UWM

users manual

by Christian Ruppert

May 4, 2013

Thanx to everybody who supports us in creating and distributing this system.

Contents

I A first touch **II**

1 `get used to it` **II**

1.1 First steps II

1.2 Some things you might want to try out (playing the piano) III

1.2.1 Raising/Lowering while moving (Tango) III

1.2.2 Multimenu selection (Cha Cha) IV

II Command line options **IV**

III The graphical user interface **V**

2 Menus **V**

3 Windows **V**

4 The Keyboard **V**

IV Configuration **VI**

5 uwmrc – the central UWM configuration file **VI**

6 Menu definition files **XI**

7 urdb files **XII**

Preface

Starting UWM for the first time you might recognize that it doesn't only look different from other window managers but also behaves not quite the way most of you would first expect such a system to do. This fact alone might be a reason for some people to throw UWM away and go back to a conventional windowing user interface.

Others might start thinking — Some of them might get used to it

Part I

A first touch

1 get used to it

UWM was designed to be easy to use once you're used to it. And although there are many people out there thinking different, getting used to UWM is not more difficult than getting used to any other window management system:

How "intuitive" is it to press a button with an X painted on to close a window or to double-click somewhere to start a program? – Well it's exactly as "intuitive" as pressing the right mouse button to start programs and pressing the middle button for dragging windows around. Only that people aren't used to this.

In this section we'll take you on a short tutorial tour through the world of UWM to give you an impression of how to use it. It was written for all those who are curious about UWM but lost in the differences of the user interface.

1.1 First steps

When UWM starts it presents a blank screen to you. There are no buttons, icons, desktop items etc. to be seen and there will never be during your whole UWM session (since they would be covered by windows most of the time anyway).

Press the right mouse button and hold it down. A menu titled Application menu appears. Select an application (the preconfigured items might not be available on all systems, but an xterm should be, so move the pointer over the submenu xterm and select the item login shell by releasing the mouse button above the item).

The application's window will appear on the screen.

Move the pointer somewhere over the window's border and hold down the left mouse button. The famous ude honeycomb will appear. Move the mouse pointer over the hex-icon on the upper left and release the mouse button. Oops... The window just disappeared together with the hex menu. It has been minimized. Some people also call this state of a window iconified. There are no icons for

such windows in UWM however. You might be wondering how to get this damn window back then if it cannot be accessed through an icon. Well, the answer is simple: through a menu!

Press the middle mouse button and hold it down. (Yes, you'll need a three-button-mouse if you want to use UWM properly. However to get a first touch perhaps your X-server's emulate3buttons will do it). A menu called Windows menu appears and either shows a list of workspaces which will pop out submenus if the pointer moves over them or (in case there is only one workspace) represents the only workspace's submenu itself. The submenus are a list of the windows on the corresponding workspace. Search the only submenu with an entry (which is your program's window of course) and select this entry. The window will deiconify and reappear on the screen.

Now move the pointer somewhere over the window's border again and hold down the middle mouse button. Move the mouse and see what happens: The window is being dragged around. Release the window by releasing the mouse button.

Try out resizing the window using the right mouse button on the window's border.

To finally close the window select the central upper button in the window's hex-menu.

Start another xterm and a second one so that you finally have two windows on the screen now. Move them around to be overlaping. Now press and release the middle mouse button somewhere on the upper window's border without moving the pointer. The window will be lowered under the other one. Reraise the window by pressing the left mouse button and releasing it again somewhere on the border without moving the mouse. (The hex menu will appear as long as the button is pressed – ignore it.)

Now try around with uwm's keyboard focus handling: Move the mouse over one of the windows. Its border will change its color. The window now has keyboard focus, try it out by typing something. Also type something into the other window.

Close both windows, the first one by typing `exit` into it, the other one by using the honeycomb's close button.

1.2 Some things you might want to try out (playing the piano)

Once you're used to uwm's basic "feel" as it is described above you might start wondering if this is all or if this piano-like mouse-usage is good for anything except of confusing new users and getting rid of the title-bar. The answer is: It is, and this is one of the things I personally like best about uwm's user interface: The chords.

1.2.1 Raising/Lowering while moving (Tango)

Open several windows on the screen. Drag a window around using the middle mouse button. You might recognize that this window does not change its

stacking position while being dragged. While this effect in most cases is quite useful there might be situations in which you want to raise or lower a window while dragging it and so not only position it two- but threedimensionally. Press the left mouse button while dragging the window (and release it again, keep the middle button pressed while doing this) – Whooops: It's risen to the top. Now press the right button and watch your window disappear behind the other ones...

