
hex pc   ; FF0000FF  ; -)
reRGBA pc ; [1 0 0 1] ; -)
```

FloodColor**FC**

outputs the flood fill color number, a nonnegative integer that is associated with a particular color by the hardware and operating system. This color is used for filling shapes.

The color can be used like the ones output by PC.

Palette *colornumber*

is not yet implemented!

outputs a list of three integers, each in the range 0-65535, representing the amount of red, green, and blue in the color associated with the given number.

PenSize**PenPattern**

output hardware-specific pen information.

pen (library *procedure*)

outputs a list containing the pen's position, mode, and hardware-specific characteristics, for use by SETPEN.

ScreenColor
SC

outputs the graphics screen background color.

Drawing Curves

Here are some commands to draw curves. The most useful one is, of course, circle, although you can also draw a circle with arc or ellipse.

These commands do not move the turtle.

Drawing Curves

- circle 254
- EllipseArc 254
- Ellipse 255
- Arc 255
- arc2 256
- cngon 256

circle *radius*

draws a circle based on the turtles position and the given arguments.

The size is based on the *radius* . The current turtle position will be at the center of the circle. Circle will also follow wrap/fence/windows modes.

Examples:

```
circle 100  
circle 50
```

EllipseArc *angle xradius yradius startAngle*

draws part of or all of an ellipse based on the turtle heading, turtle position and the given

Graphics / Drawing Curves / EllipseArc

arguments.

The ellipse starts at the rear of the turtle heading and sweeps by the amount of *angle* starting at *startAngle* .

The size is based on the *xradius* and *yradius* values. The current turtle position will be at the center of the ellipse. EllipseArc will also follow wrap/fence/windows modes.

xradius is the radius from the turtle to her right till the arc.

yradius is the radius in direction of the turtle's head till the arc.

Examples:

```
EllipseArc 360 100 200 0
cs
EllipseArc 90 50 50 0
cs
EllipseArc 90 50 50 90
```

Ellipse *xradius* *yradius*

draws an ellipse based on the turtle heading, turtle position and the given arguments.

The center of the ellipse the current turtle position. *xradius* is the ellipse radius to the turtle's right, *yradius* is the radius in direction of the turtle's head.

Ellipse will also follow wrap/fence/windows modes.

Examples:

```
Ellipse 100 200
fd 200
Ellipse 100 200
rt 30
Ellipse 100 200
```

Arc *angle radius*

draws an arc of a circle, with the turtle at the center, with the specified *radius* , starting at the turtle's heading and extending clockwise through the specified *angle* . The turtle does not move.

The size is based on the *radius* . The current turtle position will be at the center of the arc. Arc will also follow wrap/fence/windows modes.

Arc 360 *radius* will of course draw a circle.

Example:

```
Arc 360 100  
Arc 90 50
```

arc2 *angle radius*

like arc, draws an arc of a circle, but with the arc starting at the turtle, and the center to the right of the turtle. If both *angle* and *radius* are positive a clockwise arc is drawn. If both are negative, a counterclockwise arc is drawn.

Examples:

```
arc2 30 100 ;draws a right turn  
arc2 -30 -100 ;draws a left turn
```

cngon *sides radius* (library procedure)

draws a regular polygon inside of a not drawn circle.

Examples:

Graphics / Drawing Curves / cngon

```
cngon 3 100  
cngon 7 100
```

Drawing filled shapes

...is easy with those commands.

Drawing filled shapes

- PolyStart 258
- PolyEnd 259
- TessStart 259
- TessContour 260
- TessEnd 260
- setTessWindingRule 260
- endSurfaceStart 261
- SurfaceColumn 262
- SurfaceEnd 263
- GraphicStart 264
- GraphicEnd 264
- drawGraphic 265
- VideoStart 265
- VideoFrame 265
- VideoEnd 266
- fillRect 266
- fillCircle 266
- fillEllipse 267
- fillPie 267
- Sphere 267
- Ellipsoid 268
- partialEllipsoid 268

PolyStart

command which tells logo to enter polygon mode.

This is cool for drawing shapes, especially in 3D. In perspective mode the polygon will be lighted.

Be careful: the polygon must be simple and convex. For non-convex polygons and polygons with

Graphics / Drawing filled shapes / PolyStart

holes use TessStart..TessContour..TessEnd!

Example:

```
setPC 4  
PolyStart  
box  
PolyEnd
```

```
perspective  
PolyStart  
box  
PolyEnd  
rotatescene
```

PolyEnd

command which tells logo to finish polygon mode and draw the filled polygon.

This is cool for drawing shapes, esp. in 3D. In perspective mode the polygon will be lighted.

Example:

```
setPC 4  
PolyStart  
box  
PolyEnd
```

```
perspective  
PolyStart  
box  
PolyEnd  
rotatescene
```

TessStart

command which tells Logo to enter polygon tessellation mode.

This is cool for drawing shapes, esp. in 3D. In perspective mode the polygon will be lighted.

The tessellated polygons feature is needed if you want to draw non-convex polygons or polygons with holes.

Example:

```
TessStart    ;a non-convex polygon
fd 100 rt 90 fd 100 rt 90 fd 100
rt 135 fd 100/sqrt 2 lt 90 fd 100/sqrt 2
TessEnd
```

TessContour

command for use in tessellation mode to start a new contour of the polygon, i.e. a hole.

Example:

```
TessStart    ;a square with a triangular hole
repeat 4 [fd 100 rt 90]
TessContour
pu rt 45 fd 20 lt 15 pd fd 60 rt 120 fd 60 rt 120 fd 60 rt 120
TessEnd
```

TessEnd

command to leave polygon tessellation mode. See TessStart, TessContour!

setTessWindingRule *rulenr*

Graphics / Drawing filled shapes / setTessWindingRule

command to set the winding rule for the next tessellations to *rulenr* . The *rulenr* can be any of the following values:

```
0 GLU_TESS_WINDING_ODD
1 GLU_TESS_WINDING_NONZERO
2 GLU_TESS_WINDING_POSITIVE
3 GLU_TESS_WINDING_NEGATIVE
4 GLU_TESS_WINDING_ABS_GEQ_TWO
```

Example:

```
to tessstar
  cs
  ht
  image=loadImage "bricks.png"
  tex=Texture image
  enTex
  window
  setPC "white"
  back 300
  setTessWindingRule 1
  TessStart
  setTexXY 0.4 1      fd 600  rt 180-72/2
  setTexXY 1      0.2  fd 600  rt 180-72/2
  setTexXY 0      0.5  fd 600  rt 180-72/2
  setTexXY 1      0.8  fd 600  rt 180-72/2
  setTexXY 0.4 0      fd 600  rt 180-72/2
  TessEnd
end
```

endSurfaceStart

command which tells Logo to enter surface definition mode. It's very easy to define surfaces with this command, SurfaceColumn and SurfaceEnd.

The normal vectors of the first column of points is linked to the last column, like in a torus. So you can easily define closed surfaces.

See also `pretzel2.lg` and `3dsurfaces3.lg`!

Example:

```
to surfacetest
  perspective
  r=200
  setpc rgb 1 0 0
  ht
  SurfaceStart
  for [z -1 1 0.1]
  [
    pu
    setPosXYZ (list -1 (cos 90*-1)*cos 90*z z)*r
    pd
    for [x -1 1 0.1]
    [
      setPosXYZ (list x (cos 90*x)*cos 90*z z)*r
    ]
    SurfaceColumn
  ]
  SurfaceEnd
  rotateScene
end
```

SurfaceColumn

command which to call after you have defined a column of surface points, and before the next column of surface points.

Example:

```
to surfacetest
  perspective
  r=200
  setpc rgb 1 0 0
  ht
  SurfaceStart
  for [z -1 1 0.1]
    [
      pu
      setPosXYZ (list -1 (cos 90*-1)*cos 90*z z)*r
      pd
      for [x -1 1 0.1]
        [
          setPosXYZ (list x (cos 90*x)*cos 90*z z)*r
        ]
      SurfaceColumn
    ]
  SurfaceEnd
  rotateScene
end
```

SurfaceEnd

command to end surface mode.

Example:

```
to surfacetest
  perspective
  r=200
  setpc rgb 1 0 0
  ht
  SurfaceStart
  for [z -1 1 0.1]
    [
      pu
      setPosXYZ (list -1 (cos 90*-1)*cos 90*z z)*r
      pd
      for [x -1 1 0.1]
        [
          setPosXYZ (list x (cos 90*x)*cos 90*z z)*r
        ]
      SurfaceColumn
    ]
  SurfaceEnd
  rotateScene
end
```

GraphicStart

command starting a new graphic command list. All the graphic commands written on the screen till GraphicEnd are recorded to it.

Example:

```
to testDrawGraphic
  GraphicStart
  box
  abox=GraphicEnd
  cs
  rt 30 fd 100
  drawGraphic abox
  lt 40 fd 100
  (drawGraphic abox 0.5)
end
```

GraphicEnd

outputs the graphic command list recorded since the last GraphicStart. Its output is a new Logo type which cannot be converted to a text form so far.

For an example see GraphicStart.

drawGraphic *agraphic*
(**drawGraphic** *agraphic scaling*)

command which plays the graphic command list *agraphic* at the current position and orientation of the turtle. The graphic will be scaled by a factor of *scaling* if given. For an example see GraphicStart or drawasteroids.lg.

VideoStart *filename*
(**VideoStart** *filename framerate*)

command starting the creation of a Video for Windows file (.avi) with name *filename* and *framerate* (or *framerate* =30 default).

Example:

```
VideoStart "test
for [i 0 90] [cs rt i rbox updateGraph VideoFrame]
VideoEnd
```

VideoFrame

command writing a frame to the with VideoStart opened Video for Windows file (.avi).

Example:

```
VideoStart "test  
for [i 0 90] [cs rt i rbox updateGraph VideoFrame]  
VideoEnd
```

VideoEnd

command finishing the with VideoStart opened Video for Windows file (.avi).

Example:

```
VideoStart "test  
for [i 0 90] [cs rt i rbox updateGraph VideoFrame]  
VideoEnd
```

fillRect *xy1 xy2*

draws a filled rectangle between the coordinates *xy1* and *xy2* in the floodColor .

Example:

```
rt 30  
setFC RGB 1 0 0  
fillRect [0 0][200 100]
```

fillCircle *radius*

fills a circle at the turtle with *radius* in the floodColor.

Example:

Graphics / Drawing filled shapes / fillCircle

```
setFC RGB 1 0 0  
fillCircle 100
```

fillEllipse *radiusX radiusY*

draws a filled ellipse at the turtle with *radiusX* and *radiusY* in the floodColor. The radii are relative to the turtle heading.

Example:

```
setFC RGB 1 0 0  
right 30  
fillEllipse 200 100
```

fillPie *angle radius*

command drawing a filled pie at the turtle's position from the Heading clockwise.

Example:

```
cS  
rt 30  
fillPie 90 200
```

Sphere *radius*

(**Sphere** *radius slices stacks*)

command drawing a sphere with *radius* in perspective mode.

The optional arguments *slices* and *stacks* are integer numbers, which allow finer control about the drawing. They should be not too big (about 500 maximal), because else the drawing will take ages.

Example:

```
perspective
Sphere 100
rotatescene
```

Ellipsoid *radiusx radiusy radiusz*

Example:

```
cS
perspective
setPC 4
Ellipsoid 100 200 300
rotatescene
```

partialEllipsoid *radiusx radiusy radiusz sls sle sl sts ste st*

command to draw a partial ellipsoid with the three radii *radiusx* , *radiusy* and *radiusz* , relative to the current turtle orientation.

sls is slicesStart (angle in degrees, 0..360),

sle is slicesEnd (angle in degrees, 0..360),

sl is slices (integer 1..100 resonably),

sts is stacksStart (angle in degrees, 0..180),

ste is stacksEnd (angle in degrees, 0..180),

st is stacks (integer 1..100 resonably).

Examples:

Graphics / Drawing filled shapes / partialEllipsoid

```
perspective  
partialEllipsoid 100 200 300 0 180 10 0 90 10  
partialEllipsoid 100 200 300 0 360 10 0 180 10  
rotatescene
```

Lighting

Here are functions to set the lighting parameters of OpenGL.

Lighting

- `setLightPos` 270
 - `setLightAmbient` 270
 - `setLightDiffuse` 271
 - `setLightSpecular` 271
-

`setLightPos` *3darraypos*

command setting the position of light0 of OpenGL.

The argument is a array containing three numbers (the position).

Example:

```
cs
perspective
setPC 4
Sphere 300
setLightPos {0 0 100}
redraw
```

`setLightAmbient` *color*

command to set the ambient light *color* . *color* is a int build with RGB, RGBA, HSB, or HSBA.

Example:

Graphics / Lighting / setLightAmbient

```
cS
setPC 7
perspective
setLightAmbient RGB 1 0 0
Sphere 300
```

setLightDiffuse *color*

command to set the diffuse light *color* . *color* is a int build with RGB, RGBA, HSB, or HSBA.

Example:

```
cS
setPC 7
perspective
setLightDiffuse RGB 1 0 0
Sphere 300
```

setLightSpecular *color*

command to set the specular light *color* . *color* is a int build with RGB, RGBA, HSB, or HSBA.

Example:

```
cS
setPC 7
perspective
setLightSpecular RGB 1 0 0
Sphere 300
```

Fog

The nice fog effect of OpenGL is now available in aUCBLogo (4.689).

Fog

- `setFogDensity` 272
- `setFogRange` 272
- `setFogColor` 273
- `setFogMode` 273

`setFogDensity` *density*

command to set the *density* parameter of the OpenGL fog. It does not enableFog, you must call that explicitly. Typical values for *density* are 0.001...0.5.

Example:

```
cs perspective
setFogDensity 0.3
setFogColor "white
enableFog
x=FloatArray random IntArray rSeqFA 800 800 1000000
y=FloatArray random IntArray rSeqFA 400 400 1000000
z=FloatArray random IntArray rSeqFA 800 800 1000000
setPixelXYZ x-400 y-400 z-400 0
rotatescene
```

`setFogRange` *start end*

command to set the range parameter of the OpenGL fog. *start* and *end* are numbers in the range of visible coordinates, typically *start* is 200..-400, *end* is 0..1000.

Graphics / Fog / setFogRange

Example:

```
setFogRange 0 300
```

setFogColor *color*

command to set the fog *color* to the *color* argument, which must be a valid *color* . See also `setPenColor!`

Example:

```
setFogColor "white"
```

setFogMode *mode*

command to set the fog *mode* . It is recommended to read on the OpenGL docs about the fog equations.

The following constants can be used as *mode* :

```
GL_LINEAR  
GL_EXP  
GL_EXP2
```

Linear fog is currently selected on every call to perspective.

Example:

```
cs perspective
setFogDensity 0.001
setFogColor "white"
setFogMode GL_EXP
enableFog
x=FloatArray random IntArray rSeqFA 800 800 1000000
y=FloatArray random IntArray rSeqFA 400 400 1000000
z=FloatArray random IntArray rSeqFA 800 800 1000000
setPixelXYZ x-400 y-400 z-400 0
```

Pictures

Here are commands to save and load graphics.

Pictures

- `savePicture` 275, `savePic` 275
- `loadPicture` 275, `loadPic` 275
- `savePictureText` 275
- `loadPictureText` 276
- `savePostScript` 276, `savePS` 276
- `saveScreen` 276
- `saveScreenVector` 277
- `saveSize` 279
- `setSaveSize` 279

`savePicture` *filename*

`savePic` *filename*

command. Writes a file with the specified name containing the state of the graphics window in Logo's internal format. This picture can be restored to the screen using `loadPic`. The format is not portable between platforms, nor is it easily readable by other programs. See `savePS` or `savePictureText` to export Logo graphics for other programs.

`loadPicture` *filename*

`loadPic` *filename*

command. Reads the specified file, which must have been written by a `savePic` command, and restores the graphics window and color palette settings to the values stored in the file. Any drawing previously on the screen is cleared.

`savePictureText` *filename*

command. Writes a file with the specified name containing the state of the graphics window in text format. This picture can be restored to the screen using loadPictureText. The format is portable between platforms, and it is easily readable by other programs. See savePicture for a faster but non-portable method.

loadPictureText *filename*

command. Reads the specified file, which must have been written by a savePictureText command or a compatible generator, and restores the graphics window to the values stored in the file. Any drawing previously on the screen is cleared. See loadPicture for a faster but non-portable method.

savePostScript *filename*

savePS *filename*

does not work well yet!

The new primitive saveScreenVector works better.

command. Writes a file with the specified name, containing an Encapsulated Postscript (EPS) representation of the state of the graphics window. This file can be imported into other programs that understand EPS format. Restrictions: the drawing cannot use ARC, FILL, PENERASE, or PENREVERSE; any such instructions will be ignored in the translation to Postscript form.

saveScreen *filename*

command. Saves the graphics screen to the image file *filename* . The size of the resulting image depends on the SaveSize. High resolution saving is currently only supported under Windows.

Here's the list of available picture-saving file formats: .bmp .png .jpeg .pcx .pnm .tiff .xpm .ico .cur

Example:


```
setSaveSize [400 300]
cs
crbox
boundingbox
rt 30
saveScreen "test.png"
```

saveScreenVector *filename*
(saveScreenVector *filename sort options colornr*)

Command to save the currently active Graph in a vector graphics format, which will be determined by the file extension.

The used library GL2PS supports six formats so far:

PS The output stream will be in PostScript format.

EPS The output stream will be in Encapsulated PostScript format.

PDF The output stream will be in Portable Document Format.

TEX The output will be a LATEX file containing only the text strings of the plot (cf. section 2.2), as well as an `\includegraphics` command including a graphic file having the same basename as *filename*.1

GL2PS_SVG (Experimental) The output stream will be in Scalar Vector Graphics format.

GL2PS_PGF (Experimental) The output stream will be in Portable LaTeX Graphics format.

sort Specifies the sorting algorithm, chosen among:

GL2PS_NO_SORT The primitives are not sorted, and are output in stream in the order they appear in the feedback buffer. This is sufficient for two-dimensional scenes.

GL2PS_SIMPLE_SORT The primitives are sorted according to their barycenter. This can be

sufficient for simple three-dimensional scenes and/or when correctness is not crucial.

GL2PS_BSP_SORT The primitives are inserted in a Binary Space Partition (BSP) tree. The tree is then traversed back to front in a painter-like algorithm. This should be used whenever an accurate rendering of a three-dimensional scene is sought. Beware that this algorithm requires a lot more computational time (and memory) than the simple barycentric *sort*.

options Sets global plot *options*, chosen among (multiple *options* can be combined with the bitwise inclusive OR symbol |):

GL2PS_NONE No option.

GL2PS_DRAW_BACKGROUND The background frame is drawn in the plot.

GL2PS_SIMPLE_LINE_OFFSET A small offset is added in the z-buffer to all the lines in the plot. This is a simplified version of the **GL2PS_POLYGON_OFFSET_FILL** functionality (cf. section 2.4), putting all the lines of the rendered image slightly in front of their actual position. This thus performs a simple anti-aliasing solution, e.g. for finiteelement-like meshes.

GL2PS_SILENT All the messages written by GL2PS on the error stream are suppressed.

GL2PS_BEST_ROOT The construction of the BSP tree is optimized by choosing the root primitives leading to the minimum number of splits.

GL2PS_NO_TEXT All the text strings are suppressed from the output stream. This is useful to produce the image part of a LATEX plot.

GL2PS_NO_PIXMAP All the pixmaps are suppressed from the output stream.

GL2PS_LANDSCAPE The plot is output in landscape orientation instead of portrait.

GL2PS_NO_PS3_SHADING (for PostScript output only) No use is made of the `shfill` PostScript level 3 operator. Using `shfill` enhances the plotting of smooth shaded primitives but can lead to problems when converting PostScript files into PDF files. See also *options colornr* below.

GL2PS_NO_BLENDING Blending (transparency) is disabled altogether (regardless of the current **GL_BLEND** or **GL2PS_BLEND** status).

GL2PS_OCCLUSION_CULL All the hidden polygons are removed from the output, thus

Graphics / Pictures / saveScreenVector

substantially reducing the size of the output file.

GL2PS_USE_CURRENT_VIEWPORT The current OpenGL viewport is used instead of viewport.

GL2PS_TIGHT_BOUNDING_BOX The viewport is ignored and the the plot is generated with a tight bounding box, i.e., a bounding box enclosing as tightly as possible all the OpenGL entities in the scene.

colornr (for PostScript output only) Controls the number of flat-shaded (sub-)triangles used to approximate a smooth-shaded triangle when the shfill operator is not supported by the system, or when the GL2PS_NO_PS3_SHADING option is set. The argument *colornr* specify the number of values used for interpolating the full range of red, green and blue color components; that is, a triangle is recursively subdivided until the color difference between two of its vertices is smaller that $1/nr$ for the red component, $1/ng$ for the green component and $1/nb$ for the blue component. If the arguments are set to zero, default values are used.

Examples:

```
spielbrett
saveScreenVector "test.ps
saveScreenVector "test.eps
(saveScreenVector "test.pdf GL2PS_NO_SORT
    GL2PS_SIMPLE_LINE_OFFSET+GL2PS_DRAW_BACKGROUND)
```

sizelist **saveSize**

outputs the two element list sizelist containing the width and height of the virtual graphics canvas which will be used to saveScreen. It is [800 600] by default.

```
saveSize      ; [2000 1500]      ; -)
```

setSaveSize *sizeList*

command setting the *sizeList* containing width and height of the of the virtual graphics canvas which will be used to saveScreen.

Example:

```
tree  
setSaveSize [4000 3000]  
saveScreen "temp.bmp" ;then you have a highres tree image
```

Bitmaps

A Bitmap is an internal logo node type which can hold a RGBA bitmap of some specified width and height.

Bitmaps

- BitCopy 281
- BitPaste 281
- loadImage 282
- saveImage 282
- BitMakeTransparent 283, BitTrans 283
- BitSetPixel 283
- BitPixel 284
- BitMaxX 284
- BitMaxY 284

bitmap **BitCopy** *width height*

outputs a bitmap which can be assigned to a variable and again pasted to the screen with bitPaste.

Into the bitmap will be copied the part of the graphics screen right up from the turtle with *width* and *height* . See also the bitmapTest.lg example!

Example:

```
tree
b=BitCopy 200 200
bk 200
BitPaste b
```

BitPaste *bitmap*

command which pastes the *bitmap* with lower left corner at the turtle to the screen.

Example:

```
tree
b=BitCopy 200 200
bk 200
BitPaste b
```

bitmap **loadImage** *filename*

outputs a bitmap loaded from the file *filename* . File formats currently supported are:

```
wxBMPHandler   For loading and saving.
wxPNGHandler   For loading (including alpha support) and saving.
wxJPEGHandler  For loading and saving.
wxGIFHandler   Only for loading, due to legal issues.
wxPCXHandler   For loading and saving.
wxPNMHandler   For loading and saving.
wxTIFFHandler  For loading and saving.
wxIFFHandler   For loading only.
wxXPMHandler   For loading and saving.
wxICOHandler   For loading and saving.
wxCURHandler   For loading and saving.
wxANIHandler   For loading only.
```

Loading PNMs only works for ASCII or raw RGB images.

Example:

```
b=loadImage "iris.png"
BitPaste b
```

saveImage *bitmap filename*

Graphics / Bitmaps / saveImage

command saving the *bitmap* to a file. File formats currently supported are:

wxBMPHandler	For loading and saving.
wxPNGHandler	For loading (including alpha support) and saving.
wxJPEGHandler	For loading and saving.
wxGIFHandler	Only for loading, due to legal issues.
wxPCXHandler	For loading and saving.
wxPNMHandler	For loading and saving.
wxTIFFHandler	For loading and saving.
wxIFFHandler	For loading only.
wxXPMHandler	For loading and saving.
wxICOHandler	For loading and saving.
wxCURHandler	For loading and saving.
wxANIHandler	For loading only.

When saving in PCX format, wxPCXHandler will count the number of different colours in the image; if there are 256 or less colours, it will save as 8 bit, else it will save as 24 bit.

When saving in PNM format, wxPNMHandler will always save as raw RGB.

Example:

```
b=loadImage "iris.png"
saveImage b "temp.png"
```

BitMakeTransparent *bitmap color*

BitTrans *bitmap color*

command which sets the alpha value of the *color* in the *bitmap* to 0, so this *color* will be transparent on BitPaste-ing.

Example:

```
b=loadImage "iris.png"
BitTrans b rgb 1 1 1
fd 100
BitPaste b
```

BitSetPixel *bitmap x y color*

sets the pixel at position (*x y*) of the *bitmap* to *color* . For an example see throwcoin.lg.

BitPixel *bitmap x y*

outputs the color at position (*x y*) of the *bitmap* . For an example see throwcoin.lg.

BitMaxX *abitmap*

outputs the maximal X coordinate of the bitmap *abitmap* for usage with the BitSetPixel and BitPixel primitives.

Example:

```
b=BitCopy 100 100
repeat BitMaxY b [
  BitSetPixel b BitMaxX b repcount "red
]
BitPaste b
```

BitMaxY *abitmap*

outputs the maximal Y coordinate of the bitmap *abitmap* for usage with the BitSetPixel and BitPixel primitives.

Example:

Graphics / Bitmaps / BitMaxY

```
b=BitCopy 100 100
repeat BitMaxX b [
  BitSetPixel b repcount BitMaxY b "red
]
BitPaste b
```

Direct Graphics

These are primitives for direct graphics without using a turtle, which is sometimes more convenient for simulations and it is faster, too.

Also here are operations for working with colors.

Direct Graphics

- `setPixel` 286
- `setPixelXY` 287
- `setPixelXYZ` 287
- `Line` 288
- `RGB` 289
- `RGBA` 289
- `reRGB` 289
- `reRGBA` 290
- `HSB` 290
- `HSBA` 290
- `reHSB` 291
- `reHSBA` 291
- `addColors` 292
- `addColorsMod` 292
- `getColorDatabase` 293

setPixel *coordinate* (s) color(s)

sets the pixels at the *coordinate* (s) to the relating color(s). `setPixel` can now also take 3D coordinates as arguments. It is also possible to use a `FloatArray` of length 3 for the coordinates.

Examples:

Graphics / Direct Graphics / setPixel

```

setPixel [10 10] 0
setPixel [[10 10][20 10]] 0
setPixel [[10 10][20 10]] [12 14]
perspective
setPixel {[10 10 10][20 10 0]} {12 14}
setPixel FloatArray {100 2 0} 0

```

setPixelXY *Xcoors Ycoors color* (s)

For easier and faster plotting the new command has been written: It takes structured X data and equally structured Y data as separate arguments. It's not really complete and has some bugs, to compensate those you should cast your structures to int or float, like in the example. The type FloatArray is allowed for *Xcoors* and *Ycoors*, together with IntArray as colors, so it's fast. The *Ycoors* can also be just one number while the must be of type Array, so you can draw your Graph scanlinewise.

Examples:

```

setPixelXY int [0 1 2 3 4] int [100 98 96 94 92] 0
x=rSeqFA -20 20 400
y=x*x
c=mod (IntArray iSeq 1 count x) 16
setPixelXY x*20 y c
cs
x=Array rSeq -20 20 400
y=x*x
c=mod (Array iSeq 1 count x) 16
repeat 300 ~
[   setPixelXY x*20 y*#/300 c
    c=rotate c 1
]

```

setPixelXYZ *Xcoors Ycoors Zcoors color* (s)

For easier and faster plotting the new command has been written: It takes structured X data and equally structured Y and Z data as separate arguments. The type FloatArray is allowed for *Xcoors* and *Ycoors* , together with IntArray as colors, so it's fast.

Examples:

```
cs perspective
x=int [0 1 2 3 4]
y=int [100 98 96 94 92]
z=int [10 20 30 40 50]
setPixelXYZ x y z 0
rotatescene
```

```
cs
z=rSeqFA -100 100 5000
x=cos z/10*360
y=sin z/10*360
c=mod (IntArray iSeq 1 count x) 16
setPixelXYZ x*z y*z z*5 c
rotatescene
```

```
cs
z=Array rSeq -100 100 5000
x=cos z/10*360
y=sin z/10*360
c=mod (Array iSeq 1 count x) 16
repeat 10 ~
[   setPixelXYZ x*z y*z z*z*#/10 c
    c=rotate c 1
]
rotatescene
```

Line *coordinates color* (s)

draws lines between the *coordinates* in the respective *color* (s). The first *color* in a list or array of colors is ignored. There is a bug when using multicolors if the turtle is shown, so `hideTurtle` before line.

Graphics / Direct Graphics / Line

Examples:

```
Line [[0 0][100 0][100 100]] 0
ht Line [[0 0][100 0][100 100]] (list rgb 1 0 0  rgb 0 1 0  rgb 0
0 1) st
```

RGB *red green blue*

outputs an integer color of the format *red*:8 *green*:8 *blue*:8 bits from the three parameters between 0 and 1.

Example:

```
setPC RGB 1 0 0  pd  fd 100 ;draws a  red  line
```

RGBA *red green blue alpha*

outputs an integer color of the format

red:8 *green*:8 *blue*:8 *alpha*:8 bits

from the four parameters between 0 and 1. Alpha is used for transparency - 0 is transparent.

Examples:

```
hex RGBA 0.5 0.25 0 1      ;FF003F7F  ;- )
cs
repeat 200 ~
[   setPC RGBA 0 0 0 repCount/200
    box
    rt 30 fd 1 lt 30
]
```

reRGB *color*

outputs a list of the three numbers for red, green and blue of the *color* .

Example:

```
reRGB RGB 1 0 0 ;[1 0 0] ; -)
```

reRGBA *color*

outputs a list of the four numbers between 0 and 1 for red, green, blue and alpha of the *color* .

Example:

```
reRGBA RGBA 1 0.5 0.2 0.1 ;[1 0.498039 0.2 0.0980392] ; -)
```

HSB *hue saturation brightness*

outputs an integer color of the format red:8 green:8 blue:8 bits from *hue* between 0 and 360 (*hue* =0 is red, *hue* =120 is green, *hue* =240 is blue) and *saturation* and *brightness* between 0 and 1.

Example:

```
setPC HSB 120 1 1 pd fd 100 ;draws a green line
```

HSBA *hue saturation brightness*

outputs an integer color of the format

red:8 green:8 blue:8 alpha:8 bits

from *hue* between 0 and 360 (*hue* =0 is red, *hue* =120 is green, *hue* =240 is blue) and *saturation* and *brightness* between 0 and 1.

Graphics / Direct Graphics / HSBA

Alpha is used for transparency - 0 is transparent.

Example:

```
cS
repeat 200 [
  setPC HSBA 360*repcount/200 1 1 1-repCount/200
  box
  rt 30 fd 1 lt 30
]
```

reHSB *color*

outputs a list of three numbers, hue, saturation and brightness, which represent the *color* in the HSB *color* space.

You can now easily read the *color* database using `getColorDatabase`, convert all the colors to HSB, sort by hue, and write a nice HTML colortable, like in `colordb2html.lg`.

reHSB does not round the *color* values, but it can only work with integer colors using 8 bits per channel. If you need more exactness, use the Logo functions defined in `testrgb2hsb.lg`.

Example:

```
show reHSB HSB 83 0.9 0.75
;[83.0233 0.900524 0.74902]
```

reHSBA *color*

outputs a list of four numbers, hue, saturation, brightness and alpha, which represent the *color* in the HSB *color* space including an alpha channel.

Example:

```
show reHSBA HSBA 83 0.9 0.75 0.666
;[83.0233 0.900524 0.74902 0.662745]
```

resultingColor **addColors** *c1 c2*

outputs the resultingColor which consists of the two colors color1 and color2.

The colors are added channelwise (R,G,B,A) with saturation like in the following logo procedure, but faster:

```
to addColors_ ca cb
  local [ra rb r g b]
  ra=reRGB ca
  rb=reRGB cb
  r=ra.1+rb.1 if r>1 [r=1]
  g=ra.2+rb.2 if g>1 [g=1]
  b=ra.3+rb.3 if b>1 [b=1]
  a=ra.4+rb.4 if a>1 [a=1]
  output RGBA r g b a
end
```

```
reRGBA addColors RGB .8 0 1 RGB .3 .4 .5
;[1 0.4 1 1]
```

resultingColor **addColorsMod** *c1 c2*

outputs the resultingColor which consists of the two colors color1 and color2.

The colors are added channelwise (R,G,B,A) while keeping the channels in the allowed range by using mod, like in the following logo procedure, but faster:

Graphics / Direct Graphics / addColorsMod

```
to addColors_ ca cb
  local [ra rb r g b]
  ra=reRGBA ca
  rb=reRGBA cb
  r=mod ra.1+rb.1 1
  g=mod ra.2+rb.2 1
  b=mod ra.3+rb.3 1
  a=mod ra.4+rb.4 1
  output RGBA r g b a
end
```

Example:

```
reRGBA addColorsMod RGB .8 .9 1 RGB .3 .4 .5
;[0.0941176 0.294118 0.494118 1]
```

getColorDatabase

outputs a list of lists of color name, red, green and blue value of all available named colors in the color database.

Projection Matrix

Functions for work with the graphical projection matrix. So you can use a specific graphic at different positions and rotations. For an example see [spelltest.lg!](#)

Projection Matrix

- Matrix 294
 - setMatrix 295, setM 295
 - TurtleMatrix 295, TM 295
 - setTurtleMatrix 295, setTM 295
 - pushMatrix 296
 - popMatrix 296
 - IdentityMatrix 297, IDM 297
 - setIdentityMatrix 297, setIDM 297
-

Matrix

operation which outputs the current projection matrix for the graphics: This is

```
(list (list basex_x basex_y basex_z)
      (list basey_x basey_y basey_z)
      (list basez_x basez_y basez_z)
      (list center_x center_y center_z))
```

(the bases and center are internal variables not directly accessible by the user).

Examples:

Graphics / Projection Matrix / Matrix

```

setidm
cs
Matrix      ;[[1 0 0][0 1 0][0 0 1][0 0 0]] ;-)
fd 100
rt 30
setM TM
cs
rbox
Matrix
;[[0.866025 -0.5 0][0.5 0.866025 0][0 0 1][0 100 0]] ;-)

```

setMatrix *aMatrix*

setM *aMatrix*

command which sets the current OpenGL modelview matrix to matrix.

Example:

```

tree
Matrix      ;[[1 0 0][0 1 0][0 0 1][0 0 0]] ;-)
setMatrix Matrix*0.5
tree

```

TurtleMatrix

TM

operation which outputs the matrix of the turtle, ready for use with setmatrix.

Example:

```

setMatrix TM

```

setTurtleMatrix

setTM

command which sets the turtle matrix as the new projection matrix.

See also the [spelltest.lg](..\spelltest.lg) example!

Example:

```
rt 30
fd 100
setTM
pu Home pd
```

pushMatrix

command which pushes the current modelview matrix onto the matrix stack of OpenGL.

Example:

```
rt 30
fd 100
pushmatrix
setTM
popmatrix
```

popMatrix

command popping the latest pushed modelview matrix from the OpenGL matrix stack.

Example:

Graphics / Projection Matrix / popMatrix

```
rt 30  
fd 100  
pushmatrix  
setTM  
popmatrix
```

IdentityMatrix
IDM

operation which outputs the identity matrix.

Example:

```
setMatrix IDM
```

setIdentityMatrix
setIDM

command which sets the projection matrix to the identity.

Texturing

...is a cool feature of OpenGL to decorate filled shapes with a bitmap.

Texturing

- Texture 298
- setTexXY 299
- setTexPos 299
- enableTexture 300, enTex 300
- disableTexture 300, disTex 300
- deleteTextures 300

texhandle **Texture** *bitmap*

Texture *texhandle*

Texture []

Command or operation, depending on the argument.

Sets the momentary texture to *bitmap* if the argument is a *bitmap* and outputs a *texhandle* integer, by which the texture can be selected faster.

If given a *texhandle* (a integer output from an earlier call of Texture), the respective texture is selected.

Textures only are applied when enableTexture is called once.

Example:

Graphics / Texturing / Texture

```

b=loadImage "iris.png texhandle =Texture b
enTex
setPC 4
(tcube 400)
cs
enTex
Texture []
Texture texhandle
(tcube 400)

```

setTexXY *x y*

sets the OpenGL texture coordinates to (x, y) . x and y should be in the range of 0..1 for displaying only one occurrence of the texture. If the values are greater the texture is repeated.

This primitive is useful with PolyStart..PolyEnd and TessStart..TessEnd.

Example:

```

entex
bricks=loadImage "bricks.png
tex=texture bricks
setpc "white
PolyStart
setTexXY 0 1 fd 100 rt 90
setTexXY 1 1 fd 100 rt 90
setTexXY 1 0 fd 100 rt 90
setTexXY 0 0 fd 100 rt 90
PolyEnd

```

setTexPos *xylist*

sets the OpenGL texture coordinates to $[x\ y]$. x and y should be in the range of 0..1 for displaying only one occurrence of the texture. If the values are greater the texture is repeated.

This primitive is useful with PolyStart..PolyEnd and TessStart..TessEnd.

Examples:

```
setTexPos [0.2 0.9]  
x=0.2 y=0.9  
setTexPos list x y
```

enableTexture **enTex**

enables the texturing of polygonal surfaces.

See also: Texture.

Example:

