NAME

yume − Easy and flexible menu system that lets you encapsulate shell commands for few-click or no-click mouse-over operation

SYNOPSIS

yume [[*commands***] [−bu] [−do] [−dl] [−ex] [−at** *geometry***] [−in** *file***] [−iv** *value***] [−la** *label***] [−ti** *title***] [−de** *valpair***] [−bd** *buttondata***] [−bw** *widthcodes***] [−Y[a|d|e|i|l|n|o|t|u|v|w|x]] [−Y<d>]]***

DESCRIPTION

yume creates and runs a menu based on the arguments it is given. The appearance of a menu may be changed by changing the order of arguments to **yume.** Description of **yume** is continued below, after a synoptic overview. Note, yume3 uses the Gtk+ graphics API, while yume2 uses xforms. Note, each **yume** option is specified by a separate three-character word like **−bu** or **−do**.

SYNOPSIS, continued

This section gives an overview of option sequences, in a form less terse than above. For further details and examples, please see the OPTIONS section below. Options for Gtk-based yume3 are described here. For the older (and no longer developed) xforms-based yume2, see documents distributed with yume2 source.

−at geometry
: Specifies menu window geometry in widexhigh+xo+yo or similar form

−bd buttondata
: Reserved for future option

−bw widthcodes
: Use letters of widthcodes to control button widths

−bu c1 c2 ...
: Creates row of buttons labeled c1, c2, ... that do commands c1, c2 ...

−bu −la l1 c1 **−la** l2 c2 ...
: Creates row of buttons labeled l1, l2, ... that do commands c1, c2 ...

−de valpair
: (where valpair is of form name:value) changes program attribute values. See name lists in **−de** section of OPTIONS. Changeable values include label and font colors, initial auto- vs click-mode button settings, label scaling-factor, gtk resource file, verbose outputs, and initial window properties such as Iconify, Stick, On-Top, and Decorate.

−do c1 c2 ...
: Creates do-groups (with editable command boxes) in two-line form, that initially do commands c1, c2 ...

−dl c1 c2 ...
: Creates do-groups in one-line form, that do commands c1, c2 ...

−ex Creates an exit button in current row of buttons

−in file Loads options from file

−iv initialValue
Set up a value-field (a button-plus-textbox unit) for a shell variable, initialized with initialValue. Typical **−iv** sequence:

−la buttonLabel **−la** shellVar **−iv** initialValue

−la txt Push a text string onto the label stack, to specify a text item such as a button label, a variable name, or a shell command. Yume3 applies Pango markup to button labels but not to variable names or shell commands. Button label markup is described in the OPTIONS section of this man page, under two headings: "Pango markup examples" in **−de** subsection, and "Button label text markup" in **−la** subsection.

The next two examples show **−la** as used to specify button labels, variable names, and commands for the **−iv** option.

−la buttonLabel **−la** shellVar **−iv** initialValue
Set up a value-field for shell variable shellVar, initialized with string initialValue, and no-command button labeled buttonLabel.

−la cmd **−la** buttonLabel **−la** shellVar **−iv** initialValue
Set up a value-field for shell variable shellVar, initialized with string initialValue, and button labeled buttonLabel, for command cmd.

−ti title Specifies window-name for the menu window

−Y[**a**|**d**|**e**|**i**|**l**|**n**|**o**|**t**|**u**|**v**|**w**|**x**]
Aliases for all the other options

−Y<d> Set debug options. Eg, if *S* is a set of yume options, the command **yume −Y7** *S* **−Y8** will display options as yume processes them and builds a menu form, and will display commands as they are executed. Note, except for accepting larger numbers, **−de ShowMode:**n is a synonym for **−Y**n.

DESCRIPTION, continued

yume creates and runs a menu based on the arguments it's given. Order of arguments is significant; for example,

yume -bu date ls -ex &

creates a menu window with three buttons, labeled date, ls, exit (left to right); while

yume -bu ls -ex date &

creates a menu window with three buttons, labeled ls, exit, date (left to right). For either of these examples, clicking a button labeled 'date' runs the date command and displays the result in the terminal window where yume started; clicking an ls button runs an ls command for the directory current when yume started; and clicking an exit button closes the menu window.

Created menus may consist of a mix of buttons, "do-items", and "value-fields", created via **−bu**, **−ex**, **−do**, **−dl**, and **−iv** options.

A do-item is a unit with two buttons (labeled, by default, "EE" and "do:") and a command-editing box. If you click the do: button, the command list in the editing box will be performed. If you change the text in the editing box, the new command will be performed the next time you do: it. A value-field is a unit with a

command, a shell-variable name, a label, and a text-editing box. The text in the box can be used elsewhere in the menu by shell-variable substitution, as described below for the **−iv** option.

For example,

yume -do ’ls -al’ ’cal -3’ &

creates a yume menu window with two do: items. Clicking the do: button of the top item runs the command "ls -al". Clicking the other do: button runs the command "cal -3". Clicking the EE button of a do: item will cause the label text of its EE button to turn red and the do: button background to turn violet-blue to indicate that "entry execution", or "auto-mode", or "rollover mode", call it what you will, has been turned on. (The colors mentioned are defaults; see next paragraph.) When you move the mouse onto an EE-enabled do: button, its command list will be run immediately, with no click needed. (If you move the mouse cursor onto such a button and click it, the command list will get executed twice -- once for the mouse entry, once for the click.)

The violet-blue background mentioned above is used for all auto-mode buttons if you are using the supplied yume-gtkrc file and haven’t changed default settings. The **−de** section below gives examples of changing label colors and names, via Pango markup. The file examples/test-y-5 is a concrete example of label and color changes.

Command-editing text boxes for do: items may contain multiple commands, pipes, etc, punctuated by semicolons or other shell conventions. For further explanation, see "[commands]" in the next section.

Throughout this document, the term "parameter-required switch" refers to any yume option-switch that requires a parameter string. Parameter-required switches include **−at**, **−bd**, **−bw**, **−de**, **−in**, **−iv**, **−la**, or **−ti**. If you don’t provide a parameter value word after such a switch, yume will treat whatever is next as a parameter value-string anyway, or will quit with an error. Example: **yume -la D -bu date** will (correctly) make a button labeled D, with command date, while **yume -la -bu date** will make a do-group with command date. Example: **yume -la** will display a missing-argument error.

OPTIONS

−at geometry

Specify menu-window location and size, with a geometry string in the usual X Window System form, with general form

−at WIDTHxHEIGHT+XOFF+YOFF

where each of the capitalized terms is a number of pixels; and XOFF, YOFF can be positive or negative. Some examples follow.

yume −at 99x99-44-33 -ex
yume −ex −at 99x99-44-33

Either of those produces a 1-button box, labeled exit, near the lower right corner of display.