1.2.2 Multimenu selection
(Cha Cha)

Imagine you have several programs put into a subsubsubsubsubmenu of uwm's Application Menu and want to call two of them at a time. I suppose you don't want to call the Menu, work yourself through the submenus to the application, release the button, let the menu disappear and redo all this from the beginning to call the second program. So to make this a little easier simply keep the right button pressed which will keep the menu alive while selecting the programs to be loaded by clicking on the corresponding items with any other mouse button (Amiga users might remember this feature).

Using this method you can also e.g. open several xterms without having to leave and recall the menu in between. Simply click on the xterm item three times in case you want three xterms. Please note that in this context releasing the right mouse button will only load the selected program in case the corresponding item has not been selected by clicking on it directly before releasing the right button.

Part II
Command line options

Currently uwm supports the following command line options:

`--NoStartScript` will prevent UWM from executing the StartScript defined in uwmrc (see below).

`--NoStopScript` will prevent UWM from executing the StopScript defined in uwmrc (see below).

`--TryHard` conforming to icccm 2.0 UWM has the ability to recognize and replace other (fully) icccm compliant window managers. However the default behaviour is not to replace other window managers since these programs usually will be terminated then which will also terminate the whole X-session in most cases (which shouldn't be done accidentally playing around with UWM). This option uses the protocol specified by icccm 2.0 to make another window manager terminate and pass control to UWM .

`--Hostile` icccm 2.0 allows `X11` resource manager clients to terminate other resource manager clients by deleting their connection to the X-server in case

they do not react to the protocol used by `uwm --TryHard`. `uwm --Hostile` uses this in addition to the `uwm --TryHard` protocol. However there might be some really ugly side effects killing a window managers connection to the X-server so be careful with this option.

`--StayAlive` tells UWM exactly not to react on protocol requests like the ones used by `uwm --TryHarder`. However `uwm --Hostile` is able to replace a `uwm --StayAlive`.

`--help` displays a brief help screen describing UWM 's command line options.

Part III

The graphical user interface

— will be added later —

2 Menus

— will be added later —

3 Windows

— will be added later —

4 The Keyboard

Currently the following keyboard shortcuts are implemented. This is not configurable yet (but will hopefully be in future releases):

`CTRL+ALT+LEFT_ARROW`	go to previous workspace
`CTRL+ALT+RIGHT_ARROW`	go to next workspace
`CTRL+ALT+UP_ARROW`	activate next window
`CTRL+ALT+DOWN_ARROW`	activate previous window
`CTRL+ALT+DOWN_ARROW`	activate previous window
`CTRL+ALT+PG_UP`	raise active window
`CTRL+ALT+PG_DOWN`	lower active window
`CTRL+ALT+`mouse button inside any window	act as if mouse button was pressed on the corresponding window's border

— will be extended later —

Part IV

Configuration

When UWM starts it first searches and reads its central configuration file, uwmrc. Any further configuration files to be read by UWM have to be defined in this file. If no uwmrc is found, UWM will start using default values. However the values assumed might not be correct in all cases which sometimes is a reason for uwm to quit (e.g. in case no hex icons were found).

All configuration files are searched using the same pattern: UWM first looks into the directory `$HOME/.ude/config`. If the file is not found there it checks the directory `$UDEdir/config` or, if the `$UDEdir` environment variable is not set, the global UDE configuration directory which results in something like `/usr/local/share/ude/config`. The UDE default installation directory can be changed at compile/configuration time. Take a look at the `INSTALL`-Readme-File for details about this. If the configuration file is still not found, UWM takes the filename as it is (usually dereferencing any environment variables first).

If a c preprocessor is found on the system UWM is runing on, any configuration files will be passed through the c preprocessor before UWM looks at them. This will make it possible for you to use macros etc. Take a look at the manual of your preprocessor for more info.

5 uwmrc – the central UWM configuration file

uwmrc is uwms basic configuration file. It is loaded at startup and behavoir and outfit of UWM are controlled by this file as well. A typical uwrmc-line looks like this:

```
BorderWidth=10
```

As you can see it consists of a keyword, a '=' and finally a number which represents an argument.

Please note that there are several context-sensitive lines in uwrmc i.e. they have different meanings on different places.

