A Guide to wmii-3[*]

Steffen
Liebergeld

with help from
Salvador
Peiró

May 26, 2006

Contents

[*]Thanks to the wmii community, in particular all the people mentioned at WMII/people.

1 Abstract

1.1 The purpose of this document

This document tries to be a good starting point for people new to *wmii*-3. People who have used wmi, *wmii*-2.5 or even ion will get to know what is new and different in *wmii*-3, and people who have never used a tiling window manager before will fall in love with the new concept.

1.2 Wmii—the second generation of window manager improved

Wmii-3 is a new kind of window manager. It is designed to have a small memory footprint, be extremely modularised and have as little code as possible, thus ensuring as few bugs as possible. In fact, one of our official goals is to not exceed $10k$ lines of code [1].

Wmii tries to be very portable and to give the user as much freedom as possible.

Wmii-3 is the third major release of the second generation of the window manager improved [2]. *Wmii* first introduced a new paradigm in version 2.5, namely dynamic window management, that overcomes the limitations imposed by the WIMP paradigm (see also the companion *wmii.tex*).

1.3 Target audience

I presume the reader already has experience with Unix, knows all the basic terminology and concepts like files and editors.

I hope you are open minded towards new ideas, and willing to spend some time learning *wmii*-3 [3].

If you only want to know how to operate *wmii*-3 and are not interested in the inner workings or in scripting, you may read sections 2, 3 and subsection 4.1 and skip the rest.

However, to get the most out of *wmii*-3 you should probably read the whole document "sequentially", i.e. from beginning to end. Another possibility is to read/consume the guide "on demand" as you notice you need more information or details to understand better some concept. We recommend you to read the introductory chapters first, use some time to get settled in the *wmii*-world and read the scripting chapters later on.

2 Configuration and install

2.1 Obtaining wmii

Wmii is licensed under the MIT/X Consortium License, which basically means it is free software, and you are free to download it from http://wmii.de free of charge [4].

[1] benefit of the $10k$ SLOC restriction is that it's easier to read/understand, thus it's easier to use and get used to it

[2] the ii is actually the roman numeral for 2.

[3] remember the refrain: "nobody can teach you what you don't want to know".

[4] please have a look at http://wmii.de/repos/wmii/LICENSE for details

2.2 Configuration and installation

First of all, have a look if there are binary packages of *wmii* in your distribution. Debian, Ubuntu and Gentoo should already have good packages. If you found a trustworthy package, you may now safely skip this section.

For all those who are still reading this, let me tell you that you are on the good side because if you grab the sources and compile them yourself you'll benefit from having everything in its original place, which will ease your use of *wmii*.

1. Uninstalling a previous version:

   ```
   cd /path/to/wmii-previous
   make uninstall && make clean
   ```

   In case you're installing a newer version of *wmii*, this is the first thing you should do otherwise you'll end up mixing binaries, configuration files and manual-pages of different and potentially incompatible versions.

2. Unpack it:

   ```
   tar xzf wmii-3.tar.gz
   cd wmii-3
   ```

3. Edit the configuration:

   ```
   vim config.mk
   ```

   The most important variable to set is the `PREFIX`, which states, where you want *wmii*-3 to be installed to. If you are unsure, keep the default, it won't break your system.

4. Run make and make install:

   ```
   make && make install
   ```

5. Setup the X-server to start *wmii* as your default window manager. You may do that by editing the file *~/.xinitrc*.

   ```
   #!/bin/sh
   exec wmii
   ```

Now you are finished. Please note that autoconf tools are not used for various reasons [5]. Please don't ask the *wmii* developers to use autoconf, they won't listen to you.

[5] read http://www.ohse.de/uwe/articles/aal.html and http://lists.cse.psu.edu/archives/9fans/2003-November/029714.html for further details

3 Terminology

Before you actually start doing your first steps in *wmii*, first the terminology has to be clarified.

3.1 Clients

A client is a program, that provides you a graphical user interface for a special purpose, e.g. a web browser, or a terminal.