If the parameters to yume do not include an **−at** option, the menu-window size will depend on sizes of labels and commands, and it will be placed by window-manager rules. In any case, the user can move and resize the menu-window, after it appears, using ordinary window-manager controls. Note, yume treats "-geometry" as a synonym for "**−at**". If a -geometry switch appears, yume converts it to **−at**.

Yume does not validate geometry specifications. It tests whether gtk_window_parse_geometry() says the spec is valid; if not, it exits with error YERR_GEOM. However, a specification can be bad in spite of being valid. A user may give a valid specification that places the menu apparently off-screen and perhaps inaccessible. Yume tests for this condition and may print a warning message if it occurs.

When multiple screens are in use, you might purposely set coordinates exceeding screen size, to put a menu on a particular window. You can suppress, allow, or force the coordinates message via

−de XYshow as explained below.

During startup, yume parses geometry, shows the menu, sets a 1/3-second timer, and enters its main event loop. The window manager places the menu and its buttons become active. When the timer expires, yume tests if any menu corner coordinate is negative or exceeds screen size, and may print a message if enabled. The message is like "yume3 menu for pid %n appears at %x %y" (where %n, %x, %y are filled in with pid# and menu corner coordinates).

You can control the coordinates message via **−de XYshow**. By default, XYshow is 1, to show menu corner coordinates only when a coordinate is negative or exceeds screen size. To suppress message, use **−de XYshow:0**. To force message, use **−de XYshow:2**. XYshow can be abbreviated; eg: **−de xys:2**

−bd buttondata
This option is not implemented in current release of yume3. It is reserved for future use. It will specify button options applicable to subsequent buttons.

−bu Select Button Mode and start a new row.

Commands and switches following a **−bu** (up until another **−bu** or a **−do** appears) will form one row of the menu. For each command after a **−bu**, yume will store the command while processing parameters; will compute button sizes while preparing the menu; and will perform the command when the menu is used.

The next three paragraphs expand upon the **−bu** explanation just given, and perhaps can be skipped on first reading.

While parameters are being processed: If the label stack is empty, push the command onto it for use as a button label. Get a label from the label stack. Make an items-list entry with the label and command and data from earlier **−de** and **−bd** parameters. If **−Y2** or similar has turned on ShoParProc, print out the command and some related data.

While menu is being prepared: Compute size of button, based on label text length, or on a specified **−bw** width code. If **−Y4** or similar has turned on ShoAddObj, print out size of button for command.

While menu is being used: If **−Y8** or similar has turned on ShoAssert, print button data whenever it’s clicked. If **−Y1** or similar has turned on ShoWExp and the button action uses wordexp, report results of each use. Execute the command as described in "[commands]".

−bw widthcodes
Set button width ratios, forward from where the **−bw** option appears, until end of parameters or another **−bw** occurs. The k’th letter of widthcodes specifies number of unit widths for the k’th button in a row, left to right. Unit width (in pixels) for each element of a row is the menu’s whole pixel width divided by the sum of relative widths in a row.

Width codes are letters, a...z, that stand for 1...26 unit widths. For example: "**−bw** cd" says that in a row of two buttons, the left button size will be given 3 unit widths, and the right, 4. If a row has more buttons than the widthcodes has letters, excess buttons are treated not via letters but via a calculation. For example, if some rows after "**−bw** cd" have more than two buttons, the widths of buttons after the first two will be calculated per next paragraph. Note, one letter applies to one display element; so **−iv** groups use up two letters, and **−dl** or **−do** groups use up three letters each.

Calculations: If no **−bw** setting occurs among yume parameters, or if a button number exceeds the

−bw widthcodes string length, yume will compute button widths via sizes of the text strings that label the buttons. By default, a button is given one unit width for each 5 characters in its label. (That is, buQuanta defaults to 5; see examples mentioned in next paragraph.)

Examples: In the yume examples directory, see button-widths, bw-and-quanta, example-starter3, and example-starter3.txt. The first of these creates three menus with "**−bw** dc", "**−de**buq:6", and "**−bw** bbbb" to illustrate various **−bw** and **−de** buq usages.

Note, **−bw** processing does not check whether widthcodes are lower case letters; it merely uses the low 5 bits of each character it's given. For ASCII characters, **−bw** 43 has the same effect as **−bw** dc. See e.g. examples/calc-wrap2, which egregiously creates eight menus, with **−bw** settings identified in menu-window title bars.

[commands]

Parameters that are not option switches (and not part of a parameter-required switch, **−at**, **−bd**, **−bw**, **−de**, **−in**, **−iv**, **−la**, or **−ti**) are stored as commands to be performed when the user desires. Yume has several switches that store commands for user-controlled execution: **−bu**, **−dl**, **−do**, and **−iv**.

- In **−bu** mode, the command is represented in the displayed menu by a labeled button; the command will be performed when the user clicks the button.
- In **−dl** or **−do** mode, the command is represented in the menu by a do: item, as mentioned above. It will be performed when the user clicks the Do: button, or (if EE-enabled is on) when the user moves the mouse onto the Do: button.
- The command for an **−iv** option will be performed when the user clicks the labeled button next to a value-edit-box in the displayed menu. By default, the command associated with an **−iv** button is "true", effectively a no-operation command. To associate a real command with an **−iv** button, use the
  −la shellCommand −la buttonLabel −la shellVar −iv initialVal
  format described in the **−la** section.

In all modes, command lists are executed the same way. Usually this way is analogous to
execl("/bin/sh", -c , commandlist , 0)
in a forked copy of yume. See 'COMMENTS: Commands' section for other ways commands can be performed, which include %% or %# command echoing for shell-command testing or debugging; %+ or %- non-forked; and %- or %: non-SHELL.

Basically, command lists that don't begin with % are sent to a shell, where they are treated much like ordinary shell commands. Output from command execution appears just as output from yume would appear; for example, on the terminal window where yume was started. If yume output is redirected, so is command output, unless the executed command list has redirection of its own. For example,
yume ./abc ./def "./ghi > u" > t &
directs output from programs abc and def to file t, and output from program ghi to file u. See examples/abc-def-ghi for a slightly-extended version of this example; when you click the abc or def buttons, output is fed to t, as you can see after the command finishes, or while it is running by clicking the 'cat t to stderr' button. As stderr is not redirected, its output will appear on the terminal where the example is started.

−de valpair

Decode valpair to override default attribute values. For example, the sequence
−de nogtkrc −de gtkrc:mygtkrcfile
tells yume to use gtkrc file **mygtkrcfile** and not use **$HOME/.yume3-gtkrc.**

Each valpair must be a single argument to yume, so whitespace and special characters must be quoted or escaped. Within a valpair, any non-alphanumeric character except underline can be used

to separate name and value.