Here's a description of all uwmrc-Keywords:

`BorderWidth = <nr>` sets the width of window borders to `<nr>` pixels. Defaults to 10

`TitleHeight = <nr>` sets the number of pixels the northern border is wider than the other ones. Defaults to 0.

`ScreenColor = <col>` sets the current workspace's screen background color.

`ScreenPixmap = <filename>` sets the current workspace's screen background image to the image file defined by `<filename>`. Overrides `ScreenColor` if set. Currently xpm and jpeg file formats are supported.

`InactiveWin = <col>` sets the color of inactive windows' borders on the current workspace.

`ActiveWin = <col>` sets the color of the active window's border on the current workspace.

`MenuFont = <font>` sets the font used in menus.

`BackColor = <col>` sets the current workspace's background color.

`FontColor = <col>` sets the current workspace's default color for standard text.

`MenuFile = <filename>` makes uwm reading its application menu from `<filename>`. Please note that uwm reads exactly one menu file after finishing reading `uwmrc`. This means that a `MenuFile` line overwrites any previous `MenuFile` lines. If you want to merge several menu files into one application menu, please use the features menu files offer to do this.

`StartScript = <filename>` sets uwms StartScript to `<filename>`. The file must be an executable or a shell script for /bin/sh. If set, this file will be executed when uwm starts. A `StartScript` line overrides any previous `StartScript` lines.

`RubberMove = {0|1}` enables (0) or disables (1) opaque window movement.

`MenuSize = <triple>` sets ude's menu layout: `#1` defines the width of menu bevels, `#2` and `#3` define the label's x- and y-offset.

`NarrowBorderWidth = <nr>` defines the widht of so-called transient window's borders. Most applications mark requesters and other dialogue windows etc. as transient.

`UWMMenuButton = {1|2|3}` defines the button which starts the UWM Menu from the root window. Since uwm doesn't check if all root window menus are accessible, this option should always be used together with `DeiconifyButton` and `AppMenuButton` to make sure all three menus can be reached. Defaults to 1.

`DeiconifyButton = {1|2|3}` defines the button which starts the Windows Menu from the root window. Since uwm doesn't check if all root window menus are accessible, this option should always be used together with `UWMMenuButton` and `AppMenuButton` to make sure all three menus can be reached. Defaults to 2.

`AppMenuButton = {1|2|3}` defines the button which starts the Application Menu from the root window. Since uwm doesn't check if all root window menus are accessible, this option should always be used together with `DeiconifyButton` and `DeiconifyButton` to make sure all three menus can be reached. Defaults to 3.

`TransientMenues = {0|1}` lets you choose if you want your menus disappear when you release the mouse pointer (1) or if you want them to stay until you either select an item or click somewhere outside of the menu (0). Defaults to 1.

`SubMenuTitles = {0|1}` Lets you choose whether redundant titles are displayed for submenus. Defaults to 0.

`WinMenuButton = <triple>` lets you change the button behavior for hex menus. `#1` specifies the button to make the hex menu appear, `#2` specifies the button used to move the window to the previous workspace and `#3` specifies the button used to move the window to the next workspace. Defaults to `1;2;3`. **The use of this option is disencouraged!**

`DragButtons = <triple>` lets you change the button behavior for dragging windows. `#1` specifies the button to enter dragging mode, `#2` specifies the button to raise windows and `#3` specifies the button for lowering windows. Defaults to `2;1;3`. **The use of this option is disencouraged!**

`ResizeButtons = <triple>` specifies the button behaviour for resizing windows. `#1` specifies the button to enter resizing mode, `#2` specifies the button for autoraising and `#3` specifies the button to quit autoraise mode and to activate the 'oldsize'-function. Defaults to `3;1;2`. **The use of this option is disencouraged!**

`WorkSpaces = <nr>` lets you specify the number of workspaces you want.

`WorkSpaceName = <string>` lets you specify a name for the current workspace.

`WorkSpaceNr = <nr>` sets the current workspace. Any options that take effect on the 'current workspace' apply to the workspace set to be the current workspace most recently.

`PlacementStrategy = {0-7}` defines the placement strategy to be used. There are the following possibilities:

0	no placement strategy
2	agressive gradient-placement (place **all** windows automatically)
1 or 3	gradient-placement (automatic placement)
4	agressive interactive placement (place **all** windows semi-automatically)
5	interactive placement (semi-automatic placement)
6	agressive user placement (place **all** windows manually)
7	user placement (manual placement)

`PlacementThreshold = <nr>` defines the overlapping value in pixels from which on you want to place your windows manually in interactive placement stratrgy. This is useless in other placement strategies. In most other WMs 0 is used here without any comments or a way to change. 0 is the default value. If you want this option to make sense your values shouldn't be too small (I tried out 100000 to be quite a good value at a screen-size of 1200×1024).