```
entex  
bricks=loadImage "bricks.png"  
tex=texture bricks  
setfc "white"  
fillRect [-100 -100][100 100]
```

disableTexture **disTex**

disables the texturing of polygonal surfaces.

See also: Texture.

deleteTextures

command which serves as a possible cleanup of texture memory, so a texture memory overflow can be prevented, if you create many textures which you maybe only use once or a few times, after which they are obsolete.

A good example for this command is candle.lg.

Shadows

Shadow casting is a difficult-to-implement feature, but it's now available in aUCBLogo-4.69. It still has some bugs, which you can see as graphic noise or artefacts. I don't know how to completely remove this noise until now. Also cylinderlines wont cast a shadow yet. But you can use SurfaceStart..SurfaceColumn..SurfaceEnd, PolyStart..PolyEnd, TessStart..TessContour..TessEnd, Sphere and partialEllipsoid, which will cast shadows.

Shadows

- enableShadows 302
- disableShadows 302
- castShadows 303
- clearShadows 303
- setShadowColor 303
- ShadowColor 304

enableShadows

command to set the internal flag to enable shadow creating. It must be set before you draw graphics, if you want to see any shadows.

Additionally you must either call the primitive castShadows maybe followed by updateGraph, or redraw to produce the shadows.

Example:

```
cs perspective window
enableShadows
Sphere 100
setPenColor "green
back 1000 sphere 800
castShadows
```

disableShadows

command to reset the internal flag to disable shadow creating.

castShadows

command to draw the shadows which have been constructed by SurfaceStart..SurfaceColumn..SurfaceEnd, PolyStart..PolyEnd, TessStart..TessContour..TessEnd, Sphere and partialEllipsoid, after a call to enableShadows.

If you can't see the shadow of a specific surface, it is likely that you must draw it with the opposite normal. This is easy to accomplish by drawing reversly, so if you had used rt, use lt instead, then it should work.

Example:

```
perspective cs clearshadows
enableShadows
setpc "blue
pcube
back 1000
setpc "green
sphere 900
castshadows
rotatescene
```

clearShadows

command to clear the shadow objects buffer. This can be necessary if you want to draw something completely different in a new scene. I did not make clearScreen clear the shadow objects, because it might be necessary to be able to redraw the scene using a different viewpoint, and this is much faster than recreating the whole scene.

setShadowColor *acolor*

command to set the color of future shadows to *acolor* . You can use all color formats that setPenColor supports.

Example:

```
perspective cs clearshadows
enableShadows
setpc "blue
pcube
back 1000
setpc "green
sphere 900
setShadowColor "red
castshadows
rotatescene
```

ShadowColor

outputs the currently selected color for the next shadows which you draw with castShadows.

Workspace Management

Workspace Management

- Procedure Definition 306
 - Variable Definition 312
 - Property Lists 315
 - Workspace Predicates 318
 - Workspace Queries 321
 - Inspection 328
 - Workspace Control 334
 - Editing 347
 - Erasing 350
-

Procedure Definition

Procedure Definition

- to 306
- define 308
- Text 309
- fullText 310
- copyDef 310

to *procname* :*input1* :*input2* ... (special form)

command. Prepares Logo to accept a procedure definition. The procedure will be named "*procname*" and there must not already be a procedure by that name. The inputs will be called "input1" etc. Any number of inputs are allowed, including none. Names of procedures and inputs are case-insensitive.

Unlike every other Logo procedure, TO takes as its inputs the actual words typed in the instruction line, as if they were all quoted, rather than the results of evaluating expressions to provide the inputs. (That's what "special form" means.)

This version of Logo allows variable numbers of inputs to a procedure. Every procedure has a MINIMUM, DEFAULT, and MAXIMUM number of inputs. (The latter can be infinite.)

The MINIMUM number of inputs is the number of required inputs, which must come first. A required input is indicated by the

:inputname
notation.

Examples:

```
to go
  fd 100
end
```

```
to printabc :a :b :c
  (pr :a :b :c)
end
```

As a test you can type "go", which will move the turtle 100 forward. When you type "printabc 1 2 3", printabc will be invoked and will print 1 2 3.

After all the required inputs can be zero or more optional inputs, represented by the following notation:

```
[ :inputname default_value_expression ]
```

When the procedure is invoked, if actual inputs are not supplied for these optional inputs, the default value expressions are evaluated to set values for the corresponding input names. The inputs are processed from left to right, so a default value expression can be based on earlier inputs.

Examples:

```
to proc :inlist [:startvalue first :inlist]
;...some code here
end
```

If the procedure is invoked by saying

```
proc [a b c]
```

then the variable INLIST will have the value [A B C] and the variable STARTVALUE will have the value A. If the procedure is invoked by saying

```
(proc [a b c] "x")
```

then INLIST will have the value [A B C] and STARTVALUE will have the value X.

Example:

```
to go [:steps 100]
  fd :steps
end
```

You can then type "go" which will do "fd 100", or you can write "(go 10)" which will do "fd 10".

After all the required and optional input can come a single "rest" input, represented by the following notation:

```
[ :inputname ]
```

This is a rest input rather than an optional input because there is no default value expression. There can be at most one rest input. When the procedure is invoked, the value of this input will be a list

containing all of the actual inputs provided that were not used for required or optional inputs.

Example:

```
to proc :in1 [:in2 "foo"] [:in3]  
;...some code here  
end
```

If this procedure is invoked by saying

```
proc "x
```

then IN1 has the value X, IN2 has the value FOO, and IN3 has the value [] (the empty list). If it's invoked by saying

```
(proc "a "b "c "d)
```

then IN1 has the value A, IN2 has the value B, and IN3 has the value [C D].

The MAXIMUM number of inputs for a procedure is infinite if a rest input is given; otherwise, it is the number of required inputs plus the number of optional inputs.

The DEFAULT number of inputs for a procedure, which is the number of inputs that it will accept if its invocation is not enclosed in parentheses, is ordinarily equal to the minimum number. If you want a different default number you can indicate that by putting the desired default number as the last thing on the TO line.

Example:

```
to proc :in1 [:in2 "foo"] [:in3] 3  
;...some code here  
end
```

This procedure has a minimum of one input, a default of three inputs, and an infinite maximum.

Logo responds to the TO command by entering procedure definition mode. The prompt character changes from "?" to ">" and whatever instructions you type become part of the definition until you type a line containing only the word END.

define *procname text*

Workspace Management / Procedure Definition / define

command. Defines a procedure with name "*procname*" and *text* "*text*". If there is already a procedure with the same name, the new definition replaces the old one. The *text* input must be a list whose members are lists. The first member is a list of inputs; it looks like a TO line but without the word TO, without the procedure name, and without the colons before input names. In other words, the members of this first sublist are words for the names of required inputs and lists for the names of optional or rest inputs. The remaining sublists of the *text* input make up the body of the procedure, with one sublist for each instruction line of the body. (There is no END line in the *text* input.) It is an error to redefine a primitive procedure unless the variable REDEFP has the value TRUE.

Examples:

```
define "go [[] [fd 100]]
go
cs
define "go [[steps] [fd 300*steps]]           ;so go is a "scaled fd"
go .5
go .25
go .125
cs
define "go [[[steps 1]] [fd 300*steps]]
go      ;moves the turtle to the upper border, drawing
cs
(go 0.5)      ;moves the turtle till half the upper border
```

Text *procname*

outputs the text of the procedure named "*procname*" in the form expected by DEFINE: a list of lists, the first of which describes the inputs to the procedure and the rest of which are the lines of its body. The text does not reflect formatting information used when the procedure was defined, such as continuation lines and extra spaces.

Example:

```
hex pc      ;FF000000  ;-)
Text "hex
  [      [num]
    [op intForm :num 8 16]]
  ;-)
```

fullText *procname*

outputs a representation of the procedure "*procname*" in which formatting information is preserved. If the procedure was defined with TO, EDIT, or LOAD, then the output is a list of words. Each word represents one entire line of the definition in the form output by READWORD, including extra spaces and continuation lines. The last member of the output represents the END line. If the procedure was defined with DEFINE, then the output is a list of lists. If these lists are printed, one per line, the result will look like a definition using TO. Note: the output from FULLTEXT is not suitable for use as input to DEFINE!

Example:

```
hex pc      ;FF000000  ;-)
fullText "hex
  [to\ hex\ \:num \    op\ intForm\ \:num\ 8\ 16 end]
  ;-)
```

copyDef *newname oldname*

command. Makes "*newname*" a procedure identical to "*oldname*". The latter may be a primitive. If "*newname*" was already defined, its previous definition is lost. If "*newname*" was already a primitive, the redefinition is not permitted unless the variable REDEFP has the value TRUE. Definitions created by COPYDEF are not saved by SAVE; primitives are never saved, and user-defined procedures created by COPYDEF are buried. (You are likely to be confused if you PO or POT a procedure defined with COPYDEF because its title line will contain the old name. This is why it's buried.)

Note: dialects of Logo differ as to the order of inputs to COPYDEF. This dialect uses "MAKE

order," not "NAME order."

Example:

```
copyDef "write "print  
write 1234      ;1234
```

Variable Definition

Variable Definition

- make 312
- name 312
- local 313
- localmake 313
- Thing 314

make *varname value*

varname = value

command. Assigns the *value* " *value* " to the variable named " *varname* ", which must be a word. Variable names are case-insensitive. If a variable with the same name already exists, the *value* of that variable is changed. If not, a new global variable is created.

Examples:

```
make "a 1234
:a      ;1234  ;-)
a=2345
a      ;2345  ;-)
a=[Hallo World!]
pr a      ;Hallo World!
```

name *value varname* (library procedure)

command. Same as MAKE but with the inputs in reverse order.

Example:

Workspace Management / Variable Definition / `name`

```
name 1234 "a
a      ;1234  ;-
```

I think one assignment primitive is enough, so I leave `name` as a library procedure.

local *varname*

local *varnamelist*

(local *varname1 varname2 ...*)

`command`. Accepts as inputs one or more words, or a list of words. A variable is created for each of these words, with that word as its name. The variables are local to the currently running procedure. Logo variables follow dynamic scope rules; a variable that is local to a procedure is available to any subprocedure invoked by that procedure. The variables created by `LOCAL` have no initial value; they must be assigned a value (e.g., with `MAKE`) before the procedure attempts to read their value.

Examples:

```
to proc
  local [a b c]
  a=1234
  b=[Hallo World!]
  c={1 2 3}
  (pr a b c)
end
;proc defined
proc      ;1234 Hallo World! {1 2 3}
a      ; I don't know how to a
b      ; I don't know how to b
c      ; I don't know how to c
```

localmake *varname value* (library procedure)

`command`. Makes the named variable local, like `LOCAL`, and assigns it the given value, like

MAKE.

Example:

```
to proc
  localmake "a 1234
  pr a
end
;proc defined
proc      ;1234
a      ; I don't know how to a
```

I think it's better to separate the declaration from the initialization and use local and make rather than localmake.

Because of this I don't make localmake a primitive.

Thing *varname*

:quoted_varname

outputs the value of the variable whose name is the input. If there is more than one such variable, the innermost local variable of that name is chosen. The colon notation is an abbreviation not for THING but for the combination

```
thing "
```

so that

```
:FOO means THING "FOO.
```

Examples:

```
a=1234
Thing "a      ;1234  ;-) here Thing "  is not really neccessary.
ab=[Hallo World!]
Thing Word "a "b      ;[Hallo World!] ;-)
;this second example is the strength of thing!
```

Property Lists

...can now fully be replaced by Table, using Item, setItem and removeItem. The PropertyList functions are left here for compatibility only.

Note: Names of property lists are always case-insensitive. Names of individual properties are case-sensitive or case-insensitive depending on the value of CaseIgnoredP, which is true by default.

Examples for PropertyList usage:

```
pProp "myplist" "a" 1234
pProp "myplist" "b" 5678
PList "myplist"      ;[b 5678 a 1234] ;-)
gProp "myplist" "a"    ;1234 ;-)
remProp "myplist" "a"
PList "myplist"      ;[b 5678] ;-)
```

Property Lists

- putProperty 315, pProp 315
- getProperty 316, gProp 316
- removeProperty 316, remProp 316
- PropertyList 316, PList 316

putProperty *plistname propname value*
pProp *plistname propname value*

command. Adds a property to the " *plistname* " property list with name " *propname* " and *value* " *value* ".

Example:

```
pProp "myplist" "a 1234"
PList "myplist"      "[a 1234]" ; -)
```

getProperty *plistname* *propname*
gProp *plistname* *propname*

outputs the value of the "*propname*" property in the "*plistname*" property list, or the empty list if there is no such property.

Example:

```
pProp "myplist" "a 1234"
gProp "myplist" "a"      ;1234 ; -)
```

removeProperty *plistname* *propname*
remProp *plistname* *propname*

command. Removes the property named "*propname*" from the property list named "*plistname*".

Example:

```
pProp "myplist" "a 1234"
remProp "myplist" "a"
PList "myplist"      "[]" ; -)
```

PropertyList *plistname*
PList *plistname*

outputs a list whose odd-numbered members are the names, and whose even-numbered members are the values, of the properties in the property list named "*plistname*". The output is a copy of the

actual property list; changing properties later will not magically change a list output earlier by PLIST.

Example:

```
pProp "myplist" "a 1234"
PList "myplist"    ;[a 1234] ;-
```

Workspace Predicates

...ask boolean questions on root table entries, i.e. if something is a procedure, a primitive or a variable.

Workspace Predicates

- ProcedureP 318
- PrimitiveP 318
- definedP 319
- NameP 319
- CaseIgnoredP 319

ProcedureP *name*

Procedure? *name*

outputs TRUE if the input is the *name* of a procedure. Logo does not know a procedure which has not yet been loaded. It only knows the primitives at startup.

Examples:

```
Procedure? "fd      ;true  ;-)
Procedure? "tree    ;false  ;-)
tree
Procedure? "tree    ;true   ;-)
```

PrimitiveP *name*

Primitive? *name*

outputs TRUE if the input is the *name* of a primitive procedure (one built into Logo). Note that some of the procedures described in this document are library procedures, not primitives.

Examples:

```
Primitive? "fd      ;true  ;-)
tree
Primitive? "tree    ;false  ;-)
```

definedP *name*

defined? *name*

outputs TRUE if the input is the *name* of a user-defined procedure, including a library procedure. (However, Logo does not know about a library procedure until that procedure has been invoked.)

Examples:

```
defined? "fd      ;false  ;-)
defined? "tree    ;false  ;-)
tree
defined? "tree    ;true   ;-)
```

NameP *name*

Name? *name*

outputs TRUE if the input is the *name* of a variable.

Examples:

```
make "a 1234
Name? "a      ;true   ;-)
b=[something]
Name? "b      ;true   ;-)
```

CaseIgnoredP

CaseIgnored?

if TRUE, indicates that lower case and upper case letters should be considered equal by EQUALP, BEFOREP, MEMBERP, etc. Logo initially sets this internal state TRUE.

Example:

```
setCaseIgnored false
CaseIgnored?          ;false  ;-)
bUtFirst "Hallo        ; I don't know how to bUtFirst
butFirst "World        ;orld   ;-)
butfirst [so so]       ;[so]   ;-)
```

Workspace Queries

...ask questions on the root table and its members.

Note: All procedures whose input is indicated as "contentslist" will accept a single word (taken as a procedure name), a list of words (taken as names of procedures), or a list of three lists as described under the CONTENTS command.

Workspace Queries

- Contents 321
- buried 322
- traced 322
- Primitives 323
- Procedures 323
- Names 323
- PropertyLists 324, PLists 324
- namelist 324
- pllist 325
- Arity 325
- Nodes 326

Contents

outputs a "contents list," i.e., a list of three lists containing names of defined procedures, variables, and property lists respectively. This list includes all unburied named items in the workspace.

Examples:

```
tree
contents      ;[[tree tree2][[]]] ;-)
a=1234
contents      ;[[tree tree2][a][[]] ;-)
pprop "b "greetings [Hallo World]
contents      ;[[tree tree2][a][b]] ;-)
reset
contents      ;[[[]][[]]] ;-)
```

buried

outputs a contents list including all buried named items in the workspace.

Examples:

```
buried
  [  []
    [-1 0 1 determinant false floatmax intmax pi true]
    []
  ]
;-)
foreach [Hallo World] [pr ?]
Hallo
World
buried
  [  [?rest foreach foreach1 foreach_done]
    [-1 0 1 determinant false floatmax intmax pi true]
    []
  ]
;-)
```

traced

outputs a contents list including all traced named items in the workspace.

Example:

```
trace "fd
traced
  [  [fd]
    []
    []
  ]
;-)
fd 100
( fd 100 )
fd stops
```

Primitives

outputs a list of the names of all primitives.

Examples:

```
show primitives      ;then you'll see the long list of primitives
count primitives     ;970  ;-)
count remdup lowercase primitives ;551  ;-)
```

Procedures

outputs a list of the names of all unburied user-defined procedures in the workspace. Note that this is a list of names, not a contents list. (However, procedures that require a contents list as input will accept this list.)

Examples:

```
Procedures      ;[]  ;-)
tree
Procedures      ;[tree tree2]  ;-)
```

Names

outputs a contents list consisting of an empty list (indicating no procedure names) followed by a list of all unburied variable names in the workspace.

Example:

```
a=1234
b=[Hallo World!]
Names      ;[[[] [a b]] ; -)
```

PropertyLists**PLists**

outputs a contents list consisting of two empty lists (indicating no procedures or variables) followed by a list of all unburied property lists in the workspace.

Example:

```
pProp "myplist" "a 1234"
pProp "myplist2" "a 4321"
PLists      ;[[[] [] [myplist myplist2]] ; -)
```

namelist *varname* (library procedure)

namelist *varnamelist*

outputs a contents list consisting of an empty list followed by a list of the name or names given as input. This is useful in conjunction with workspace control procedures that require a contents list as input.

Examples:


```
namelist "a
  [  []
    [a]
  ]
;-)
namelist [a b c]
  [  []
    [a b c]
  ]
;-)
```

plist *pname* (library procedure)

plist *plnamelist*

outputs a contents list consisting of two empty lists followed by a list of the name or names given as input. This is useful in conjunction with workspace control procedures that require a contents list as input.

Examples:

```
plist "a
  [  []
    []
    [a]
  ]
;-)
plist [a b c]
  [  []
    []
    [a b c]
  ]
;-)
```

Arity *procedurename*

outputs a list of four numbers: the minimum, default, and maximum number of inputs and the priority for the procedure whose name is the input. It is an error if there is no such procedure. A maximum of -1 means that the number of inputs is unlimited.

Examples:

```
arity "fd      ;[1 1 1 10] ;-)
arity "item    ;[2 2 3 10] ;-)
arity "sum     ;[0 2 -1 10] ;-)
arity "\+      ;[1 1 1 12] ;-)
arity "\*      ;[1 1 1 13] ;-)
arity "\^      ;[1 1 1 14] ;-)
arity "\- \-   ;[1 1 1 16] ;-)
```

Nodes

outputs a list of two numbers. The first represents the number of nodes of memory currently in use. The second shows the maximum number of nodes that have been in use at any time since the last invocation of NODES. (A node is a small block of computer memory as used by Logo. Each number uses one node. Each non-numeric word uses one node, plus some non-node memory for the characters in the word. Each array takes one node, plus some non-node memory, as well as the memory required by its elements. Each list requires one node per element, as well as the memory within the elements.)

If you want to track the memory use of an algorithm, it is best if you invoke GC at the beginning of each iteration, since otherwise the maximum will include storage that is unused but not yet collected.

Examples:

Workspace Management / Workspace Queries / Nodes

```
Nodes      ;[3105 3471] ;-)
Nodes      ;[3105 3125] ;-)
a=1234
Nodes      ;[3108 3143] ;-)
Nodes      ;[3108 3128] ;-)
erase [[]a]
Nodes      ;[3105 3145] ;-)
Nodes      ;[3105 3125] ;-)
```

Inspection

These are print out functions with many abbreviations.

Inspection

- `printOut` 328, `pO` 328
 - `poall` 329
 - `pops` 329
 - `pons` 330
 - `popls` 330
 - `pon` 331
 - `popl` 331
 - `printOutTitles` 332, `poT` 332
 - `pots` 332
-

`printOut` *contentslist*

`pO` *contentslist*

command. Prints to the write stream the definitions of all procedures, variables, and property lists named in the input contents list.

Example:

```
a=1234
to t
  pr a
end
;t defined
po [[t][a]]
```

which will print:

```
to t
  pr a
end
```

```
Make "a 1234
```

poall (library *procedure*)

command. Prints all unburied definitions in the workspace. Abbreviates PO CONTENTS.

Example:

```
a=1234
to t
  pr a
end
;t defined
po contents
```

which will print:

```
to t
  pr a
end
```

```
Make "a 1234
```

pops (library *procedure*)

command. Prints the definitions of all unburied procedures in the workspace. Abbreviates PO PROCEDURES.

Example:

```
to t
  pr 1234
end
t defined
pops
```

which will print:

```
to t
  pr 1234
end
```

pops (**library procedure**)

command. Prints the definitions of all unburied variables in the workspace. Abbreviates PO NAMES.

Example:

```
a=1234
b=[Hallo World!]
pops
```

which will print:

```
Make "a 1234
Make "b [Hallo World!]
```

popls (**library procedure**)

command. Prints the contents of all unburied property lists in the workspace. Abbreviates PO PLISTS.

Example:

```
pProp "pa "a 1234
pProp "pb "b [Hallo]
pProp "pf "d "Hallo
popls
```

which will print:

```
pProp "pa" "a 1234"
pProp "pb" "b [Hallo]"
pProp "pf" "d "hallo"
```

pon *varname* (library procedure)
pon *varnamelist*

command. Prints the definitions of the named variable(s). Abbreviates PO NAMELIST
 varname(list).

Examples:

```
a=1234
b=[Hallo World!]
pon "a"
```

will print:

```
Make "a 1234"
pon [a b]
```

will print:

```
Make "a 1234"
Make "b [Hallo World!]"
```

popl *pname* (library procedure)
popl *plnamelist*

command. Prints the definitions of the named property list(s). Abbreviates PO PLLIST
 pname(list).

Examples:

```
pProp "myplist" "a 1234"
pProp "myplistb" "b [Hallo World!]"
popl "myplist"
```

will print:

```
pProp "myplist" "a 1234"
popl [myplist myplistb]
```

will print:

```
pProp "myplist" "a 1234"
pProp "myplistb" "b [Hallo World!]"
```

printOutTitles *contentslist*
poT *contentslist*

command. Prints the title lines of the named procedures and the definitions of the named variables and property lists. For property lists, the entire list is shown on one line instead of as a series of PPROP instructions as in PO.

Example:

```
a=1234
to t
  pr 5
end
;t defined
pProp "myplist" "b 4321"
pot [[t][a][myplist]]
```

will print:

```
to t
Make "a 1234"
PList "myplist=b 4321"
```


pots (*library procedure*)

command. Prints the title lines of all unburied procedures in the workspace. Abbreviates POT PROCEDURES.

Example:

```
tree
pots
```

will print:

```
to tree [level 7][size 100]
to tree2 level size
```

Workspace Control

Workspace Control

- `setCaseIgnored` 334
- `bury` 335
- `buryall` 335
- `buryname` 336
- `unbury` 336
- `unburyall` 336
- `unburyname` 337
- `trace` 337
- `traceall` 338
- `untrace` 338
- `step` 338
- `stepall` 339
- `unstep` 339
- `save` 340
- `savel` 340
- `load` 341
- `help` 342
- `h` 343
- `GC` 344
- `shrinkStacks` 345
- `setStackNoisy` 345

`setCaseIgnored` *bool*

Sets the internal state of being case-sensitive. Set it to false if you want to discern upper and lower case symbols.

But be careful: Also the procedure and primitive names are then case-sensitive.

There are two case forms of each primitive, one completely lowercase, the other like in the help.

Workspace Management / Workspace Control / setCaseIgnored

Example:

```
setCaseIgnored false
clearscreen ;works!
clearScreen ;works!
ClearScreen
; I don't know how to ClearScreen
```

bury *contentslist*

command. Buries the procedures, variables, and property lists named in the input. A buried item is not included in the lists output by CONTENTS, PROCEDURES, VARIABLES, and PLISTS, but is included in the list output by BURIED. By implication, buried things are not printed by POALL or saved by SAVE.

Example:

```
to t
end
;t defined
Contents      ;[[t]][] [] ;-)
bury "t
Contents      ;[[] [] []] ;-)
buried
  [  [t]
    [-1 0 1 determinant false floatmax intmax pi true]
    []
  ]
;-)
```

buryall (library *procedure*)

command. Abbreviates BURY CONTENTS.

Example:

```
a=1234
tree
buryall
Contents      ;[[[]]][] ;-)
buried
  [  [buryall tree tree2]
    [-1 0 1 a determinant false floatmax intmax pi true]
    []
  ]
;-)
```

buryname *varname* (library procedure)

buryname *varnamelist*

command. Abbreviates BURY NAMELIST varname(list).

unbury *contentslist*

command. Unburies the procedures, variables, and property lists named in the input. That is, the named items will be returned to view in CONTENTS, etc.

Example:

```
a=1234
bury namelist "a
Contents      ;[[[]]][] ;-)
unbury namelist "a
Contents      ;[[[]a]][] ;-)
```

unburyall (library procedure)

Workspace Management / Workspace Control / unburyall

command. Abbreviates UNBURY BURIED.

Example:

```
a=1234
bury namelist "a
unburyall
Contents      ;[[namelist unburyall][a][[]] ;-)
```

unburyname *varname* (library procedure)

unburyname *varnamelist*

command. Abbreviates UNBURY NAMELIST varname(list).

Example:

```
a=1234
buryname "a
buried      ;[[buryname namelist][a][[]] ;-)
unburyname "a
Contents    ;[[[]][a][[]] ;-)
```

trace *contentslist*

command. Marks the named items for tracing. A message is printed whenever a traced procedure is invoked, giving the actual input values, and whenever a traced procedure STOPS or OUTPUTs. A message is printed whenever a new value is assigned to a traced variable using MAKE. A message is printed whenever a new property is given to a traced property list using PPROP.

Examples:

```
a=1234
trace namelist "a
a=4321;      Make "a 4321
```

```
load "tree.lg
trace "tree
tree
```

The second example will print a quite long transcript of tree invocations.

traceall (library *procedure*)

command. Abbreviates TRACE CONTENTS.

Example:

```
load "am.lg
traceall
am
```

will print a long transcript of the asteroid miner game.

untrace *contentslist*

command. Turns off tracing for the named items.

Example:

```
trace namelist "a
a=1234
Make "a 1234
untrace namelist "a
a=4321
```

step *contentslist*

Workspace Management / Workspace Control / step

command. Marks the named items for stepping. Whenever a stepped procedure is invoked, each instruction line in the procedure body is printed before being executed, and Logo waits for the user to type a newline at the terminal. A message is printed whenever a stepped variable name is "shadowed" because a local variable of the same name is created either as a procedure input or by the LOCAL command.

Examples:

```
step namelist "a
a=1234
to t
  local "a
  a=4321
  pr a
end
t defined
step "t
t
local "a >>>
; a shadowed by local in procedure call in t line 1
```

```
= "a 4321 >>>
pr :a >>>
4321
```

stepall (**library procedure**)

command. Abbreviates STEP CONTENTS.

Example:

```
load "bounce3.lg
stepall
bounce3
```

unstep *contentslist*

command. Turns off stepping for the named items.

Example:

```
load "bounce3.lg
stepall
unstep Contents
bounce3
```

save *filename*

command. Saves the definitions of all unburied procedures, variables, and property lists in the named file. Equivalent to

```
poall
setwriter :oldwriter
close : filename
end
```

Example:

```
load "si.lg
save "temp.txt
```

Now temp.txt contains the same definitions as si.lg.

savel *contentslist filename* (library procedure)

command. Saves the definitions of the procedures, variables, and property lists specified by "*contentslist*" to the file named "*filename*".

Example:


```
load "si.lg
contents      ;[[fractal image_right si triangle][[]]] ;-)
savel "triangle "temp.txt
```

Now temp.txt contains the definition of triangle.

load *filename*

command. Reads instructions from the named file and executes them. The file can include procedure definitions with TO, and these are accepted even if a procedure by the same name already exists. If the file assigns a list value to a variable named STARTUP, then that list is run as an instructionlist after the file is loaded.

Example:

```
load "bounce3.lg
pots
```

will print:

```
to ballinit
to bat
to batinit
to bounce3
to circ size
to courtinit
to drawBall
to drawExplosions
to explosioninit
to fractal size level
to image_right tilt
to moveBall
to moveBat
to myfrbox size
to noball
to nobat
to si
to smallfrbox size size2
to stonesinit
to triangle
```

help *name*
(help)

command. Prints information from the reference manual about the primitive procedure named by the input. With no input, lists all the primitives about which help is available. If there is an environment variable ALOGOHELP, then its value is taken as the directory in which to look for help files, instead of the default help directory.

Exceptionally, the HELP command can be used without its default input and without parentheses provided that nothing follows it on the instruction line.

Example:

```
help [word]
```

will print:

```
Help Contents / Data Structure Primitives / Constructors / Word  
Word word1 word2  
(Word word1 word2 word3 ...)
```

outputs a word formed by concatenating its inputs.

Examples:

```
show word "hal "lo      ;hallo  
show (word [wor ld])    ;wor ld  
show (word {a b c}) ;{a b c}  
show word "12 "34      ;1234
```

next: List

h *listOfSymbols* (library procedure)

h *symbolWord*

searches the help pages for matches of one or more of the symbols in the *listOfSymbols* or the *symbolWord* and either opens a web browser showing that topic or prints out h commands nearly matching the *listOfSymbols*.

Examples:

```
h [graphic]
Loading help index file c:\auchblog\help\indexarr.txt ...
I don't know " graphic " directly, therefore I'm searching the
semantics now...
h "bg      ; 789
h "loadpic ; 789
h "clearscreen ; 789
h "savepic   ; 938
h "clean    ; 938
h "right    ; 938
h "refresh  ; 1154
h "pc       ; 1154
h "label    ; 1154
h "background ; 1218
h "not      ; 1875
h "eps pict ; 2182
h "turtle-motion-queries ; 3750
h "fillrect ; 3750
h "hsb      ; 3750
h "turtle-&-window-queries ; 3750
h "line     ; 3750
h "turtle-&-window-control ; 3750
h "moveline ; 3750
h "rgb      ; 3750
h "movepixel ; 3750
h "rergb    ; 3750
h "pen-queries ; 3750
h "projection-matrix ; 3750
h "pictures  ; 3750
h "pen-&-background-control ; 4056
h "turtle-motion ; 4125
h "graphics   ; 5536
h "setpixel   ; 5893
h "direct-graphics ; 8750
```

GC

(GC *anything*)

Workspace Management / Workspace Control / GC

command. Runs the garbage collector, reclaiming unused nodes. Logo does this when necessary anyway, but you may want to use this command to control exactly when Logo does it. In particular, the numbers output by the NODES operation will not be very meaningful unless garbage has been collected. Another reason to use GC is that a garbage collection takes a noticeable fraction of a second, and you may want to schedule collections for times before or after some time-critical animation. If invoked with an input (of any value), GC runs a full garbage collection, including GCTWA (Garbage Collect Truly Worthless Atoms, which means that it removes from Logo's memory words that used to be procedure or variable names but aren't any more); without an input, GC does a generational garbage collection, which means that only recently created nodes are examined (The latter is usually good enough).

Example:

```
(gc true)
```

shrinkStacks

command to shrink all stacks of aUCBLogo to the least power of 2 greater than PAGE_SIZE (0x100) and the number of elements on that respective stack.

Example:

```
to teststack
  clearText
  fillstack 1000000
end
to fillstack n
  if n > 0 [fillstack n-1]
  n=n
end
teststack
shrinkStacks
```

setStackNoisy *state*

command to set the *state* of the "noisy" flag for the Stacks. If *state* is true, the Stacks will print their size when they grow or shrink.

Example:

```
setStackNoisy true
```

Editing

...commands call an external editor with either a temporary file (edit) or a specified file (editFile). And there are lots of abbreviations.

But probably you won't need those commands if you edit your programs in an external editor like Crimson Editor. If you have a program x.lg and a procedure x in x.lg, then you can run x by calling "logo.exe x.lg". Such a command you can assign to a hotkey like F9 in you editor and then debugging is quite more comfortable than using edall. One more advantage on using "logo.exe x.lg" rather than edall is that the sequence of your procedures does not change.

Editing

- edit 347, ed 347
- editFile 348
- edall 348, e 348
- edps 348
- edns 348
- edpls 349
- edn 349
- edpl 349

edit *contentslist*

ed *contentslist*

(edit)

(ed)

command. If invoked with an input, EDIT writes the definitions of the named items into a temporary file and edits that file, using your favorite editor as determined by the AEDITOR environment variable. If you don't have an AEDITOR variable, edits the definitions using the internal editor. If invoked without an input, EDIT edits the same file left over from a previous EDIT or EDITFILE instruction. When you leave the editor, Logo reads the revised definitions and modifies the workspace accordingly. It is not an error if the input includes names for which there is no previous definition.

If there is an environment variable called ATEMP, then Logo uses its value as the directory in which to write the temporary file used for editing.

Exceptionally, the EDIT command can be used without its default input and without parentheses provided that nothing follows it on the instruction line.

Example:

```
si      ;227.375  
ed "si
```

editFile *filename*

command. Starts the Logo editor, like EDIT, but instead of editing a temporary file it edits the file specified by the input. When you leave the editor, Logo reads the revised file, as for EDIT.

EDITFILE also remembers the *filename*, so that a subsequent EDIT command with no input will re-edit the same file.

EDITFILE is intended as an alternative to LOAD and SAVE. You can maintain a workspace file yourself, controlling the order in which definitions appear, maintaining comments in the file, and so on.

Example:

```
editfile "si.lg
```

edall (library *procedure*) **e**

command. Abbreviates EDIT CONTENTS.

edps (library *procedure*)

command. Abbreviates EDIT PROCEDURES.

edns (**library procedure**)

command. Abbreviates EDIT NAMES.

edpls (**library procedure**)

command. Abbreviates EDIT PLISTS.

edn varname (library procedure)
edn varnamelist

command. Abbreviates EDIT NAMELIST varname(list).

edpl pname (library procedure)
edpl plnamelist

command. Abbreviates EDIT PLLIST pname(list).

Erasing

...is often more convenient than exiting logo and start it again. Because of this there is the erase command and its abbreviations.

But don't confuse erase with eraseFile! erase and its abbreviations will only erase something from the workspace.

Erasing

- erase 350, er 350
- eraseAll 350, erAll 350
- eraseProcedures 351, erPs 351
- eraseNames 351, erNs 351
- erasePropertyLists 351, erPLs 351
- ern 351
- erpl 352
- reset 352

erase *contentslist*

er *contentslist*

command. Erases from the workspace the procedures, variables, and property lists named in the input. Primitive procedures may not be erased unless the variable REDEFP has the value TRUE.

Example:

```
si      ;170.634
Contents      ;[[fractal image_right si triangle][t]] ;-)
er "si
Contents      ;[[fractal image_right triangle][t]] ;-)
```

eraseAll

erAll

Workspace Management / Erasing / eraseAll

command. Erases all unburied procedures, variables, and property lists from the workspace. Abbreviates ERASE CONTENTS.

eraseProcedures
erPs

command. Erases all unburied procedures from the workspace. Abbreviates ERASE PROCEDURES.

eraseNames
erNs

command. Erases all unburied variables from the workspace. Abbreviates ERASE NAMES.

erasePropertyLists
erPLs

command. Erases all unburied property lists from the workspace. Abbreviates ERASE PLISTS.

ern *varname* (library procedure)
ern *varnamelist*

command. Erases from the workspace the variable(s) named in the input. Abbreviates ERASE NAMELIST *varname*(list).

Example:

```
a=1234
ern "a
a      ; I don't know how to a
```

erpl *pname* (library procedure)
erpl *plnamelist*

command. Erases from the workspace the property list(s) named in the input. Abbreviates ERASE PLLIST pname(list).

reset (library *procedure*)

command calling erall and several resettings to reset the interpreter to a known state.

reset.lg looks now like this:

Workspace Management / Erasing / reset

```
to reset
  setCaseIgnored true
  erAll
  clearScreen
  setUpdateGraph true
  setPenColor 0
  setScreenColor 7
  setFloodColor 0
  setPenSize [1 1]
  disableCylinderLines
  enableLineSmooth
  setLabelFont [Times]
  showTurtle
  PenDown
  setLightAmbient rgb 0.1 0.1 0.1
  setLightDiffuse rgb 1 1 1
  unperspective
  refresh
  doubleBuffer
  wrap
  insertMode
end
```

Control Structures

Note: in the following descriptions, an "instructionlist" can be a list or a word. In the latter case, the word is parsed into list form before it is run. Thus, RUN READWORD or RUN READLIST will work. The former is slightly preferable because it allows for a continued line (with ~) that includes a comment (with ;) on the first line.

Control Structures

- bye 354
- stop 354
- goTo 355
- Tag 355
- output 356, op 356
- ignore 356, comment 356
- ` 356
- run 357
- runResult 358
- wait 358
- waitMS 359
- waituS 359
- pause 359
- continue 360, co 360
- check 361
- profile 362

bye

command. Exits from Logo; returns to the operating system.

stop

command. Ends the running of the procedure in which it appears. Control is returned to the context in which that procedure was invoked. The stopped procedure does not output a value.

Control Structures / stop

Example:

```
to t
  forever [if key? [stop]]
    pr [Hallo!]
end
;t defined
t
```

Then, if you press any key t will stop and the pr will not be run, because stop immediatly exits the currently running procedure.

goTo word

command. Looks for a Tag command with the same input in the same procedure, and continues running the procedure from the location of that tag. It is meaningless to use GOTO outside of a procedure.

But be aware, goTo is a rather slow command, as it searches the current procedure until it finds the tag or till its end (with an error message).

Examples:

```
to t
  Tag "one
  pr 1
  goTo "two
  pr 5
  Tag "two
  pr 2
  goTo "one
end
;t defined
t
```

will print 1 and 2 alternately until you press control-q (stop).

Tag *quoted_word*

command. Does nothing. The input must be a literal word following a quotation mark ("), not the result of a computation. Tags are used by the GOTO command.

output *value***op** *value*

command. Ends the running of the procedure in which it appears. That procedure outputs the *value* " *value* " to the context in which it was invoked. Don't be confused: OUTPUT itself is a command, but the procedure that invokes OUTPUT is an operation.

Example:

```
to t
  pr 1
  output [Hallo!]
  pr 2
end
;t defined
show t
```

will print:

```
1
[Hallo!]
```

ignore *value***comment** *value* (library procedure)

command. Does nothing. Used when an expression is evaluated for a side effect and its actual *value* is unimportant.

Control Structures / `

` *list* (library procedure)

outputs a list equal to its input but with certain substitutions. If a member of the input list is the word "," (comma) then the following member should be an instructionlist that produces an output when run. That output value replaces the comma and the instructionlist. If a member of the input list is the word ",@" (comma atsign) then the following member should be an instructionlist that outputs a list when run. The members of that list replace the ,@ and the instructionlist.

Example:

```
show `[foo baz ,[bf [a b c]] garply ,@[bf [a b c]]]
will print
  [foo baz [b c] garply b c]
```

run *instructionlist*

command or operation. Runs the Logo instructions in the input list; outputs if the list contains an expression that outputs.

Examples:

```
run [pr 1 pr 2 pr 3]
prints:
```

```
1
2
3
```

```
to a
  pr "a
end
a defined
to b
  pr "b
end
b defined
repeat 10 [run pick [a b]]
```

prints something like:

```
a
a
b
b
a
b
a
a
a
a
```

runResult *instructionlist*

runs the instructions in the input; outputs an empty list if those instructions produce no output, or a list whose only member is the output from running the input *instructionlist* . Useful for inventing command-or-operation control structures.

Example:

```
to t something
  local "result
  make "result runresult something
  if empty? :result [stop]
  output first :result
end
;t defined
t [1] ;1 ;-)
```

outputs 1, but

```
t [pr 5] ;5
```

does not output anything, but just prints 5.

Control Structures / wait

wait *time*

command. Delays further execution for "*time*" 60ths of a second. Also causes any buffered characters destined for the terminal to be printed immediately. WAIT 0 can be used to achieve this buffer flushing without actually waiting.

Example:

```
wait 60      ;waits one second
```

waitMS *milliseconds*

command. Delays further execution for *milliseconds* .

Example:

```
waitMS 1000   ;waits one second
```

waituS *microseconds*

command. Delays further execution for *microseconds* . This command tries to be as exact as possible, depending on the possibilities of the operating system and the hardware.

Example:

```
waituS 10500   ;waits 10.5 milliseconds
```

pause

command or operation. Enters an interactive pause. The user is prompted for instructions, as at toplevel, but with a prompt that includes the name of the procedure in which PAUSE was invoked. Local variables of that procedure are available during the pause. PAUSE outputs if the pause is ended by a CONTINUE with an input.

If the variable `ERRACT` exists, and an error condition occurs, the contents of that variable are run as an instructionlist. Typically `ERRACT` is given the value `[PAUSE]` so that an interactive pause will be entered on the event of an error. This allows the user to check values of local variables at the time of the error.

Typing the key `[Pause]` will also enter a pause.

Example:

```
to t
  forever
    [ pause
    ]
end
;t defined
t ;continue ; Pausing t in line 2...
stop
```

continue *value*

co *value*

(continue)

(co)

command. Ends the current interactive pause, returning to the context of the `PAUSE` invocation that began it. If `CONTINUE` is given an input, that *value* is used as the output from the `PAUSE`. If not, the `PAUSE` does not output.

Exceptionally, the `CONTINUE` command can be used without its default input and without parentheses provided that nothing follows it on the instruction line.

Examples:

Control Structures / continue

```
to t
  pause
  pr 5
end
;t defined
t
continue ; Pausing t in line 2...
;5
to tt
  pr pause
end
;tt defined
tt
continue 1234; Pausing tt in line 1...
;1234
```

check *instructionlist value* (library procedure)

command which runs the *instructionlist* and compares the result output by the run with value. It's good for consistency checking of operations.

Example:

```

to checkplists
  local [ok]
  ok=true
  putProperty "opposite_of" "yes" "no"
  pProp "opposite_of" "black" "white"
  pprop "opposite_of" "up" "down"
  check [getProperty "opposite_of" "black"] "white"
  check [plist "opposite_of"] [up down black white yes no]

  removeProperty "opposite_of" "black"
  check [plist "opposite_of"] [up down yes no]

  remProp "opposite_of" "yes"
  check [plist "opposite_of"] [up down]

  pprop "greetings" "english" "hello"
  pprop "greetings" "german" "hallo"
  check [plist "greetings"] [german hallo english hello]
  check [plists] [[] [] [greetings opposite_of]]

  remProp "opposite_of" "up"
  check [plist "opposite_of"] []
  check [plists] [[]][greetings]]

  type [\; The Property List functions\ ]
  ifelse not :ok
  [ pr [ARE NOT CORRECT!]
  ][ pr [look OK.]
  ]
  output :ok
end

```

profile *instructionlist* (library procedure)

runs the *instructionlist* after renaming all user procedures and defining caller procedures, which save the execution time and number of the renamed procedures. These data are showed in a table as a profile after the run of the *instructionlist*.

Control Structures / profile

Example:

```
load "bounce3.lg  
profile [bounce3]
```

Conditional execution

Conditional execution

- if 364
- ifElse 365
- ifTrue 365, ifT 365
- ifFalse 365, ifF 365
- test 366
- case 366
- cond 366
- throw 367
- catch 368
- error 369
- _maybeOutput 369

if *tf instructionlist*
(**if** *tf instructionlist1 instructionlist2*)

command. If the first input has the value TRUE, then IF runs the second input. If the first input has the value FALSE, then IF does nothing. (If given a third input, IF acts like IFELSE, as described below.) It is an error if the first input is not either TRUE or FALSE.

For compatibility with earlier versions of Logo, if an IF instruction is not enclosed in parentheses, but the first thing on the instruction line after the second input expression is a literal list (i.e., a list in square brackets), the IF is treated as if it were IFELSE, but a warning message is given. If this aberrant IF appears in a procedure body, the warning is given only the first time the procedure is invoked in each Logo session.

Examples:

Conditional execution / if

```

if true [pr true]      ;true
if true [true]        ;true  ;-) (outputs true)
if false [pr false]
if 1==1 [pr true]      ;true

```

ifElse *tf instructionlist1 instructionlist2*

command or operation. If the first input has the value TRUE, then IFELSE runs the second input. If the first input has the value FALSE, then IFELSE runs the third input. IFELSE outputs a value if the instructionlist contains an expression that outputs a value.

Examples:

```

ifElse true [pr true][pr false]      ;true
ifElse false [pr true][pr false]      ;false

```

ifTrue *instructionlist***ifT** *instructionlist*

command. Runs its input if the most recent TEST instruction had a TRUE input. The TEST must have been in the same procedure or a superprocedure.

Example:

```

to t
  test true
  iftrue [pr true]
end
;t defined
t      ;true

```

ifFalse *instructionlist*

ifF *instructionlist*

command. Runs its input if the most recent TEST instruction had a FALSE input. The TEST must have been in the same procedure or a superprocedure.

Example:

```
to t
  test false
  iffalse [pr false]
end
;t defined
t      ;false
```

test *tf*

command. Remembers its input, which must be TRUE or FALSE, for use by later IFTRUE or IFFALSE instructions. The effect of TEST is local to the procedure in which it is used; any corresponding IFTRUE or IFFALSE must be in the same procedure or a subprocedure.

result **case** *value clist*

outputs as result the run butfirst entry of the list of the *clist* list which has the entry *value* as its first element. If the first entry of a *clist* sublist is else then if no previous condition is fulfilled the corresponding rest of that sublist gets executed.

Example:

```
case 2 [[1 "a] [2 "b] [3 "c]]      ;b  ;-)
case 2 [[1 "a][else "b]]          ;b  ;-)
case 2 [[1 pr "a] [(Cos 0)+Cos 0] pr "b] [else pr
"els]]                          ;b
```

Conditional execution / cond

result **cond** *clist* (library procedure)

command or operation. *clist* is a list of [condition instructions] lists. result is the value which instructions output. But don't use the logo word output in instructions because that would end the cond caller.

Example:

```
cond [[false 1] [true 2] [else 3]]      ;2    ;-)
cond [[1==2 1234][3==3 4321][else [Hallo]]] ;4321    ;-)
```

throw *tag*
(**throw** *tag value*)

command. Must be used within the scope of a CATCH with an equal *tag* . Ends the running of the instructionlist of the CATCH. If THROW is used with only one input, the corresponding CATCH does not output a *value* . If THROW is used with two inputs, the second provides an output for the CATCH.

THROW "TOPLEVEL

can be used to terminate all running procedures and interactive pauses, and return to the toplevel instruction prompt. Typing the system interrupt character [Ctrl-Pause] has the same effect.

THROW "ERROR

can be used to generate an error condition. If the error is not caught, it prints a message (THROW "ERROR) with the usual indication of where the error (in this case the THROW) occurred. If a second input is used along with a *tag* of ERROR, that second input is used as the text of the error message instead of the standard message. Also, in this case, the location indicated for the error will be not the location of the THROW, but the location where the procedure containing the THROW was invoked. This allows user-defined procedures to generate error messages as if they were primitives. Note: in this case the corresponding

CATCH "ERROR,

if any, does not output, since the second input to THROW is not considered a return *value* .

THROW "SYSTEM

immediately leaves Logo, returning to the operating system, without printing the usual parting message and without deleting any editor temporary file written by EDIT.

Example:

```
to t
  catch "Hallo
  [   pr 1
      throw "Hallo
      pr 2
  ]
  pr 3
end
;t defined
t
;1
;3
```

catch *tag instructionlist*

command or operation. Runs its second input. Outputs if that *instructionlist* outputs. If, while running the *instructionlist*, a THROW instruction is executed with a *tag* equal to the first input (case-insensitive comparison), then the running of the *instructionlist* is terminated immediately. In this case the CATCH outputs if a value input is given to THROW. The *tag* must be a word.

If the *tag* is the word ERROR, then any error condition that arises during the running of the *instructionlist* has the effect of

```
THROW "ERROR
```

instead of printing an error message and returning to toplevel. The CATCH does not output if an error is caught. Also, during the running of the *instructionlist*, the variable ERRACT is temporarily unbound. (If there is an error while ERRACT has a value, that value is taken as an *instructionlist* to be run after printing the error message. Typically the value of ERRACT, if any, is the list [PAUSE].)

Example:

Conditional execution / catch

```

to t
  output catch "Hallo
  [   pr 1
      (throw "Hallo 1234)
      pr 2
  ]
  pr 3
end
;t defined
t
;1
;1234  ;-)

```

error

outputs a list describing the error just caught, if any. If there was not an error caught since the last use of **ERROR**, the empty list will be output. The error list contains four members: an integer code corresponding to the type of error, the text of the error message, the name of the procedure in which the error occurred, and the instruction line on which the error occurred.

Example:

```

to t
  pr 1
  catch "error
  [   asdfsfg
  ]
  show error
end
;t defined
t
;1
;[13 [; I don't know how to asdfsfg] t 3]

```

_maybeOutput *value* (special form)

works like `OUTPUT` except that the expression that provides the input value might not, in fact, output a value, in which case the effect is like `STOP`. This is intended for use in control structure definitions, for cases in which you don't know whether or not some expression produces a value.

Example:

```
to invoke :function [:inputs] 2
  _maybeOutput apply :function :inputs
end
```

```
(invoke "print "a "b "c)      ;a b c
print (invoke "word "a "b "c) ;abc
```

This is an alternative to `RUNRESULT`. It's fast and easy to use, at the cost of being an exception to Logo's evaluation rules. (Ordinarily, it should be an error if the expression that's supposed to provide an input to something doesn't have a value.)

Loops

Loops

Loops

- repeat 371
- repeatCount 371, repCount 371
- forever 372
- for 372
- while 373
- until 373
- do_while 374
- do_until 374
- break 374
- continueLoop 375

repeat *num instructionlist*

command. Runs the instructionList repeatedly, *num* times.

Example:

```
repeat 4
[   fd 100 rt 90
    pr repCount
]
```

repeatCount
repCount

outputs the repetition count of the innermost current REPEAT, starting from 1. If no REPEAT is active, outputs 0.

Example:

```
show repCount    ;0
repeat 4
[   print repCount    ;1 2 3 4
]
show repCount    ;0
```

forever *instructionList*

command. Runs the *instructionList* repeatedly, infinite times.

Example:

```
forever
[   print repcount
]
```

for *forcontrol instructionlist*

command. The first input must be a list containing three or four members:

- (1) a word, which will be used as the name of a local variable;
- (2) a word or list that will be evaluated as by RUN to determine a number, the starting value of the variable;
- (3) a word or list that will be evaluated to determine a number, the limit value of the variable;
- (4) an optional word or list that will be evaluated to determine the step size. If the fourth member is missing, the step size will be 1 or -1 depending on whether the limit value is greater than or less than the starting value, respectively.

The second input is an *instructionlist* .

Loops / for

The effect of FOR is to run that *instructionlist* repeatedly, assigning a new value to the control variable (the one named by the first member of the *forcontrol* list) each time. First the starting value is assigned to the control variable. Then the value is compared to the limit value. FOR is complete when the sign of (current - limit) is the same as the sign of the step size. (If no explicit step size is provided, the *instructionlist* is always run at least once. An explicit step size can lead to a zero-trip FOR, e.g., FOR [I 1 0 1] ...) Otherwise, the *instructionlist* is run, then the step is added to the current value of the control variable and FOR returns to the comparison step.

Examples:

```
for [i 2 7 1.5] [print :i]
;2
;3.5
;5
;6.5
```

```
for [i [cos 0] 3+1][pr i]
;1
;2
;3
;4
```

while *tfexpression instructionlist*

command. Repeatedly evaluates the "*instructionlist*" as long as the evaluated "*tfexpression*" remains TRUE. Evaluates the first input first, so the "*instructionlist*" may never be run at all. The "*tfexpression*" must be an expressionlist whose value when evaluated is TRUE or FALSE.

While is good if you don't know before the loop how many times the loop should execute, but it's a little bit slower than repeat.

Examples:

```
a=Int 0 while [a<1000000] [a+=1] pr a ;1000000
while [not key?][pr repcount]
```

until *tfexpression instructionlist*

command. Repeatedly evaluates the "*instructionlist*" as long as the evaluated "*tfexpression*" remains FALSE. Evaluates the first input first, so the "*instructionlist*" may never be run at all. The "*tfexpression*" must be an expressionlist whose value when evaluated is TRUE or FALSE.

Examples:

```
until [true][pr repcount]      ;does nothing
until [key?][pr repcount]      ;prints numbers until a key is
pressed
```

do_while *instructionlist tfexpression*

command. Repeatedly evaluates the "*instructionlist*" as long as the evaluated "*tfexpression*" remains TRUE. Evaluates the first input first, so the "*instructionlist*" is always run at least once. The "*tfexpression*" must be an expressionlist whose value when evaluated is TRUE or FALSE.

Examples:

```
do_while [pr repcount][false]   ;1
do_while [pr repcount][repcount < 3] ;1 2 3
```

do_until *instructionlist tfexpression*

command. Repeatedly evaluates the "*instructionlist*" as long as the evaluated "*tfexpression*" remains FALSE. Evaluates the first input first, so the "*instructionlist*" is always run at least once. The "*tfexpression*" must be an expressionlist whose value when evaluated is TRUE or FALSE.

Examples:

```
do_until [pr repcount][repcount > 2] ;1 2 3
do_until [pr repcount][key?]
```

Loops / break

break

command which stops the currently executing loop immediately and continues execution behind the loop.

Examples:

```
repeat 10 [pr # if #==5 [break] pr "so]
while [# < 10][pr # if #==5 [break] pr "so]
until [# >= 10][pr # if #==5 [break] pr "so]
do_while [pr # if #==5 [break] pr "so"][# < 10]
do_until [pr # if #==5 [break] pr "so"][# >= 10]
for [i 1 10][pr i if i==5 [break] pr "so]
for [i 1.1 10.1][pr i if i==5.1 [break] pr "so]
foreach [1 2 3 4 5 6 7 8 9 10][pr # if #==5 [break] pr "so]
```

continueLoop

command which continues the currently executing loop at the beginning of the loop body while incrementing the repcount.

Examples:

```
repeat 10 [pr # if #==5 [continueLoop] pr "so]
while [# < 10][pr # if #==5 [continueLoop] pr "so]
until [# >= 10][pr # if #==5 [continueLoop] pr "so]
do_while [pr # if #==5 [continueLoop] pr "so"][# < 10]
do_until [pr # if #==5 [continueLoop] pr "so"][# >= 10]
for [i 1 10][pr i if i==5 [continueLoop] pr "so]
for [i 1.1 10.1][pr i if i==5.1 [continueLoop] pr "so]
foreach [1 2 3 4 5 6 7 8 9 10][pr # if #==5 [continueLoop] pr "so]
```

Template Based Iteration

The procedures in this section are iteration tools based on the idea of a "template." This is a generalization of an instruction list or an expression list in which "slots" are provided for the tool to insert varying data. Four different forms of template can be used.

The most commonly used form for a template is "explicit-slot" form, or "question mark" form.

Example:

```
show map [? * ?] [2 3 4 5] ;[4 9 16 25]
```

In this example, the MAP tool evaluated the template `[? * ?]` repeatedly, with each of the members of the data list `[2 3 4 5]` substituted in turn for the question marks. The same value was used for every question mark in a given evaluation. Some tools allow for more than one datum to be substituted in parallel; in these cases the slots are indicated by `?1` for the first datum, `?2` for the second, and so on:

```
show (map [(word ?1 ?2 ?1)] [a b c] [d e f]) ;[ada beb cfc]
```

If the template wishes to compute the datum number, the form `(? 1)` is equivalent to `?1`, so `(? ?1)` means the datum whose number is given in datum number 1. Some tools allow additional slot designations, as shown in the individual descriptions.

The second form of template is the "named-procedure" form. If the template is a word rather than a list, it is taken as the name of a procedure. That procedure must accept a number of inputs equal to the number of parallel data slots provided by the tool; the procedure is applied to all of the available data in order. That is, if data `?1` through `?3` are available, the template "PROC is equivalent to `[PROC ?1 ?2 ?3]`.

```
show (map "word [a b c] [d e f]) ;[ad be cf]
to dotprod :a :b ; vector dot product
  op apply "sum (map "product :a :b)
end
```

The third form of template is "named-slot" or "lambda" form. This form is indicated by a template list containing more than one member, whose first member is itself a list. The first member is taken as a list of names; local variables are created with those names and given the available data in order

Template Based Iteration

as their values. The number of names must equal the number of available data. This form is needed primarily when one iteration tool must be used within the template list of another, and the ? notation would be ambiguous in the inner template.

Example:

```
to matmul :m1 :m2 [:tm2 transpose :m2] ; multiply two matrices
  output map [[row] map [[col] dotprod :row :col] :tm2] :m1
end
```

The fourth form is "procedure text" form, a variant of lambda form. In this form, the template list contains at least two members, all of which are lists. This is the form used by the DEFINE and TEXT primitives, and APPLY accepts it so that the text of a defined procedure can be used as a template.

Note: The fourth form of template is interpreted differently from the others, in that Logo considers it to be an independent defined procedure for the purposes of OUTPUT and STOP.

For example, the following two instructions are identical:

```
show apply [[x] :x+3] [5] ;8
show apply [[x] [output :x+3]] [5] ;8
```

although the first instruction is in named-slot form and the second is in procedure-text form. The named-slot form can be understood as telling Logo to evaluate the expression :x+3 in place of the entire invocation of apply, with the variable x temporarily given the value 5. The procedure-text form can be understood as invoking the procedure

```
to foo :x
  output :x+3
end
```

with input 5, but without actually giving the procedure a name. If the use of OUTPUT were interchanged in these two examples, we'd get errors:

```
print apply [[x] output :x+3] [5]
;Can only use output inside a procedure
print apply [[x] [:x+3]] [5] ;8 ;-)
```

The named-slot form can be used with STOP or OUTPUT inside a procedure, to stop the enclosing

procedure.

The following iteration tools are extended versions of the ones in

Appendix B of the book
Computer Science Logo Style, Volume 3:
Advanced Topics
by Brian Harvey [MIT Press, 1987].

The extensions are primarily to allow for variable numbers of inputs.

Template Based Iteration

- apply 378
- invoke 379
- foreach 379
- map 380
- map_se 381
- filter 382
- find 382
- reduce 383
- crossmap 383
- cascade 384
- cascade2 385
- transfer 385

apply *template inputlist*

command or operation. Runs the " *template* ," filling its slots with the members of " *inputlist* ." The number of members in " *inputlist* " must be an acceptable number of slots for " *template* ." It is illegal to apply the primitive TO as a *template* , but anything else is okay. APPLY outputs what " *template* " outputs, if anything.

Examples:

Template Based Iteration / apply

```
show apply [?+?] [1 3 5]      ;2
show apply [?1+?2] [1 3 5]    ;4
show apply "sum [1 2 3]       ;6
show apply [[x] :x+3] [5]      ;8
show apply [[x] [output :x+3]] [5] ;8
```

invoke *template input* (library procedure)
(invoke *template input1 input2 ...*)

command or operation. Exactly like APPLY except that the inputs are provided as separate expressions rather than in a list.

Example:

```
show (invoke [?1+?2*?3] 1 2 3) ;7
```

foreach *data template*
(foreach *data1 data2 ... template*)

command. Evaluates the *template* list repeatedly, once for each member of the *data* list. If more than one *data* list are given, each of them must be the same length (The *data* inputs can be words, in which case the *template* is evaluated once for each character).

In a *template* , the symbol ?REST represents the portion of the *data* input to the right of the member currently being used as the ? slot-filler. That is, if the *data* input is [A B C D E] and the *template* is being evaluated with ? replaced by B, then ?REST would be replaced by [C D E]. If multiple parallel slots are used, then (?REST 1) goes with ?1, etc.

In a *template* , the symbol # represents the position in the *data* input of the member currently being used as the ? slot-filler. That is, if the *data* input is [A B C D E] and the *template* is being evaluated with ? replaced by B, then # would be replaced by 2.

Examples:

```

foreach [A B C] [(show # ? ?rest)]
;1 A [B C]
;2 B [C]
;3 C []
(foreach [1 2 3][4 5 6] [(show ?1 ?2)])
;1 4
;2 5
;3 6
(foreach [1 2 3][4 5 6] [[a b] (show a b)])
;1 4
;2 5
;3 6
(foreach [1 2 3][4 5 6] [[a b] [(show a b)])])
;1 4
;2 5
;3 6
foreach "Hallo [show ?]
;h
;a
;l
;l
;o
foreach [Abc Def Ghi] [foreach ? [(show # ? ?rest)]]
;1 A bc
;2 b c
;3 c
;1 D ef
;2 e f
;3 f
;1 G hi
;2 h i
;3 i

```

map *template data* (library procedure)
(map *template data1 data2 ...*)

outputs a word or list, depending on the type of the data input, of the same length as that data input.

Template Based Iteration / map

(If more than one data input are given, the output is of the same type as *data1* .) Each member of the output is the result of evaluating the *template* list, filling the slots with the corresponding member(s) of the data input(s). (All data inputs must be the same length.) In the case of a word output, the results of the *template* evaluation must be words, and they are concatenated with WORD.

In a *template* , the symbol ?REST represents the portion of the data input to the right of the member currently being used as the ? slot-filler. That is, if the data input is [A B C D E] and the *template* is being evaluated with ? replaced by B, then ?REST would be replaced by [C D E]. If multiple parallel slots are used, then (?REST 1) goes with ?1, etc.

In a *template* , the symbol # represents the position in the data input of the member currently being used as the ? slot-filler. That is, if the data input is [A B C D E] and the *template* is being evaluated with ? replaced by B, then # would be replaced by 2.

Examples:

```
map [??] [2 3 4 5] ;[4 9 16 25] ;-)
(map "word [a b c] [d e f]) ;[ad be cf] ;-)
(map [(word ?1 ?2 ?1)] [a b c] [d e f]) ;[ada beb cfc] ;-)
```

map_se *template data* (library procedure)
(map_se *template data1 data2 ...*)

outputs a list formed by evaluating the *template* list repeatedly and concatenating the results using SENTENCE. That is, the members of the output are the members of the results of the evaluations. The output list might, therefore, be of a different length from that of the data input(s). (If the result of an evaluation is the empty list, it contributes nothing to the final output.) The data inputs may be words or lists.

In a *template* , the symbol ?REST represents the portion of the data input to the right of the member currently being used as the ? slot-filler. That is, if the data input is [A B C D E] and the *template* is being evaluated with ? replaced by B, then ?REST would be replaced by [C D E]. If multiple parallel slots are used, then (?REST 1) goes with ?1, etc.

In a *template* , the symbol # represents the position in the data input of the member currently being used as the ? slot-filler. That is, if the data input is [A B C D E] and the *template* is being

evaluated with ? replaced by B, then # would be replaced by 2.

Example:

```
(map_se [list ?1 ?2] [a b c][d e f])      ;[a d b e c f] ; -)
```

filter *tftemplate data* (library procedure)

outputs a word or list, depending on the type of the data input, containing a subset of the members (for a list) or characters (for a word) of the input. The template is evaluated once for each member or character of the data, and it must produce a TRUE or FALSE value. If the value is TRUE, then the corresponding input constituent is included in the output.

Example:

```
filter "vowelp" "elephant" ;eea ; -)
```

In a template, the symbol ?REST represents the portion of the data input to the right of the member currently being used as the ? slot-filler. That is, if the data input is [A B C D E] and the template is being evaluated with ? replaced by B, then ?REST would be replaced by [C D E].

In a template, the symbol # represents the position in the data input of the member currently being used as the ? slot-filler. That is, if the data input is [A B C D E] and the template is being evaluated with ? replaced by B, then # would be replaced by 2.

find *tftemplate data* (library procedure)

outputs the first constituent of the data input (the first member of a list, or the first character of a word) for which the value produced by evaluating the template with that constituent in its slot is TRUE. If there is no such constituent, the empty list is output.

In a template, the symbol ?REST represents the portion of the data input to the right of the member currently being used as the ? slot-filler. That is, if the data input is [A B C D E] and the template is being evaluated with ? replaced by B, then ?REST would be replaced by [C D E].

In a template, the symbol # represents the position in the data input of the member currently being

Template Based Iteration / find

used as the ? slot-filler. That is, if the data input is [A B C D E] and the template is being evaluated with ? replaced by B, then # would be replaced by 2.

Examples:

```
find [?=="a] [H a l l o]      ;a  ;-)
find [?=="b] [H a l l o]      ;[] ;-)
```

reduce *template data* (library procedure)

outputs the result of applying the *template* to accumulate the members of the data input. The *template* must be a two-slot function. Typically it is an associative function name like "SUM. If the data input has only one constituent (member in a list or character in a word), the output is that constituent. Otherwise, the *template* is first applied with ?1 filled with the next-to-last constituent and ?2 with the last constituent. Then, if there are more constituents, the *template* is applied with ?1 filled with the next constituent to the left and ?2 with the result from the previous evaluation. This process continues until all constituents have been used. The data input may not be empty.

Note: If the *template* is, like SUM, the name of a procedure that is capable of accepting arbitrarily many inputs, it is more efficient to use APPLY instead of REDUCE. The latter is good for associative procedures that have been written to accept exactly two inputs:

```
to max :a :b
  output ifelse :a > :b [:a] [:b]
end
```

```
pr reduce "max_ [5 2 6 8 4 2 6 0 6]      ;8
```

Alternatively, REDUCE can be used to write MAX as a procedure that accepts any number of inputs, as SUM does:

```
to max [:inputs] 2
  if empty? :inputs ~
    [(throw "error [not enough inputs to max]]]
  output reduce [ifelse ?1 > ?2 [?1] [?2]] :inputs
end
```

Template Based Iteration / crossmap

crossmap *template listlist* (library procedure)
(crossmap *template data1 data2 ...*)

outputs a list containing the results of *template* evaluations. Each data list contributes to a slot in the *template* ; the number of slots is equal to the number of data list inputs. As a special case, if only one data list input is given, that list is taken as a list of data lists, and each of its members contributes values to a slot. CROSSMAP differs from MAP in that instead of taking members from the data inputs in parallel, it takes all possible combinations of members of data inputs, which need not be the same length.

```
show (crossmap [word ?1 ?2] [a b c] [1 2 3 4])
;[a1 a2 a3 a4 b1 b2 b3 b4 c1 c2 c3 c4]
```

For compatibility with the version in the first edition of CSLS, CROSSMAP templates may use the notation :1 instead of ?1 to indicate slots.

cascade endtest *template startvalue* (library procedure)
(cascade endtest *tmp1 sv1 tmp2 sv2 ...*)
(cascade endtest *tmp1 sv1 tmp2 sv2 ... finaltemplate*)

outputs the result of applying a *template* (or several templates, as explained below) repeatedly, with a given value filling the slot the first time, and the result of each application filling the slot for the following application.

In the simplest case, CASCADE has three inputs. The second input is a one-slot expression *template* . That *template* is evaluated some number of times (perhaps zero). On the first evaluation, the slot is filled with the third input; on subsequent evaluations, the slot is filled with the result of the previous evaluation. The number of evaluations is determined by the first input. This can be either a nonnegative integer, in which case the *template* is evaluated that many times, or a predicate expression *template* , in which case it is evaluated (with the same slot filler that will be used for the evaluation of the second input) repeatedly, and the CASCADE evaluation continues as long as the predicate value is FALSE. (In other words, the predicate *template* indicates the condition for stopping.)

If the *template* is evaluated zero times, the output from CASCADE is the third (startvalue) input. Otherwise, the output is the value produced by the last *template* evaluation.

Template Based Iteration / cascade

CASCADE templates may include the symbol # to represent the number of times the *template* has been evaluated. This slot is filled with 1 for the first evaluation, 2 for the second, and so on.

```
show cascade 5 [lput # ?] [] ;[1 2 3 4 5]
show cascade [vowelp first ?] [bf ?] "spring ;ing
show cascade 5 [# * ?] 1 ;120
```

Several cascaded results can be computed in parallel by providing additional *template* -startvalue pairs as inputs to CASCADE. In this case, all templates (including the *endtest template* , if used) are multi-slot, with the number of slots equal to the number of pairs of inputs. In each round of evaluations, ?2 represents the result of evaluating the second *template* in the previous round. If the total number of inputs (including the first *endtest* input) is odd, then the output from CASCADE is the final value of the first *template* . If the total number of inputs is even, then the last input is a *template* that is evaluated once, after the end test is satisfied, to determine the output from CASCADE.

```
to fibonacci :n
  output (cascade :n [?1 + ?2] 1 [?1] 0)
end
```

```
to piglatin :aword
  output (cascade [vowelp first ?]
    [word bf ? first ?]
    :aword
    [word ? "ay])
end
;piglatin defined
piglatin "greetings ;eetingsgray i-)
```

cascade2 endtest temp1 startval1 temp2 startval2 (library procedure)

outputs the result of invoking CASCADE with the same inputs. The only difference is that the default number of inputs is five instead of three.

transfer endtest template inbasket (library procedure)

outputs the result of repeated evaluation of the *template*. The *template* is evaluated once for each member of the list "inbasket." TRANSFER maintains an "outbasket" that is initially the empty list. After each evaluation of the *template*, the resulting value becomes the new outbasket.

In the *template*, the symbol ?IN represents the current member from the inbasket; the symbol ?OUT represents the entire current outbasket. Other slot symbols should not be used.

If the first (*endtest*) input is an empty list, evaluation continues until all inbasket members have been used. If not, the first input must be a predicate expression *template*, and evaluation continues until either that *template*'s value is TRUE or the inbasket is used up.

Macros

Macros

- `_Macro` 387
- `_defMacro` 387
- `MacroP` 390
- `macroexpand` 390

`_Macro procname :input1 :input2 ...` (special form)

`_defMacro procname text`

A macro is a special kind of procedure whose output is evaluated as Logo instructions in the context of the macro's caller. `_macro` is exactly like `TO` except that the new procedure becomes a macro; `_defmacro` is exactly like `DEFINE` with the same exception.

Macros are useful for inventing new control structures comparable to `REPEAT`, `IF`, and so on. Such control structures can almost, but not quite, be duplicated by ordinary Logo procedures.

For example, here is an ordinary procedure version of `REPEAT`:

```
to my_repeat :num :instructions
  if :num=0 [stop]
  run :instructions
  my_repeat :num-1 :instructions
end
```

This version works fine for most purposes, e.g.,

```
my_repeat 5 [print "hello]
```

But it doesn't work if the instructions to be carried out include `OUTPUT`, `STOP`, or `LOCAL`.

For example, consider this procedure:

```
to example
  print [Guess my secret word.  You get three guesses.]
  repeat 3 ~
    [
      type "|?? |
      if readword = "secret [pr "Right! stop]
    ]
  print [Sorry, the word was "secret"!]
```

This procedure works as written, but if MY_REPEAT is used instead of REPEAT, it won't work because the STOP will stop MY_REPEAT instead of stopping EXAMPLE as desired.

The solution is to make MY_REPEAT a macro. Instead of actually carrying out the computation, a macro must return a list containing Logo instructions. The contents of that list are evaluated as if they appeared in place of the call to the macro. Here's a macro version of REPEAT:

```
_macro my_repeat :num :instructions
  if :num=0 [output []]
  output sentence :instructions ~
    (list "my_repeat :num-1 :instructions)
end
```

Every macro is an operation -- it must always output something. Even in the base case, MY_REPEAT outputs an empty instruction list.

To show how MY_REPEAT works, let's take the example

```
my_repeat 5 [print "hello]
```

For this example, MY_REPEAT will output the instruction list

```
[print "hello my_repeat 4 [print "hello]]
```

Logo then executes these instructions in place of the original invocation of MY_REPEAT; this prints "hello" once and invokes another repetition.

The technique just shown, although fairly easy to understand, has the defect of slowness because each repetition has to construct an instruction list for evaluation. Another approach is to make my_repeat a macro that works just like the non-macro version unless the instructions to be repeated

Macros / _defMacro

include OUTPUT or STOP:

```
_macro my_repeat :num :instructions
  catch "repeat_catchtag ~
    [op repeat_done runresult [repeat1 :num :instructions]]
  op []
end
```

```
to repeat1 :num :instructions
  if :num=0 [throw "repeat_catchtag]
  run :instructions
  _maybeOutput repeat1 :num-1 :instructions
end
```

```
to repeat_done :repeat_result
  if empty? :repeat_result [op [stop]]
  op list "output quoted first :repeat_result
end
```

If the instructions do not include STOP or OUTPUT, then REPEAT1 will reach its base case and invoke THROW. As a result, my_repeat's last instruction line will output an empty list, so the second evaluation of the macro result will do nothing. But if a STOP or OUTPUT happens, then REPEAT_DONE will output a STOP or OUTPUT instruction that will be re-executed in the caller's context.

The macro-defining commands have names starting with a dot because macros are an advanced feature of Logo; it's easy to get in trouble by defining a macro that doesn't terminate, or by failing to construct the instruction list properly.

Lisp users should note that Logo macros are NOT special forms. That is, the inputs to the macro are evaluated normally, as they would be for any other Logo procedure. It's only the output from the macro that's handled unusually.

Here's another example:

```
_macro localmake :name :value
  output (list "local      ~
             word " " :name  ~
             "apply      ~
             "make      ~
             (list :name :value))
end
```

It's used this way:

```
to try
  localmake "garply "hello
  print :garply
end
```

LOCALMAKE outputs the list

```
[local "garply apply "make [garply hello]]
```

The reason for the use of APPLY is to avoid having to decide whether or not the second input to MAKE requires a quotation mark before it. (In this case it would -- MAKE "GARPLY "HELLO -- but the quotation mark would be wrong if the value were a list.)

It's often convenient to use the ` function to construct the instruction list:

```
_macro localmake :name :value
  op `[local ,[word " " :name] apply "make [[:name] ,[:value]]]
end
```

On the other hand, ` is pretty slow, since it's tree recursive and written in Logo.

MacroP *name*

Macro? *name*

outputs TRUE if its input is the *name* of a macro.

macroexpand *expr* (library procedure)

Macros / macroexpand

takes as its input a Logo expression that invokes a macro (that is, one that begins with the name of a macro) and outputs the the Logo expression into which the macro would translate the input expression.