Kinds of attributes and their associated names and purposes include (1) label and font colors, (2) initial on/off-settings for auto- vs click-mode button settings, (3) button locations, (4) label scaling-factor, (4b) inclusion-limit, (5a) Gtk-parameters, (5) gtk resource file, (6) verbose outputs, and (7) initial window properties such as Iconify, Stick, On-Top, and Decorate. In numbered subsections (2)-(7) below, attributes are categorized, named, tagged, and described in a format like following:

(subsection number) category: attribute-names ˜[data-type]˜ description.

Data types include flags, numbers, and strings.

(1) label-colors:

Label-color string names are listed in column 1 of the following table. The strings may contain Pango markup and are used as formats for labels of buttons as noted in column 3.

1. Name	2. Value	3. Purpose
y3db	"do:"	do: button, in button mode
y3da	"Do:"	do: button, in roll mode
y3eb	"EE"	EE button, button mode
y3ea	"EE"	EE button, roll mode
y3fb	"%s"	fixed-command button
y3fa	"%s"	fixed-command button, r.m.
y3gb	"%s"	general-command button
y3ga	"%s"	general-command button, r.m.
y3hb	"%s"	hidden-command button
y3ha	"%s"	hidden-command button, r.m.
y3ib	"%s"	iv group’s button
y3ia	"%s"	iv group’s button, r.m.
y3xb	"exit"	Exit button, button mode
y3xa	"exit"	Exit button, roll mode
y3td	""	do: group’s text field
y3tv	""	iv group’s text field

Column 2 shows initial format values for the several kinds of buttons. User-supplied labels replace %s occurrences. This same set of names is used as style names in ˜/.yume-gtkrc, which as supplied has a single style for each button-mode/roll-mode pair except y3db/y3da.

The effect of a **−de** label-color setting carries forward from the point of definition. If you want a set of colors to apply to all yume menus, change settings via the ˜/.yume-gtkrc file. If you want to change label font sizes for all or many yume menus, put appropriate **−de** label-color settings in file ˜/.yume-rc or in an option file included via **−in** option. Also see example test-y-5 and its yume3-gtkrc-5C resource file.

Pango markup examples

Example: Create a menu with first button labeled "date" in ordinary text, and other button "exit" in big bold text:

yume −bu date -de ’y3ex:<big>%s</big>’ -ex

Example: Create a menu with first button labeled "(Show date)", and other button "Bye", both in usual text:

yume -de ’y3bg:(Show %s)’ -bu date -la Bye -ex

Example: Create a menu where do: button for date command is labeled "(((Do)))" whenever roll-over mode is on. Note, yume starts out in **−do** mode, not **−bu** mode; **−ex** automatically turns on **−bu** mode for itself, so the first date command in this example is in a do: group while the second one is on a **−bu** button.

yume -de 'y3dr:(((Do)))' date -ex date

Example: Create a menu with two do: groups and a button labeled "exit". The second EE button's label text is green and it is labeled Hey rather than EE.

yume 'ls -al' -de 'y3ee:Hey
' 'cal -3' -ex

On many Linux systems, color constant values for use in Pango markup are shown in file /usr/lib/X11/rgb.txt. Hexadecimal forms like #RRGGBB can also be used. For Pango markup text-size and text-style constant values, see <http://www.pygtk.org/docs/pygtk/pango-markup-language.html>.

(2) initial on/off-settings:

ba, bb, da, db, ea, eb, ia, ib, xa, xb ˜[flags]˜ If fname is a flag name, the form **−de** fname turns on its flag. For each pair of these flags, exactly one is on at a time. (One of **ba** and **bb**, one of **da** and **db**, and so forth.)

These ten flags control the initial auto-mode vs click-mode state of yume's five kinds of buttons (-bu, do:, ee, -iv, -ex). For example, **−bu** buttons created after **−de ba** are created in auto-mode and operate when the mouse enters the button.

In the menu created by the following example, the current date prints whenever the mouse crosses the "date" button or whenever that button is clicked, and the menu exits whenever the mouse enters the button labeled Exit. The button labeled ls only operates when clicked.

yume -de xa -bu ls -de ba date -ex

If you use the supplied yume-gtkrc file as ˜/.yume-gtkrc and haven't changed default settings within that file, then buttons with auto-mode enabled have a violet-blue normal background, pink prelight background, and white lettering. Click-mode buttons by default are displayed with sky-blue backgrounds and black lettering.

A future version of yume may provide right-click dropdown menus to change auto- vs click-mode states. In current version, except for **−do** and **−dl** do: buttons controlled by EE buttons, auto-mode vs click-mode states are static during yume's execution.

(3) button-locations:

attach ˜[numbers]˜ [Future]

(4) label scaling-factor:

buQuanta ˜[number]˜ Set the number of label characters per "unit width". Default value is 5. Consider the following four examples.

1. **yume -bu 12345 12345678 123456789ABC**
2. **yume -bw aaa -bu 12345 12345678 123456789ABC**
3. **yume -de buq:4 -bu 12345 12345678 123456789ABC**
4. **yume -bw aaa -de buq:4 -bu 12345 12345678 123456789ABC**

In example 1, the menu has a 1-unit-wide button, a 2-unit-wide button, and a 3-unit-wide button, because ceiling((letter count)/(quantum size)) is 1, 2, 3 for ratios 5/5, 8/5, and 12/5.

In examples 2 and 4, the menu has three 1-unit-wide buttons, because **-bw aaa** says to use width 1 for buttons 1, 2, 3. (Note that **−bw** options override **buQuanta** calculations.)

In example 3, the menu has two 2-unit-wide buttons and a 3-unit-wide button, because ceiling((letter count)/(quantum size)) is 2, 2, 3 for ratios 5/4, 8/4, and 12/4. If this menu appears 350 pixels wide on-screen, then this row's unit width is 50 pixels.

Unit widths are calculated on a row-by-row basis. For any given menu width, a row with many buttons will have fewer pixels per unit width than one with few buttons.

(4b) inclusion-limit:

FStackLim ˜[number]˜ This number controls the number of levels of file inclusion that yume allows. It defaults to 400. Yume reports an error if inclusion depth exceeds the limit. Such an error will occur if an included file includes itself or an ancestor.

(5a) Gtk-parameters:

gtk ˜[string]˜ The Gtk initialization routine gtk_init() can accept command line parameters such as --gtk-module, --g-fatal-warnings, and a few others. **−de** gtk:string gives parameters in string to gtk_init(), after shell-globbing via wordexp(). For additional information about gtk options, see the man page for gtk-options, and see <http://library.gnome.org/devel/gtk/unstable/gtk-running.html>.