`ScreenCommand = <string>` defines a command line which is run when the current workspace is entered and killed when it is left. You can e.g. use this to have xearth on the background of one workspace while xsnow makes it winter on another one. Nice toy...

`ReadFrom = <filename>` reads another config file in `uwmrc` format immediately. The file is interpreted as if it was inserted at the place of the corresponding `ReadFrom` line.

`BevelFactor = <float>` defines how extreme bevels are drawn. A value of 1 draws no bevels, values $0 <$ `<float>` < 1 draw deep bevels and values greater than 1 draw usual high bevels. A `BevelFactor` setting affects all 3d-color definitons between itself and the next `BevelFactor` line.

`FrameBevelWidth = <nr>` specifies the bevel width used for window frames.

`OpaqueMoveSize = <nr>` specifies the size in pixels from which on windows are no longer moved opaquely but transparently. A value of 0, which is also the default, means move always opaque, any other value means move transparent from that size on. Values greater than 0 might be useful on slower machines with some applications. You should try out your favourite value or if e.g. transparent movement for all windows works better on your machine with your frequently used applications.

`TitleFont = <font>` specifies the font used for window titles.

`ActiveTitle = <col>` specifies the color the name of active windows is drawn with.

`InactiveTitle = <col>` specifies the color the name of inactive windows is drawn with.

`FrameFlags = <nr>` specifies the layout and behaviour of window titles and borders. Expects a sum of the following values:

1	Groove	draw the groove on window borders if there's enough space.
2	Line	draw a black separation line along the inside of window borders.
4	Inactive Title	display inactive windows' titles.
8	Active Title	display active window's title.
16	Dodgy Title	hide active window's title when hit by the mouse pointer.
32	Center Title	display titles in the center of the top border instead of the northeastern corner.

E.g. use a value of $1 + 2 + 4 + 32 = 39$ if you want titles in the center position disappearing when the window is activated and grooves drawn on your window borders.

`BehaviourFlags = <nr>` specifies parts of uwm's behaviour. Expects a sum of the following values:

1	AllMouse	do not ignore mouse events passed on to uwm by some client windows (e.g. xosview can be moved easily clicking somewhere in the window using this option).

`OtherWMs = <string>` expects a comma seperated list of shell command lines used to start other window managers out of uwm.

`MaxWinWidth = <nr>` defines the maximum window width allowed during this session. This is useful on displays with low resolutions. Unfortunately there might be some problems with applications not regarding the most basic X11 specifications.

`MaxWinHeight = <nr>` defines the maximum window height allowed during this session. This is useful on displays with low resolutions. Unfortunately there might be some problems with applications not regarding the most basic X11 specifications.

`StopScript = <filename>` sets uwms StopScript to `<filename>`. The file must be an executable or a shell script for /bin/sh. If set, this file will be executed when uwm starts. A `StopScript` line overrides any previous `StopScript` lines.

`WarpPointerToNewWinH = <nr>` allows you to make uwm warp the pointer to any newly mapped window. Any value between 0 and 100 defines the X-position in the window (in percent) the pointer is warped to. A value of -1 means don't warp the pointer and a value of -2 means warp the pointer to the upper left corner of the window's border. Only takes effect if `WarpPointerToNewWinV` is set between 0 and 100.

`WarpPointerToNewWinV = <nr>` allows you to make uwm warp the pointer to any newly mapped window. Any value between 0 and 100 defines the Y-position in the window (in percent) the pointer is warped to. A value of -1 means don't warp the pointer and a value of -2 means warp the pointer to the upper left corner of the window's border. Only takes effect if `WarpPointerToNewWinH` is set between 0 and 100.

`InactiveText = <col>` defines the color of any inactive text.

`HighlightedText = <col>` sets font used for highlighted text.

`HighlightedBgr = <col>` sets the background color of highlighted text.

`TextColor = <col>` sets the text color for text windows, e.g. editors, terminals etc.

`TextBgr = <col>` sets the background color for text windows, e.g. editors, terminals etc.