```
_macro localmake :name :value  
  op `[local ,[word " " :name] apply "make [[:name] ,[:value]]]  
end
```

```
show macroexpand [localmake "pi 3.14159]  
;[local "pi apply "make [pi 3.14159]]
```

Error Processing

If an error occurs, Logo takes the following steps. First, if there is an available variable named `ERRACT`, Logo takes its value as an instructionlist and runs the instructions. The operation `ERROR` may be used within the instructions (once) to examine the error condition. If the instructionlist invokes `PAUSE`, the error message is printed before the pause happens. Certain errors are "recoverable"; for one of those errors, if the instructionlist outputs a value, that value is used in place of the expression that caused the error. (If `ERRACT` invokes `PAUSE` and the user then invokes `CONTINUE` with an input, that input becomes the output from `PAUSE` and therefore the output from the `ERRACT` instructionlist.)

It is possible for an `ERRACT` instructionlist to produce an inappropriate value or no value where one is needed. As a result, the same error condition could recur forever because of this mechanism. To avoid that danger, if the same error condition occurs twice in a row from an `ERRACT` instructionlist without user interaction, the message "Erract loop" is printed and control returns to toplevel. "Without user interaction" means that if `ERRACT` invokes `PAUSE` and the user provides an incorrect value, this loop prevention mechanism does not take effect and the user gets to try again.

During the running of the `ERRACT` instructionlist, `ERRACT` is locally unbound, so an error in the `ERRACT` instructions themselves will not cause a loop. In particular, an error during a pause will not cause a pause-within-a-pause unless the user reassigns the value `[PAUSE]` to `ERRACT` during the pause. But such an error will not return to toplevel; it will remain within the original pause loop.

Error Processing

If there is no available ERRACT value, Logo handles the error by generating an internal THROW "ERROR. (A user program can also generate an error condition deliberately by invoking THROW.) If this throw is not caught by a CATCH "ERROR in the user program, it is eventually caught either by the toplevel instruction loop or by a pause loop, which prints the error message. An invocation of CATCH "ERROR in a user program locally unbinds ERRACT, so the effect is that whichever of ERRACT and CATCH "ERROR is more local will take precedence.

If a floating point overflow occurs during an arithmetic operation, or a two-input mathematical function (like POWER) is invoked with an illegal combination of inputs, the "doesn't like" message refers to the second operand, but should be taken as meaning the combination.

Here are the numeric codes that appear as the first member of the list output by ERROR when an error is caught, with the corresponding messages. Some messages may have two different codes depending on whether or not the error is recoverable (that is, a substitute value can be provided through the ERRACT mechanism) in the specific context. Some messages are warnings rather than errors; these will not be caught. Errors 0 and 32 are so bad that Logo exits immediately.

Error Processing

```
0 Fatal internal error (can't be caught)
1 Out of memory
2 Stack overflow
3 Turtle out of bounds
4 PROC doesn't like DATUM as input (not recoverable)
5 PROC didn't output to PROC
6 Not enough inputs to PROC
7 PROC doesn't like DATUM as input (recoverable)
8 Too much inside ()'s
9 You don't say what to do with DATUM
10 ')' not found
11 VAR has no value
12 Unexpected ')'
13 I don't know how to PROC (recoverable)
14 Can't find catch tag for THROWTAG
15 PROC is already defined
16 Stopped
17 Already dribbling
18 File system error
19 Assuming you mean IFELSE, not IF (warning only)
20 VAR shadowed by local in procedure call (warning only)
21 Throw "Error
22 PROC is a primitive
23 Can't use TO inside a procedure
24 I don't know how to PROC (not recoverable)
25 IFTRUE/IFFALSE without TEST
26 Unexpected ']'
27 Unexpected '}'
28 Couldn't initialize graphics
29 Macro returned VALUE instead of a list
30 You don't say what to do with VALUE
31 Can only use STOP or OUTPUT inside a procedure
32 APPLY doesn't like BADTHING as input
33 END inside multi-line instruction
34 Really out of memory (can't be caught)
```

Special Variables

Logo takes special action if any of the following variable names exists. They follow the normal scoping rules, so a procedure can locally set one of them to limit the scope of its effect. Initially, no variables exist.

Special Variables

- `erract` 394
- `loadNoisily` 394
- `printDepthLimit` 394
- `printWidthLimit` 394
- `reDefP` 395
- `Startup` 395

erract

an instructionlist that will be run in the event of an error. Typically has the value [PAUSE] to allow interactive debugging.

loadNoisily

if TRUE, prints the names of procedures defined when loading from a file (including the temporary file made by EDIT).

printDepthLimit

if a nonnegative integer, indicates the maximum depth of sublist structure that will be printed by PRINT, etc.

Special Variables / printWidthLimit

printWidthLimit

if a nonnegative integer, indicates the maximum number of members in any one list that will be printed by PRINT, etc.

reDefP

if TRUE, allows primitives to be erased (ERASE) or redefined (COPYDEF).

Startup

if assigned a list value in a file loaded by LOAD, that value is run as an instructionlist after the loading.

GUI programming

GUI means Graphical User Interface, that's something which Windows, X11, or GTK implements. Among GUI programming are Custom Event Handlers, Standard Dialogs, Windows (=Frames), Graphs (=Logo drawing areas), Sizers and Controls.

Controls are little windows showing some content, numerical, textual or list values, maybe editable or readonly, or clickable.

How to start? It's best to read on. The index page of each of the following sub-chapters helps getting you started, and they mostly point to a demo Logo program, i.e. `buttontest.lg` or `frametest.lg`.

GUI programming

- Window Styles 396
 - Custom Event Handlers 399
 - Standard Dialogs 413
 - Frames 422
 - Graphs 432
 - BoxSizers 439
 - Buttons 443
 - CheckBoxes 446
 - ChoiceBoxes 450
 - ComboBoxes 461
 - FloatControls 484
 - Gauges 489
 - IntControls 493
 - ListBoxes 497
 - ListControls 507
 - RadioButtons 530
 - Sliders 534
 - StaticTexts 539
 - TextControls 544
 - ToggleButtons 556
 - Miscellaneous GUI elements 560
-

Window Styles

The following styles can apply to all windows, although they will not always make sense for a particular window class or on all platforms.

`wxSIMPLE_BORDER` Displays a thin border around the window. `wxBORDER` is the old name for this style.

`wxDOUBLE_BORDER` Displays a double border. Windows and Mac only.

`wxSUNKEN_BORDER` Displays a sunken border.

`wxRAISED_BORDER` Displays a raised border.

`wxSTATIC_BORDER` Displays a border suitable for a static control. Windows only.

`wxNO_BORDER` Displays no border, overriding the default border style for the window.

`wxTRANSPARENT_WINDOW` The window is transparent, that is, it will not receive paint events. Windows only.

`wxTAB_TRAVERSAL` Use this to enable tab traversal for non-dialog windows.

`wxWANTS_CHARS` Use this to indicate that the window wants to get all char/key events for all keys - even for keys like TAB or ENTER which are usually used for dialog navigation and which wouldn't be generated without this style. If you need to use this style in order to get the arrows or etc., but would still like to have normal keyboard navigation take place, you should create and send a `wxNavigationKeyEvent` in response to the key events for Tab and Shift-Tab.

`wxNO_FULL_REPAINT_ON_RESIZE` Disables repainting the window completely when its size is changed - you will have to repaint the new window area manually if you use this style. Currently only has an effect for Windows.

`wxVSCROLL` Use this style to enable a vertical scrollbar.

`wxHSCROLL` Use this style to enable a horizontal scrollbar.

`wxALWAYS_SHOW_SB` If a window has scrollbars, disable them instead of hiding them when they are not needed (i.e. when the size of the window is big enough to not require the scrollbars to navigate it). This style is currently only implemented for `wxMSW` and `wxUniversal` and does nothing on the other platforms.

`wxCLIP_CHILDREN` Use this style to eliminate flicker caused by the background being repainted, then children being painted over them. Windows only.

`wxFULL_REPAINT_ON_RESIZE` Use this style to force a complete redraw of the window whenever it is resized instead of redrawing just the part of the window affected by resizing. Note that this was the behaviour by default before 2.5.1 release and that if you experience redraw problems with the code which previously used to work you may want to try this.

Custom Event Handlers

...are "short" Logo routines which are executed when a specific event is raised, i.e. a keystroke or a mouse click.

"short" means here short in execution time relative to the number of events per second. If there are many events in one second, as it can be with mouse events, then your custom event handler shouldn't block, and at least it should stop at some time, it should terminate.

If there are more events generated than could be processed, eventually a fatal stack overflow will occur and Logo will crash.

For the console are the following keyboard and mouse handlers:

```
OnChar OnKeyDown OnKeyUp KeyboardValue
OnTextMouseLeftDown OnTextMouseRightDown OnTextMouseMiddleDown
OnTextMouseLeftUp OnTextMouseRightUp OnTextMouseMiddleUp
OnTextMouseLeftDClick OnTextMouseRightDClick OnTextMouseMiddleDClick
OnTextMouseMotion
```

For the main graph window are those mouse handlers:

```
OnMouseLeftDown OnMouseRightDown OnMouseMiddleDown
OnMouseLeftUp OnMouseRightUp OnMouseMiddleUp
OnMouseLeftDClick OnMouseRightDClick OnMouseMiddleDClick OnMouseMotion
```

The event handlers are presented in the example eventtest.lg.

Custom Event Handlers

- OnChar 400
- OnKeyDown 400
- OnKeyUp 400
- KeyboardValue 401
- OnTextMouseLeftDown 405
- OnTextMouseRightDown 405
- OnTextMouseMiddleDown 405
- OnTextMouseLeftUp 406
- OnTextMouseRightUp 406

- OnTextMouseMiddleUp 407
 - OnTextMouseLeftDClick 407
 - OnTextMouseRightDClick 407
 - OnTextMouseMiddleDClick 408
 - OnTextMouseMotion 408
 - OnMouseLeftDown 409
 - OnMouseRightDown 409
 - OnMouseMiddleDown 409
 - OnMouseLeftUp 410
 - OnMouseRightUp 410
 - OnMouseMiddleUp 410
 - OnMouseLeftDClick 411
 - OnMouseRightDClick 411
 - OnMouseMiddleDClick 411
 - OnMouseMotion 411
-

OnChar *commands*

This command sets the Logo event handler for a char event of the console. A char event is generated when a key is pressed or held down longer.

Example:

```
OnChar [pr KeyboardValue] ;press Ctrl-Break to stop
```

OnKeyDown *commands*

This command sets the Logo event handler for a key down event of the console. A key down event is generated when a key is pressed or held down longer.

Example:

```
OnKeyDown [pr KeyboardValue]
```

OnKeyUp *commands*

GUI programming / Custom Event Handlers / OnKeyUp

This command sets the Logo event handler for a key up event of the console. A key up event is generated when a key is released from the pressed state.

Example:

```
OnKeyUp [pr KeyboardValue]
```

KeyboardValue

returns the key number of the latest processed key event, this is a char event, key down event or key up event, all just from the console window.

The keycodes are for normal keys their ASCII values, and for special keys they correspond to the following constants:

```
WXX_BACK  
WXX_TAB  
WXX_RETURN  
WXX_ESCAPE  
WXX_SPACE  
WXX_DELETE
```

```
WXX_START
WXX_LBUTTON
WXX_RBUTTON
WXX_CANCEL
WXX_MBUTTON
WXX_CLEAR
WXX_SHIFT
WXX_CONTROL
WXX_MENU
WXX_PAUSE
WXX_CAPITAL
WXX_PRIOR
WXX_NEXT
WXX_END
WXX_HOME
WXX_LEFT
WXX_UP
WXX_RIGHT
WXX_DOWN
WXX_SELECT
WXX_PRINT
WXX_EXECUTE
WXX_SNAPSHOT
WXX_INSERT
WXX_HELP
WXX_NUMPAD0
WXX_NUMPAD1
WXX_NUMPAD2
WXX_NUMPAD3
WXX_NUMPAD4
WXX_NUMPAD5
WXX_NUMPAD6
WXX_NUMPAD7
WXX_NUMPAD8
WXX_NUMPAD9
WXX_MULTIPLY
WXX_ADD
WXX_SEPARATOR
WXX_SUBTRACT
WXX_DECIMAL
```

WXXK_DIVIDE
WXXK_F1
WXXK_F2
WXXK_F3
WXXK_F4
WXXK_F5
WXXK_F6
WXXK_F7
WXXK_F8
WXXK_F9
WXXK_F10
WXXK_F11
WXXK_F12
WXXK_F13
WXXK_F14
WXXK_F15
WXXK_F16
WXXK_F17
WXXK_F18
WXXK_F19
WXXK_F20
WXXK_F21
WXXK_F22
WXXK_F23
WXXK_F24
WXXK_NUMLOCK
WXXK_SCROLL
WXXK_PAGEUP
WXXK_PAGEDOWN

```
WXX_NUMPAD_SPACE  
WXX_NUMPAD_TAB  
WXX_NUMPAD_ENTER  
WXX_NUMPAD_F1  
WXX_NUMPAD_F2  
WXX_NUMPAD_F3  
WXX_NUMPAD_F4  
WXX_NUMPAD_HOME  
WXX_NUMPAD_LEFT  
WXX_NUMPAD_UP  
WXX_NUMPAD_RIGHT  
WXX_NUMPAD_DOWN  
WXX_NUMPAD_PRIOR  
WXX_NUMPAD_PAGEUP  
WXX_NUMPAD_NEXT  
WXX_NUMPAD_PAGEDOWN  
WXX_NUMPAD_END  
WXX_NUMPAD_BEGIN  
WXX_NUMPAD_INSERT  
WXX_NUMPAD_DELETE  
WXX_NUMPAD_EQUAL  
WXX_NUMPAD_MULTIPLY  
WXX_NUMPAD_ADD  
WXX_NUMPAD_SEPARATOR  
WXX_NUMPAD_SUBTRACT  
WXX_NUMPAD_DECIMAL  
WXX_NUMPAD_DIVIDE
```

The following key codes are only generated under Windows currently:

```
WXX_WINDOWS_LEFT  
WXX_WINDOWS_RIGHT  
WXX_WINDOWS_MENU  
WXX_COMMAND
```

Example:

GUI programming / Custom Event Handlers / KeyboardValue

```
OnChar [kv=KeyboardValue
  pr kv
  if kv==WXK_ESCAPE [OnChar []] ;press Esc to stop
```

OnTextMouseLeftDown *commands*

This command sets the Logo event handler for a mouse left button down event of the console. This way it enables custom mouse behavior of the console.

Example:

```
OnTextMouseLeftDown [
  cs ht
  setH 90
  Label WordUnderCursor
  updateGraph]
```

OnTextMouseRightDown *commands*

This command sets the Logo event handler for a mouse right button down event of the console. This way it enables custom mouse behavior of the console.

Example:

```
OnTextMouseRightDown [
  disableTextMouseEvents ;neccessary to prevent popup menu
  cs ht
  setH 90
  Label WordUnderCursor
  updateGraph]
OnTextMouseLeftDown [
  enabletextmouseevents
  setCursor TextMousePos]
```

OnTextMouseMiddleDown *commands*

This command sets the Logo event handler for a mouse middle button down event of the console. This way it enables custom mouse behavior of the console.

Example:

```
OnTextMouseMiddleDown [  
  cs ht  
  setH 90  
  Label WordUnderCursor  
  updateGraph]
```

OnTextMouseLeftUp *commands*

This command sets the Logo event handler for a mouse left button up (=release) event of the console. This way it enables custom mouse behavior of the console.

Example:

```
OnTextMouseLeftUp [  
  cs ht  
  seth 90  
  Label TextMousePos  
  updateGraph]
```

OnTextMouseRightUp *commands*

This command sets the Logo event handler for a mouse right button up (=release) event of the console. This way it enables custom mouse behavior of the console.

Example:

```
OnTextMouseRightUp [  
    cs ht  
    seth 90  
    Label TextMousePos  
    updateGraph]
```

OnTextMouseMiddleUp *commands*

This command sets the Logo event handler for a mouse middle button up (=release) event of the console. This way it enables custom mouse behavior of the console.

Example:

```
OnTextMouseMiddleUp [  
    cs ht  
    seth 90  
    Label TextMousePos  
    updateGraph]
```

OnTextMouseLeftDClick *commands*

This command sets the Logo event handler for a mouse left button double click event of the console. This way it enables custom mouse behavior of the console.

Example:

```
OnTextMouseLeftDClick [  
    setXY  
        10*first TextMousePos  
        -20*first bf TextMousePos  
    updateGraph]
```

OnTextMouseRightDClick *commands*

This command sets the Logo event handler for a mouse right button double click event of the console. This way it enables custom mouse behavior of the console.

Example:

```
OnTextMouseRightDown [
  disableTextMouseEvents] ;neccessary to prevent popup menu
OnTextMouseRightDClick [
  setXY
    10*first TextMousePos
    -20*first bf TextMousePos
  updateGraph]
OnTextMouseLeftDown [
  enabletextmouseevents
  setCursor TextMousePos]
```

OnTextMouseMiddleDClick *commands*

This command sets the Logo event handler for a mouse middle button double click event of the console. This way it enables custom mouse behavior of the console.

Example:

```
OnTextMouseMiddleDClick [
  setXY
    10*first TextMousePos
    -20*first bf TextMousePos
  updateGraph]
```

OnTextMouseMotion *commands*

GUI programming / Custom Event Handlers / OnTextMouseMotion

This command sets the Logo event handler for a mouse motion event of the console. This way it enables custom mouse behavior of the console.

Example:

```
OnTextMouseMotion [  
  ifelse MouseButtons != 0 [PenDown][PenUp]  
  setXY  
    10*first TextMousePos  
    -20*first bf TextMousePos  
  updateGraph]
```

OnMouseLeftDown *commands*

This command sets the Logo event handler for a mouse left button down event of the main graph window.

Example:

```
OnMouseLeftDown [PenDown]  
OnMouseLeftUp [PenUp]  
OnMouseMotion [setPosXYZ MousePos  updateGraph]
```

OnMouseRightDown *commands*

This command sets the Logo event handler for a mouse right button down event of the main graph window.

Example:

```
OnMouseRightDown [setPosXYZ MousePos  fill  updateGraph]
```

OnMouseMiddleDown *commands*

This command sets the Logo event handler for a mouse middle button down event of the main graph window.

Example:

```
OnMouseMiddleDown [pr MousePos]
```

OnMouseLeftUp *commands*

This command sets the Logo event handler for a mouse left button up (=release) event of the main graph window.

Example:

```
OnMouseLeftUp [pr [up!]]
```

OnMouseRightUp *commands*

This command sets the Logo event handler for a mouse right button up (=release) event of the main graph window.

Example:

```
OnMouseRightUp [pr [up!]]
```

OnMouseMiddleUp *commands*

This command sets the Logo event handler for a mouse middle button up (=release) event of the main graph window.

Example:

```
OnMouseMiddleUp [pr [up!]]
```

OnMouseLeftDClick *commands*

This command sets the Logo event handler for a mouse left button double click event of the main graph window.

Example:

```
OnMouseLeftDClick [pr [doubleclick!]]
```

OnMouseRightDClick *commands*

This command sets the Logo event handler for a mouse right button double click event of the main graph window.

Example:

```
OnMouseRightDClick [pr [doubleclick!]]
```

OnMouseMiddleDClick *commands*

This command sets the Logo event handler for a mouse middle button double click event of the main graph window.

Example:

```
OnMouseMiddleDClick [pr [doubleclick!]]
```

OnMouseMotion *commands*

This command sets the Logo event handler for a mouse motion event of the main graph window.

Example:

```
OnMouseMotion [setPosXYZ MousePos  updateGraph]
```

Standard Dialogs

The following standard dialogs for getting information from the user are now available.

They are presented in the example `dialogstest.lg`.

Standard Dialogs

- `DirSelector` 413
- `FileSelector` 414
- `getColorFromUser` 414
- `getFontFromUser` 415
- `getMultipleChoices` 415
- `getNumberFromUser` 416
- `getPasswordFromUser` 417
- `getTextFromUser` 417
- `getSingleChoice` 418
- `getSingleChoiceIndex` 419
- `MessageBox` 419

DirSelector

(DirSelector *message defaultPath pos*)

operation to show a directory selector dialog, where the user can choose a directory.

message is the input prompt in the dialog.

defaultPath is the standard path, where the selector starts.

pos is a list of two integer numbers representing x and y position.

Output is the choosen directory as a word.

Examples:

```
pr DirSelector  
pr (DirSelector [This is the input prompt][C:\\][100 200])
```

FileSelector *message***(FileSelector *message stdPath stdFilename stdExtension wildcard flags pos*)**

Operation to show a file selector dialog, where the user can choose a file from a directory listing.

message is the input prompt text in the dialog.

stdPath is the default path, where the selector starts.

stdFilename is the default filename.

stdExtension is the default file extension.

wildcard is a DOS or unix *wildcard* for the selection of a subset of files to display.

flags may be a combination of wxOPEN wxSAVE wxOVERWRITE_PROMPT
wxFILE_MUST_EXIST wxMULTIPLE or 0

pos is a list of two integer numbers representing x and y position.

Output is the choosen filename as a word.

Examples:

```
pr FileSelector [Choose a file]  
pr (FileSelector [Choose a file] " [dialogtest.lg][lg][*.lg]  
    wxOPEN [400 100])
```

(getColorFromUser)

getColorFromUser *stdColor*

GUI programming / Standard Dialogs / getColorFromUser

Operation to show a color selector dialog, where the user can choose a color with either RGB numbers, from a set of standard colors, or by clicking on a color gradient and choosing the brightness.

stdColor is the initial selected color.

Output is the choosen color as a int.

Examples:

```
pr reRGB (getColorFromUser)
pr reRGB getColorFromUser RGB 1 0.5 0.25
```

getFontFromUser

Operation to show a font selector dialog, where the user can select a font.

Output is a list with the following structure:

[PointSize [PixelSizeX PixelSizeY] Familiy Style Weight FaceName]

Example:

```
pr getFontFromUser
```

getMultipleChoices *message caption choices*
(getMultipleChoices *message caption choices pos centre size*)

Operation to show a dialog to choose some of multiple *choices* .

message is the input prompt text.

caption is the window title text.

choices is a list of items which may be lists or words to select from.

pos is a list of two integer numbers representing x and y position.

If *centre* is true, the *message* text (which may include new line characters) is centred; if false, the *message* is left-justified.

size is a list of two numbers representing width and height.

Output is a list of integers representing the selections, starting from 1 for the first choice.

Examples:

```
show (getMultipleChoices [a Message][a Caption]
    [[Choice Nr. 1][Second Choice][Third Choice]])
```

```
show (getMultipleChoices [a Message][a Caption]
    [[Choice Nr. 1][Second Choice][Third Choice]]
    [200 200] false [400 300])
```

getNumberFromUser *message prompt caption value min max*
(**getNumberFromUser** *message prompt caption value min max pos*)

Operation to show a dialog to enter a integer number.

message is the dialog text.

prompt is the input *prompt* text.

caption is the window title text.

value is the standard number showing at the opening of the dialog.

min is the minimal *value* which can be entered.

max is the maximal *value* .

pos is a list of two integer numbers representing x and y position.

GUI programming / Standard Dialogs / `getNumberFromUser`

Output is the entered integer number.

Example:

```
show (getNumberFromUser [a Message][a Prompt\
    maybe multiline][A caption ] 1234 42 4321)
```

`getPasswordFromUser` *message caption stdValue*
(**`getPasswordFromUser`** *message caption stdValue pos centre*)

Operation to show a dialog to enter a password.

message is the dialog text.

caption is the window title text.

stdValue is the standard password showing at the opening of the dialog.

pos is a list of two integer numbers representing x and y position.

If *centre* is true, the *message* text (which may include new line characters) is centred; if false, the *message* is left-justified.

Output is the entered password.

Examples:

```
show getPasswordFromUser [A Message][A Caption][default_value]
show (getPasswordFromUser [A Message][A Caption][default_value]
    [200 100])
show (getPasswordFromUser [A Message][A Caption][default_value]
    [200 100] false)
```

`getTextFromUser` *message caption stdValue*
(**`getTextFromUser`** *message caption stdValue pos centre*)

Operation to show a dialog to enter text.

message is the dialog text.

caption is the window title text.

stdValue is the standard password showing at the opening of the dialog.

pos is a list of two integer numbers representing x and y position.

If *centre* is true, the *message* text (which may include new line characters) is centred; if false, the *message* is left-justified.

Output is the entered text.

Examples:

```
show (getTextFromUser [A Message][A Caption][default_value]
    [200 100])
show (getTextFromUser [A Message][A Caption][default_value]
    [200 100] false)
```

getSingleChoice *message caption choices*
(**getSingleChoice** *message caption choices pos centre size*)

Operation to show a dialog to choose one of multiple *choices* .

message is the input prompt text.

caption is the window title text.

choices is a list of items which may be lists or words to select from.

pos is a list of two integer numbers representing x and y position.

If *centre* is true, the *message* text (which may include new line characters) is centred; if false, the *message* is left-justified.

GUI programming / Standard Dialogs / `getSingleChoice`

size is a list of two numbers representing width and height.

Output the text of the selected choice.

Examples:

```
show (getSingleChoice [A Message][A Caption]
    [[Choice Nr. 1][Second Choice][Third Choice]])
```

`getSingleChoiceIndex` *message caption choices*
(**`getSingleChoiceIndex`** *message caption choices pos centre size*)

Operation to show a dialog to choose one of multiple *choices* .

message is the input prompt text.

caption is the window title text.

choices is a list of items which may be lists or words to select from.

pos is a list of two integer numbers representing x and y position.

If *centre* is true, the *message* text (which may include new line characters) is centred; if false, the *message* is left-justified.

size is a list of two numbers representing width and height.

Output is the zero-based index representing the selected string. If the user pressed cancel, -1 is returned.

Example:

```
show (getSingleChoiceIndex [A Message][A Caption]
    [[Choice Nr. 1][Second Choice][Third Choice]])
```

MessageBox *message*
(**MessageBox** *message caption style pos*)

Operation to show a dialog with a short *message* text and one or more buttons, depending on the *style* .

message is the MessageBox text.

caption is the window title text.

style may be a bit list of the following identifiers:

wxYES_NO Puts Yes and No buttons on the *message* box. May be combined with **wxCANCEL**.

wxCANCEL Puts a Cancel button on the *message* box. May be combined with **wxYES_NO** or **wxOK**.

wxOK Puts an Ok button on the *message* box. May be combined with **wxCANCEL**.

wxICON_EXCLAMATION Displays an exclamation mark symbol.

wxICON_HAND Displays an error symbol.

wxICON_ERROR Displays an error symbol - the same as **wxICON_HAND**.

wxICON_QUESTION Displays a question mark symbol.

wxICON_INFORMATION Displays an information symbol.

pos is a list of two integer numbers representing x and y position.

Output is one of the constants **wxYES** **wxNO** **wxCANCEL** **wxOK**.

Examples:

GUI programming / Standard Dialogs / MessageBox

```
show MessageBox [This is a Message]
show (MessageBox [This is a Message][A Caption]
      wxYes_No+wxCancel [400 200])
```

Frames

...are windows typically having a border around it and mostly a title bar, maybe with a system menu on it. You can place Controls or Graphs in a Frame, specifying the Frame as parent of the Control.

They are presented in the example frametest.lg.

Frames

- Frame 422
- FrameDestroy 424
- FrameOnChar 425
- FrameOnKeyDown 425
- FrameOnKeyUp 425
- FrameSetFocus 426
- FrameEnable 426
- FrameMaximize 427
- FrameIconize 427
- FrameFullScreen 427
- FrameSetClientSize 428
- FrameSetColor 428
- FrameSetBackgroundColor 429
- FrameSetFontSize 429
- FrameSetFontName 429
- FrameSetFontStyle 430
- FrameSetFontWeight 430
- FrameSetShape 430
- FrameSetSizer 431

Frame *parent title style pos size*
(Frame *parent title style*)
(Frame *parent title style pos size name*)

Operation which outputs a new Frame (a window with maybe a border around). This Frame can be assigned to a variable, and it will be deleted, when the variable is being deleted. So with local variables you can emulate Dialogs with Frames until those will be included into aUCBLogo.

GUI programming / Frames / Frame

parent can be a Frame or a Graph.

title is the window *title* text.

style is the Window Style and can be a combination (with +) of the following constants:

`wxDEFAULT_FRAME_STYLE` Defined as `wxMINIMIZE_BOX` | `wxMAXIMIZE_BOX` | `wxRESIZE_BORDER` | `wxSYSTEM_MENU` | `wxCAPTION` | `wxCLOSE_BOX` | `wxCLIP_CHILDREN`.

`wxICONIZE` Display the frame iconized (minimized). Windows only.

`wxCAPTION` Puts a caption on the frame.

`wxMINIMIZE` Identical to `wxICONIZE`. Windows only.

`wxMINIMIZE_BOX` Displays a minimize box on the frame.

`wxMAXIMIZE` Displays the frame maximized. Windows only.

`wxMAXIMIZE_BOX` Displays a maximize box on the frame.

`wxCLOSE_BOX` Displays a close box on the frame.

`wxSTAY_ON_TOP` Stay on top of all other windows, see also `wxFRAME_FLOAT_ON_PARENT`.

`wxSYSTEM_MENU` Displays a system menu.

`wxRESIZE_BORDER` Displays a resizable border around the window.

`wxFRAME_TOOL_WINDOW` Causes a frame with a small titlebar to be created; the frame does not appear in the taskbar under Windows or GTK+.

`wxFRAME_NO_TASKBAR` Creates an otherwise normal frame but it does not appear in the taskbar under Windows or GTK+ (note that it will minimize to the desktop window under Windows which may seem strange to the users and thus it might be better to use this *style* only without `wxMINIMIZE_BOX` *style*). In wxGTK, the flag is respected only if GTK+ is at least version 2.2 and the window manager supports `_NET_WM_STATE_SKIP_TASKBAR` hint. Has no effect

under other platforms.

`wxFRAME_FLOAT_ON_PARENT` The frame will always be on top of its *parent* (unlike `wxSTAY_ON_TOP`). A frame created with this *style* must have a non-NULL *parent* .

`wxFRAME_SHAPED` Windows with this *style* are allowed to have their shape changed with the `SetShape` method.

pos is a list of two integer numbers representing x and y position.

size is a list of two integer numbers representing width and height.

Like all window or control positions and sizes the value -1 for a coordinate means that `wxWidgets` will use the default value. This is especially useful when you use sizers, so you mostly don't need to figure out the coordinates yourself, for i.e. Buttons or other controls.

name is the internal *name* of the window, it's mostly unused.

Output is the created Frame.

Examples:

```
f=Frame [][MyFrame][][400 200][200 100]
f2=(Frame [][MyFrame2]
    wxDefault_Frame_Style+wxStay_on_Top
    [400 300][300 300])
```

```
f3=(Frame f2 [MyFrame3]
    wxDefault_Frame_Style+wxFRAME_float_on_parent+wxFRAME_shaped
    [500 400][260 300])
```

```
f4=(Frame f2 [MyFrame4]
    wxDefault_Frame_Style+wxFRAME_shaped+wxFRAME_float_on_parent
    [100 400][260 300]
    [noname])
```

```
OnChar [ern [f f2 f3 f4] GC OnChar []]
```

FrameDestroy *aframe*

Command to destroy the frame *aframe* . When a variable still points to the frame and you call another FrameXXX primitive with that var then an error occurs, because the frame has then been destroyed and cannot be used anymore.

Example:

```
f=Frame [][MyFrame][][400 200][200 100]
FrameDestroy f
```

FrameOnChar *aframe commands*

Command to set the custom event handler for the char event of the frame *aframe* to *commands* . See also OnChar!

Example:

```
f=Frame [][MyFrame][][400 200][200 100]
FrameOnChar f [pr KeyboardValue]
```

FrameOnKeyDown *aframe commands*

Command to set the custom event handler for the key down event of the frame *aframe* to *commands* . See also OnKeyDown!

Example:

```
f=Frame [][MyFrame][][400 200][200 100]
FrameOnKeyDown f [pr KeyboardValue]
```

FrameOnKeyUp *aframe commands*

Command to set the custom event handler for the key up (=release) event of the frame *aframe* to *commands* . See also OnKeyUp!

Example:

```
f=Frame [][MyFrame][][][400 200][200 100]  
FrameOnKeyUp f [pr KeyboardValue]
```

FrameSetFocus *aframe*

Command to set the keyboard and mouse input focus to the frame *aframe* .

Example:

```
f=Frame [][MyFrame][][][100 200][200 100]  
f2=Frame [][MyFrame][][][400 200][200 100]  
FrameSetFocus f
```

FrameEnable *aframe state*

Command to enable or disable the frame *aframe* .

state is a boolean. If *state* is false, *aframe* and all its children will be disabled, that means, no more user input is accepted.

Examples:

GUI programming / Frames / FrameEnable

```
f=Frame [][MyFrame] wxStay_on_top [100 200][200 100]
b=Button f [Close][ern "f  GC]
FrameEnable f false
;try to press the button - nothing happens!
FrameEnable f true
;try to press the button again - fd 100 executed!
```

FrameMaximize *aframe state*

command to maximize the frame *aframe* if *state* is true. Else the normal layout will be restored.

Example:

```
f=Frame [][MyFrame] wxDefault_Frame_Style [100 200][200 100]
FrameMaximize f true
FrameMaximize f false
```

FrameIconize *aframe state*

command to iconize the frame *aframe* if *state* is true. Else the normal layout will be restored.

Example:

```
f=Frame [][MyFrame] wxDefault_Frame_Style [100 200][200 100]
FrameIconize f true
FrameIconize f false
```

FrameFullScreen *aframe state style*

command to show the frame *aframe* fullscreen if *state* is true. Else the normal layout will be restored.

style can be a combination of the following constants, which indicate what elements of the window to hide in full-screen mode:

```
wxFULLSCREEN_NOMENUBAR  
wxFULLSCREEN_NOTOOLBAR  
wxFULLSCREEN_NOSTATUSBAR  
wxFULLSCREEN_NOBORDER  
wxFULLSCREEN_NOCAPTION  
wxFULLSCREEN_ALL (all of the above)
```

Example:

```
f=Frame [][MyFrame] wxDefault_Frame_Style [100 200][200 100]  
FrameFullScreen f true wxFULLSCREEN_ALL  
FrameFullScreen f false 0
```

FrameSetClientSize *aframe width height*

Command to set the client size of the Frame *aframe* . The client size is the interior of the frame, which is, what mostly has to be set instead of the complete window size in Frame.

width and *height* are integer numbers.

Example:

```
f=Frame [][MyFrame] wxStay_on_top [100 200][200 100]  
b=(Button f [Close][ern "f GC] 0 [-1 -1][100 40])  
FrameSetClientSize f 100 40
```

FrameSetColor *aframe acolor*

Command to set the foreground color of the Frame *aframe* to *acolor* , which must be a valid color (see SetPenColor!).

GUI programming / Frames / FrameSetColor

Example:

```
f=Frame [][MyFrame] wxdefault_frame_style [100 200][200 100]
FrameSetColor f "red"
ic=IntControl f [number one] 0 1234 4321 []
```

FrameSetBackgroundColor *aframe* *acolor*

Command to set the background color of the Frame *aframe* to *acolor* , which must be a valid color (see SetPenColor!).

Example:

```
f=Frame [][MyFrame] wxdefault_frame_style [100 200][200 100]
FrameSetBackgroundColor f rgb 1 0 0
b=(Button f [Close][ern "f GC] 0 [-1 -1][100 40])
```

FrameSetFontSize *aframe* *size*

Command to set the font *size* of the Frame *aframe* to *size* , which must be a integer number.

Example:

```
f=Frame [][MyFrame] wxdefault_frame_style [100 200][200 100]
FrameSetFontSize f 50
b=(Button f [Close][ern "f GC] 0 [-1 -1][200 70])
```

FrameSetFontName *aframe* *size*

Command to set the font name of the Frame *aframe* to *name*, which must be a valid font name.

Example:

```
f=Frame [][MyFrame] wxdefault_frame_style [100 200][200 100]
FrameSetFontName f [Courier]
b=(Button f [Close][ern "f  GC] 0 [-1 -1][200 70])
```

FrameSetFontStyle *aframe style*

Command to set the font *style* of the Frame *aframe* .

style can be one of the constants wxFONTSTYLE_NORMAL wxFONTSTYLE_SLANT wxFONTSTYLE_ITALIC.

Example:

```
f=Frame [][MyFrame] wxdefault_frame_style [100 200][200 100]
FrameSetFontStyle f wxFONTSTYLE_ITALIC
b=(Button f [Close][ern "f  GC] 0 [-1 -1][200 70])
```

FrameSetFontWeight *aframe weight*

Command to set the font *weight* of the Frame *aframe* .

weight can be one of the constants wxFONTWEIGHT_NORMAL wxFONTWEIGHT_LIGHT wxFONTWEIGHT_BOLD

Example:

```
f=Frame [][MyFrame] wxdefault_frame_style [100 200][200 100]
FrameSetFontWeight f wxFONTWEIGHT_BOLD
b=(Button f [Close][ern "f  GC] 0 [-1 -1][200 70])
```

GUI programming / Frames / FrameSetShape

FrameSetShape *aframe ashape*

Command to set the shape of the Frame *aframe* to the shape *ashape* .

ashape must be a list of positions.

A position in this context is a list of two integer numbers representing x and y. [0 0] is at the top left position of the client area.

Example:

```
f=(Frame [][MyFrame]
  wxdefault_frame_style+wxFRAME_SHAPED
  [100 200][200 100])
b=(Button f [Close][ern "f GC] 0 [-1 -1][200 100])
FrameSetClientSize f 200 100
FrameSetShape f [[0 0][200 0][100 100]]
```

FrameSetSizer *aframe asizer*

Command to set the sizer *asizer* for the Frame *aframe* . Currently only BoxSizer is supported as sizer.

Example:

```
f=Frame [][MyFrame] wxdefault_frame_style [100 200][100 200]
b1=Button f [Close][ern "f GC]
b2=Button f [fd 100][fd 100 updateGraph]
b3=Button f [back 100][back 100 updateGraph]
bs=BoxSizer wxVertical
BoxSizerAdd bs b1 100 wxExpand 0
BoxSizerAdd bs b2 200 wxExpand 10
BoxSizerAdd bs b3 200 wxExpand 20
FrameSetSizer f bs
```

Graphs

A Graph is a Logo drawing area, where you can draw anything Logo can draw in the main Graph window. Using a Graph and its event handlers you can write your own Control in Logo.

Graphs

- Graph 432
- GraphDestroy 433
- GraphCurrent 433
- GraphSetCurrent 434
- GraphOnChar 434
- GraphOnKeyDown 435
- GraphOnKeyUp 435
- GraphOnMouseLeftDown 435
- GraphOnMouseRightDown 436
- GraphOnMouseMiddleDown 436
- GraphOnMouseLeftUp 436
- GraphOnMouseRightUp 437
- GraphOnMouseMiddleUp 437
- GraphOnMouseLeftDClick 437
- GraphOnMouseRightDClick 437
- GraphOnMouseMiddleDClick 438
- GraphOnMouseMotion 438

Graph *parent*

(Graph *parent style pos size name*)

Operation which outputs a new Graph, a Logo drawing area.

parent can be a Frame or a Graph.

style can be a combination of the standard Window Styles, but probably will be just `wxFULL_REPAINT_ON_RESIZE`, which is the default value and will also be set if *style* is the empty list [].

pos is a list of two integer numbers representing x and y position.

GUI programming / Graphs / Graph

size is a list of two integer numbers representing width and height.

Like all window or control positions and sizes the value -1 for a coordinate means that wxWidgets will use the default value. This is especially useful when you use sizers, so you mostly don't need to figure out the coordinates yourself, for i.e. Buttons or other controls.

name is the internal *name* of the window, it's mostly unused.

Output is the created Graph.

Example:

```
f=(Frame [][MyFrame]
  wxResize_Border+wxCaption+wxSystem_Menu+wxClose_Box
  +wxFull_Repaint_on_Resize+wxStay_on_Top
  ;wxDefault_Frame_Style+wxStay_on_Top)
  [100 100][400 300])
FrameSetClientSize f 400 300
g=(Graph f
  wxDefault_Frame_Style+wxFull_Repaint_on_Resize+wxStay_on_Top
  [0 0][400 300][Graph])
```

GraphDestroy *agraph*

Command to destroy the Graph *agraph* . When a variable still points to the graph and you call another GraphXXX primitive with that var then an error occurs, because the graph has then been destroyed and cannot be used anymore. Sometimes you need to resize the parent frame a bit to clean rubbish from the graph, I don't know why but it helps.

Example:

```
f=Frame [][MyFrame][][100 100][400 300]
g=Graph f
FrameSetClientSize f 400 300
boundingbox
rbox
GraphDestroy g
```

GraphCurrent

outputs the currently active Graph.

Example:

```
g=(Graph GraphCurrent
    wxFull_Repaint_on_Resize
    [100 100][200 150])
boundingbox
rbox
```

GraphSetCurrent *agraph*

Command to set the Graph *agraph* as current graph.

Examples:

```
g=GraphCurrent
g1=(Graph g
    wxFull_Repaint_on_Resize
    [100 100][200 150])
g2=(Graph g
    wxFull_Repaint_on_Resize
    [400 100][200 150])
boundingbox
rbox
GraphSetCurrent g1
tree
GraphSetCurrent []
rboxes
```

GraphOnChar *agraph commands*

This command sets the Logo event handler for a char event of the Graph *agraph* . A char event is generated when a key is pressed or held down longer.

Example:

```
GraphOnChar GraphCurrent [pr KeyboardValue]
;activate the graph and type something!
GraphOnChar GraphCurrent [] ;reset
```

GraphOnKeyDown *agraph commands*

This command sets the Logo event handler for a key down event of the Graph *agraph* . A key down event is generated when a key is pressed or held down longer.

Example:

```
GraphOnKeyDown GraphCurrent [pr KeyboardValue]
;activate the graph and type something!
GraphOnKeyDown GraphCurrent [] ;reset
```

GraphOnKeyUp *agraph commands*

This command sets the Logo event handler for a key up event of the Graph *agraph* . A key up event is generated when a key is released from the pressed state.

Example:

```
GraphOnKeyUp GraphCurrent [pr KeyboardValue]
;activate the graph and type something!
GraphOnKeyUp GraphCurrent [] ;reset
```

GraphOnMouseLeftDown *agraph commands*

This command sets the Logo event handler for a mouse left button down event of the Graph window *agraph* .

Example:

```
g=(Graph GraphCurrent
  wxFull_Repaint_on_Resize
  [100 100][200 150])
GraphOnMouseLeftDown g [PenDown]
GraphOnMouseLeftUp g [PenUp]
GraphOnMouseMotion g [setPosXYZ MousePos  updateGraph]
```

GraphOnMouseRightDown *agraph commands*

This command sets the Logo event handler for a mouse right button down event of the Graph window *agraph* .

Example:

```
(GraphOnMouseRightDown GraphCurrent
  [setPosXYZ MousePos  fill  updateGraph])
```

GraphOnMouseMiddleDown *agraph commands*

This command sets the Logo event handler for a mouse middle button down event of the Graph window *agraph* .

Example:

```
GraphOnMouseMiddleDown GraphCurrent [pr MousePos]
```

GraphOnMouseLeftUp *agraph commands*

GUI programming / Graphs / GraphOnMouseLeftUp

This command sets the Logo event handler for a mouse left button up (=release) event of the Graph window *agraph* .

Example:

```
GraphOnMouseLeftUp GraphCurrent [pr [up!]]
```

GraphOnMouseRightUp *agraph commands*

This command sets the Logo event handler for a mouse right button up (=release) event of the Graph window *agraph* .

Example:

```
GraphOnMouseRightUp GraphCurrent [pr [up!]]
```

GraphOnMouseMiddleUp *agraph commands*

This command sets the Logo event handler for a mouse middle button up (=release) event of the Graph window *agraph* .

Example:

```
GraphOnMouseMiddleUp GraphCurrent [pr [up!]]
```

GraphOnMouseLeftDClick *agraph commands*

This command sets the Logo event handler for a mouse left button double click event of the Graph window *agraph* .

Example:

```
GraphOnMouseLeftDClick GraphCurrent [pr [doubleclick!]]
```

GraphOnMouseRightDClick *agraph commands*

This command sets the Logo event handler for a mouse right button double click event of the Graph window *agraph* .

Example:

```
GraphOnMouseRightDClick GraphCurrent [pr [doubleclick!]]
```

GraphOnMouseMiddleDClick *agraph commands*

This command sets the Logo event handler for a mouse middle button double click event of the Graph window *agraph* .

Example:

```
GraphOnMouseMiddleDClick GraphCurrent [pr [doubleclick!]]
```

GraphOnMouseMotion *agraph commands*

This command sets the Logo event handler for a mouse motion event of the Graph window *agraph* .

Example:

```
GraphOnMouseMotion GraphCurrent [setPosXYZ MousePos  
updateGraph]
```

BoxSizers

The main window layout mechanism of wxWidgets is supported with the constructor `BoxSizer` and the applicable primitives.

Be aware of the command `FrameSetSizer`: it enables a sizer on a `Frame`.

The best demo for the `BoxSizers` is probably `buttontest.lg`, because here you see the benefit of not having to figure out all the coordinates yourself, but letting this tedious work do the `BoxSizers`.

BoxSizers

- `BoxSizer` 439
- `BoxSizerAdd` 440
- `BoxSizerDestroy` 442

`BoxSizer orient`

outputs a new `BoxSizer` with the orientation *orient* .

orient can be either `wxHORIZONTAL` or `wxVERTICAL`.

Examples:

```

f=(Frame [][MyFrame]
    wxFrame_Tool_Window+wxCaption+wxClose_Box
    +wxSystem_Menu+wxResize_Border+wxTab_traversal+wxStay_on_Top
    [200 200][-1 -1])
bfd=(Button f [&forward]
[    forward 100
    updateGraph
])
blt=(Button f [&left]
[    left 30
    updateGraph
])
brt=(Button f [&right]
[    right 30
    updateGraph
])
bbk=(Button f [&back]
[    back 100
    updateGraph
])
bs=BoxSizer wxVertical
BoxSizerAdd bs bfd 100 wxExpand 0
bsrl=BoxSizer wxHorizontal
BoxSizerAdd bsrl blt 100 wxExpand 0
BoxSizerAdd bsrl brt 100 wxExpand 0
BoxSizerAdd bs bsrl 100 wxExpand 0
BoxSizerAdd bs bbk 100 wxExpand 0
FrameSetSizer f bs

```

BoxSizerAdd *aboxsizer acontrol proportion flag border*

Command to add the control *acontrol* to the BoxSizer *aboxsizer* .

proportion indicates, if a child of a sizer can change its size in the main orientation of the BoxSizer - where 0 stands for not changeable and a value of more than zero is interpreted relative to the value of other children of the same BoxSizer. For example, you might have a horizontal BoxSizer with three children, two of which are supposed to change their size with the sizer. Then

the two stretchable windows would get a value of 1 each to make them grow and shrink equally with the sizer's horizontal dimension. As another example you can have a BoxSizer with two children of whom the first shall be 150% of the size of the second, then you'd add the first child with a *proportion* of 150, and the second child with a *proportion* of 100.

flag can be used to set a number of flags which can be combined using the binary OR BitOr (or + in most cases). Two main behaviours are defined using these flags. One is the *border* around a window: the *border* parameter determines the *border* width whereas the flags given here determine which side(s) of the item that the *border* will be added. The other flags determine how the sizer item behaves when the space allotted to the sizer changes, and is somewhat dependent on the specific kind of sizer used.

wxTOP

wxBOTTOM

wxLEFT

wxRIGHT

wxALL These flags are used to specify which side(s) of the sizer item the *border* width will apply to.

wxEXPAND The item will be expanded to fill the space assigned to the item.

wxSHAPED The item will be expanded as much as possible while also maintaining its aspect ratio

wxFIXED_MINSIZE Normally wxSizers will use GetAdjustedBestSize to determine what the minimal size of window items should be, and will use that size to calculate the layout. This allows layouts to adjust when an item changes and its best size becomes different. If you would rather have a window item stay the size it started with then use wxFIXED_MINSIZE.

wxALIGN_CENTER

wxALIGN_LEFT

wxALIGN_RIGHT

wxALIGN_TOP

wxALIGN_BOTTOM

wxALIGN_CENTER_VERTICAL

wxALIGN_CENTER_HORIZONTAL The wxALIGN flags allow you to specify the alignment of the item within the space allotted to it by the sizer, adjusted for the *border* if any.

border determines the *border* width, if the *flag* parameter is set to include any *border flag* .

For examples see BoxSizer!

BoxSizerDestroy *aboxsizer*

Command to destroy the BoxSizer *aboxsizer* .

Buttons

...are little windows containing a text label, which can run a Logo instructionlist when the user clicks with the mouse on them. A good start to check out the ButtonXXX primitives is `buttontest.lg`.

Buttons

- Button 443
- ButtonDestroy 444
- ButtonOnClick 444
- ButtonEnable 444

Button *parent alabel onclick*
(Button *parent alabel onclick style apos size*)

outputs a new Button on the *parent* (a Frame or a Graph),

having the text *alabel* on it,

and running the *onclick* instructionlist when the user clicks with the mouse on it.

style can be a combination (use +) of the following constants:

`wxBU_LEFT` Left-justifies the label. Windows and GTK+ only.

`wxBU_TOP` Aligns the label to the top of the button. Windows and GTK+ only.

`wxBU_RIGHT` Right-justifies the bitmap label. Windows and GTK+ only.

`wxBU_BOTTOM` Aligns the label to the bottom of the button. Windows and GTK+ only.

`wxBU_EXACTFIT` Creates the button as small as possible instead of making it of the standard *size* (which is the default behaviour).

`wxNO_BORDER` Creates a flat button. Windows and GTK+ only.

apos is the position of the Button (a list of two integer numbers, x and y),

size is the *size* of the Button (again a list of two integer numbers, width and height).

Examples:

```
bfd=Button [][Go!][fd 100 updateGraph]
bback=(Button [][Go!][back 100 updateGraph]
        wxBU_LEFT+wxBU_TOP [0 100][200 100])
```

ButtonDestroy *abutton*

destroys the Button *abutton* .

Example:

```
bfd=Button [][Go!][fd 100 updateGraph]
ButtonDestroy bfd
```

ButtonOnClick *abutton instructionlist*

sets the onClick event handler of the Button *abutton* to the commands in the *instructionlist* .

Examples:

```
bfd=Button [][Go!][fd 100 updateGraph]
ButtonOnClick bfd [fd 10 updateGraph] ;slow motion!
ButtonOnClick bfd [] ;like disabling the Button
```

GUI programming / Buttons / ButtonEnable

ButtonEnable *abutton state*

if *state* is true then it enables the Button *abutton* , if *state* is false then it disables the Button.

Example:

```
bfd=Button [[Go!]][fd 100 updateGraph]
;Click on the Button - it goes
ButtonEnable bfd false
;Click on the Button again --nothing happens
ButtonEnable bfd true ;enabled again
```

CheckBoxes

...are little windows containing a check box and a text label, which can run a Logo instructionlist when the user clicks with the mouse on them and checks or unchecks them. CheckBoxes are demonstrated in `buttontest.lg`.

CheckBoxes

- `CheckBox` 446
- `CheckBoxDestroy` 447
- `CheckBoxOnClick` 447
- `CheckBoxValue` 448
- `CheckBoxSet` 448
- `CheckBoxEnable` 448

`CheckBox` *parent label onclick*
(**`CheckBox`** *parent label onclick style apos size*)

outputs a new `CheckBox` on the *parent* (a `Frame` or a `Graph`),

having the text *label* on it,

and running the *onclick* instructionlist when the user clicks with the mouse on it.

style can be a combination (use +) of the following constants:

`wxCHK_2STATE` Create a 2-state checkbox. This is the default.

`wxCHK_3STATE` Create a 3-state checkbox. Not implemented in `wxMGL`, `wxOS2` and `wxGTK` built against `GTK+ 1.2`.

`wxCHK_ALLOW_3RD_STATE_FOR_USER` By default a user can't set a 3-state checkbox to the third state. It can only be done from code. Using this flags allows the user to set the checkbox to the third state by clicking.

GUI programming / CheckBoxes / CheckBox

`wxALIGN_RIGHT` Makes the text appear on the left of the checkbox.

apos is the position of the Button (a list of two integer numbers, x and y),

size is the *size* of the Button (again a list of two integer numbers, width and height).

Examples:

```
cbpd=CheckBox [][Pen Down][
    ifelse CheckBoxValue cbpd [PenDown][PenUp]
    updateGraph]
cbht=(CheckBox [][Hide Turtle][
    ifelse CheckBoxValue cbht [hideTurtle][showTurtle]
    updateGraph]
wxBU_LEFT+wxBU_TOP [0 100][200 100])
```

CheckBoxDestroy *checkbox*

Command that destroys the CheckBox *checkbox* .

Example:

```
cbht=CheckBox [][Hide Turtle][
    ifelse CheckBoxValue cbht [hideTurtle][showTurtle]
    updateGraph]
CheckBoxDestroy cbht
```

CheckBoxOnClick *checkbox instructionlist*

Command that sets the onClick event handler of the CheckBox *checkbox* to the commands in the *instructionlist* .

Examples:

```
cb=CheckBox [][Hide Turtle][
    ifelse CheckBoxValue cb [hideTurtle][showTurtle]
    updateGraph]
CheckBoxOnClick cb [
    ifelse CheckBoxValue cb [pr [Yes, Sir!]][pr [No, Sir!]]
    updateGraph]
CheckBoxOnClick cb [] ;like disabling the CheckBox
```

CheckBoxValue *checkbox*

outputs true if the CheckBox *checkbox* is checked, false otherwise.

Example:

```
cb=CheckBox [][Hide Turtle][
    ifelse CheckBoxValue cb [hideTurtle][showTurtle]
    updateGraph]
```

CheckBoxSet *checkbox state*

Command that sets the CheckBox *checkbox* to checked if *state* is true, to unchecked if false.

state must be a boolean.

Example:

```
cb=CheckBox [][Hide Turtle][
    ifelse CheckBoxValue cb [hideTurtle][showTurtle]
    updateGraph]
CheckBoxSet bc shown?
```

CheckBoxEnable *checkbox state*

Command. If *state* is true then it enables the CheckBox *checkbox*, if *state* is false then it disables the CheckBox.

state must be a boolean.

Example:

```
cb=CheckBox [][Hide Turtle][
  ifelse CheckBoxValue cb [hideTurtle][showTurtle]
  updateGraph]
CheckBoxEnable bfd false
;Click on the CheckBox again --nothing happens
CheckBoxEnable bfd true ;enabled again
```

ChoiceBoxes

...are little windows containing a text label and a pulldown menulike list of text choices, which can run a Logo instructionlist when the user clicks with the mouse on them and selects a choice, and when a key event is received. The corresponding demo is `choiceboxtest.lg`.

ChoiceBoxes

- ChoiceBox 450
- ChoiceBoxDestroy 451
- ChoiceBoxSelection 451
- ChoiceBoxSetSelection 452
- ChoiceBoxSetChoices 452
- ChoiceBoxAppend 453
- ChoiceBoxSetItem 453
- ChoiceBoxRemoveItem 454
- ChoiceBoxCount 454
- ChoiceBoxSetBackgroundColor 455
- ChoiceBoxSetColor 455
- ChoiceBoxSetFontSize 456
- ChoiceBoxSetFontName 456
- ChoiceBoxSetFontStyle 457
- ChoiceBoxSetFontWeight 457
- ChoiceBoxOnChar 458
- ChoiceBoxOnKeyDown 458
- ChoiceBoxOnKeyUp 459
- ChoiceBoxOnSelect 459
- ChoiceBoxEnable 460

ChoiceBox *parent name choices onSelect*

(ChoiceBox *parent name choices onSelect style pos size*)

outputs a new ChoiceBox and shows it on the *parent* (a Frame or a Graph, main Graph if []). Unlike a listbox, only the selection is visible until the user pulls down the menu of *choices* .

name is its label text.

GUI programming / ChoiceBoxes / ChoiceBox

choices is a list of items from whom the user can select one.

onSelect is a Logo instructionlist which will be run when the user selects a choice.

style is the window *style* , nothing special here, it defaults to zero.

pos is the position of the ChoiceBox (a list of two integer numbers, x and y),

size is the *size* of the ChoiceBox (a list of two integer numbers, width and height).

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[  [String 1]
    [and a second string]
    [and the last string]
]
[  (show "|MyChoiceBox| ChoiceBoxSelection)
])
```

ChoiceBoxDestroy *achoicebox*

Command that destroys the ChoiceBox *achoicebox* .

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[  [String 1]
    [and a second string]
    [and the last string]
]
[  (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxDestroy cb
```

ChoiceBoxSelection**(ChoiceBoxSelection *achoicebox*)**

outputs the selection of the ChoiceBox *achoicebox* , or if called without an argument, of the ChoiceBox that processed the last OnSelect event.

The output value is a zero-based integer, representing the nth choice.

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
```

ChoiceBoxSetSelection *achoicebox selection*

Command to set the *selection* of the ChoiceBox *achoicebox* to the value *selection* .

selection is a zero-based integer index into the choices, which must be less than the number of choices in the control.

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxSetSelection cb 2
```

ChoiceBoxSetChoices *achoicebox choices*

Command to set the contents of the ChoiceBox *achoicebox* to *choices* .

choices must be a list of items, whose texts will be the *choices* in the ChoiceBox.

Example:

```
cb=(ChoiceBox [][MyChoiceBox][ ]
[ (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxSetChoices cb [
    [String 1]
    [and a second string]
    [and the last string]
]
```

ChoiceBoxAppend *achoicebox choice*

Command to append the text of the thing *choice* to the choices of ChoiceBox *achoicebox* .

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[ [String 1]
  [and a second string]
]
[ (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxAppend cb [and the last string]
```

ChoiceBoxSetItem *achoicebox index choice*

Command to change the item at position *index* of the ChoiceBox *achoicebox* to the text of the thing *choice* .

index must be an integer number $index \geq 0$ and $index < \text{number of choices in the ChoiceBox}$.

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[  [String 1]
    [and a second string]
    [and the last string]
]
[  (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxSetItem cb 1 [a changed item]
```

ChoiceBoxRemoveItem *achoicebox index*

Command to remove the choice item at position *index* from the ChoiceBox *achoicebox* .

index must be an integer number $index \geq 0$ and $index < \text{number of choices in the ChoiceBox}$.

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[  [String 1]
    [and a second string]
    [and the last string]
]
[  (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxRemoveItem cb 1
```

ChoiceBoxCount *achoicebox*

GUI programming / ChoiceBoxes / ChoiceBoxCount

outputs the number of choices in the ChoiceBox *achoicebox* .

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
show ChoiceBoxCount cb
```

ChoiceBoxSetBackgroundColor *achoicebox color*

Command to set the background *color* of the ChoiceBox *achoicebox* to the *color color* .

color must be a valid *color* , see also setPenColor!

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxSetBackgroundColor cb "red
```

ChoiceBoxSetColor *achoicebox color*

Command to set the foreground *color* of the ChoiceBox *achoicebox* to the *color color* .

color must be a valid *color* , see also setPenColor!

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxSetColor cb "red"
```

ChoiceBoxSetFontSize *achoicebox size*

Command to set the font *size* of the ChoiceBox *achoicebox* to *size* , which must be a integer number.

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxSetFontSize cb 50
```

ChoiceBoxSetFontName *achoicebox name*

Command to set the font *name* of the ChoiceBox *achoicebox* to *name* , which must be a valid font *name* .

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxSetFontName cb [Courier]
```

ChoiceBoxSetFontStyle *achoicebox style*

Command to set the font *style* of the ChoiceBox *achoicebox* .

style can be one of the constants wxFONTSTYLE_NORMAL wxFONTSTYLE_SLANT wxFONTSTYLE_ITALIC.

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxSetFontStyle cb wxFONTSTYLE_ITALIC
```

ChoiceBoxSetFontWeight *achoicebox weight*

Command to set the font *weight* of the ChoiceBox *achoicebox* .

weight can be one of the constants wxFONTWEIGHT_NORMAL wxFONTWEIGHT_LIGHT

wxFONTWEIGHT_BOLD

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxSetFontWeight cb wxFONTWEIGHT_BOLD
```

ChoiceBoxOnChar *achoicebox commands*

Command to set the custom event handler for the char event of the ChoiceBox *achoicebox* to *commands* . See also OnChar!

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxOnChar cb [pr KeyboardValue]
```

ChoiceBoxOnKeyDown *achoicebox commands*

Command to set the custom event handler for the key down event of the ChoiceBox *achoicebox* to *commands* . See also OnKeyDown!

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxOnKeyDown cb [pr KeyboardValue]
```

ChoiceBoxOnKeyUp *achoicebox commands*

Command to set the custom event handler for the key up (=release) event of the ChoiceBox *achoicebox* to *commands* . See also OnKeyUp!

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxOnKeyUp cb [pr KeyboardValue]
```

ChoiceBoxOnSelect *achoicebox commands*

Command to set the custom event handler for the select event of the ChoiceBox *achoicebox* to *commands* . The select event is generated when the user selects a choice with a mouseclick or a keypress.

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxOnSelect cb [pr ChoiceBoxSelection]
```

ChoiceBoxEnable *achoicebox state*

Command. If *state* is true then it enables the ChoiceBox *achoicebox* , if *state* is false then it disables the ChoiceBox.

state must be a boolean.

Example:

```
cb=(ChoiceBox [][MyChoiceBox]
[   [String 1]
    [and a second string]
    [and the last string]
]
[   (show "|MyChoiceBox| ChoiceBoxSelection)
])
ChoiceBoxEnable cb false
;Click on the ChoiceBox again --nothing happens
ChoiceBoxEnable cb true ;enabled again
```

ComboBoxes

A ComboBox is like a combination of an edit control and a ListBox. It can be displayed as static list with editable or read-only text field; or a drop-down list with text field; or a drop-down list without a text field.

A ComboBox permits a single selection only. ComboBox items are numbered from zero.

A ComboBox can run a Logo instructionlist when the user clicks with the mouse on an item and selects a choice, when the user changes the text or presses [Enter], and when any key event is received. The corresponding demo is comboboxtest.lg.

ComboBoxes

- ComboBox 461
 - ComboBoxDestroy 463
 - ComboBoxSelection 463
 - ComboBoxSetSelection 464
 - ComboBoxSetChoices 465
 - ComboBoxAppend 466
 - ComboBoxSetItem 466
 - ComboBoxRemoveItem 467
 - ComboBoxCount 468
 - ComboBoxValue 468
 - ComboBoxSetValue 469
 - ComboBoxSetBackgroundColor 470
 - ComboBoxSetColor 471
 - ComboBoxSetFontSize 472
 - ComboBoxSetFontName 473
 - ComboBoxSetFontStyle 474
 - ComboBoxSetFontWeight 475
 - ComboBoxOnChar 476
 - ComboBoxOnKeyDown 477
 - ComboBoxOnKeyUp 478
 - ComboBoxOnSelect 479
 - ComboBoxOnChange 480
 - ComboBoxOnEnter 481
 - ComboBoxEnable 482
-

ComboBox *parent name text choices onSelect onChange onEnter*
(ComboBox *parent name text choices*)
(ComboBox *parent name text choices onSelect onChange onEnter style pos size*)

outputs a new ComboBox and shows it on the *parent* (a Frame or a Graph, main Graph if []).

name is its label *text* .

text is the standard *text* shown in the edit control when the ComboBox is created.

choices is a list of items from whom the user can select one.

onSelect is a Logo instructionlist which will be run when the user selects a choice.

onChange is a Logo instructionlist which will be run when the user changes the *text* in the edit control.

onEnter is a Logo instructionlist which will be run when the user presses the [Enter] key.

style is the window *style* and can be a combination of the following constants:

wxCB_SIMPLE Creates a ComboBox with a permanently displayed list. Windows only.

wxCB_DROPDOWN Creates a ComboBox with a drop-down list.

wxCB_READONLY Same as wxCB_DROPDOWN but only the strings specified as the ComboBox *choices* can be selected, it is impossible to select (even from a program) a string which is not in the *choices* list.

wxCB_SORT Sorts the entries in the list alphabetically.

pos is the position of the ComboBox (a list of two integer numbers, x and y),

size is the *size* of the ComboBox (a list of two integer numbers, width and height).

Example:

```

cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])

```

ComboBoxDestroy *acombox*

Command that destroys the ComboBox *acombox* .

Example:

```

cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxDestroy cb

```

ComboBoxSelection**(ComboBoxSelection *acombo*box)**

outputs the selection of the ComboBox *acombo*box , or if called without an argument, of the ComboBox that processed the last ComboBox event.

The output value is a zero-based integer, representing the nth choice.

Example:

```
cb=(ComboBox [][MyComboBox][Initial string]
  [ [String 1]
    [and a second string]
    [and the last string]
  ]
  [ (show "|MyComboBox| ComboBoxSelection)
  ]
  [ (pr [change] ComboBoxValue)
  ]
  [ v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
```

ComboBoxSetSelection *acombo*box *selection*

Command to set the *selection* of the ComboBox *acombo*box to the value *selection* .

selection is a zero-based integer index into the choices, which must be less than the number of choices in the control.

Example:

```

cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxSetSelection cb 1

```

ComboBoxSetChoices *acombox choices*

Command to set the contents of the ComboBox *acombox* to *choices* .

choices must be a list of items, whose texts will be the *choices* in the ComboBox.

Example:

```

cb=(ComboBox [][MyComboBox][Initial string][
  (show "|MyComboBox| ComboBoxSelection)
]
  (pr [change] ComboBoxValue)
]
  v=ComboBoxValue
  (pr [enter] v)
  ComboBoxAppend cb v
]
wxCB_simple [-1 -1][300 200])
ComboBoxSetChoices cb [
  [String 1]
  [and a second string]
  [and the last string]
]

```

ComboBoxAppend *acombox choice*

Command to append the text of the thing *choice* to the choices of ComboBox *acombox* .

Example:

```

cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
]
  (pr [change] ComboBoxValue)
]
  v=ComboBoxValue
  (pr [enter] v)
  ComboBoxAppend cb v
]
wxCB_simple [-1 -1][300 200])

```

ComboBoxSetItem *acombox index choice*

Command to change the item at position *index* of the ComboBox *acombox* to the text of the thing *choice* .

index must be an integer number $index \geq 0$ and $index < (\text{number of choices in the ComboBox})$.

Example:

```
cb=(ComboBox [][MyComboBox][Initial string]
  [  [String 1]
    [and a second string]
    [and the last string]
  ]
  [  (show "|MyComboBox| ComboBoxSelection)
  ]
  [  (pr [change] ComboBoxValue)
  ]
  [  v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxSetItem cb 1 [Item 1 now]
```

ComboBoxRemoveItem *acombox index*

Command to remove the choice item at position *index* from the ComboBox *acombox* .

index must be an integer number $index \geq 0$ and $index < (\text{number of choices in the ComboBox})$.

Example:

```

cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxRemoveItem cb 1

```

ComboBoxCount *acombox*

outputs the number of choices in the ComboBox *acombox* .

Example:

```

cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
show ComboBoxCount cb

```

ComboBoxValue**(ComboBoxValue *acombox*)**

outputs the text in the edit control of the ComboBox *acombox* or if called without argument, of the ComboBox whose event is being processed lately.

Example:

```
cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  [
    (show "|MyComboBox| ComboBoxSelection)
  ]
  [
    (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
```

ComboBoxSetValue *acombox text*

Command to set the *text* in the edit control of the ComboBox *acombox* to *text* which will be converted to a word first.

Example:

```
cb=(ComboBox [][MyComboBox][Initial string]
  [  [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [  v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxSetValue cb [Hallo World!]
```

ComboBoxSetBackgroundColor *acombox color*

Command to set the background *color* of the ComboBox *acombox* to the *color color* .

color must be a valid *color* , see also setPenColor!

Example:

```

cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxSetBackgroundColor cb "red

```

ComboBoxSetColor *acombox color*

Command to set the foreground *color* of the ComboBox *acombox* to the *color color* .

color must be a valid *color* , see also setPenColor!

Example:

```
cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxSetColor cb "red"
```

ComboBoxSetFontSize *acombox size*

Command to set the font *size* of the ComboBox *acombox* to *size* , which must be a integer number.

Example:

```

cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxSetFontSize cb 50

```

ComboBoxSetFontName *acombox name*

Command to set the font *name* of the ComboBox *acombox* to *name* , which must be a valid font *name* .

Example:

```
cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxSetFontName cb [Courier]
```

ComboBoxSetFontStyle *acombox style*

Command to set the font *style* of the ComboBox *acombox* .

style can be one of the constants wxFONTSTYLE_NORMAL wxFONTSTYLE_SLANT wxFONTSTYLE_ITALIC.

Example:

```

cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxSetFontStyle cb wxFONTSTYLE_ITALIC

```

ComboBoxSetFontWeight *acombox weight*

Command to set the font *weight* of the ComboBox *acombox* .

weight can be one of the constants wxFONTWEIGHT_NORMAL wxFONTWEIGHT_LIGHT wxFONTWEIGHT_BOLD

Example:

```
cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxSetFontWeight cb wxFONTWEIGHT_BOLD
```

ComboBoxOnChar *acombox commands*

Command to set the custom event handler for the char event of the ComboBox *acombox* to *commands* . See also OnChar!

Example:


```

cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxOnChar cb [pr KeyboardValue]

```

ComboBoxOnKeyDown *acombox commands*

Command to set the custom event handler for the key down event of the ComboBox *acombox* to *commands* . See also OnKeyDown!

Example:

```
cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxOnKeyDown cb [pr KeyboardValue]
```

ComboBoxOnKeyUp *acombox commands*

Command to set the custom event handler for the key up (=release) event of the ComboBox *acombox* to *commands* . See also OnKeyUp!

Example:

```

cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxOnKeyUp cb [pr KeyboardValue]

```

ComboBoxOnSelect *acombox commands*

Command to set the custom event handler for the select event of the ComboBox *acombox* to *commands* . The select event is generated when the user selects a choice with a mouseclick or a keypress.

Example:

```
cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxOnSelect cb [pr ComboBoxSelection]
```

ComboBoxOnChange *acombox commands*

Command to set the custom event handler for the change text event of the ComboBox *acombox* to *commands* . The change text event is generated when the user types some text into the edit control.

Example:

```
cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxOnChange cb [pr ComboBoxValue]
```

ComboBoxOnEnter *acombox commands*

Command to set the custom event handler for the enter key event of the ComboBox *acombox commands* . The text enter event is generated when the user presses the enter or the return key.

Example:

```
cb=(ComboBox [][MyComboBox][Initial string]
  [
    [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
  ]
  (pr [change] ComboBoxValue)
  ]
  [
    v=ComboBoxValue
    (pr [enter] v)
    ComboBoxAppend cb v
  ]
  wxCB_simple [-1 -1][300 200])
ComboBoxOnEnter cb [pr ComboBoxValue]
```

ComboBoxEnable *acombox* *state*

Command. If *state* is true then it enables the ComboBox *acombox* , if *state* is false then it disables the ComboBox.

state must be a boolean.

Example:

GUI programming / ComboBoxes / ComboBoxEnable

```
cb=(ComboBox [][MyComboBox][Initial string]
  [  [String 1]
    [and a second string]
    [and the last string]
  ]
  (show "|MyComboBox| ComboBoxSelection)
]
(pr [change] ComboBoxValue)
]
[  v=ComboBoxValue
  (pr [enter] v)
  ComboBoxAppend cb v
]
wxCB_simple [-1 -1][300 200])
ComboBoxEnable cb false
;Click on the ComboBox again --nothing happens
ComboBoxEnable cb true ;enabled again
```

FloatControls

A FloatControl is a little window containing a text label, an edit control for use with floating point numbers, and a spinbutton. It holds the value of a floating point number, which can be changed by the user by either typing, or with the arrow keys, or by clicking on the spinbutton with the mouse.

FloatControl is demonstrated in floatcontroltest.lg.

FloatControls

- FloatControl 484
- FloatControlDestroy 485
- FloatControlValue 486
- FloatControlSetValue 486
- FloatControlSetRange 487
- FloatControlOnChange 487
- FloatControlEnable 488

FloatControl *parent name min value max increment digits onChange*
(FloatControl *parent name min value max increment* **)**
(FloatControl *parent name min value max increment digits onChange format style pos*
size **)**

outputs a new FloatControl on the *parent* (a Frame or a Graph, main Graph if []).

name is the label text and the *name* of the control.

min is the minimal allowed *value* that the control accepts.

value is the number shown at startup of the control.

max is the maximal allowed *value* that the control accepts.

increment is the amount by which the arrow keys increase or decrease the *value* .

min , *value* , *max* and *increment* are Logo numbers that will be converted to Float's internally.

GUI programming / FloatControls / FloatControl

digits are the number of *digits* of the number behind the dot. It's an integer number and defaults to 3 if not supplied.

onChange is a command list that will be run if the OnChange event of the control is processed. That's whenever the number in the control changes anyhow.

format is an integer between 1 and 3, meaning 1="%f" (floating point), 2="%e" (exponential) and 3="%g" (Signed *value* printed in f or e *format* , whichever is more compact for the given *value* and precision. The e *format* is used only when the exponent of the *value* is less than 4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more *digits* follow it.).

style is the window *style* and can be the special *value* wxSP_ARROW_KEYS to which it defaults.

pos is the position of the ComboBox (a list of two integer numbers, x and y),

size is the *size* of the ComboBox (a list of two integer numbers, width and height).

Example:

```
fctx=(FloatControl [] [X] -400 0 400 1.01 5
      [   x=FloatControlValue
          ;(pr "x= x)
          setX x
          updateGraph
      ] 2 0 [10 20] [-1 -1])
```

FloatControlDestroy *afloatcontrol*

Command to destroy the FloatControl *afloatcontrol* .

Example:

```
fcx=(FloatControl [] [X] -400 0 400 1.01 5
    [   x=FloatControlValue
        ;(pr "x= x)
        setX x
        updateGraph
    ] 2 0 [10 20][ -1 -1])
FloatControlDestroy fcx
```

FloatControlValue**(FloatControlValue *afloatcontrol*)**

outputs the value of the FloatControl *afloatcontrol* or if called without arguments, of the FloatControl whose event is being processed lately.

Example:

```
fcx=(FloatControl [] [X] -400 0 400 1.01 5
    [   x=FloatControlValue
        ;(pr "x= x)
        setX x
        updateGraph
    ] 2 0 [10 20][ -1 -1])
```

FloatControlSetValue *afloatcontrol value*

Command to set the *value* of the FloatControl *afloatcontrol* to the number *value* .

Example:

GUI programming / FloatControls / FloatControlSetValue

```
fcx=(FloatControl [] [X] -400 0 400 1.01 5
    [   x=FloatControlValue
        ;(pr "x= x)
        setX x
        updateGraph
    ] 2 0 [10 20][-1 -1])
FloatControlSetValue fcx 123.456
```

FloatControlSetRange *afloatcontrol min max*

Command to set the range of valid numbers of the FloatControl *afloatcontrol* to stay between the numbers *min* and *max* inclusive.

Example:

```
fcx=(FloatControl [] [X] -400 0 400 1.01 5
    [   x=FloatControlValue
        ;(pr "x= x)
        setX x
        updateGraph
    ] 2 0 [10 20][-1 -1])
FloatControlSetRange fcx -10.11 20.22
```

FloatControlOnChange *afloatcontrol commands*

Command to set the OnChange event handler of the FloatControl *afloatcontrol* to the commandlist *commands* .

Example:

```
fcx=(FloatControl [] [X] -400 0 400 1.01 5
      [] 2 0 [10 20][-1 -1])
FloatControlOnChange fcx [
    x=FloatControlValue
    ;(pr "x= x)
    setX x
    updateGraph
]
```

FloatControlEnable *afloatcontrol state*

Command. If *state* is true then it enables the FloatControl *afloatcontrol* , if *state* is false then it disables the FloatControl.

state must be a boolean.

Example:

```
fcx=(FloatControl [] [X] -400 0 400 1.01 5
      [    x=FloatControlValue
          ;(pr "x= x)
          setX x
          updateGraph
      ] 2 0 [10 20][-1 -1])
FloatControlEnable fcx false
;Click on the FloatControl again --nothing happens
FloatControlEnable fcx true ;enabled again
```

Gauges

A Gauge is a little window containing a colored bar, that takes up some percentage of the width or height of the Control. It shows the value of a integer number, represented by the size of the bar. Gauge is read-only.

Gauge is demonstrated in `gaugetest.lg`.

Gauges

- Gauge 489
- GaugeDestroy 490
- GaugeValue 490
- GaugeSetValue 490
- GaugeSetRange 491
- GaugeSetColor 491
- GaugeSetBackgroundColor 491

Gauge *parent name range*

(Gauge *parent name range value style pos size*)

outputs a new Gauge on the *parent* (a Frame or a Graph, main Graph if []).

name is the *name* of the control.

range is the maximal allowed *value* that the control accepts.

value is the number shown at startup of the control.

range and *value* must be integer numbers.

style is the window *style* and can be a combination of (wxGA_HORIZONTAL or wxGA_VERTICAL) with wxGA_SMOOTH.

wxGA_HORIZONTAL Creates a horizontal Gauge.

wxGA_VERTICAL Creates a vertical Gauge.

wxGA_SMOOTH Creates smooth progress bar with one pixel wide update step (not supported by all platforms).

pos is the position of the Gauge (a list of two integer numbers, x and y),

size is the *size* of the Gauge (a list of two integer numbers, width and height).

Example:

```
g=Gauge [][Gauge1] 400
g2=(Gauge [][Gauge1] 400 100 wxGA_horizontal+wxGA_smooth
    [0 200])
```

GaugeDestroy *agauge*

Command to destroy the Gauge *agauge* .

Example:

```
g=Gauge [][Gauge1] 400
GaugeDestroy g
```

GaugeValue *agauge*

outputs the integer value of the Gauge *agauge* .

Example:

```
g=(Gauge [][Gauge1] 400 123 wxGA_smooth)
show GaugeValue g
```

GaugeSetValue *agauge value*

GUI programming / Gauges / GaugeSetValue

Command to set the *value* of the Gauge *agauge* to the integer *value* .

Example:

```
g=(Gauge [][Gauge1] 400 123 wxGA_smooth)
GaugeSetValue g 321
```

GaugeSetRange *agauge max*

Command to set the range of valid numbers of the Gauge *agauge* to stay between 0 and the integer number *max* .

Example:

```
g=(Gauge [][Gauge1] 400 123 wxGA_smooth)
GaugeSetRange g 333
```

GaugeSetColor *agauge color*

Command to set the foreground *color* of the Gauge *agauge* to the *color color* .

color must be a valid *color* , see also setPenColor!

Example:

```
g=(Gauge [][Gauge1] 400 123 wxGA_smooth)
GaugeSetColor g "red"
```

GaugeSetBackgroundColor *agauge color*

Command to set the background *color* of the Gauge *agauge* to the *color color* .

color must be a valid *color* , see also setPenColor!

Example:

```
g=(Gauge [][Gauge1] 400 123 wxGA_smooth)
GaugeSetBackgroundColor g "blue"
```

IntControls

A IntControl is a little window containing a text label, an edit control for use with integer numbers, and a spinbutton. It holds the value of an integer number, which can be changed by the user by either typing, or with the arrow keys, or by clicking on the spinbutton with the mouse.

IntControl is demonstrated in intcontroltest.lg.

IntControls

- IntControl 493
- IntControlDestroy 494
- IntControlValue 494
- IntControlSetValue 494
- IntControlSetRange 495
- IntControlOnChange 495
- IntControlEnable 495

IntControl *parent name min value max onChange*

(IntControl *parent name min value max*)

(IntControl *parent name min value max onChange style pos size*)

outputs a new IntControl on the *parent* (a Frame or a Graph, main Graph if []).

name is the label text and the *name* of the control.

min is the minimal allowed *value* that the control accepts.

value is the number shown at startup of the control.

max is the maximal allowed *value* that the control accepts.

min , *value* and *max* must be integer numbers.

onChange is a command list that will be run if the OnChange event of the control is processed. That's whenever the number in the control changes anyhow.

style is the window *style* and can be the special *value* `wxSP_ARROW_KEYS` to which it defaults.

pos is the position of the ComboBox (a list of two integer numbers, x and y),

size is the *size* of the ComboBox (a list of two integer numbers, width and height).

Examples:

```
[    y=IntControlValue
    (pr "y= y)
] wxSP_Arrow_Keys [10 100][80 -1])
```

IntControlDestroy *aintcontrol*

Command to destroy the IntControl *aintcontrol* .

Example:

```
ic=(IntControl [][X] -400 0 400 [x=IntControlValue (pr "x= x)])
IntControlDestroy ic
```

IntControlValue (IntControlValue *aintcontrol*)

outputs the integer value of the IntControl *aintcontrol* or if called without arguments, of the IntControl whose event is being processed lately.

Example:

```
ic=(IntControl [][X] -400 0 400 [x=IntControlValue (pr "x= x)])
```

IntControlSetValue *aintcontrol value*

GUI programming / IntControls / IntControlSetValue

Command to set the *value* of the IntControl *aintcontrol* to the integer number *value* .

Example:

```
ic=(IntControl [][X] -400 0 400 [x=IntControlValue (pr "x= x)])
IntControlSetValue ic 123.456
```

IntControlSetRange *aintcontrol min max*

Command to set the range of valid numbers of the IntControl *aintcontrol* to stay between the numbers *min* and *max* inclusive.

Example:

```
ic=(IntControl [][X] -400 0 400 [x=IntControlValue (pr "x= x)])
IntControlSetRange ic -10.11 20.22
```

IntControlOnChange *aintcontrol commands*

Command to set the OnChange event handler of the IntControl *aintcontrol* to the commandlist *commands* .

Example:

```
ic=IntControl [][X] -400 0 400 []
IntControlOnChange ic [
    x=IntControlValue
    ;(pr "x= x)
    setX x
    updateGraph
]
```

IntControlEnable *aintcontrol state*

Command. If *state* is true then it enables the IntControl *aintcontrol* , if *state* is false then it disables the IntControl.

state must be a boolean.

Example:

```
ic=(IntControl [][X] -400 0 400 [x=IntControlValue (pr "x= x)])
IntControlEnable fcx false
;Click on the FloatControl again --nothing happens
IntControlEnable fcx true ;enabled again
```

ListBoxes

...are little windows containing a text label and a list of text choices, which can run a Logo instructionlist when the user clicks with the mouse on them and selects a choice, and when a key event is received.

A listbox is used to select one or more of a list of strings. The strings are displayed in a scrolling box, with the selected string(s) marked in reverse video. A listbox can be single selection (if an item is selected, the previous selection is removed) or multiple selection (clicking an item toggles the item on or off independently of other selections).

List box elements are numbered from zero. Their number is limited in some platforms (e.g. ca. 2000 on GTK).

The corresponding demo is listboxtest.lg.

ListBoxes

- ListBox 497
 - ListBoxDestroy 499
 - ListBoxSelections 499
 - ListBoxSetSelections 500
 - ListBoxSetChoices 500
 - ListBoxAppend 500
 - ListBoxSetItem 501
 - ListBoxRemoveItem 501
 - ListBoxCount 501
 - ListBoxSetBackgroundColor 502
 - ListBoxSetColor 502
 - ListBoxSetFontSize 502
 - ListBoxSetFontName 503
 - ListBoxSetFontStyle 503
 - ListBoxSetFontWeight 503
 - ListBoxOnChar 504
 - ListBoxOnKeyDown 504
 - ListBoxOnKeyUp 504
 - ListBoxOnSelect 505
 - ListBoxOnDClick 505
 - ListBoxEnable 505
-

ListBox *parent name choices onSelect onDClick*
(ListBox *parent name choices*)
(ListBox *parent name choices onSelect onDClick style pos size*)

outputs a new ListBox and shows it on the *parent* (a Frame or a Graph, main Graph if []).

name is its label text.

choices is a list of items from whom the user can select one.

onSelect is a Logo instructionlist which will be run when the user selects a choice.

onDClick is a Logo instructionlist which will be run when the user double-clicks on a choice.

style is the window *style* and can be a combination of the following constants:

wxLB_SINGLE Single-selection list.

wxLB_MULTIPLE Multiple-selection list: the user can toggle multiple items on and off.

wxLB_EXTENDED Extended-selection list: the user can select multiple items using the SHIFT key and the mouse or special key combinations.

wxLB_HSCROLL Create horizontal scrollbar if contents are too wide (Windows only).

wxLB_ALWAYS_SB Always show a vertical scrollbar.

wxLB_NEEDED_SB Only create a vertical scrollbar if needed.

wxLB_SORT The listbox contents are sorted in alphabetical order.

Note that wxLB_SINGLE, wxLB_MULTIPLE and wxLB_EXTENDED styles are mutually exclusive and you can specify at most one of them (single selection is the default).

pos is the position of the ChoiceBox (a list of two integer numbers, x and y),

GUI programming / ListBoxes / ListBox

size is the *size* of the ChoiceBox (a list of two integer numbers, width and height).

Example:

```
lb=(ListBox [][MyListBox]
  [  [String 1]
    [and a second string]
    [a third string]
    [and the last string]
  ]
  [  (show "|MyListBox2| ListBoxSelections)
  ]
  [  pr [Doubleclick]
  ]
  wxLB_multiple [0 0][300 200])
```

ListBoxDestroy *alistbox*

Command that destroys the ListBox *alistbox* .

Example:

```
lb=ListBox [][LB] [[String 1][S2][S3]] [[]]
ListBoxDestroy lb
```

ListBoxSelections (**ListBoxSelections** *alistbox*)

outputs the selections of the ListBox *alistbox* , or if called without an argument, of the ListBox that processed the last event.

The output value is a list of zero-based integers, representing the *nth* choice.

Example:

```
lb=(ListBox [][LB] [[String 1][S2][S3]]  
    [show ListBoxSelections][] wxLB_multiple)
```

ListBoxSetSelections *achoicebox selections*

Command to set the *selections* of the ListBox *alistbox* to the item numbers in the list *selections* .

selections is a list of zero-based integer indices into the choices, which must be less than the number of choices in the control.

Example:

```
lb=(ListBox [][LB] [[String 1][S2][S3]] [[]] wxLB_multiple)  
ListBoxSetSelections lb [0 2]
```

ListBoxSetChoices *alistbox choices*

Command to set the contents of the ListBox *alistbox* to *choices* .

choices must be a list of items, whose texts will be the *choices* in the ListBox.

Example:

```
lb=ListBox [][LB] [[]][[]]  
ListBoxSetChoices lb [  
    [First string]  
    [and a second string]  
    [and the last string]  
]
```

ListBoxAppend *alistbox choice*

GUI programming / ListBoxes / ListBoxAppend

Command to append the text of the thing *choice* to the choices of ListBox *alistbox* .

Examples:

```
lb=ListBox [][LB] [[]][  
ListBoxAppend lb [First string]  
ListBoxAppend lb [and a second string]  
ListBoxAppend lb [and the last string]
```

ListBoxSetItem *alistbox index choice*

Command to change the item at position *index* of the ListBox *alistbox* to the text of the thing *choice* .

index must be an integer number $index \geq 0$ and $index < (\text{number of choices in the ListBox})$.

Example:

```
lb=ListBox [][LB] [[The first String][S2][S3]] [[]]  
ListBoxSetItem lb 1 [a changed item]
```

ListBoxRemoveItem *alistbox index*

Command to remove the choice item at position *index* from the ListBox *alistbox* .

index must be an integer number $index \geq 0$ and $index < \text{number of choices in the ListBox}$.

Example:

```
lb=ListBox [][LB] [[The first String][S2][Last String]] [[]]  
ListBoxRemoveItem lb 1
```

ListBoxCount *alistbox*

outputs the number of choices in the ListBox *alistbox* .

Example:

```
lb=ListBox [[LB] [[The first String][S2][S3]] [][]  
show ListBoxCount lb
```

ListBoxSetBackgroundColor *alistbox color*

Command to set the background *color* of the ListBox *alistbox* to the *color color* .

color must be a valid *color* , see also setPenColor!

Example:

```
lb=ListBox [[LB] [[The first String][S2][S3]] [][]  
ListBoxSetBackgroundColor lb "red
```

ListBoxSetColor *alistbox color*

Command to set the foreground *color* of the ListBox *alistbox* to the *color color* .

color must be a valid *color* , see also setPenColor!

Example:

```
lb=ListBox [[LB] [[The first String][S2][S3]] [][]  
ListBoxSetColor lb "red
```

ListBoxSetFontSize *alistbox size*

Command to set the font *size* of the ListBox *alistbox* to *size* , which must be a integer number.

Example:

```
lb=ListBox [[LB] [[The first String][S2][S3]] []]  
ListBoxSetFontSize lb 50
```

ListBoxSetFontName *alistbox name*

Command to set the font *name* of the ListBox *alistbox* to *name* , which must be a valid font *name* .

Example:

```
lb=ListBox [[LB] [[The first String][S2][S3]] []]  
ListBoxSetFontName lb [Courier]
```

ListBoxSetFontStyle *alistbox style*

Command to set the font *style* of the ListBox *alistbox* .

style can be one of the constants wxFONTSTYLE_NORMAL wxFONTSTYLE_SLANT wxFONTSTYLE_ITALIC.

Example:

```
lb=ListBox [[LB] [[The first String][S2][S3]] []]  
ListBoxSetFontStyle lb wxFONTSTYLE_ITALIC
```

ListBoxSetFontWeight *alistbox weight*

Command to set the font *weight* of the ListBox *alistbox* .

weight can be one of the constants wxFONTWEIGHT_NORMAL wxFONTWEIGHT_LIGHT wxFONTWEIGHT_BOLD

Example:

```
lb=ListBox [][LB] [[The first String][S2][S3]] [][]  
ListBoxSetFontWeight lb wxFONTWEIGHT_BOLD
```

ListBoxOnChar *alistbox commands*

Command to set the custom event handler for the char event of the ListBox *alistbox* to *commands* . See also OnChar!

Example:

```
lb=ListBox [][LB] [[The first String][S2][S3]] [][]  
ListBoxOnChar lb [pr KeyboardValue]
```

ListBoxOnKeyDown *alistbox commands*

Command to set the custom event handler for the key down event of the ListBox *alistbox* to *commands* . See also OnKeyDown!

Example:

```
lb=ListBox [][LB] [[The first String][S2][S3]] [][]  
ListBoxOnKeyDown lb [pr KeyboardValue]
```

ListBoxOnKeyUp *alistbox commands*

Command to set the custom event handler for the key up (=release) event of the ListBox *alistbox* to *commands* . See also OnKeyUp!

Example:

```
lb=ListBox [[LB] [[The first String][S2][S3]] []]  
ListBoxOnKeyUp lb [pr KeyboardValue]
```

ListBoxOnSelect *alistbox commands*

Command to set the custom event handler for the select event of the ListBox *alistbox* to *commands* . The select event is generated when the user selects a choice with a mouseclick or a keypress.

Example:

```
lb=ListBox [[LB] [[The first String][S2][S3]] []]  
ListBoxOnSelect lb [pr ListBoxSelections]
```

ListBoxOnDClick *alistbox commands*

Command to set the custom event handler for the mouse double-click event of the ListBox *alistbox* to *commands* .

Example:

```
lb=ListBox [[LB] [[The first String][S2][S3]] []]  
ListBoxOnDClick lb [pr ListBoxSelections]
```

ListBoxEnable *alistbox state*

Command. If *state* is true then it enables the ListBox *alistbox* , if *state* is false then it disables the ListBox.

state must be a boolean.

Example:

```
lb=ListBox [[LB] [[The first String][S2][S3]] []]  
ListBoxEnable lb false  
;Click on the ListBox again --nothing happens  
ListBoxEnable lb true    ;enabled again
```

ListControls

...are windows containing a text label and a list of lists of text choices, which can run a Logo instructionlist when the user clicks with the mouse on them and selects an item, and when a key event is received.

A list control presents lists in a number of formats: list view, report view, icon view and small icon view. In any case, elements are numbered from zero. For all these modes, the items are stored in the control and must be added to it using InsertItem method.

ListControls are demonstrated in listcontroltest.lg.

ListControls

- ListControl 508
- ListControlDestroy 510
- ListControlInsertColumn 511
- ListControlInsertItem 511
- ListControlSetItem 512
- ListControlGetItem 512
- ListControlDeleteItem 513
- ListControlSetRow 513
- ListControlSetColumn 514
- ListControlSet 515
- ListControlGetRow 515
- ListControlGetColumn 516
- ListControlGet 516
- ListControlItemCount 517
- ListControlColumnCount 517
- ListControlColumn 518
- ListControlRow 519
- ListControlText 520
- ListControlSort 521
- ListControlSetBackgroundColor 522
- ListControlSetColor 522
- ListControlSetFontsize 523
- ListControlSetFontName 523
- ListControlSetFontStyle 524
- ListControlSetFontWeight 524

- ListControlOnChar 525
- ListControlOnKeyDown 525
- ListControlOnKeyUp 526
- ListControlOnItemSelected 526
- ListControlOnItemActivated 527
- ListControlOnColClick 527
- ListControlEnable 528

ListControl *parent name onItemSelected onItemActivated onColClick*

(**ListControl** *parent name*)

(**ListControl** *parent name onItemSelected onItemActivated onColClick style pos size*)

outputs a new ListControl *lc* and shows it on the *parent* (a Frame or a Graph, main Graph if []).

name is its label text.

onItemSelected is a Logo instructionlist which will be run when the user selects a list item.

onItemActivated is a Logo instructionlist which will be run when the user activates a list item, using either the [Enter] or [Space] key double-clicking with the mouse.

onColClick is a Logo instructionlist which will be run when the user left-clicks a column label with the mouse.

style is the window *style* and can be a combination of the following constants:

wxLC_LIST Multicolumn list view, with optional small icons. Columns are computed automatically, i.e. you don't set columns as in **wxLC_REPORT**. In other words, the list wraps, unlike a **wxListBox**.

wxLC_REPORT Single or multicolumn report view, with optional header.

wxLC_VIRTUAL Don't use this, it won't work! (The application provides items text on demand. May only be used with **wxLC_REPORT**.)

wxLC_ICON Large icon view, with optional labels.

GUI programming / ListControls / ListControl

`wxLC_SMALL_ICON` Small icon view, with optional labels.

`wxLC_ALIGN_TOP` Icons align to the top. Win32 default, Win32 only.

`wxLC_ALIGN_LEFT` Icons align to the left.

`wxLC_AUTOARRANGE` Icons arrange themselves. Win32 only.

`wxLC_EDIT_LABELS` Labels are editable: the application will be notified when editing starts.

`wxLC_NO_HEADER` No header in report mode.

`wxLC_SINGLE_SEL` Single selection (default is multiple).

`wxLC_SORT_ASCENDING` Don't use this, it wont work! (Sort in ascending order (must still supply a comparison callback in `SortItems`..))

`wxLC_SORT_DESCENDING` Don't use this, it wont work! (Sort in descending order (must still supply a comparison callback in `SortItems`..))

`wxLC_HRULES` Draws light horizontal rules between rows in report mode.

`wxLC_VRULES` Draws light vertical rules between columns in report mode.

pos is the position of the ChoiceBox (a list of two integer numbers, x and y),

size is the *size* of the ChoiceBox (a list of two integer numbers, width and height).

Example:

```
lc=(ListControl [][MyListControl]
  [ row=ListControlRow
    (pr word [row=] row
      word [column=] ListControlColumn
      ListControlText
      [Selected])
    id=Int (ListControlGetItem lc row 0)
  ]
  [ (pr word [row=] ListControlRow
    word [column=] ListControlColumn
    ListControlText
    [Activated])
  ]
  [ c=ListControlColumn
    (pr word [column=] c
      [Columnclick])
    ListControlSort lc c
  ] wxLC_REPORT [10 20][400 300])
```

```
foreach [[[ID] 40][[First Name] 80][[Name] 120][[Logo] 120]] [
  (ListControlInsertColumn lc # (?).1 (?).2)
]
```

```
data=[
  [1 Brian Harvey UCBLogo]
  [2 George Mills MSWLogo]
  [3 Pavel Boytchev Elica]
  [4 Lionel Laske Liogo]
  [5 Andreas Micheler aUCBLogo]
]
ListControlSet lc data
```

ListControlDestroy *alistcontrol*

Command that destroys the ListControl *alistcontrol*.

Example:

```
lc=(ListControl [][MyListControl][][][])
    wxLC_LIST [0 0][300 200])
ListControlInsertColumn lc 1 "Logo
data=[[UCBLogo][MSWLogo][Elica][Liogo][aUCBLogo]]
ListControlSet lc data
ListControlDestroy lc
```

ListControlInsertColumn *alistcontrol col labl*
(ListControlInsertColumn *alistcontrol col labl awidth format*)

Command to insert a new column into the ListControl *alistcontrol* at the column *col* with the column label *labl* and column width *awidth* .

col must be a positive integer number.

labl is the column label text.

awidth is the column width.

format can be wxLIST_FORMAT_LEFT, wxLIST_FORMAT_RIGHT or wxLIST_FORMAT_CENTRE.

Examples:

```
lc=(ListControl [][MyListControl][][][])
    wxLC_REPORT [0 0][300 200])
(ListControlInsertColumn lc 1 [ID] 30)
(ListControlInsertColumn lc 2 [First Name]
    100 wxLIST_FORMAT_RIGHT)
ListControlInsertColumn lc 3 [Name]
```

ListControlInsertItem *alistcontrol arow*

Command inserting a new item (a row in wxLC_REPORT mode) into the ListControl *alistcontrol*

at row *arow* .

arow is the row of the new item, a zero-based integer number.

Example:

```
lc=(ListControl [[MyListControl]][][])
    wxLC_LIST [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertItem lc 0
ListControlSetItem lc 0 0 [UCBLogo]
```

ListControlSetItem *alistcontrol arow col aitem*

Command to set the item *aitem* at position (column *col* , row *arow*) of the ListControl *alistcontrol* .

arow is the row (Y in wxLC_REPORT mode) of the item that's set. It's a zero-based integer.

col is the column (X in wxLC_REPORT mode) of the item. It's a zero-based integer.

aitem is the item text that's set.

Examples:

```
lc=(ListControl [[MyListControl]][][])
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlInsertItem lc 0
ListControlInsertItem lc 1
ListControlSetItem lc 0 0 [UCBLogo]
ListControlSetItem lc 0 1 [Brian Harvey]
ListControlSetItem lc 1 0 [MSWLogo]
ListControlSetItem lc 1 1 [George Mills]
```

ListControlItem *alistcontrol arow col*

outputs the item's text of the item at position (column *col* , row *arow*) of the ListControl *alistcontrol* .

arow is the row (Y in wxLC_REPORT mode) of the item that's fetched. It's a zero-based integer.

col is the column (X in wxLC_REPORT mode) of the item. It's a zero-based integer.

Example:

```
lc=(ListControl [][MyListControl]][][][]  
    wxLC_REPORT [0 0][300 200])  
ListControlItemInsertColumn lc 0 [Logo]  
ListControlItemInsertColumn lc 1 [Main Author]  
ListControlItemSet lc [  
    [UCBLogo [Brian Harvey]]  
    [MSWLogo [George Mills]]]  
show ListControlItemGetItem lc 0 1
```

ListControlItemDeleteItem *alistcontrol arow*

Command to delete the row *arow* of the ListControl *alistcontrol* .

arow is the row (Y in wxLC_REPORT mode) that's deleted. It's a zero-based integer.

Example:

```
lc=(ListControl [][MyListControl]][][][]  
    wxLC_REPORT [0 0][300 200])  
ListControlItemInsertColumn lc 0 [Logo]  
ListControlItemInsertColumn lc 1 [Main Author]  
ListControlItemSet lc [  
    [UCBLogo [Brian Harvey]]  
    [MSWLogo [George Mills]]]  
ListControlItemDeleteItem lc 0
```

ListControlSetRow *alistcontrol arow data*

Command to set the row at *arow* of the ListControl *alistcontrol* to the list of items in *data* .

arow is the row (Y in wxLC_REPORT mode) that's set. It's a zero-based integer.

data is a list of items corresponding to the columns.

Example:

```
lc=(ListControl [[MyListControl]][][])
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
ListControlSetRow lc 0 [Liogo [Lionel Laske]]
```

ListControlSetColumn *alistcontrol col data*

Command to set the column at *col* of the ListControl *alistcontrol* to the list of items in *data* .

col is the column (X in wxLC_REPORT mode) that's set. It's a zero-based integer.

data is a list of items corresponding to the rows.

Example:

```
lc=(ListControl [][MyListControl][[]][[]]
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
ListControlSetColumn lc 1 [Brian George]
```

ListControlSet *alistcontrol data*

Command to set the items of the ListControl *alistcontrol* to the list of rows (which are lists of column items) in *data* .

data is a list of rows. A element of rows is a list of items corresponding to the columns.

Example:

```
lc=(ListControl [][MyListControl][[]][[]]
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
```

ListControlGetRow *alistcontrol arow*

outputs the row at *arow* of the ListControl *alistcontrol* .

arow is the row (Y in wxLC_REPORT mode) that's fetched. It's a zero-based integer.

Output is a list of items (words) corresponding to the columns.

Example:

```
lc=(ListControl [][MyListControl][[]][[]]
      wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
  [UCBLogo [Brian Harvey]]
  [MSWLogo [George Mills]]]
show ListControlGetRow lc 0
show first bf ListControlGetRow lc 0      ;It's a word
```

ListControlGetColumn *alistcontrol col data*

outputs the column at *col* of the ListControl *alistcontrol* .

col is the column (X in wxLC_REPORT mode) that's fetched. It's a zero-based integer.

Output is a list of items (words) corresponding to the rows.

Example:

```
lc=(ListControl [][MyListControl][[]][[]]
      wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
  [UCBLogo [Brian Harvey]]
  [MSWLogo [George Mills]]]
show ListControlGetColumn lc 1
```

ListControlGet *alistcontrol*

outputs the items of the ListControl *alistcontrol* .

Output is a list of rows. A element of rows is a list of items (words) corresponding to the columns.

Example:

```
lc=(ListControl [][MyListControl][[]][[]]
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
show ListControlGet lc
```

ListControlItemCount *alistcontrol*

outputs the item count (the number of rows in wxLC_REPORT mode) of the ListControl *alistcontrol* .

Example:

```
lc=(ListControl [][MyListControl][[]][[]]
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
show ListControlItemCount lc
```

ListControlColumnCount *alistcontrol*

outputs the column count (the number of columns in wxLC_REPORT mode) of the ListControl *alistcontrol* .

Example:

```
lc=(ListControl [[MyListControl]][][])
  wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
  [UCBLogo [Brian Harvey]]
  [MSWLogo [George Mills]]]
show ListControlColumnCount lc
```

ListControlColumn

outputs the column where the latest event of a ListControl was processed.

Examples:

```

lc=(ListControl [][MyListControl]
  [
    row=ListControlRow
    (pr word [row=] row
      word [column=] ListControlColumn
      ListControlText
      [Selected])
  ]
  [
    (pr word [row=] ListControlRow
      word [column=] ListControlColumn
      ListControlText
      [Activated])
  ]
  [
    c=ListControlColumn
    (pr word [column=] c
      [Columnclick])
    ListControlSort lc c
  ] wxLC_REPORT [10 20][400 300])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
  [UCBLogo [Brian Harvey]]
  [MSWLogo [George Mills]]]

```

ListControlRow

outputs the row where the latest event of a ListControl was processed.

Examples:

```

lc=(ListControl [][MyListControl]
  [
    row=ListControlRow
    (pr word [row=] row
      word [column=] ListControlColumn
      ListControlText
      [Selected])
  ]
  [
    (pr word [row=] ListControlRow
      word [column=] ListControlColumn
      ListControlText
      [Activated])
  ]
  [
    c=ListControlColumn
    (pr word [column=] c
      [Columnclick])
    ListControlSort lc c
  ] wxLC_REPORT [10 20][400 300])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
  [UCBLogo [Brian Harvey]]
  [MSWLogo [George Mills]]]

```

ListControlText

outputs the item text of the first column in the row where the latest event of a ListControl was processed.

Examples:

```

lc=(ListControl [][MyListControl]
  [
    row=ListControlRow
    (pr word [row=] row
      word [column=] ListControlColumn
      ListControlText
      [Selected])
  ]
  [
    (pr word [row=] ListControlRow
      word [column=] ListControlColumn
      ListControlText
      [Activated])
  ]
  [
    c=ListControlColumn
    (pr word [column=] c
      [Columnclick])
    ListControlSort lc c
  ] wxLC_REPORT [10 20][400 300])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
  [UCBLogo [Brian Harvey]]
  [MSWLogo [George Mills]]]

```

ListControlSort *alistcontrol col*

Command to sort the ListControl *alistcontrol* by column number *col*.

Example:

```
lc=(ListControl [][MyListControl][[]][[]]
      wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [col0]
ListControlInsertColumn lc 1 [col1]
ListControlInsertColumn lc 2 [col2]
ListControlSet lc [
    [1 3 A]
    [2 1 B]
    [3 10 C]]
ListControlSort lc 1
ListControlSort lc 2
```

ListControlSetBackgroundColor *alistcontrol color*

Command to set the background *color* of the ListControl *alistcontrol* to the *color color*.

color must be a valid *color*, see also setPenColor!

Example:

```
lc=(ListControl [][MyListControl][[]][[]]
      wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
ListControlSetBackgroundColor lc "red"
```

ListControlSetColor *alistcontrol color*

Command to set the foreground *color* of the ListControl *alistcontrol* to the *color color*.

color must be a valid *color*, see also setPenColor!

Example:

```
lc=(ListControl [[MyListControl]][][])
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
ListControlSetColor lc "red
```

ListControlSetFontSize *alistcontrol size*

Command to set the font *size* of the ListControl *alistcontrol* to *size* , which must be a integer number.

Example:

```
lc=(ListControl [[MyListControl]][][])
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
ListControlSetFontSize lc 50
```

ListControlSetFontName *alistcontrol name*

Command to set the font *name* of the ListControl *alistcontrol* to *name* , which must be a valid font *name* .

Example:

```
lc=(ListControl [][MyListControl][][][])
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
ListControlSetFontName lc [Courier]
```

ListControlSetFontStyle *alistcontrol style*

Command to set the font *style* of the ListControl *alistcontrol* .

style can be one of the constants wxFONTSTYLE_NORMAL wxFONTSTYLE_SLANT wxFONTSTYLE_ITALIC.

Example:

```
lc=(ListControl [][MyListControl][][][])
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
ListControlSetFontStyle lc wxFONTSTYLE_ITALIC
```

ListControlSetFontWeight *alistbox weight*

Command to set the font *weight* of the ListControl *alistcontrol*.

weight can be one of the constants wxFONTWEIGHT_NORMAL wxFONTWEIGHT_LIGHT wxFONTWEIGHT_BOLD

Example:

```
lc=(ListControl [[MyListControl]][][])
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
ListControlSetFontWeight lc wxFONTWEIGHT_BOLD
```

ListControlOnChar *alistcontrol commands*

Command to set the custom event handler for the char event of the ListControl *alistcontrol* to *commands* . See also OnChar!

Example:

```
lc=(ListControl [[MyListControl]][][])
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
ListControlOnChar lc [pr KeyboardValue]
```

ListControlOnKeyDown *alistcontrol commands*

Command to set the custom event handler for the key down event of the ListControl *alistcontrol* to *commands* . See also OnKeyDown!

Example:

```
lc=(ListControl [][MyListControl][][][])
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
ListControlOnKeyDown lc [pr KeyboardValue]
```

ListControlOnKeyUp *alistcontrol commands*

Command to set the custom event handler for the key up (=release) event of the ListControl *alistcontrol* to *commands* . See also OnKeyUp!

Example:

```
lc=(ListControl [][MyListControl][][][])
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]]
ListControlOnKeyUp lc [pr KeyboardValue]
```

ListControlOnItemSelected *alistcontrol commands*

Command to set the custom event handler for the LIST_ITEM_SELECTED event of the ListControl *alistcontrol* to *commands* .

Example:

```

lc=(ListControl [][MyListControl][][][]
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]
]
ListControlOnItemSelected lc [
    (pr word [row=] ListControlRow
        word [column=] ListControlColumn
        ListControlText
        [Selected])
]

```

ListControlOnItemActivated *alistcontrol commands*

Command to set the custom event handler for the LIST_ITEM_ACTIVATED event (ENTER or double-click) of the ListControl *alistcontrol* to *commands*.

Example:

```

lc=(ListControl [][MyListControl][][][]
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]
]
ListControlOnItemActivated lc [
    (pr word [row=] ListControlRow
        word [column=] ListControlColumn
        ListControlText
        [Activated])
]

```

ListControlOnColClick *alistcontrol commands*

Command to set the custom event handler for the LIST_COL_CLICK event (click on a column label) of the ListControl *alistcontrol* to *commands* .

Example:

```
lc=(ListControl [[MyListControl]][][])
  wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
  [UCBLogo [Brian Harvey]]
  [MSWLogo [George Mills]]
]
ListControlOnColClick lc [
  c=ListControlColumn
  (pr word [column=] c
    [Columnclick])
  ListControlSort lc c
]
```

ListControlEnable *alistcontrol state*

Command. If *state* is true then it enables the ListControl *alistcontrol* , if *state* is false then it disables the ListControl.

state must be a boolean.

Example:

```
lc=(ListControl [][MyListControl][[]][[]]
    wxLC_REPORT [0 0][300 200])
ListControlInsertColumn lc 0 [Logo]
ListControlInsertColumn lc 1 [Main Author]
ListControlSet lc [
    [UCBLogo [Brian Harvey]]
    [MSWLogo [George Mills]]
]
ListControlEnable lc false
;Click on the FloatControl again --nothing happens
ListControlEnable lc true ;enabled again
```

RadioButtons

...are little windows containing a radio button and a text label, which can run a Logo instructionlist when the user clicks with the mouse on them and checks them. CheckBoxes are demonstrated in `buttontest.lg`.

RadioButtons

- `RadioButton` 530
 - `RadioButtonDestroy` 531
 - `RadioButtonOnClick` 531
 - `RadioButtonValue` 532
 - `RadioButtonSet` 532
 - `RadioButtonEnable` 532
-

`RadioButton` *parent label onclick*
(**`RadioButton`** *parent label onclick style pos size*)

outputs a new `RadioButton` on the *parent* (a Frame or a Graph),

having the text *label* on it,

and running the *onclick* instructionlist when the user clicks with the mouse on it.

style can be a combination (use +) of the following constants:

`wxRB_GROUP` Marks the beginning of a new group of radio buttons.

`wxRB_SINGLE` In some circumstances, radio buttons that are not consecutive siblings trigger a hang bug in Windows (only). If this happens, add this *style* to mark the button as not belonging to a group, and implement the mutually-exclusive group behaviour yourself.

`wxRB_USE_CHECKBOX` Use a checkbox button instead of radio button (currently supported only on PalmOS).

GUI programming / RadioButtons / RadioButton

pos is the position of the Button (a list of two integer numbers, x and y),

size is the *size* of the Button (again a list of two integer numbers, width and height).

Examples:

```
rb1=(RadioButton [][PenPaint]
[   if RadioButtonValue rb1 [PenPaint setPC 0]
] wxRB_GROUP)
rb2=(RadioButton [][PenErase]
[   if RadioButtonValue rb2 [PenErase]
] 0 [0 30])
rb3=(RadioButton [][PenReverse]
[   if RadioButtonValue rb3 [PenReverse]
] 0 [0 60])
```

RadioButtonDestroy *aradiobutton*

Command that destroys the RadioButton *aradiobutton* .

Example:

```
rb=(RadioButton [][PenPaint]
[   if RadioButtonValue rb [PenPaint setPC 0]
] wxRB_GROUP)
RadioButtonDestroy rb
```

RadioButtonOnClick *aradiobutton instructionlist*

Command that sets the onClick event handler of the RadioButton *aradiobutton* to the commands in the *instructionlist* .

Examples:

```
rb=(RadioButton [] [PenPaint] [] wxRB_GROUP)
RadioButtonOnClick rb [
    if RadioButtonValue rb [PenPaint setPC 0]
]
RadioButtonOnClick cb [] ;like disabling the RadioButton
```

RadioButtonValue *aradiobutton*

outputs true if the RadioButton *aradiobutton* is checked, false otherwise.

Example:

```
rb=(RadioButton [] [PenPaint]
[   if RadioButtonValue rb [pr [checked]]
] wxRB_GROUP)
```

RadioButtonSet *acheckbox state*

Command that sets the RadioButton *aradiobutton* to checked if *state* is true, to unchecked if false.

state must be a boolean.

Example:

```
rb=(RadioButton [] [PenPaint]
[   if RadioButtonValue rb [pr [checked]]
] wxRB_GROUP)
RadioButtonSet rb true
RadioButtonSet rb false
```

RadioButtonEnable *aradiobutton state*

GUI programming / RadioButtons / RadioButtonEnable

Command. If *state* is true then it enables the RadioButton *aradiobutton* , if *state* is false then it disables the RadioButton.

state must be a boolean.

Example:

```
rb=(RadioButton [] [PenPaint]
[   if RadioButtonValue rb [pr [checked]]
] wxRB_GROUP)
RadioButtonEnable rb false
;Click on the RadioButton again --nothing happens
RadioButtonEnable rb true    ;enabled again
```

Sliders

A Slider is a little window containing a text label, and a slider control for use with integer numbers. It holds the value of an integer number, which can be changed by the user with the arrow keys, or by dragging of or clicking on the slider with the mouse.

Slider is demonstrated in `slidertest.lg`.

Sliders

- Slider 534
- SliderDestroy 535
- SliderValue 536
- SliderSetValue 536
- SliderSetRange 536
- SliderOnScroll 537
- SliderEnable 537

Slider *parent name min value max onScroll*
 (Slider *parent name min value max*)
 (Slider *parent name min value max onScroll style pos size*)

outputs a new Slider on the *parent* (a Frame or a Graph, main Graph if []).

name is the label text and the *name* of the control.

min is the minimal allowed *value* that the control accepts.

value is the number shown at startup of the control.

max is the maximal allowed *value* that the control accepts.

min , *value* and *max* must be integer numbers.

onChange is a command list that will be run if the OnChange event of the control is processed. That's whenever the number in the control changes anyhow.

GUI programming / Sliders / Slider

style is the window *style* and can be a combination of the following constants:

`wxSL_HORIZONTAL` Displays the slider horizontally (this is the default).

`wxSL_VERTICAL` Displays the slider vertically.

`wxSL_AUTOTICKS` Displays tick marks.

`wxSL_LABELS` Displays minimum, maximum and *value* labels.

`wxSL_LEFT` Displays ticks on the left and forces the slider to be vertical.

`wxSL_RIGHT` Displays ticks on the right and forces the slider to be vertical.

`wxSL_TOP` Displays ticks on the top.

`wxSL_BOTTOM` Displays ticks on the bottom (this is the default).

`wxSL_SELRANGE` Allows the user to select a range on the slider. Windows only.

`wxSL_INVERSE` Inverses the minimum and maximum endpoints on the slider. Not compatible with `wxSL_SELRANGE`.

pos is the position of the Slider (a list of two integer numbers, x and y),

size is the *size* of the Slider (a list of two integer numbers, width and height).

Example:

```
s=(Slider [][X] -400 0 400
  [
    x=SliderValue
    (pr "x= x)
    setXY x y
    updateGraph
  ]
  wxSL_horizontal+wxSL_Labels+wxSL_Ticks
  [10 220][200 80])
```

SliderDestroy *aslider*

Command to destroy the Slider *aslider* .

Example:

```
s=(Slider [][X] -400 0 400 [x=SliderValue (pr "x= x)]
    wxSL_horizontal+wxSL_Labels+wxSL_Ticks
    [10 220][200 80])
SliderDestroy s
```

SliderValue

(SliderValue *aslider*)

outputs the integer value of the Slider *aslider* or if called without arguments, of the Slider whose event is being processed lately.

Example:

```
s=(Slider [][X] -400 0 400 [x=SliderValue (pr "x= x)]
    wxSL_horizontal+wxSL_Labels+wxSL_Ticks
    [10 220][200 80])
```

SliderSetValue *aslider value*

Command to set the *value* of the Slider *aslider* to the integer number *value* .

Example:

```
s=(Slider [][X] -400 0 400 [x=SliderValue (pr "x= x)]
    wxSL_horizontal+wxSL_Labels+wxSL_Ticks
    [10 220][200 80])
SliderSetValue s 123
```

GUI programming / Sliders / SliderSetRange

SliderSetRange *aslider min max*

Command to set the range of valid numbers of the Slider *aslider* to stay between the numbers *min* and *max* inclusive.

Example:

```
s=(Slider [][X] -400 0 400 [x=SliderValue (pr "x= x)"]
  wxSL_horizontal+wxSL_Labels+wxSL_Ticks
  [10 220][200 80])
SliderSetRange s -10 20
```

SliderOnScroll *aslider commands*

Command to set the OnScroll event handler of the Slider *aslider* to the commandlist *commands* .

Example:

```
s=(Slider [][X] -400 0 400 []
  wxSL_horizontal+wxSL_Labels+wxSL_Ticks
  [10 220][200 80])
SliderOnScroll s [x=SliderValue (pr "x= x)"]
```

SliderEnable *aslider state*

Command. If *state* is true then it enables the Slider *aslider* , if *state* is false then it disables the Slider.

state must be a boolean.

Example:

```
s=(Slider [][X] -400 0 400 [x=SliderValue (pr "x= x)]
    wxSL_horizontal+wxSL_Labels+wxSL_Ticks
    [10 220][200 80])
SliderEnable s false
;Click on the Slider again --nothing happens
SliderEnable s true ;enabled again
```

StaticTexts

...are windows containing one or more lines of read-only text.

The corresponding demo is `statictexttest.lg`.

StaticTexts

- `StaticText` 539
- `StaticTextDestroy` 540
- `StaticTextLabel` 540
- `StaticTextSetLabel` 540
- `StaticTextSetColor` 541
- `StaticTextSetBackgroundColor` 541
- `StaticTextSetFontSize` 541
- `StaticTextSetFontName` 542
- `StaticTextSetFontStyle` 542
- `StaticTextSetFontWeight` 543

StaticText *parent label*

(StaticText *parent label style pos size name* **)**

outputs a new `StaticText` control on the *parent* (a Frame or a Graph, or if [] the main Graph).

label can be anything that can be converted to a word. That word is shown in the `StaticText` control.

style is the window *style* and can be a combination of the following constants:

`wxALIGN_LEFT` Align the text to the left

`wxALIGN_RIGHT` Align the text to the right

`wxALIGN_CENTRE` Center the text (horizontally)

`wxST_NO_AUTORESIZE` By default, the control will adjust its *size* to exactly fit to the *size* of the text when `SetLabel` is called. If this *style* flag is given, the control will not change its *size* (this

style is especially useful with controls which also have wxALIGN_RIGHT or CENTER *style* because otherwise they won't make sense any longer after a call to SetLabel)

pos is the position of the StaticText (a list of two integer numbers, x and y),

size is the *size* of the StaticText (a list of two integer numbers, width and height).

Example:

```
s=(StaticText [][This is\  
    a multiline\  
    static\  
    Text] wxAlign_centre [100 200][100 100][Text1])
```

StaticTextDestroy *astatictext*

Command that destroys the StaticText *astatictext* .

Example:

```
s=(StaticText [][This is a static Text]  
    wxAlign_centre [100 200][100 100][Text1])  
StaticTextDestroy s
```

StaticTextLabel *astatictext*

outputs the text contents of the StaticText control.

Example:

```
s=(StaticText [][This is a static Text]  
    wxAlign_centre [100 200][100 100][Text1])  
show StaticTextLabel s
```

StaticTextSetLabel *astatictext label*

Command to set the text contained in the StaticText control *astatictext* .

label is anything printable. It will be converted to a word.

Example:

```
s=(StaticText [] "  
    wxAlign_centre [100 200][100 100][Text1])  
StaticTextSetLabel s [This is a static Text]
```

StaticTextSetColor *astatictext color*

Command to set the foreground *color* of the StaticText *astatictext* to *color* , which must be a valid *color* (see SetPenColor!).

Example:

```
s=(StaticText [] [This is a static Text]  
    wxAlign_centre [100 200][100 100][Text1])  
StaticTextSetColor s "red"
```

StaticTextSetBackgroundColor *astatictext color*

Command to set the background *color* of the StaticText *astatictext* to *color* , which must be a valid *color* (see SetPenColor!).

Example:

```
s=(StaticText [] [This is a static Text]  
    wxAlign_centre [100 200][100 100][Text1])  
StaticTextSetBackgroundColor s rgb 1 0 0
```

StaticTextSetFontSize *astatictext size*

Command to set the font *size* of the StaticText *astatictext* to *size* , which must be a integer number.

Example:

```
s=(StaticText [][This is a static Text]
    wxAlign_centre [100 200][100 100][Text1])
StaticTextSetFontSize s 50
```

StaticTextSetFontName *astatictext size*

Command to set the font name of the StaticText *astatictext* to name, which must be a valid font name.

Example:

```
s=(StaticText [][This is a static Text]
    wxAlign_centre [100 200][100 100][Text1])
StaticTextSetFontName s [Courier]
```

StaticTextSetFontStyle *astatictext style*

Command to set the font *style* of the StaticText *astatictext* .

style can be one of the constants wxFONTSTYLE_NORMAL wxFONTSTYLE_SLANT wxFONTSTYLE_ITALIC.

Example:

```
s=(StaticText [][This is a static Text]
    wxAlign_centre [100 200][100 100][Text1])
StaticTextSetFontStyle s wxFONTSTYLE_ITALIC
```

StaticTextSetFontWeight *astatictext weight*

Command to set the font *weight* of the StaticText *astatictext* .

weight can be one of the constants wxFONTWEIGHT_NORMAL wxFONTWEIGHT_LIGHT wxFONTWEIGHT_BOLD

Example:

```
s=(StaticText [][This is a static Text]
    wxAlign_centre [100 200][100 100][Text1])
StaticTextSetFontWeight f wxFONTWEIGHT_BOLD
```

TextControls

...is a window with a text label and an edit control, which might be multiline, and maybe formatted.

A text control allows text to be displayed and edited. It may be single line or multi-line.

The corresponding demo is textcontroltest.lg.

TextControls

- TextControl 544
- TextControlDestroy 546
- TextControlValue 547
- TextControlSetValue 547
- TextControlWrite 547
- TextControlAppend 548
- TextControlSetInsertionPointEnd 548
- TextControlCursor 548
- TextControlSetCursor 549
- TextControlInsertMode 549
- TextControlOverwriteMode 550
- TextControlSetColor 550
- TextControlSetBackgroundColor 550
- TextControlSetFontSize 551
- TextControlSetFontName 551
- TextControlSetFontStyle 552
- TextControlSetFontWeight 552
- TextControlOnChar 552
- TextControlOnKeyDown 553
- TextControlOnKeyUp 553
- TextControlOnChange 554
- TextControlOnEnter 554
- TextControlEnable 554

TextControl *parent name value*

(TextControl *parent name* **)**

(TextControl *parent name value onChange onEnter style pos size* **)**

outputs a new TextControl on the *parent* (a Frame or a Graph, or if [] the main Graph).

value can be anything that can be converted to a word. That word is shown in the TextControl at the beginning.

style is the window *style* and can be a combination of the following constants:

wxTE_PROCESS_ENTER The control will generate the event

wxEVT_COMMAND_TEXT_ENTER (otherwise pressing Enter key is either processed internally by the control or used for navigation between dialog controls).

wxTE_PROCESS_TAB The control will receive **wxEVT_CHAR** events for TAB pressed - normally, TAB is used for passing to the next control in a dialog instead. For the control created with this *style*, you can still use Ctrl-Enter to pass to the next control from the keyboard.

wxTE_MULTILINE The text control allows multiple lines.

wxTE_PASSWORD The text will be echoed as asterisks.

wxTE_READONLY The text will not be user-editable.

wxTE_RICH Use rich text control under Win32, this allows to have more than 64KB of text in the control even under Win9x. This *style* is ignored under other platforms.

wxTE_RICH2 Use rich text control version 2.0 or 3.0 under Win32, this *style* is ignored under other platforms

wxTE_AUTO_URL Highlight the URLs and generate the **wxTextUrlEvents** when mouse events occur over them. This *style* is only supported for **wxTE_RICH** Win32 and multi-line **wxGTK2** text controls.

wxTE_NOHIDESEL By default, the Windows text control doesn't show the selection when it doesn't have focus - use this *style* to force it to always show it. It doesn't do anything under other platforms.

wxHSCROLL A horizontal scrollbar will be created and used, so that text won't be wrapped. No effect under **wxGTK1**.

wxTE_LEFT The text in the control will be left-justified (default).

wxTE_CENTRE The text in the control will be centered (currently **wxMSW** and **wxGTK2** only).

wxTE_RIGHT The text in the control will be right-justified (currently wxMSW and wxGTK2 only).

wxTE_DONTWRAP Same as wxHSCROLL *style* : don't wrap at all, show horizontal scrollbar instead.

wxTE_CHARWRAP Wrap the lines too long to be shown entirely at any position (wxUniv and wxGTK2 only).

wxTE_WORDWRAP Wrap the lines too long to be shown entirely at word boundaries (wxUniv and wxGTK2 only).

wxTE_BESTWRAP Wrap the lines at word boundaries or at any other character if there are words longer than the window width (this is the default).

wxTE_CAPITALIZE On PocketPC and Smartphone, causes the first letter to be capitalized.

pos is the position of the TextControl (a list of two integer numbers, x and y),

size is the *size* of the TextControl (a list of two integer numbers, width and height).

Example:

```
t=(TextControl [[TextControl1][This is\  
    some multiline\  
    text]  
    []]  
    wxTE_multiline+wxTE_Rich2+wxTE_NoHideSel  
    [0 0][300 200])
```

TextControlDestroy *atextcontrol*

Command to destroy the TextControl *atextcontrol* .

Example:

GUI programming / TextControls / TextControlDestroy

```
t=(TextControl [][TextControl1][This is some text][[]]
    wxTE_CENTRE [0 0][300 200])
TextControlDestroy t
```

TextControlValue

(TextControlValue *atextcontrol*)

outputs the word value of the TextControl *atextcontrol* or if called without arguments, of the TextControl whose event is being processed lately.

Examples:

```
t=(TextControl [][TextControl1][This is some text]
    [ (pr [onChange] TextControlValue)
    ][ (pr [onEnter] TextControlValue)
    ]
    wxTE_MULTILINE+wxTE_RICH2 [0 0][300 200])
```

TextControlSetValue *atextcontrol value*

Command to set the text *value* of the TextControl *atextcontrol* to *value* .

Example:

```
t=(TextControl [][TextControl1]
    [This is the initial text][[]]
    wxTE_CENTRE [0 0][300 200])
TextControlSetValue t [That's the new text!]
```

TextControlWrite *atextcontrol value*

Command to write the text of *value* at the current cursor position into the TextControl *atextcontrol* .

Example:

```
t=(TextControl [][TextControl1]
    [This is the initial text.][[]]
    wxTE_CENTRE [0 0][300 200])
TextControlSetCursor t [10 0]
TextControlWrite t [!That's the new text!]
```

TextControlAppend *atextcontrol value*

Command to append the text of *value* to the TextControl *atextcontrol* .

Example:

```
t=(TextControl [][TextControl1]
    [This is the initial text.][[]]
    wxTE_CENTRE [0 0][300 200])
TextControlAppend t [!That's the new text!]
```

TextControlSetInsertionPointEnd *atextcontrol*

Command to set the insertion point (the cursor) to the end of the text of the TextControl *atextcontrol* .

Example:

```
t=(TextControl [][TextControl1]
    [This is the initial text.][[]]
    wxTE_CENTRE [0 0][300 200])
TextControlSetInsertionPointEnd t ;like with append
TextControlWrite t [!That's the new text!]
```

TextControlCursor *atextcontrol*

outputs the cursor position of the TextControl *atextcontrol* as a list of two zero-based integers, [column row].

Example:

```
t=(TextControl [][TextControl1]
    [This is the initial text.][][]
    wxTE_MULTILINE [0 0][300 200])
TextControlSetCursor t [10 2]
TextControlWrite t [!That's the new text!]
show TextControlCursor t
```

TextControlSetCursor *atextcontrol pos*

Command to set the insertion point, the cursor, of the TextControl *atextcontrol* to the *pos* list, which consists of [column row].

Example:

```
t=(TextControl [][TextControl1]
    [This is the initial text.][][]
    wxTE_MULTILINE [0 0][300 200])
TextControlSetCursor t [10 2]
TextControlWrite t [!That's the new text!]
```

TextControlInsertMode *atextcontrol*

Command to set the write mode of the TextControl *atextcontrol* to insert.

Example:

```
t=(TextControl [][TextControl1]
  [This is the initial text.][][]
  wxTE_MULTILINE [0 0][300 200])
TextControlOverwriteMode t ;type something
TextControlInsertMode t ;type something more
```

TextControlOverwriteMode *atextcontrol*

Does not work yet!

Command to set the write mode of the TextControl *atextcontrol* to overwrite.

Example:

```
t=(TextControl [][TextControl1]
  [This is the initial text.][][]
  wxTE_MULTILINE+wxTE_RICH2 [0 0][300 200])
TextControlOverwriteMode t ;type something
TextControlInsertMode t ;type something more
```

TextControlSetColor *aTextControl color*

Command to set the foreground *color* of the TextControl *aTextControl* to *color* , which must be a valid *color* (see SetPenColor!).

Example:

```
t=(TextControl [][TextControl1]
  [This is the initial text.][][]
  wxTE_MULTILINE [0 0][300 200])
TextControlSetColor t "red"
```

TextControlSetBackgroundColor *aTextControl color*

Command to set the background *color* of the TextControl *aTextControl* to *color* , which must be a valid *color* (see SetPenColor!).

Example:

```
t=(TextControl [][TextControl1]
  [This is the initial text.][[]]
  wxTE_MULTILINE [0 0][300 200])
TextControlSetBackgroundColor t rgb 1 0 0
```

TextControlSetFontSize *aTextControl size*

Command to set the font *size* of the TextControl *aTextControl* to *size* , which must be a integer number.

Example:

```
t=(TextControl [][TextControl1]
  [This is the initial text.][[]]
  wxTE_MULTILINE [0 0][300 200])
TextControlSetFontSize t 50
```

TextControlSetFontName *aTextControl size*

Command to set the font name of the TextControl *aTextControl* to *name*, which must be a valid font name.

Example:

```
t=(TextControl [][TextControl1]  
    [This is the initial text.][[]]  
    wxTE_MULTILINE [0 0][300 200])  
TextControlSetFontName t [Courier]
```

TextControlSetFontStyle *aTextControl style*

Command to set the font *style* of the TextControl *aTextControl* .

style can be one of the constants wxFONTSTYLE_NORMAL wxFONTSTYLE_SLANT wxFONTSTYLE_ITALIC.

Example:

```
t=(TextControl [][TextControl1]  
    [This is the initial text.][[]]  
    wxTE_MULTILINE [0 0][300 200])  
TextControlSetFontStyle t wxFONTSTYLE_ITALIC
```

TextControlSetFontWeight *aTextControl weight*

Command to set the font *weight* of the TextControl *aTextControl* .

weight can be one of the constants wxFONTWEIGHT_NORMAL wxFONTWEIGHT_LIGHT wxFONTWEIGHT_BOLD

Example:

```
t=(TextControl [][TextControl1]  
    [This is the initial text.][[]]  
    wxTE_MULTILINE [0 0][300 200])  
TextControlSetFontWeight t wxFONTWEIGHT_BOLD
```

TextControlOnChar *aTextControl commands*

Command to set the custom event handler for the char event of the TextControl *aTextControl* to *commands* . See also OnChar!

Example:

```
t=(TextControl [][TextControl1]  
  [This is the initial text.][[]]  
  wxTE_MULTILINE [0 0][300 200])  
TextControlOnChar lc [pr KeyboardValue]
```

TextControlOnKeyDown *aTextControl commands*

Command to set the custom event handler for the key down event of the TextControl *aTextControl* to *commands* . See also OnKeyDown!

Example:

```
t=(TextControl [][TextControl1]  
  [This is the initial text.][[]]  
  wxTE_MULTILINE [0 0][300 200])  
TextControlOnKeyDown lc [pr KeyboardValue]
```

TextControlOnKeyUp *aTextControl commands*

Command to set the custom event handler for the key up (=release) event of the TextControl *aTextControl* to *commands* . See also OnKeyUp!

Example:

```
t=(TextControl [][TextControl1]
  [This is the initial text.][[]]
  wxTE_MULTILINE [0 0][300 200])
TextControlOnKeyUp lc [pr KeyboardValue]
```

TextControlOnChange *aTextControl commands*

Command to set the custom event handler for the text change event of the TextControl *aTextControl* to *commands* . See also OnKeyUp!

Example:

```
t=(TextControl [][TextControl1]
  [This is the initial text.][[]]
  wxTE_MULTILINE [0 0][300 200])
TextControlOnChange lc [pr KeyboardValue]
```

TextControlOnEnter *aTextControl commands*

Command to set the custom event handler for the text enter event (when [Enter] was pressed) of the TextControl *aTextControl* to *commands* . See also OnKeyUp!

Example:

```
t=(TextControl [][TextControl1]
  [This is the initial text.][[]]
  wxTE_MULTILINE [0 0][300 200])
TextControlOnEnter lc [pr KeyboardValue]
```

TextControlEnable *atextcontrol state*

GUI programming / TextControls / TextControlEnable

Command. If *state* is true then it enables the TextControl aTextControl, if *state* is false then it disables the TextControl.

state must be a boolean.

Example:

```
t=(TextControl [][TextControl1]
  [This is the initial text.][[]]
  wxTE_MULTILINE [0 0][300 200])
TextControlEnable t false
;Click on the TextControl again --nothing happens
TextControlEnable t true      ;enabled again
```

ToggleButtons

...are little windows containing a button with two states, which can run a Logo instructionlist when the user clicks with the mouse on them and toggles them on and off. ToggleButtons are demonstrated in `buttontest.lg`.

ToggleButtons

- `ToggleButton` 556
 - `ToggleButtonDestroy` 557
 - `ToggleButtonOnClick` 557
 - `ToggleButtonValue` 557
 - `ToggleButtonSetValue` 558
 - `ToggleButtonEnable` 558
-

`ToggleButton` *parent alabel onclick*
(**`ToggleButton`** *parent alabel onclick style apos size*)

outputs a new `ToggleButton` on the *parent* (a Frame or a Graph),

having the text *alabel* on it,

and running the *onclick* instructionlist when the user clicks with the mouse on it.

style is the window *style* , defaults to 0.

apos is the position of the Button (a list of two integer numbers, x and y),

size is the *size* of the Button (again a list of two integer numbers, width and height).

Examples:


```

tbpd=ToggleButton [][Pen Down][
    ifelse ToggleButtonValue tbpd [PenDown][PenUp]
    updateGraph]
tbht=(ToggleButton [][Hide Turtle][
    ifelse ToggleButtonValue tbht [hideTurtle][showTurtle]
    updateGraph]
    wxBU_LEFT+wxBU_TOP [0 100][200 100])

```

ToggleButtonDestroy *aToggleButton*

Command that destroys the ToggleButton *aToggleButton* .

Example:

```

tbht=ToggleButton [][Hide Turtle][
    ifelse ToggleButtonValue tbht [hideTurtle][showTurtle]
    updateGraph]
ToggleButtonDestroy tbht

```

ToggleButtonOnClick *aToggleButton instructionlist*

Command that sets the onClick event handler of the ToggleButton *aToggleButton* to the commands in the *instructionlist* .

Examples:

```

tb=ToggleButton [][Hide Turtle][
    ifelse ToggleButtonValue tb [hideTurtle][showTurtle]
    updateGraph]
ToggleButtonOnClick tb [
    ifelse ToggleButtonValue tb [pr [Yes, Sir!]][pr [No, Sir!]]
    updateGraph]
ToggleButtonOnClick tb [] ;like disabling the ToggleButton

```

ToggleButtonValue *aToggleButton*

outputs true if the ToggleButton *aToggleButton* is checked, false otherwise.

Example:

```
tb=ToggleButton [[][Hide Turtle]]  
  ifelse ToggleButtonValue tb [hideTurtle][showTurtle]  
  updateGraph]
```

ToggleButtonSetValue *aToggleButton state*

Command that sets the ToggleButton *aToggleButton* to checked if *state* is true, to unchecked if false.

state must be a boolean.

Example:

```
tb=ToggleButton [[][Hide Turtle]]  
  ifelse ToggleButtonValue tb [hideTurtle][showTurtle]  
  updateGraph]  
ToggleButtonSetValue tb shown?
```

ToggleButtonEnable *aToggleButton state*

Command. If *state* is true then it enables the ToggleButton *aToggleButton* , if *state* is false then it disables the ToggleButton.

state must be a boolean.

Example:

GUI programming / ToggleButtons / ToggleButtonEnable

```
tb=ToggleButton [[Hide Turtle][
    ifelse ToggleButtonValue tb [hideTurtle][showTurtle]
    updateGraph]
ToggleButtonEnable tb false
;Click on the ToggleButton again --nothing happens
ToggleButtonEnable tb true ;enabled again
```

Miscellaneous GUI elements

Miscellaneous GUI elements

- `beginBusyCursor` 560
 - `endBusyCursor` 560
 - `ConsoleSetFocus` 560
-

`beginBusyCursor`

Command to show a busy mouse cursor (hour glass).

`endBusyCursor`

Command to show the standard mouse cursor.

`BusyCursor?`

outputs true if `beginBusyCursor` was called lately, and false if `endBusyCursor` was called lately.

`ConsoleSetFocus`

sets the keyboard focus to the console, which can be wanted if user input is needed.

Sound programming

Sound programming

Here are some commands to control the PC speaker, to play wave sound on the analog sound card output, and for Midi music.

Sound programming

- PC speaker 562
 - Wave Sound 566
 - Midi 569
 - Midi Table 577
 - Midi Control Table 577
 - Midi Glossary 578
 - Midi Instruments 580
-

PC speaker

PC speaker

- Tone 562
 - Tones 562
 - TonesStop 563
 - TonesFinished 563
 - Sound 564
 - Sounds 564
 - SoundsStop 565
 - SoundsFinished 565
-

Tone *pitch duration*

makes the PC speaker output a tone of *pitch* lasting *duration* seconds. The *pitch* is a number, normally an integer, representing a musical note. 0 means A. Its frequency is computed by $440 \cdot 2^{(pitch/12)}$.

Examples:

```
tone 3 1000 ;outputs a C on the PC speaker
tone -12 1000 ;outputs a A (220Hz) on the PC speaker
```

Tones *pitchDurationList* (Tones *pitchs duration* (s))

command that starts a new thread that plays the notes in either the *pitchDurationList* [pitch1 duration1 pitch2 duration2 ...] or from the two lists *pitchs* and durations. *duration* can also be one number.

Examples:

Sound programming / PC speaker / Tones

```
Tones [0 1000 2 1000 4 1000]  
(Tones [0 2 4 5 7 9 11 12] [400 350 300 250 200 150 100 50])  
(Tones [0 2 4 5 7 9 11 12] 400)
```

TonesStop

stops one thread created by Tones. If you have called more than one Tones thread, you can call TonesStop that many times.

TonesFinished

outputs an integer number representing the number of Tones threads that are finished.

Example:

```

to testtones
  tones [0 1000 2 1000 4 1000 5 1000] waitMS 300
  tones [4 1000 5 1000 7 1000] waitMS 300
  tones [7 1000 9 1000]
  tf=TonesFinished
  otf=tf
  print tf
  tf=tf+3
  overwriteMode
  y=Cursor.2-1
  while [TonesFinished < tf]
    [ if TonesFinished != otf
      [ otf=TonesFinished
        setCursor list 0 y
        type otf
      ]
    ]
  insertMode
  pr []
end

```

Sound *frequency duration*

makes the PC speaker output a tone of *frequency* lasting *duration* seconds. The *frequency* must be a positive number. If the *duration* is -1, the sound never stops.

Example:

```
Sound 440 1000
```

Sounds *frequencyDurationList* (Sound *frequencies durations*)

command that starts a new thread that plays the *frequencies* in either the *frequencyDurationList* [frequency1 duration1 frequency2 duration2 ...] or from the two lists *frequencies* and *durations* .

Sound programming / PC speaker / Sounds

durations can also be one number.

Examples:

```
Sounds [440 400 880 300 1760 200]
(Sounds [440 880 1760] [400 300 200])
(Sounds [440 880 1760] 400)
(Sounds (se
  rSeq 20 10000 100
  rSeq 10000 20 100) 10)
(Sounds (sin rSeq 0 15*360 200)*4000+4000 10)
waitMS 2000
(Sounds (sin rSeq 0 3*360 200)*1000+1000 10)
```

SoundsStop

stops one thread created by Sounds. If you have called more than one Sounds thread, you can call SoundsStop that many times to stop all threads.

SoundsFinished

outputs an integer number representing the number of Tones threads that are finished. See also TonesFinished.

Wave Sound

Wave Sound

- playWave 566
- playWaveFast 567

playWave *wavefile flags*

playWave *wavedata flags*

The *wavefile* must be a filename of a .WAV file. The *wavedata* can be an IntArray or an Int16Array loaded with a correct wave file contents including the header.

The *flags* describe how the sound should be played.

However this command does not require a true sound card to enjoy.

There is a publicly available driver from Microsoft called SPEAKER(.EXE, .DRV, .ZIP ?) which will emulate the wave file capabilities over the pc speaker.

wavefile (String): The name of .WAV file.

flags (Integer): A flag to indicate how you want the sound played.

<p>0 = Synchronous does not return until completed. 1 = Asynchronous returns immediately while sound is still playing. 2 = Dont use the default sound if the specified one cannot be found. 4 = In memory sound (automatically set if <i>wavedata</i> is given). 8 = Continue to loop the sound until another sound command is issued. 16 = Dont stop an already playing sound.</p>

Note: these *flags* can be combined by adding them together.

Sound programming / Wave Sound / playWave

Under Linux those *flags* have no meaning at the moment.

Hint: You can write portable sounds if you check the LogoVersion.

Example:

```
playwave "c:\\windows\\tada.wav 1+8  
playwave [ ] 0
```

See also [makewav3.lg](..\makewav3.lg).

playWaveFast *wavedata*

The *wavedata* can be an IntArray or an Int16Array loaded with a correct wave file contents including the header.

Up to 32 waves can be simultaneously playing, and the startup time is very short, so it's ideal for games and music programs.

Example:

```
to loadwav f
  local [size wav]
  openReadBin f
  setReader f
  size=FileSize f
  wav=readInt16ArrayBin size/2
  setReader []
  close f
; (pr f "loaded)
  output wav
end
to resizeWav wav factor
  local "w
  w=Int16Array 44+round ((count wav)-44)*factor
  setItems 1 w Items 1 44 wav
  setItems 45 w resize
    (Items 45 count wav wav) (count w)-44
  output w
end
clickWav=loadwav "start.wav
cmajor=[0 2 4 5 7 9 11 12]
stoneWav=Array 8
repeat 8 [
  i=repcount
  stoneWav.i=resizeWav clickWav 2^((12-cmajor.i)/12)
]
repeat 100 [playWaveFast stonewav.(1+random 8) waitms 10]
```

Midi

Midi

- MidiCountDevices 569
- MidiDeviceInfo 569
- MidiOpen 570
- MidiClose 571
- MidiMessage 571
- MidiProgramChange 573
- MidiNoteOn 574
- MidiNoteOff 574
- MidiAllSoundsOff 575
- MidiOutputStream 575
- MidiOutputStreamsStart 576
- MidiOutputStreamsStop 576
- MidiOutputStreamsFinished 576

devnr **MidiCountDevices**

outputs the number of available Midi devices devnr.

Example:

<pre>MidiCountDevices ; 4 ; -)</pre>

devInfo **MidiDeviceInfo** *devnr*

outputs hardware-specific device information on the Midi device number *devnr* .

The first item of the output is either "input" or "output", which means that device is either a input or output device.

Example:

```
repeat MidiCountDevices [print MidiDeviceInfo repcount-1]  
;output Microsoft MIDI-Mapper MMSystem  
;input MPU-401 MMSystem  
;output Microsoft GS Wavetable SW Synth MMSystem  
;output MPU-401 MMSystem
```

MidiOpen *outputDeviceNr*

This command opens the Midi device number *outputDeviceNr* and accesses it through a Midi device driver.

The device driver chosen depends on several things.

It lets you choose any available Midi driver available on your system.

The id starts at 0 up to 1 less than number of Midi drivers available.

To determine which id maps to which driver try several MidiOpen commands with increasing id.

MidiOpen will output the name of the driver being used.

device:(List) When successfully opened outputs the name of the driver being used.

id:(Integer) Is an index that specifies which Midi device driver you wish to open. When no id given it selects the Midi Mapper device driver.

Basically Midi allows you to generate sound on your sound card.

You will need to install the appropriate drivers under Windows for your sound card to work.

Most sound cards come with a Midi player which basically reads Midi messages from the file and passes them to the Midi driver.

Sound programming / Midi / MidiOpen

If you have your Midi player working under Windows then aUCBLogo should work too.

Midi commands in aUCBLogo do not yet support Midi files (.MID or .MDI).

Instead you directly build up sequences of Midi messages and send them directly to the Midi device.

You can think of Midi commands in aUCBLogo as a programmable keyboard.

It just so happens that the link between your programmable keyboard (Midi commands) and your Speaker is Midi.

Example:

```
MidiOpen 0  
MidiNoteOn 0 60 127  
MidiClose
```

MidiClose

This command closes the Midi device. There are no inputs or outputs.

See also the MIDIOPEN command.

Example:

```
MidiOpen 0  
MidiNoteOn 0 60 127  
MidiClose
```

MidiMessage *message*

The input must be a list. You must already of issued a MidiOpen command to use this command.

message :(List) A Midi *message* in one of 3 forms explained below.

There are 3 forms of the Midi *message* , the Short form, the Long form and the System Exclusive form.

Short form *message* :

[status data1 data2]

The Short form is the common form and always has a 3 integer list.

The first integer is known as the Status BYTE (it can also be thought of as a COMMAND BYTE).

It must be followed by 2 data bytes even if the *message* requires only 1 (just use 0).

status:(Integer) In Midi terminology this is the "status byte" but I call it the "command code" in the table below.

data1:(Integer) Data byte 1 of the Midi "command code".

data2:(Integer) Data byte 2 of the Midi "command code".

Long form *message* :

[status data1 data2 status data1 data2 ...]

The Long form is similar to the Short form but integer list contains many short messages (triples).

System Exclusive form *message* :

[240 data1 data2 data3 data4 ...]

Sound programming / Midi / MidiMessage

The SYSTEM EXCLUSIVE form must be led by the system exclusive status byte 240 (F0 hex).

It can then be followed by any amount of data bytes.

data1:(Integer) First data byte specific to your midi device.

data2:(Integer) First data byte specific to your midi device.

and so on.

See the MIDI TABLE which basically is a Specification of the Midi Message.

This documentation is not attempt to teach you Midi. But there is enough information here to hopefully get you started.

For more information you may be interested in purchasing a book on Midi such as:

Midi BASICS by Akira Otsuka and Akihiko Nakajima.

Example:

```
MidiOpen 0
MidiMessage (List 192+13 56 0 192+13 56 0)
MidiMessage (List 144+13 58 100)
;Listen to tone.
MidiClose
```

MidiProgramChange *channel program*

command to change the *program* (the timbre) of the Midi *channel channel* to *program* .

channel must be an integer in the range 0..15.

program must be an integer in the range 0..127.

Example:

```
MidiOpen 0
MidiProgramChange 0 127
MidiNoteOn 0 58 127
MidiClose
```

MidiNoteOn *channel pitch velocity*

command to turn the note with *pitch* of the *channel* on. The *velocity* is the speed with which a key is pressed and directly maps to the volume of the sound.

channel must be an integer in the range 0..15.

pitch must be a number in the range 0..127.

velocity must be an integer in the range 0..127.

Example:

```
MidiOpen 0
MidiNoteOn 0 58 127
MidiClose
```

MidiNoteOff *channel pitch velocity*

command to turn the note with *pitch* of the *channel* off. The *velocity* is the speed with which a key is released.

channel must be an integer in the range 0..15.

pitch must be a number in the range 0..127.

velocity must be an integer in the range 0..127.

Example:

Sound programming / Midi / MidiNoteOff

```
MidiOpen 0
MidiNoteOn 0 58 127
MidiNoteOff 0 58 127
MidiClose
```

MidiAllSoundsOff

command to turn off all running Midi sounds of the selected device.

Example:

```
MidiOpen 0
MidiProgramChange 0 16
MidiNoteOn 0 58 127
MidiNoteOn 0 58+5 127
MidiNoteOn 0 58+9 127
MidiAllSoundsOff
MidiClose
```

MidiOutputStream *channel tonelist*

(**MidiOutputStream** *channel pitches durations velocities*)

command to start a new thread which will play the Midi tones in *tonelist* or in the lists *pitches* , *durations* and *velocities* .

The created threads must be activated using the command **MidiOutputStreamsStart**.

tonelist is a list of triples of numbers: [pitch1 duration1 velocity1 pitch2 duration2 velocity2 ...]

channel must be an integer in the range 0..15.

pitch must be a number in the range 0..127.

velocity must be an integer in the range 0..127.

duration must be a number and is measured in milliseconds.

Example:

```
MidiOpen 0
MidiOutputStream 0 [60 1000 127 62 1000 127 64 1000 127 65 1000
127]
MidiOutputStream 1 [64 1000 127 65 1000 127 67 1000 127]
MidiOutputStream 2 [67 1000 127 69 1000 127]
MidiOutputStreamsStart
MidiClose
```

MidiOutputStreamsStart

command to activate the waiting threads which were created with MidiOutputStream.

Example:

```
MidiOpen 0
MidiOutputStream 0 [60 1000 127 62 1000 127 64 1000 127 65 1000
127]
MidiOutputStream 1 [64 1000 127 65 1000 127 67 1000 127]
MidiOutputStream 2 [67 1000 127 69 1000 127]
MidiOutputStreamsStart
MidiClose
```

MidiOutputStreamsStop

command to stop the threads created by MidiOutputStream and maybe activated by MidiOutputStreamsStart.

MidiOutputStreamsFinished

Sound programming / Midi / MidiOutputStreamFinished

outputs the number of finished MidiOutputStream threads. This is useful for determining the end of a Midi sound.

Midi Table

Command Name	Command Code	Data Byte 1	Data Byte 2
Note Off	128 + Channel	0-127 Pitch	0-127 Velocity
Note On	144 + Channel	0-127 Pitch	0-127 Velocity
Poly Pressure	160 + Channel	0-127 Pitch	0-127 Pressure
Control Change	176 + Channel	0-127 Midi Control	0-127 MSB
Program Change	192 + Channel	0-127 Program	Not used
Channel Pressure	208 + Channel	0-127 Pressure	Not used
Pitch Wheel	224 + Channel	0-127 LSB	0-127 MSB
System Exclusive	240	0-127 Id Code	Any number of bytes
Undefined	241	Not used	Not used
Song Position	242	0-127 LSB	0-127 MSB
Song Select	243	0-127 Song	Not used
Undefined	244	Not used	Not used
Undefined	245	Not used	Not used
Tune Request	246	Not used	Not used
End of Exclusive	247	Not used	Not used
Timing Clock	248	Not used	Not used
Undefined	249	Not used	Not used
Start	250	Not used	Not used
Continue	251	Not used	Not used
Stop	252	Not used	Not used
Undefined	253	Not used	Not used
Active Sensing	254	Not used	Not used
System Reset	255	Not used	Not used

See also the MIDI GLOSSARY.

Midi Control Table

Command Name	Command Code	Data Byte 1	Data Byte 2
Control Change	176 + Channel 0	Undefined	0-127 MSB
Control Change	176 + Channel 1	Modulation Wheel	0-127 MSB
Control Change	176 + Channel 2	Breath Controller	0-127 MSB
Control Change	176 + Channel 3	After Touch	0-127 MSB
Control Change	176 + Channel 4	Foot Controller	0-127 MSB
Control Change	176 + Channel 5	Portamento Time	0-127 MSB
Control Change	176 + Channel 6	Data Entry	0-127 MSB
Control Change	176 + Channel 7	Main Volume	0-127 MSB
Control Change	176 + Channel 8-31	Undefined	0-127 MSB
Control Change	176 + Channel 32-63	LSB of 0-31	0-127 LSB
Control Change	176 + Channel 64	Damper Pedal	0:Off 127:On
Control Change	176 + Channel 65	Portamento	0:Off 127:On
Control Change	176 + Channel 66	Sostenuto	0:Off 127:On
Control Change	176 + Channel 67	Soft Pedal	0:Off 127:On
Control Change	176 + Channel 68-92	Undefined	0:Off 127:On
Control Change	176 + Channel 93	Chorus	0:Off 127:On
Control Change	176 + Channel 94	Celeste	0:Off 127:On
Control Change	176 + Channel 95	Phaser	0:Off 127:On
Control Change	176 + Channel 96	Data Entry + 1	0:Off 127:On
Control Change	176 + Channel 97	Data Entry - 1	0:Off 127:On
Control Change	176 + Channel 98-121	Undefined	0:Off 127:On
Control Change	176 + Channel 122	Local Control	0-127
Control Change	176 + Channel 123	All Notes Off	0
Control Change	176 + Channel 124	Omni Mode off	0-15
Control Change	176 + Channel 125	Omni Mode on	0
Control Change	176 + Channel 126	Mono on/Poly off	0
Control Change	176 + Channel 127	Poly on/Mono off	0

See also the MIDI GLOSSARY.

Midi Glossary

Sound programming / Midi Glossary

Channel: A channel is a number from 0-15 which corresponds to channels 1-16.

Pitch: A pitch is a number from 0-127 and corresponds to a note on the instrument.

Velocity: A velocity is a number from 0-127 and corresponds to how fast the key (or string) is pressed or released (most terminology is in reference to keyboards). 0 means is released.

Pressure: A pressure is a number from 0-127 and corresponds to the characteristics of how the key is hit.

Program: A program is a number from 0-127 and corresponds to the instrument to use. See the MIDI INSTRUMENTS table.

MSB: Most Significant Bits.

LSB: Least Significant Bits.

Id Code: Manufactures Id Code. Used to enter System Exclusive Mode which is specific to the Manufacturer of the device.

Song: A song is a rhythm machine.

Midi Instruments

Midi Instruments

- Pianos 580
- Chromatic 580
- Organs 581
- Guitars 581
- Basses 581
- Strings 582
- Ensembles 582
- Brass 582
- Reed 583
- Pipes 583
- Synth-Lead 583
- Synth-Pad 584
- Synth-Effects 584
- Ethnic 584
- Percussive 585
- Soundeffects 585

Pianos

0	- Acoustic Grand Piano
1	- Bright Acoustic Piano
2	- Electric Grand Piano
3	- Honky-tonk Piano
4	- Rhodes Piano
5	- Chorused Piano
6	- Harpsichord
7	- Clavinet

Chromatic *Percussion*

8 -	Celesta
9 -	Glockenspiel
10 -	Music box
11 -	Vibraphone
12 -	Marimba
13 -	Xylophone
14 -	Tubular Bells
15 -	Dulcimer

Organs

16 -	Hammond Organ
17 -	Percussive Organ
18 -	Rock Organ
19 -	Church Organ
20 -	Reed Organ
21 -	Accordion
22 -	Harmonica
23 -	Tango Accordion

Guitars

24 -	Acoustic Guitar (nylon)
25 -	Acoustic Guitar (steel)
26 -	Electric Guitar (jazz)
27 -	Electric Guitar (clean)
28 -	Electric Guitar (muted)
29 -	Overdriven Guitar
30 -	Distortion Guitar
31 -	Guitar Harmonics

Basses

32 -	Acoustic Bass
33 -	Electric Bass (finger)
34 -	Electric Bass (pick)
35 -	Fretless Bass
36 -	Slap Bass 1
37 -	Slap Bass 2
38 -	Synth Bass 1
39 -	Synth Bass 2

Strings

40 -	Violin
41 -	Viola
42 -	Cello
43 -	Contrabass
44 -	Tremolo Strings
45 -	Pizzicato Strings
46 -	Orchestral Harp
47 -	Timpani

Ensembles

48 -	String Ensemble 1
49 -	String Ensemble 2
50 -	Synth Strings 1
51 -	Synth Strings 2
52 -	Choir Aahs
53 -	Voice Oohs
54 -	Synth Voice
55 -	Orchestra Hit

Sound programming / Midi Instruments / Brass

Brass

56 -	Trumpet
57 -	Trombone
58 -	Tuba
59 -	Muted Trumpet
60 -	French Horn
61 -	Brass Section
62 -	Synth Brass 1
63 -	Synth Brass 2

Reed

64 -	Soprano Sax
65 -	Alto Sax
66 -	Tenor Sax
67 -	Baritone Sax
68 -	Oboe
69 -	English Horn
70 -	Bassoon
71 -	Clarinet

Pipes

72 -	Piccolo
73 -	Flute
74 -	Recorder
75 -	Pan Flute
76 -	Bottle Blow
77 -	Shakuhachi
78 -	Whistle
79 -	Ocarina

Synth-Lead

80 -	Lead 1 (square)
81 -	Lead 2 (sawtooth)
82 -	Lead 3 (caliope lead)
83 -	Lead 4 (chiff lead)
84 -	Lead 5 (charang)
85 -	Lead 6 (voice)
86 -	Lead 7 (fifths)
87 -	Lead 8 (brass + lead)

Synth-Pad

88 -	Pad 1 (new age)
89 -	Pad 2 (warm)
90 -	Pad 3 (polysynth)
91 -	Pad 4 (choir)
92 -	Pad 5 (bowed)
93 -	Pad 6 (metallic)
94 -	Pad 7 (halo)
95 -	Pad 8 (sweep)

Synth-Effects

96 -	FX 1 (rain)
97 -	FX 2 (soundtrack)
98 -	FX 3 (crystal)
99 -	FX 4 (atmosphere)
100 -	FX 5 (brightness)
101 -	FX 6 (goblins)
102 -	FX 7 (echoes)
103 -	FX 8 (sci-fi)

Ethnic

104 - Sitar
105 - Banjo
106 - Shamisen
107 - Koto
108 - Kalimba
109 - Bagpipe
110 - Fiddle
111 - Shanai

Percussive

112 - Tinkle Bell
113 - Agogo
114 - Steel Drums
115 - Woodblock
116 - Taiko Drum
117 - Melodic Tom
118 - Synth Drum
119 - Reverse Cymbal

Soundeffects

120 - Guitar Fret Noise
121 - Breath Noise
122 - Seashore
123 - Bird Tweet
124 - Telephone Ring
125 - Helicopter
126 - Applause

- `_defMacro` 387
- `_eq` 67
- `_Macro` 387
- `_maybeOutput` 369
- `_setBF` 60
- `_setButFirst` 60
- `_setFirst` 60
- `_setItem` 58
- `_setPos` 205
- `_setPosXYZ` 206
- ``` 356
- `abs` 143
- Absolute Turtle Motion 200
- `addColors` 292
- `addColorsMod` 292
- `allFullScreen` 228
- `allOpen` 106
- `AlNumP` 74
- `AlphaP` 74
- `and` 175
- `and2` 176
- `apply` 378
- `Arc` 255
- `arc2` 256
- `ArcCos` 150
- `ArcSin` 149
- `ArcTan` 150
- Arithmetic 133
- Arithmetic Mutators 160
- Arithmetic Predicates 162
- `Arity` 325
- `Array` 37
- `ArrayP` 65
- ASCII 83
- ASCIIP 75
- `aShift` 173
- `axes` 232
- `back` 196
- `backslashedP` 73
- `BackslashEncode` 86
- Basses 581
- `beforeP` 67
- `beginBusyCursor` 560
- Berkeley Logo User Manual 27
- `butFirst` 51
- `butFirsts` 51
- `BigFloat` 140
- `BigFloatSetPrecision` 140
- `BitAnd` 172
- `BitCopy` 281
- `BitItem` 54
- `BitMakeTransparent` 283
- `Bitmaps` 281
- `BitMaxX` 284
- `BitMaxY` 284
- `BitNot` 173
- `BitOr` 172
- `BitPaste` 281
- `BitPixel` 284
- `BitSetPixel` 283
- `BitMakeTransparent` 283
- Bitwise Operations 172
- `BitXOr` 173
- `back` 196
- `butLast` 52
- `boldTextMode` 122
- `BoxSizer` 439
- `BoxSizerAdd` 440
- `BoxSizerDestroy` 442
- `BoxSizers` 439
- Brass 582
- `break` 374
- Buglist for version 4.65 21
- `buried` 322
- `bury` 335
- `buryall` 335
- `buryname` 336
- `butFirst` 51
- `butFirsts` 51
- `butLast` 52
- `Button` 443
- `ButtonDestroy` 444
- `ButtonEnable` 444
- `ButtonOnClick` 444
- `Buttons` 443
- `bye` 354
- Calls 231
- `cascade` 384
- `cascade2` 385
- `case` 366
- `CaseIgnoredP` 319
- `castShadows` 303
- `catch` 368
- `changeDir` 112
- `changeDir` 112
- `Char` 84
- `CharP` 70
- `CharUnderCursor` 120
- `check` 361
- `CheckBox` 446
- `CheckBoxDestroy` 447
- `CheckBoxEnable` 448
- `CheckBoxes` 446
- `CheckBoxOnClick` 447
- `CheckBoxSet` 448
- `CheckBoxValue` 448
- `ChoiceBox` 450
- `ChoiceBoxAppend` 453
- `ChoiceBoxCount` 454
- `ChoiceBoxDestroy` 451
- `ChoiceBoxEnable` 460
- `ChoiceBoxes` 450
- `ChoiceBoxOnChar` 458
- `ChoiceBoxOnKeyDown` 458
- `ChoiceBoxOnKeyUp` 459
- `ChoiceBoxOnSelect` 459
- `ChoiceBoxRemoveItem` 454
- `ChoiceBoxSelection` 451
- `ChoiceBoxSetBackgroundColor` 455
- `ChoiceBoxSetChoices` 452
- `ChoiceBoxSetColor` 455
- `ChoiceBoxSetFontName` 456
- `ChoiceBoxSetFontSize` 456
- `ChoiceBoxSetFontStyle` 457
- `ChoiceBoxSetFontWeight` 457
- `ChoiceBoxSetItem` 453
- `ChoiceBoxSetSelection` 452
- Chromatic 580
- `circle` 254

Index

- circularP 74
- clean 220
- clearPortBit 125
- clearScreen 220
- clearShadows 303
- clearText 119
- close 105
- closeall 106
- congong 256
- CntrlP 75
- continue 360
- Color database 179
- combine 44
- ComboBox 461
- ComboBoxAppend 466
- ComboBoxCount 468
- ComboBoxDestroy 463
- ComboBoxEnable 482
- ComboBoxes 461
- ComboBoxOnChange 480
- ComboBoxOnChar 476
- ComboBoxOnEnter 481
- ComboBoxOnKeyDown 477
- ComboBoxOnKeyUp 478
- ComboBoxOnSelect 479
- ComboBoxRemoveItem 467
- ComboBoxSelection 463
- ComboBoxSetBackgroundColor 470
- ComboBoxSetChoices 465
- ComboBoxSetColor 471
- ComboBoxSetFontName 473
- ComboBoxSetFontSize 472
- ComboBoxSetFontStyle 474
- ComboBoxSetFontWeight 475
- ComboBoxSetItem 466
- ComboBoxSetSelection 464
- ComboBoxSetValue 469
- ComboBoxValue 468
- ignore 356
- Communication 88
- complexP 73
- cond 366
- Conditional execution 364
- conjugate 56
- ConsoleSetFocus 560
- Constructors 34
- Contents 3
- Contents 321
- continue 360
- continueLoop 375
- Control Structures 354
- copyDef 310
- Cos 147
- count 81
- cross 157
- crossmap 383
- clearScreen 220
- CSymP 76
- clearText 119
- Cursor 120
- Custom Event Handlers 399
- Data Structure Primitives 33
- deepCopy 48
- define 308
- definedP 319
- deleteTextures 300
- dequeue 62
- Difference 136
- DigitP 76
- dir 91
- Direct Graphics 286
- DirectoryP 111
- dirlg 91
- DirSelector 413
- disableCylinderLines 248
- disableDepthTest 248
- disableDither 249
- disableFog 250
- disableLighting 249
- disableLineSmooth 245
- disablePointSmooth 249
- disablePolySmooth 246
- disableRoundLineEnds 247
- disableShadows 302
- disableTextMouseEvents 123
- disableTexture 300
- disableCylinderLines 248
- disableDepthTest 248
- disableLineSmooth 245
- dispatchMessages 231
- displaymatrix 91
- disablePolySmooth 246
- disableRoundLineEnds 247
- Distance 216
- DistanceXYZ 216
- disableTexture 300
- DynamicLibraryCall 130
- do_until 374
- do_while 374
- doubleBuffer 229
- downPitch 198
- downPitch 198
- drawGraphic 265
- Drawing Curves 254
- Drawing filled shapes 258
- dribble 107
- Dynamic Libraries 129
- DynamicLibrary 129
- DynamicLibraryCall 130
- edall 348
- edit 347
- edall 348
- edit 347
- editFile 348
- Editing 347
- edn 349
- edns 348
- edpl 349
- edpls 349
- edps 348
- Ellipse 255
- EllipseArc 254
- Ellipsoid 268
- emptyP 66
- enable and disable flags 245
- enableCylinderLines 247
- enableDepthTest 248
- enableDither 249
- enableFog 249

- enableLighting 248
- enableLineSmooth 245
- enablePointSmooth 249
- enablePolySmooth 246
- enableRoundLineEnds 246
- enableShadows 302
- enableTextMouseEvents 123
- enableTexture 300
- enableCylinderLines 247
- endBusyCursor 560
- endSurfaceStart 261
- enableDepthTest 248
- enableLineSmooth 245
- enablePolySmooth 246
- enableRoundLineEnds 246
- Ensembles 582
- Entering and leaving Logo 28
- enableTexture 300
- Environment 115
- EofP 110
- equalP 66
- erase 350
- eraseAll 350
- erase 350
- eraseAll 350
- eraseFile 106
- eraseNames 351
- eraseProcedures 351
- erasePropertyLists 351
- Erasing 350
- eraseFile 106
- ern 351
- eraseNames 351
- erpl 352
- erasePropertyLists 351
- eraseProcedures 351
- erract 394
- error 369
- Error Processing 391
- Ethnic 584
- exp 145
- factorize 152
- Faculty 151
- FloodColor 252
- forward 195
- Fence 221
- FileAccess 100
- FileP 111
- Files 113
- FileSelector 414
- FileSize 111
- FileTime 111
- fill 223
- fillCircle 266
- fillEllipse 267
- fillPie 267
- fillRect 266
- filter 382
- find 382
- first 49
- firsts 50
- Float 139
- float2ratio 167
- FloatArray 40
- FloatControl 484
- FloatControlDestroy 485
- FloatControlEnable 488
- FloatControlOnChange 487
- FloatControls 484
- FloatControlSetRange 487
- FloatControlSetValue 486
- FloatControlValue 486
- floatP 72
- FloodColor 252
- Fog 272
- for 372
- foreach 379
- forever 372
- Form 170
- forward 195
- fPut 36
- Frame 422
- FrameDestroy 424
- FrameEnable 426
- FrameFullScreen 427
- FrameIconize 427
- FrameMaximize 427
- FrameOnChar 425
- FrameOnKeyDown 425
- FrameOnKeyUp 425
- Frames 422
- FrameSetBackgroundColor 429
- FrameSetClientSize 428
- FrameSetColor 428
- FrameSetFocus 426
- FrameSetFontName 429
- FrameSetFontSize 429
- FrameSetFontStyle 430
- FrameSetFontWeight 430
- FrameSetShape 430
- FrameSetSizer 431
- fullScreen 227
- fullScreen 227
- fullText 310
- Gauge 489
- GaugeDestroy 490
- Gauges 489
- GaugeSetBackgroundColor 491
- GaugeSetColor 491
- GaugeSetRange 491
- GaugeSetValue 490
- GaugeValue 490
- GC 344
- gcd 155
- genSym 47
- getColorDatabase 293
- getColorFromUser 414
- getFontFromUser 415
- getMultipleChoices 415
- getNumberFromUser 416
- getPasswordFromUser 417
- getPortBit 125
- getProperty 316
- getSingleChoice 418
- getSingleChoiceIndex 419
- getTextFromUser 417
- getWorkingDirectory 112
- goTo 355
- getProperty 316

Index

- Graph 432
- GraphCurrent 433
- GraphDestroy 433
- GraphicEnd 264
- Graphics 178
- GraphicStart 264
- GraphOnChar 434
- GraphOnKeyDown 435
- GraphOnKeyUp 435
- GraphOnMouseLeftDClick 437
- GraphOnMouseLeftDown 435
- GraphOnMouseLeftUp 436
- GraphOnMouseMiddleDClick 438
- GraphOnMouseMiddleDown 436
- GraphOnMouseMiddleUp 437
- GraphOnMouseMotion 438
- GraphOnMouseRightDClick 437
- GraphOnMouseRightDown 436
- GraphOnMouseRightUp 437
- GraphP 77
- Graphs 432
- GraphSetCurrent 434
- greaterEqualP 163
- greaterP 162
- GUI programming 396
- Guitars 581
- h 343
- Heading 214
- help 342
- hex 171
- hideTurtle 219
- Home 200
- HSB 290
- HSBA 290
- hideTurtle 219
- IdentityMatrix 297
- IdentityMatrix 297
- if 364
- ifElse 365
- ifFalse 365
- ifFalse 365
- ifTrue 365
- ifTrue 365
- ignore 356
- imag 56
- insertMode 122
- Inspection 328
- Int 140
- Int16 141
- Int16Array 39
- Int16P 71
- Int8 141
- Int8P 71
- IntArray 38
- IntControl 493
- IntControlDestroy 494
- IntControlEnable 495
- IntControlOnChange 495
- IntControls 493
- IntControlSetRange 495
- IntControlSetValue 494
- IntControlValue 494
- intForm 170
- IntP 70
- invertMatrix 157
- invoke 379
- iSeq 154
- Item 52
- items 54
- KeyboardValue 401
- KeyP 119
- Label 223
- LabelAlign 226
- LabelFont 224
- LabelSize 224
- LabelWeight 225
- last 50
- lcm 155
- left 196
- leftPortShift 125
- leftRoll 197
- lessEqualP 163
- lessP 162
- Lighting 270
- Line 288
- List 35
- ListBox 497
- ListBoxAppend 500
- ListBoxCount 501
- ListBoxDestroy 499
- ListBoxEnable 505
- ListBoxes 497
- ListBoxOnChar 504
- ListBoxOnDClick 505
- ListBoxOnKeyDown 504
- ListBoxOnKeyUp 504
- ListBoxOnSelect 505
- ListBoxRemoveItem 501
- ListBoxSelections 499
- ListBoxSetBackgroundColor 502
- ListBoxSetChoices 500
- ListBoxSetColor 502
- ListBoxSetFontName 503
- ListBoxSetFontSize 502
- ListBoxSetFontStyle 503
- ListBoxSetFontWeight 503
- ListBoxSetItem 501
- ListBoxSetSelections 500
- ListControl 508
- ListControlColumn 518
- ListControlColumnCount 517
- ListControlDeleteItem 513
- ListControlDestroy 510
- ListControlEnable 528
- ListControlGet 516
- ListControlGetColumn 516
- ListControlGetItem 512
- ListControlGetRow 515
- ListControlInsertColumn 511
- ListControlInsertItem 511
- ListControlItemCount 517
- ListControlOnChar 525
- ListControlOnColClick 527
- ListControlOnItemActivated 527
- ListControlOnItemSelected 526
- ListControlOnKeyDown 525
- ListControlOnKeyUp 526
- ListControlRow 519
- ListControls 507

- ListControlSet 515
- ListControlSetBackgroundColor 522
- ListControlSetColor 522
- ListControlSetColumn 514
- ListControlSetFontName 523
- ListControlSetFontSize 523
- ListControlSetFontStyle 524
- ListControlSetFontWeight 524
- ListControlSetItem 512
- ListControlSetRow 513
- ListControlSort 521
- ListControlText 520
- ListP 65
- LN 146
- load 341
- loadImage 282
- loadNoisily 394
- loadpalette 113
- loadPicture 275
- loadPicture 275
- loadPictureText 276
- local 313
- localmake 313
- Log10 145
- Logical Operations 175
- LogoComspec 115
- LogoEditor 116
- LogoHelpDir 116
- LogoLibDir 117
- LogoTempDir 117
- LogoVersion 98
- Loops 371
- lowerCase 84
- LowerP 77
- lowPassFilter 156
- lPut 37
- leftRoll 197
- lShift 174
- left 196
- macroexpand 390
- MacroP 390
- Macros 387
- make 312
- makeDirectory 112
- MandelIterate 158
- map 380
- map_se 381
- Matrix 294
- max 153
- maxNorm 153
- mdarray 40
- mdItem 53
- mdSetItem 58
- Member 84
- MemberP 68
- merge 46
- mergePairs 46
- mergeSort 47
- MessageBox 419
- Midi 569
- Midi Control Table 577
- Midi Glossary 578
- Midi Instruments 580
- Midi Table 577
- MidiAllSoundsOff 575
- MidiClose 571
- MidiCountDevices 569
- MidiDeviceInfo 569
- MidiMessage 571
- MidiNoteOff 574
- MidiNoteOn 574
- MidiOpen 570
- MidiOutputStream 575
- MidiOutputStreamsFinished 576
- MidiOutputStreamsStart 576
- MidiOutputStreamsStop 576
- MidiProgramChange 573
- min 152
- Minus 137
- MIPS 128
- Miscellaneous GUI elements 560
- Modulo 139
- Modulo 139
- MouseButtons 235
- MousePos 234
- Multiple Turtles 237
- Mutators 57
- name 312
- NameInTableP 69
- namelist 324
- NameP 319
- Names 323
- newTurtle 237
- Nodes 326
- noDribble 107
- noRefresh 229
- Norm 153
- not 177
- notFullScreen 228
- notPortBit 125
- NumberP 69
- Numeric Operations 134
- OnChar 400
- OnKeyDown 400
- OnKeyUp 400
- OnMouseLeftDClick 411
- OnMouseLeftDown 409
- OnMouseLeftUp 410
- OnMouseMiddleDClick 411
- OnMouseMiddleDown 409
- OnMouseMiddleUp 410
- OnMouseMotion 411
- OnMouseRightDClick 411
- OnMouseRightDown 409
- OnMouseRightUp 410
- OnTextMouseLeftDClick 407
- OnTextMouseLeftDown 405
- OnTextMouseLeftUp 406
- OnTextMouseMiddleDClick 408
- OnTextMouseMiddleDown 405
- OnTextMouseMiddleUp 407
- OnTextMouseMotion 408
- OnTextMouseRightDClick 407
- OnTextMouseRightDown 405
- OnTextMouseRightUp 406
- output 356
- openAppend 104
- openAppendBin 104
- openRead 100

Index

- openReadBin 102
- openUpdate 105
- openUpdateBin 105
- openWrite 103
- openWriteBin 103
- or 175
- or2 176
- Organs 581
- Orientation 215
- OSScreenSize 98
- output 356
- overwriteMode 122
- Palette 252
- parse 86
- partialEllipsoid 268
- pause 359
- PenColor 251
- PC speaker 562
- PenDown 238
- PenErase 239
- pen 252
- Pen and Background Control 238
- Pen Queries 251
- PenColor 251
- PenDown 238
- PenDownP 251
- PenErase 239
- PenMode 251
- PenPaint 239
- PenSize 252
- PenReverse 239
- PenSize 252
- PenUp 238
- Percussive 585
- perspective 221
- Pianos 580
- pick 54
- Pictures 275
- Pipes 583
- Pitch 214
- Pixel 217
- plainTextMode 122
- playWave 566
- playWaveFast 567
- PropertyList 316
- PropertyLists 324
- plist 325
- printOut 328
- poall 329
- PolyEnd 259
- PolyStart 258
- pon 331
- pons 330
- pop 61
- popl 331
- popls 330
- popMatrix 296
- pops 329
- Port Input and Output 124
- PortIn 124
- PortOut 124
- Pos 213
- PosXYZ 213
- printOutTitles 332
- pots 332
- Power 145
- putProperty 315
- PenPaint 239
- print 89
- Predicates 64
- primeP 164
- PrimitiveP 318
- Primitives 323
- print 89
- Print formatting 170
- printDepthLimit 394
- printOut 328
- printOutTitles 332
- PrintP 78
- printWidthLimit 394
- Procedure Definition 306
- ProcedureP 318
- Procedures 323
- Product 137
- profile 362
- Projection Matrix 294
- Property Lists 315
- PropertyList 316
- PropertyLists 324
- PenUp 238
- PunctP 78
- push 61
- pushMatrix 296
- putProperty 315
- PenReverse 239
- Queries 81
- queue 62
- quoted 55
- Quotient 138
- radArcCos 150
- radArcSin 149
- radArcTan 151
- radCos 148
- radd 168
- RadioButton 530
- RadioButtonDestroy 531
- RadioButtonEnable 532
- RadioButtonOnClick 531
- RadioButtons 530
- RadioButtonSet 532
- RadioButtonValue 532
- radSin 147
- radTan 148
- random 165
- Random Numbers 165
- ratio 167
- ratio2float 168
- Rational numbers 167
- rawASCII 83
- readChar 93
- readCharExt 94
- readChars 95
- rdiv 169
- readChar 93
- readCharExt 94
- readChars 95
- readComplexBin 96
- Reader 109
- ReaderPos 110

- readFloatArrayBin 96
- readFloatBin 96
- readInt16ArrayBin 96
- readInt16Bin 95
- readInt8Bin 95
- readIntArrayBin 96
- readIntBin 95
- readList 92
- readStructBin 97
- readUInt8Bin 96
- readWord 93
- real 55
- Receivers 92
- reDefP 395
- redraw 230
- reduce 383
- Reed 583
- refresh 229
- refreshP 229
- reHSB 291
- reHSBA 291
- Relative Turtle Motion 195
- Release Notes 4
- Release Notes for Version 4.64 24
- Release Notes for Version 4.65 18
- Release Notes for Version 4.66 18
- Release Notes for Version 4.67 17
- Release Notes for Version 4.672 16
- Release Notes for Version 4.68 15
- Release Notes for Version 4.682 15
- Release Notes for Version 4.683 14
- Release Notes for Version 4.684 13
- Release Notes for Version 4.685 7
- Release Notes for Version 4.686 6
- Release Notes for Version 4.687 5
- Release Notes for Version 4.688 5
- Release Notes for Version 4.689 5
- Release Notes for Version 4.69 4
- Remainder 138
- remDup 55
- remove 55
- removeItem 59
- removeProperty 316
- removeProperty 316
- repeatCount 371
- repeat 371
- repeatCount 371
- replace 47
- reRandom 165
- reRGB 289
- reRGBA 290
- reset 352
- resize 155
- reverse 44
- RGB 289
- RGBA 289
- right 197
- rightPortShift 125
- rightRoll 198
- readList 92
- rmul 168
- rnd 165
- Roll 214
- rotate 45
- rotatescene 233
- round 142
- rightRoll 198
- rSeq 154
- rSeqFloatArray 154
- rSeqFloatArray 154
- rsub 168
- right 197
- run 357
- runParse 86
- runResult 358
- readWord 93
- saturateAbove 156
- saturateBelow 157
- save 340
- saveImage 282
- savel 340
- savePicture 275
- savePicture 275
- savePictureText 275
- savePostScript 276
- savePostScript 276
- saveScreen 276
- saveScreenVector 277
- saveSize 279
- ScreenColor 253
- ScreenColor 253
- scroll 232
- scrollCalibrate 232
- scrollCalibrate 232
- Scrunch 217
- Sentence 36
- Selectors 49
- Sentence 36
- setCallsSplitter 227
- setCaseIgnored 334
- setCursor 120
- setCylinderPos 207
- setDepthFunc 243
- setEye 222
- setFloodColor 240
- setFloodColor 240
- setFogColor 273
- setFogDensity 272
- setFogMode 273
- setFogRange 272
- setHeading 208
- setHeading 208
- setIdentityMatrix 297
- setIdentityMatrix 297
- setItem 57
- setItems 59
- setLabelAlign 226
- setLabelFont 225
- setLabelSize 224
- setLabelWeight 225
- setLightAmbient 270
- setLightDiffuse 271
- setLightPos 270
- setLightSpecular 271
- setLogoComspec 115
- setLogoEditor 116
- setLogoHelpDir 116
- setLogoLibDir 117
- setLogoTempDir 117

Index

- setMatrix 295
- setMargins 121
- setMaterialAmbient 241
- setMaterialDiffuse 241
- setMaterialEmission 241
- setMaterialShininess 241
- setMaterialSpecular 241
- setMatrix 295
- setOrientation 209
- setPenColor 239
- setpen 243
- setPenColor 239
- setPenPattern 242
- setPenSize 242
- setPitch 208
- setPixel 286
- setPixelXY 287
- setPixelXYZ 287
- setPortBit 124
- setPos 204
- setPosXYZ 205
- setPrintPrecision 171
- setPenSize 242
- setReader 107
- setReaderPos 109
- setRoll 209
- setSaveSize 279
- setScreenColor 242
- setScreenColor 242
- setScreenRange 233
- setScrunch 228
- setShadowColor 303
- setSpherePos 206
- setStackNoisy 345
- setTextColor 121
- setTessWindingRule 260
- setTexPos 299
- setTextColor 121
- setTextFont 121
- setTextSelection 123
- setTextSize 122
- setTexXY 299
- setTurtleMatrix 295
- setTurtle 237
- setTurtleMatrix 295
- setUpdateGraph 230
- setVarsSplitter 227
- setWriter 108
- setWriterPos 110
- setX 201
- setXY 202
- setXYZ 202
- setY 201
- setZ 201
- ShadowColor 304
- Shadows 302
- Shell 97
- ShellSpawn 97
- show 90
- shownP 234
- showTurtle 219
- shrinkStacks 345
- shuffle 45
- Signum 143
- Sin 146
- singleBuffer 229
- SizeOf 82
- Slider 534
- SliderDestroy 535
- SliderEnable 537
- SliderOnScroll 537
- Sliders 534
- SliderSetRange 536
- SliderSetValue 536
- SliderValue 536
- Sound 564
- Sound programming 561
- Soundeffects 585
- Sounds 564
- SoundsFinished 565
- SoundsStop 565
- SpaceP 79
- Special Variables 394
- Sphere 267
- spinX 210
- spinY 210
- spinZ 210
- splitScreen 227
- Sqr 144
- Sqrt 144
- splitScreen 227
- showTurtle 219
- Standard Dialogs 413
- standout 85
- Startup 395
- StaticText 539
- StaticTextDestroy 540
- StaticTextLabel 540
- StaticTexts 539
- StaticTextSetBackgroundColor 541
- StaticTextSetColor 541
- StaticTextSetFontName 542
- StaticTextSetFontSize 541
- StaticTextSetFontStyle 542
- StaticTextSetFontWeight 543
- StaticTextSetLabel 540
- step 338
- stepall 339
- stop 354
- StringBuffer 35
- StringBufferToWord 35
- Strings 582
- Struct 42
- subStringP 69
- Sum 135
- SurfaceColumn 262
- SurfaceEnd 263
- Synth-Effects 584
- Synth-Lead 583
- Synth-Pad 584
- Table 41
- Tag 355
- Tan 148
- Template Based Iteration 376
- Terminal Access 119
- TessContour 260
- TessEnd 260
- TessStart 259
- test 366

- Text 309
- TextControl 544
- TextControlAppend 548
- TextControlCursor 548
- TextControlDestroy 546
- TextControlEnable 554
- TextControlInsertMode 549
- TextControlOnChange 554
- TextControlOnChar 552
- TextControlOnEnter 554
- TextControlOnKeyDown 553
- TextControlOnKeyUp 553
- TextControlOverwriteMode 550
- TextControls 544
- TextControlSetBackgroundColor 550
- TextControlSetColor 550
- TextControlSetCursor 549
- TextControlSetFontName 551
- TextControlSetFontSize 551
- TextControlSetFontStyle 552
- TextControlSetFontWeight 552
- TextControlSetInsertionPointEnd 548
- TextControlSetValue 547
- TextControlValue 547
- TextControlWrite 547
- TextMousePos 234
- TextMouseX 235
- TextMouseY 235
- TextScreen 226
- Texture 298
- Texturing 298
- Thing 314
- throw 367
- Time 128
- TimeFine 127
- TimeMilli 127
- TimerFreq 128
- TimeU 127
- TimeURes 127
- Timing 127
- TurtleMatrix 295
- to 306
- ToggleButton 556
- ToggleButtonDestroy 557
- ToggleButtonEnable 558
- ToggleButtonOnClick 557
- ToggleButtons 556
- ToggleButtonSetValue 558
- ToggleButtonValue 557
- Tokenization 29
- toList 43
- toListList 46
- Tone 562
- Tones 562
- TonesFinished 563
- TonesStop 563
- towards 215
- towardsXYZ 215
- trace 337
- traceall 338
- traced 322
- transfer 385
- Transmitters 89
- transposematrix 158
- transposematrix 158
- truncate 143
- truncate 143
- TextScreen 226
- Turtle 237
- Turtle and Window Control 218
- Turtle and Window Queries 234
- Turtle Motion Queries 212
- TurtleMatrix 295
- type 90
- TypeOf 82
- UInt8 142
- UInt8P 72
- unbury 336
- unburyall 336
- unburyname 337
- unperspective 222
- unstep 339
- until 373
- untrace 338
- upPitch 198
- updateCalls 231
- updateGraph 230
- updateVars 231
- updateVarsOnStep 231
- upperCase 85
- UpperP 79
- upPitch 198
- Variable Definition 312
- VideoEnd 266
- VideoFrame 265
- VideoStart 265
- wait 358
- waitMS 359
- waituS 359
- Wave Sound 566
- while 373
- Window 221
- Window Styles 396
- Word 34
- WordP 64
- WordUnderCursor 120
- Workspace Control 334
- Workspace Management 305
- Workspace Predicates 318
- Workspace Queries 321
- wrap 220
- Writer 109
- WriterPos 110
- xAdd 160
- xCopy 160
- xCor 212
- xDigitP 80
- xDiv 161
- xMod 161
- xMul 161
- xSub 160
- yCor 212
- zCor 213