(5) gtk-resource-file:

gtkrc ˜[string]˜ Use **-de gtkrc:***filename* to have yume use *filename* as a GTK resource file.

nogtkrc ˜[flag]˜ Use **-de nogtkrc** to tell yume to not use **˜/.yume-gtkrc** as a GTK resource file. For example, the menus

1. **yume -ex**
2. **yume -ex -de nogtkrc**
3. **yume -ex -de gtkrc:file13**
4. **yume -ex -de gtkrc:file13 -de nogtkrc**

use Gtk resource files respectively as follows: (1) only ˜/.yume-gtkrc, (2) none, (3) both ˜/.yume-gtkrc and ./file13, (4) only ./file13. If you don't want ˜/.yume-gtkrc to be used for any of your yume menus, delete it. If you have some resource file that you want used for all of your yume menus, copy it over ˜/.yume-gtkrc.

(6) verbose-outputs:

ShowMode, XYshow ˜[number]˜ ShowMode:n is a synonym for **−Y**n but with larger n allowed. XYshow:n controls menu coordinates test and display as described earlier in **−at** section.

(7) initial-window-properties:

wIconic, wOnTop, wStick, wUndec ˜[flags]˜ By default, a yume menu window is created with Decorated turned on, and Iconic, Keep-Above, and Stick properties turned off. The defaults can be overridden as follows:

−de wIcon will cause menu to be created iconified, rather than as an open window.

−de wOnTop will designate menu to stay on top of other windows.

−de wStick will cause menu to be visible on all workspaces.

−de wUndec will cause menu to be created without a window-manager frame, as one might want for small menus where decorations (such as window manager ops bar) may be larger than the menu itself. Note, if you subsequently need to move or resize an undecorated menu, use a window-manager menu. Eg, on metacity press alt-F7 and arrow keys; see <http://library.gnome.org/users/gnome-access-guide/stable/keynav-14.html.en>.

Valpair forms that can appear as parameters for the -de option include name; name:numbervalue; name:stringvalue; and name:multiple#values. In each form, names can be abbreviated to any unambiguous shorter form, and the case of letters in names is ignored. For example, the name buQuanta can be abbreviated as buQ, buq, buqu, buQua, etc. Some examples:

-de FStackLim:10
-de gtkrc:mygtkrcfile
-de buq:7
-de buQuanta:7
-de bOn
-de attach:3,5,4,7

Concise lists of constants known to yume appear in yume3-trec.h, a file that is created by program yume3-init when yume is built.

In yume2, recognized constant values for setting BuDef were BuFix, BuHid, and BuVar, which are not recognized in yume3.

−dl Turn on **−dl** mode for following commands. That is, commands among the following parameters will be put into **−dl** items, until a **−bu**, **−do**, **−ex**, or **−iv** appears.

Briefly, **−dl** items are treated much like **−do** items, except are embedded within a button row, instead of taking two rows as **−do** items do.

That is, like each **−do** item, each **−dl** item has an EE button, a do: button, and a command editing box. Each **−do** item is presented on two rows of a menu; the upper row has an EE button and a do: button, while the lower row has a command editing box. By contrast, the elements of a **−dl** item are in line within a button row. A button row can contain any number of **−dl** items. In the sequence of commands after a **−dl**, other **−dl** switches would be redundant. For example, these are equivalent:

... **−dl** c1 **−dl** c2 ...
... **−dl** c1 c2 ...

After **−do**, **−dl** will start a new row in the menu; after **−bu**, it won't. For example,

yume -do -dl c1 -do -dl c2 -do c3 -bu -ex -dl c4

creates a menu with 5 rows; the first two are **−dl** items for commands c1 and c2, the third and fourth rows are a **−do** item for command c3, and the fifth row contains an exit button and a **−dl** item for c4.

−do Start a new row in the menu and turn on do: mode for following commands. That is, commands among the parameters after **−do** will be put into do: items, until a **−bu**, **−dl**, **−ex**, or **−iv** appears. By default, yume is in do: mode when it begins parameter parsing.

The general idea of the **−do** switch is that commands and switches following it, up until a **−bu**, **−dl**, **−ex**, or **−iv** appears, will form do: entries in the menu. A do: entry occupies two rows of the menu layout; the upper row has an EE button and a do: button, while the lower row has a command editing box. In the sequence of commands after a **−do** before a **−bu**, **−dl**, **−ex**, or **−iv** switch, other **−do** switches would be redundant. Thus, all three of the following are equivalent, and produce three do: items for commands c1, c2, c3, in a menu with 6 rows:

yume c1 c2 c3
yume -do c1 c2 c3
yume -do c1 -do c2 -do c3

When you click do: (or move the mouse across an auto-mode Do: button), the text in the editing box will be processed as a command.

−ex Creates an Exit button in button mode, labeled from top of label stack.

In more detail, **−ex** acts as follows: (1) If **−do** mode is on, then turn on **−bu** mode and start a row of buttons. (2) If label stack is empty, push "exit". (3) In current row of buttons, pop label from stack and create a labeled button, with exit-from-yume action.

−in file Process yume options from specified file. The file contents should look like an ordinary yume parameter list, except that line continuations are optional in the file, and comments are allowed. As usual, if a line ends with backslash, then backslash and newline characters are stripped; in other lines, a blank replaces newline. Lines that begin with a number mark (#) are treated as comments and ignored. Expansion of unquoted variables in a file occurs when the file is processed, in yume's runtime environment.

After all lines are read in and concatenated, the text is shell-expanded with wordexp(3) and the result is processed as a yume parameter list. File-includes may be nested to depth given by parameter FStackLim, with default value 400 (ie: **−de FStackLim:400**).

If a user-options-file $HOME/.yume-rc exists, it is read at the outset of parameter processing. It is treated like any other included file, except for having pride of place and being loaded automatically.

The button label stack can be set up with **−la** parameters in an include file and used by buttons generated after that file. But an include file cannot end with **−at**, **−bd**, **−bw**, **−de**, **−in**, **−iv**, **−la**, or **−ti**. That is, if a parameter is required for a switch in an include file, both the switch and its parameter must be in that same file.

If your file includes shell-type expressions for file-globbing, command substitution, or variable substitution, the following applies: By its specification, wordexp(3) is allowed to fail to recognize some shell special parameter expressions, such as $0, $#, and $*. See wordexp(3) and test any cases that are relevant on your own system. Also see examples/wordexp-vals which uses "**−in** wordexp-in" to test the special parameter cases just mentioned.