`BevelWidth = <nr>` sets the width of bevels for ude applications (will be used in the library)

`ResourceFile = <filename>` uwm can read a file with the format described in section 7 to set workspace specific application colors etc.

`SnapDistance = <nr>` sets the distance (in pixels) from which a window snaps to another window's or the screen's border when being moved.

`HexPath = <string>` sets the path where uwm first looks for a set of hex icons. The icons must be of .xpm format, and the named directory must contain a complete set of hex icons with the following names and meanings:

Normal State	Selected State	Meaning
autorise.xpm	autorises.xpm	autorise or resize the window
back.xpm	backs.xpm	lower the window
close.xpm	closew.xpm	close the window
iconify.xpm	iconifys.xpm	iconify/minimize the window
kill.xpm	kills.xpm	shut down the application's connection to the x-server
menu.xpm	menus.xpm	open the window menu
really.xpm	reallys.xpm	security button that X-Server connection gets killed by accident

`TextFont = <font>` sets the font used for text windows such as editors or terminals etc. The use of a fixed width font is highly recommended here.

`HighlightFont = <font>` sets the font for highlighted text.

`InactiveFont = <font>` sets the font for inactive elements such as buttons or menu items etc.

In this description the following data types are being used as arguments for the options:

`<nr>` is an integer number with the range specified in the option's description.

`<string>` represents a usual text line. it may contain any desired characters, whitespace etc. and is terminated by a linebreak.

`<font>` is an X11 font definition string. The most easy way to get such a string is to paste it directly from xfontsel into the file.

`<filename>` is the name of a file. The file is searched in the way described above and in most cases passed through the c preprocessor.

`<col>` represents an X11 color definition string. For the exact format of these strings please take a look at the man page of `XQueryColor`. All colors can be set for any workspace seperately.

`<triple>` represents a set of three semicolon seperated integers.

`<float>` represents a floating point number. Please note that the decimal expected seperation character may differ with the internationalized version of (g)libc with different `LANG`-environments set (e.g.`.` as default but `,` for `LANG=de`). Admins of multilingual systems say thanx to the big internationalisationers of libc for this feature.

`{X|Y|Z}` means one out of `X`, `Y` or `z`.

6 Menu definition files

A menu definition file is a hierarchical file made up of the following commands:

`SUBMENU "<name>" {commands to build submenu}` will create a submenu named `<name>` with the items created by the commands inside the braces.

`ITEM "<name>":"<command>";` will create an item on the corresponding position named `<name>` which will lead to the execution of `command` if selected. The item is not created in case there already exists an item with the same `<name>` in the same submenu.

`LINE;` will add a seperation line to the corresponding position. Several `LINE`s with nothing else in between will be truncatd to a single seperator.

`FILE "<filename>";` will process the named file as if its contents were in the position of the `FILE` command. The file is searched for in the way described above and passed through the preprocessor.

`PIPE "<command>";` will call `<command>` and process its standard output as if it was in the position of the `PIPE` command. The commands output is not passed through the preprocessor.

7 urdb files

urdb-files have the same format as xrdb files (see xrdb documentation for details) except that there are some additional macros defined which are replaced by uwm workspace dependent. These macros are:

`@BACKGROUND@` represents the workspace's background color set with `BackColor` in uwmrc.

`@LIGHTCOLOR@` represents the workspace's light bevel color for `BackColor`.

`@SHADOWCOLOR@` represents the workspace's shadow bevel color for `BackColor`.

`@STANDARDTEXT@` represents the workspace's standard text color (`TextColor`).

`@INACTIVETEXT@` represents the workspace's color for inactive text (`InactiveText`)

`@HIGHLIGHTEDTEXT@` represents the workspace's color for highlighted text (`HighlightedText`)

`@HIGHLIGHTEDBGR@` represents the workspace's background color for highlighted text (`HighlightedBgr`)

`@TEXTCOLOR@` represents the workspace's text color for text windows (`TextColor`)

`@TEXTBGR@` represents the workspace's background color for text windows (`TextBgr`)

`@BEVELWIDTH@` the standard bevel width (`BevelWidth`).

`@FLAGS@` 1 if transient menus are activated, if not 0.

`@STANDARDFONT@` the standard font (`MenuFont`).

`@INACTIVEFONT@` the font for inactive buttons etc (`InactiveFont`)

`@HIGHLIGHTFONT@` the font used for highlighted text (`HighlightFont`)

`@TEXTFONT@` the font used for text windows (`TextFont`)