During processing of an included file, the value of $0 will be not the script name, but instead will be the path to yume3, because the substitution will occur while yume is running, not beforehand. Usually one can work around such issues in a script by setting intermediate variables.

Use **−Y1**, **−Y2**, or other **−Y**<d> options to see some of the processed results of including a file. See the "**−Y**<d>" section for details.

−la buttonLabel **−la** shellVar **−iv** initialValue
Set up a value-field for shell variable shellVar, initialized with initialValue, and no-operation button labeled buttonLabel. [The command 'true' is used as a no-op, if no other command is specified. That is, if the label stack has fewer than three entries when the -iv option is processed, a 'true' command is attached to the command button.]

−la cmd **−la** buttonLabel **−la** shellVar **−iv** initialValue
Set up a value-field for shell variable shellVar, initialized with initialValue, and a button labeled buttonLabel that performs command cmd.

−iv initialValue
Set up a value-field, initialized with initialValue. A value-field has an optional command, a name, a label, and an editing box. (The command, name, and label are specified with **−la** entries as noted just above.) The name is a shell-variable name that can be used elsewhere in the menu to refer to the field's value. The label is text that is displayed on the label button. If you click the button, the optional command is executed. The editing box is a small text-editing window pane. As you change text in the editing box, the value of the variable changes in yume's environment as well.

Example: To construct a value-field entry for shell-variable HIPIX, with label 'Screen Height, Pixels', and initial value of 1024, use:

-la 'Screen Height, Pixels' -la HIPIX -iv 1024.

To reference the contents of the value editing field elsewhere in the menu, quote $HIPIX for substitution at command execution time. Eg: Use

-do 'echo "Screen height is $HIPIX pixels"'

(with outer single quotes and inner double quotes) and do not use

-do "echo 'Screen height is $HIPIX pixels'"

(with outer double quotes and inner single quotes) which substitutes for $HIPIX from startup environment rather than runtime environment.

If you click or operate the label button next to the edit-box, the optional command will be executed, in the usual yume manner. The command, if given, is specified by a previous **−la** parameter. For example, the one-line menu

yume -la 'echo Pay \$$D to $B' -la Amt -la D -iv 20
-de ia -la Who -la B -iv Sam

will have buttons labeled Amt and Who, and text fields initialized to 20 and Sam for shell variables D and B. If you change the name in the Who field to Sammi and then move the mouse across the Who button, that button will operate because **-de ia** put it into auto-mode. If you change the amount in the Amt field to 207 and click the Amt button, a message like "Pay $207 to Sammi" will print.

If the options for a menu include several value-fields with the same name, the last one specified gives the shell variable its initial value. That is, yume sets variables as it processes **−iv** parameters; if a variable belongs to several value-fields, it will be assigned several times during yume initialization.

When you change the text in the edit-box, yume changes the shell variable that belongs to the item, via setenv(name,value,1). In yume2 only, if the value-field setup specifies an empty name, yume2 may use putenv(value), intended to work with text in form "name=value" in the edit-box. Yume3 does not support that form.

Note, **−iv** label order is different between yume2 and releases > 16 of yume3. yume3 accepts as the shell variable the latest label defined before **−iv**.

One more example:

-la 'svn help $H' -la 'Help Topic:' -la H -iv propget

in examples/svn-buttons uses the first **-la** to specify a command **'svn help $H'** enclosed in single-quotes to avoid early shell substitution for H. It uses a second **-la** to specify the label "Help Topic:" for the button next to the edit-box, and a third **-la** to specify shell variable H. The edit-box is initialized to **propget**. If you click the "Help Topic:" button when the menu appears, the command "svn help propget" will be performed, causing svn's help-text about propget to appear on the console where examples/svn-buttons was started.

−la txt Push text string txt onto the label stack. Primarily, **−la** is used to specify button labels; it is also used to specify shell-variable-name and shell-command text for **−iv**, as mentioned above. Note, to display literal character '<' in a label, write '<'. See "Button label text markup" below.

To create a line of buttons labeled b1, b2, b3, that execute commands c1, c2, c3, use a sequence like the following:

-bu -la b1 c1 -la b2 c2 -la b3 c3

To create buttons as above but stacked in a column, use a sequence as follows. (Note additional **−bu** switches.)

-bu -la b1 c1 -bu -la b2 c2 -bu -la b3 c3

Button label text markup -- Yume applies Pango markup to button label text. You can use any UTF8 encoding and Pango Text Attribute Markup Language in button labels per
<http://library.gnome.org/devel/pango/stable/PangoMarkupFormat.html>.

Yume does not apply markup to other label text, such as shell-variable names and commands for **−iv**.

When characters in set { &, <, >, ", ’ } are to appear verbatim in a label, use escape sequences from set { &, <, >, ", ' } for correct display. Example: option sequence
... −la ’mycmd <clip>’ mycmd ...
results in a blank label for the mycmd button and run-time error message
Gtk-WARNING **: Failed to set text from markup
due to error parsing markup: Unknown tag ’clip’
while by contrast,
... −la ’mycmd <clip>’ mycmd ...
gives a button labeled ’mycmd <clip>’ for the mycmd button.

−la Usage -- As noted above, **-la txt** pushes a string value onto the label stack, from which values are popped for use as labels, names, commands, and values. When such an item is needed, it’s obtained from the stack. If the stack is empty when an item is needed, defaults are used. Some examples:
yume -la ta-ta -la al date -bu ’ls -al’ -ex
creates a menu with a do: item for command "date"; a button labeled "al", for command "ls -al"; and a button labeled "ta-ta", for exit from yume.
yume -la al date -bu -ex ’ls -al’ -ex
creates a menu with a do: item for command "date"; a button labeled "al", for exit from yume; a button labeled "ls -al", for command "ls -al"; and a button labeled "Exit", for exit from yume.

If the stack is empty,
yume -la AA -iv report.t
creates a value-field for shell variable AA with initial value "report.t", with a no-command button labeled "AA", while
yume -la ’date -r $FF’ -la ’File to Date:’
-la FF -iv report.u
creates a value-field for shell variable FF with initial value "report.u", and command ’date -r $FF’ for the button labeled "File to Date:".

Label-stack depth is limited by available memory and is not program limited. The label stack is freed up after parameter processing.

−ti title Set menu window title to specified title. If yume is not given a **−ti** option, it sets the menu window title to "yume3 nnnnn" where nnnnn is the process id number (pid#) assigned to yume3.

-Y[adefghilnotuvwx]
Each of the two-letter options **−Ya** ... **−Yx** indicated by **-Y**[aefghinotuvwx] is synonymous with another yume option with the same second letter. For example, -Ya == -la, -Ye == -de, ... -Yx == -ex. When it starts, yume converts all of the 2-letter option codes that it recognizes to -Y. form. Note, this behavior is vestigial from yume2 and *will be removed* in a future release of yume3. Yume also converts -geometry, if present, to -Yt.

−Y<d> For this option, d is a digit 0...9. The lower 4 bits of d control 4 parameter and command debugging functions; a function is turned on when its bit is set, else is turned off.

If you just want to list parameters as they are processed, put -Y2 at the beginning of the parameter list (or at the point where you want parameter listing to begin). If you just want to see commands as buttons are clicked, put **−Y8** near the end of the parameter list.

Briefly:

−Y0 Turn off parameter and command debugging.

−Y1 Use this to list the wordexp result array for %- or %:-prefaced commands, or for **−in** expansions.

−Y2 Use this to list parameters as they are processed, for example if you are having problems with correct levels of quoting.

−Y3 Like **−Y1** plus **−Y2**

−Y4 Use this to list the objects of the menu form, for example if you want to see actual sizes of menu elements.

−Y5 Like **−Y1** plus **−Y4**

−Y6 Like **−Y2** plus **−Y4**

−Y7 Like **−Y1** plus **−Y2** plus **−Y4**

−Y8 Use this to print the executed command each time a button is clicked or has a mouse event. (Or use %% at beginning of command; see "COMMENTS: Commands")

−Y9 Like **−Y1** plus **−Y8**

In more detail:

−Y1, ShoWExp
List the result array from wordexp expansion for commands prefaced with %: or %-, or for **−in** expansions.

−Y2, ShoParProc
List parameters as they are processed and show phase information. For example,
yume −Y2 -bw ab -ex -la info xwininfo
prints a heading and lines like the following as it processes parameters and completes yume startup phases:

```
      Item#  Par#  Par     Par
ParProc   0.    2   <-Yw>   <ab>
ParProc   0.    4   <-Yx>   <-Ya>
ParProc   2.    5   <-Ya>   <info>
ParProc   2.    7  <xwininfo> <(null)>
Phase I (process parameters) is done
Phase II (make display) is done
```

The first ParProc line represents the "**−bw** ab" sequence; **−bw** has been converted to -Yw. In other lines, -Yx means -ex and -Ya means -la. The two numbers in each ParProc line are item number and parameter number. As -bw and -la each require a parameter, the parameter number increases from 2 to 4 and from 5 to 7 when they are processed. -ex generated two items (see next example). Note, -Y2 turns on Y2 parameter-processing printouts; -Y7 would turn on all of Y1, Y2, and Y4 at the same time. By contrast, "-Y1 -Y2 -Y4" would turn on -Y1, then turn it off and turn on -Y2, then turn that off and turn on -Y4.

−Y4, ShoAddObj
List objects as they are added to form. For example,
yume −Y4 -bw ab -ex -la info xwininfo
prints a heading and three AddObj lines due to -Y4. The first two AddObj lines show the row-beginning RoList item and an Exitor item. The third of these,

```
AddObj 2. BuGen ( 39 0 78 28) <info> <xwininfo>
```

shows that item 2 of the form is a general button with (x=39, y=0, w=78, h=28), label="info", value="xwininfo".

-Y8, ShoAssert
List objects when they are clicked on or have a mouse event. For example,

yume -Y4 -bw ab -ex -la info xwininfo -Y8

prints the same 3 AddObj lines as the previous example, and prints a line like the following when the "info" button is clicked:

AssertET 2. BuGen 39 0 78 30 <info> <xwininfo> L235

"L235" indicates that assertET() was called from line 235 of yume2.c

Note that when a do-group button is clicked, two AssertET lines will print, one for the button and one for the edit-box. Also note that when Y8 is on, moving the mouse over a do: button will generate window entry reports labeled <do:>.

EXAMPLES

See example scripts in the yume2/examples directory. The examples listed below are described in yume-examples(1). The command yume-examples starts a small menu to display a list of yume examples. On the list of examples, "Code" and "Note" buttons appear next to the names of some examples. If you click a Code button, an xterm will appear with source code of the example. If you click a Note button, an xterm will appear with a brief description of the example. Pressing q in a Code or Note xterm will close it. The *.txt files in yume2/examples contain Note text.

lac - Latest-C script
Make menu to edit / compile / make / run latest C program

lala - Latest-Latex script
Make menu to edit / latex / pdf / view latest .tex file

yume-examples - A menu to start some yume examples
Make menu to start menu to start example scripts, such as lac, lacc, lacx, lala, find-ls-count, play-sound-delay, url-and-misc, url-clip ...

COMMENTS: Regarding 'Commands'

As noted above, most command lists are executed in a shell, like shell commands. This is done via a C-statement like "execl("/bin/sh", "-c", "commandlist", 0), which starts a shell via a forked copy of yume and passes it a command list. The first parameter (here, "/bin/sh") is the value of environment variable SHELL (except when the command list begins with a percent sign, as described later).

Use of $SHELL has several ramifications:

- SHELL should be defined in yume's environment. If it isn't, yume probably won't work.
- SHELL probably will be defined by your shell, such that menu commands will act much the same way when yume runs them as when you type them at a shell prompt.
- If your shell doesn't support or allow -c, set SHELL to be the path to a shell that does support -c. Many UNIX shells, such as ash, bash, csh, ksh, sh, tcsh, and zsh, use -c to specify a command to a shell.
- If you want to write a filter for menu commands to pass through, set SHELL to the path to the filter, and make the filter accept -c for its first parameter, and a raw command list for its second parameter.

A command list that starts with a percent sign (%) is handled differently than described above; see next section for details.

COMMENTS: Meta-mode 'Commands'

A command list that starts with a percent sign (%) is processed in "meta-mode", which allows one to control how commands are invoked, and allows echoing commands for debugging. Meta-mode is explained below. For an example, please see examples/date-of-file and examples/date-of-file.txt.

In this section, "Command" stands for the shell command sequence of a **yume** menu button. That is, "Command" is the action to be performed when an associated button is clicked or activated. If Command begins begins as shown below left, it will act as shown below right.

%%	Print command on stdout before it is executed
%#	Print command on stdout, don't execute.
%+	Execute command without forking. (Non-fork)
%-	Execute command as-is, without forking. (Non-fork, non-SHELL)
%:	Execute command as-is. (Non-SHELL)

Fork, SHELL : The usual command-processing method that yume uses for a non-meta Command is fork() to start a process to perform the command, followed by execl() to start a shell that processes it. That is, for a Command not starting with %, Command is expanded via POSIX routine wordexp() per usual shell globbing rules, after which yume forks and uses execl($SHELL,"-c",...) to start a shell to process the expanded parameter list. ($SHELL represents a value from yume's environment, like "/bin/sh", that indicates what shell to use.)

Non-fork, %+ and %- : When a meta-mode Command is executed without forking, in essence yume exits before the Command is performed. This allows one to write menus with "popup"-like behavior. For example:

- Menu A can have buttons that start menus B, C, ..., within which Commands all start with %+ (non-fork) or %- (non-fork, non-SHELL). Clicking a menu-A button starts (or pops-up) a secondary menu; when a menu-B, menu-C, ... button is clicked, that menu goes away and the button's associated command gets executed.
- %+ facilitates buttons for editing a menu script file and restarting it, via formulas like:
  -bu -la 'Edit menu' "gedit $0" -la 'Reload menu' "%+$0"

Non-SHELL, %- and %: : When a meta-mode Command is executed non-SHELL, it is expanded by wordexp as usual, but the first word after %- or %: should be a program name. yume will present the balance of the Command list to that program, as its arguments. Executing Command non-SHELL reduces the number of exec() levels and number of processes started by yume. The Command may run a few microseconds faster and take a few less bytes of memory vs. starting via shell.

DIAGNOSTICS, WARNINGS, CAVEATS

This section includes:

- yume's error messages and error exits;
- how to deal with errors in shell commands in menus;
- what environment is available to shell commands;
- yume interaction with
- visibility of parameter lists;
- piping a parameter list into yume3;
- safety and SUID and shell history lists.

yume's error messages and error exits:

Messages as listed below may appear on stderr if any of the following errors occur. This section

shows symbolic names for the numbers that will appear in actual error messages. In revision 19, error names ERR_EXECV, ERR_SEGV, ERR_FILE, ERR_MALLOC, ERR_MISSARG, ERR_GEOM, ERR_WEXP, ERR_BADET, and ERR_DECODE correspond in that order to numbers 10 through 18. For current list, see typedef enum {...} ErrorCode, near the beginning of yume-initA.h.

The yume error number is the numeric result returned to the shell if a yume error occurs. That is, for error number n, yume quits with exit(n). In each message, the phrase "at F:m L:n D:n" shows F, the source-file name; L, the source-file line number where the error was detected; and D, the current depth of file-inclusion. Note, no message appears for error ERR_EXECV.

*** Error ERR_EXECV, execv problem
If a child process started by yume is unable to execv its command, it uses _exit(ERR_EXECV) to terminate. This has no effect on yume and no messages are printed.

*** Error ERR_MISSARG at F:m L:n D:n Switch argument is missing for -Yx
(where x is a letter and -Yx stands for **−at**, **−bd**, **−bw**, **−de**, **−in**, **−iv**, **−la**, or **−ti**) An option switch that needs a parameter was the last parameter in the parameter list or in an included file. For example,

yume -at

will cause a message like above.

*** Error ERR_BADET at F:m L:n D:n Expected entry type n but have type n
(where n's are integers). This message should never appear; if it does, please report the input that caused it, and the yume version number, by email to yume@pat7.com

*** Error ERR_FILE at F:m L:n D:n Unable to open file X
(where X is a filename with path and the n's are integers) appears if a file specified in a -in option setting failed to open. The path should start from current directory or be absolute, and should name an existing readable file, and any chain of includes should be of finite depth. For example, "yume -de FStackLim:567 -in in-test-7" (where in-test-7 is a yume examples include file that includes itself) will end with a message like "*** Error 12 at L:476 D:567 Unable to open file in-test-7"

*** Error ERR_GEOM at F:m L:n D:n Geometry code X not recognized
(where X is a string). This message will appear if, according to gtk_window_parse_geometry(), the geometry parameter to -at is not in proper X Window System form.

*** Error ERR_WEXP at F:m L:n D:n wordexp failure n
(where n's are integers) appears if -in or %- or %: command processing leads to a wordexp problem.

*** Error ERR_DECODE at F:m L:n D:n Lookup error in X
(where X is a string). This message will appear if a -de string contains an unrecognized constant name.

Dealing with errors in shell commands:

If your command list is wrong or malformed, you probably will see error messages in the window where yume was started. Some general techniques for troubleshooting include:

- Try out the command at a shell prompt
- Add temporary echo commands to display command text, or preface the command list with %% (to echo command list before execution) or %# (to echo command list, but not execute it). (See yume-meta(1), examples/date-of-file, and examples/date-of-file.txt.)
- Or, add **−Y4** to the parameter list. See -Y<d> OPTIONS section.

- Add a temporary **−do** group with the problematic command

But before you can use those techniques, you need to identify what command has a problem. Don't depend on line numbers. For an example of this, run examples/test-y-6; click the buttons, and note that different lines of the menu, from different lines of the file, give exactly the same two error messages, attributed to lines 0 and 1.

/bin/bash: -c: line 0: unexpected EOF while looking for matching ‘’’
/bin/bash: -c: line 1: syntax error: unexpected end of file

Because the commands in test-y-6 are so simple, it is no problem to figure out what is causing the problem there. Just copy the text of the command you clicked on, eg "echo 'Ok, let´s check'", into a shell prompt. Then the error will be as obvious as it's going to get and you can quickly resolve the problem. Here, it is that between single quotes, backslash is literal, ie, the first two quotes quote It\, leaving the third quote dangling.

If the problem is in the command behind a button, you can add a temporary echo or %%, or a **−do** group with the same command list, so it is readily visible and possible to edit while testing. Also, read about shell quoting in various man pages; for example, see section QUOTING in bash(1), and in xargs(1) see --null and --delimiter=delim sections. The --delimiter option in xargs allows you to specify an alternative character to delimit parameters.

Environment of menu commands:

In the usual UNIX scheme of things, parent and child have separate environments; the child starts out with a copy of the parent's environment, after which they go their own ways. For example, suppose process P creates child processes R, S. R and S start out with their own copies of P's environment. After that, changes to the environment in any one of the processes has no effect upon that of the other two.

yume's -iv value-fields allows you to change variables in yume's environment while yume is running.

yume and script interaction:

script(1) is a unix command to make a typescript of a terminal session. The sequence

script t
yume ...

will result in file t containing the output of any commands started from the yume menu, along with other session activity, and that output will also appear on the terminal. (But t will not show the yume-invoked commands themselves.) On the other hand, the sequence

yume ...
script t

will result in file t not containing the output of commands started from the yume menu, although that output will appear on the terminal along with other session activity.

Visibility of parameter lists:

Historically, on multiuser unix systems parameter lists to processes could be viewed by anyone who entered a ps -f command. On most unix systems that's no longer so; superuser access is required.

To minimize parameter visibility, yume is just a startup program that runs briefly. It hands parameters over to yume3 and then exits. Parameters to yume usually are visible to ps only for a fraction

of a second. If that is not good enough and the commands of your yume menu should never be be exposed to ps -f, you can put the commands into an access-protected file and use -in option. With a little more effort, it also is possible to put commands into an encrypted file, and pipe directly from a decryptor program into yume3.

In more detail: yume-pipr.c, running as 'yume', (1) accepts parameters for yume3, (2) pipes them to yume3, and (3) exits, while yume3 continues. Both yume and yume3 must be executable and on your search paths for this startup method to work.

Piping a parameter list into yume3:

All of the scripts in the examples/ directory invoke yume via a shell variable called YUME. With pre-2018.01.22 yume versions, yume3 read parameters from stdin. Thus one could make all the examples run yume3 (instead of yume) by cd'ing to examples and entering the following command to run yume-examples:

YUME="yume3" ./yume-examples

or one could turn on -Y display options for any examples run by saying:

YUME="yume -Y4" ./yume-examples

With current versions of yume, yume3 is given a file descriptor to use when reading parameters. That FD is the sole parameter to yume3. Also, yume3 now requires a parameter count and an upper bound on total parameter size, given as the first two lines of input from the pipe. The current set of examples does not cover direct input to yume3. Here is a trivial example, where we suppose paramFile is a file of less than 400 characters, containing 7 parameters to yume, and file 7400 has two lines, containing 7 and 400 respectively:

cat 7400 paramFile | yume3 0

Note, FD 0 (corresponding to stdin) tells yume3 to read stdin.

Safety and SUID and shell history lists:

It is common advice to never use SUID shell scripts (that is, a script one user can run with another's uid). If, against this advice you make a SUID script for a yume menu, it would be blatantly unsafe to include any -do commands, and probably unsafe to have any -iv value fields. That is, limit the menu to buttons with fixed, hidden command text. This increases the difficulty of a user somehow getting shell access.

If, for whatever reason, you decide to include -do or -iv items in such a menu, then use chroot to limit file and command access, and use history -c to limit its view of past shell commands.

HISTORY

Development of yume3, the gtk-based version of yume, began in February 2011. The xforms-based source file yume2.c was split up into yume3.c and yume3-p[123].[ch], and xforms calls in yume3-p2.c and yume3-p2.c were converted to Gtk calls. Size calculations and property settings are quite different in xforms and Gtk, so substantial changes were made in yume3.c and yume3-p1.c as well.

In February, 2009, yume2.c began as a translation of Tcl/Tk program yume.tcl into C, using the xforms API. yume2 was released on SourceForge on 11 March 2009. yume.tcl began as a program called me-me in October, 1997. In December, 1997 it was renamed to yume (a Japanese word that translates to English as "dream"), and then released for public use in 1998; for example, several early releases appear(ed) at <http://yangtze.cs.uiuc.edu/~j-waldby/yume/> .

yume is not related in any way to yum(8), the Yellowdog Updater Modified, nor is it related to Yume Linux (a 2005 SourceForge project, to which virtual host yume.sourceforge.net is assigned), aside from pre-dating

both of them by most of a decade.

yume is a work in progress, and suggestions for improvement are requested. Please send useful example scripts and/or bug reports by email to yume@pat7.com

FILES

INSTALL Not needed after yume is installed, this file explains how to install yume and how to obtain libraries that yume uses to draw graphics objects. The INSTALL file distributed with yume2 explains how to obtain xforms, while that distributed with yume3 explains how to obtain Gtk.

*${yumedirectory}/examples/** Two dozen examples of yume usage appear in the examples directory of the yume distribution. See yume-examples(1) for details.

$HOME/.yume-rc An optional user configuration file for yume default values. If this file exists, it should be a text file containing yume option strings to be treated in the same way as an -in file.

yume-rc The yume3 distribution may include a file *yume-rc* containing yume3 option settings. If so, it lists all configurable items, with default values in commented-out lines. It can be used as a prototype for creating your own *$HOME/.yume3-rc*.

$HOME/.yume-gtkrc An optional user configuration file for Gtk settings. If this file exists, it should be a text file in Gtk resource-file format. Distribution includes file *yume-gtkrc* (that is installed as *$HOME/.yume-gtkrc*) that gives menus with colors as seen in the yume3-examples webpage, eg-overview.html. Other yume3-gtkrc5x files appear in the examples directory and are used by example scripts and explained in .txt files in that directory.

eg-overview.html See webpages <http://yume2.sourceforge.net/examples/eg-overview.html> and <http://yume3.sourceforge.net/examples/eg-overview.html> for numerous examples of yume use.

ENVIRONMENT

SHELL

yume uses the value of environment variable SHELL as the shell it invokes to perform commands.

Please see section DIAGNOSTICS, WARNINGS, CAVEATS regarding the environment available to the shell commands that yume runs.

BUGS

−bd option is not implemented and is reserved for future use.

−de attach is accepted but not acted upon in this release.

−iv label order is different in yume3 and yume2.
yume2: **−la** command **−la** shellvarname **−la** buttonlabel **−iv** initialvalue
yume3: **−la** command **−la** buttonlabel **−la** shellvarname **−iv** initialvalue

This change to **−iv** lets simple menus like

yume −la VVAR −iv vvarInitialValue −bu commandUsingVvar

be given without needing to separately specify a button label. (The **−iv** button in this example is labeled VVAR.) This change probably will be back-ported to yume2.

Hidden-command/general/fixed-command button types: In current release, all **−bu** buttons are "hidden-command" buttons such that command associated with a button cannot be displayed by menu user. A future release may have a right-click popup command view for fixed buttons, or view-and-modify for general buttons. That popup also will allow changing click-mode vs. auto-mode (roll-over) state of various buttons. At present, auto- vs click-mode is controlled by **−de** options.

Before Release 19, changes in **−iv** text boxes did not update environment text unless Enter was pressed or button clicked. This problem was resolved by use of signal "changed" instead of "activate" for CB_TextEntry's InitVal case so that the yume runtime environment changes whenever the text in the **−iv** text box changes.

Item 2 in Section 12.1 ("Utility Argument Syntax") of IEEE Std 1003.1-2008 Open Group Base Specifications Issue 7 states that "a conforming application shall use separate arguments for that option and its option-argument. However, a conforming implementation shall also permit applications to specify the option and option-argument in the same argument string without intervening <blank> characters." Yume does not provide the latter capability but instead always requires separate arguments for option and option-argument.

AUTHOR

James Waldby <j-waldby at pat7 dot com>
Send suggestions and/or bug notes to: yume@pat7.com

SEE ALSO

yume-examples(1)