3.2 Focus

The (input) focus is the client, which currently receives your input. In X11 exactly one client can get your input at a time. If you input some command into your terminal, the terminal window has the input focus, whereas all the other windows do not receive the input you enter.

3.3 Events

An event is a message generated by X server to notify X clients about states. For instance, X generates a button press event, if you click into a window.

3.4 Tags

A tag is an alphanumeric string you can associate to clients, which allows you to group clients in a natural way.

In *wmii*, there are no workspaces anymore. Instead, all clients matching a particular tag are displayed at a time. For instance, if you tag your browser and a terminal window with the tag "web-browser", and you request to view all clients matching this tag, *wmii* will display your browser and the terminal on the screen. It is also possible to give clients multiple tags, which is described later.

3.5 View

A view is the set of displayed clients, which match a specific single tag. A view is pretty similar to the "workspace" metaphor in other window managers, though more powerful.

Only one view can be visible at a time.

Views are related to the tags, which are currently in use. You have exactly one view for each single tag, thus you can only view sets of clients which match an existing tag.

If you destroy the last client with a tag, the view of this tag is destroyed.

3.6 Column

A column is a distinct part of a view, where clients are arranged automatically in a vertical direction.

In *wmii*, you are able to divide each view into different columns. You should be aware, that every column holds at least one client. As soon as you close the last client of a column, the column is destroyed automatically.

3.7 Layout

A layout is the arrangement of clients in a column. There are three different ways to arrange clients in a column.

default This layout arranges each client with equally vertical space fitting into the column's height.

maximum This layout arranges all clients with the same geometry as the column, showing only one of them at a time.

stacking This layout arranges all clients like a stack, where only the focused client is completely visible, and all other clients can be accessed through its title-bars. This is an alternative approach to *tabbing*.

4 Getting started

Now it is time to start diving into the *wmii* user experience. I suggest you to try everything described by yourself immediately, instead of first reading it, to avoid "memory leakage". It is very helpful, if you print this document on paper or have it available on a different screen, because you might not be able to view it during your first steps in *wmii*.

Note, that the *MOD* key I am referring to, may resemble different keys on different systems. By default it is the *Mod1* or *Alt* key in X11. Normally this is the key labelled with *Alt* on your keyboard.

The notation *MOD-Key* means to press *MOD* and *Key* both at the same time.

All key combinations can be freely configured, but for the sake of simplicity I'll stick with the default key combinations for this guide. You will learn how to alter the bindings in the section 6.

The default key combinations heavily use the home row navigation keys *h* (left), *j* (down), *k* (up), and *l* (right), which are associated with the specific direction.

4.1 First steps

Start your X session now. Since it is the first time you start *wmii*, a window with a little tutorial will show up. You are free to read it, but you may also follow this guide :-)

First of all, press *MOD-Enter* to start an xterm. It will take half of the vertical space, so you have two equally arranged windows. If you press *MOD-Enter* again, you have three windows that are arranged equally.

To switch between the three windows, press *MOD-j*, which cycles the focus between the three windows.

You can also press *MOD-k* to switch to the window above or *MOD-j* to switch to the window below the current.

Now look at the title-bars of those windows. They display important information: the first label contains the tag of the window. The second label displays the window's title.

Similar information is displayed in the status-bar at the bottom. The first labels display the tags currently in use and highlight the currently selected view. On the right side some status information is displayed, by default the system load and the current time (see subsection 6.5 for details).

4.2 Using columns

As described earlier, *wmii* uses columns to arrange your windows. Your view already consists of a single column. Next, you will create a new column.

In *wmii* columns always consists of at least a single client, thus to create a new column, you need at least two clients at hand.

Now focus a client of your choice and press *MOD-Shift-l*, which moves the client rightwards. As you see, *wmii* creates a new column by dividing the view horizontally in two equally big areas. The focused client has been moved into the new column.

If you close the last client of a column, the column is destroyed immediately. If the last client of the current view is closed, the view will be removed accordingly as well.

If you press *MOD-j* to change focus, you will see that *wmii* actually cycles the focus in the current column only.

To change the focus to a different column, you can press *MOD-l* (right) and *MOD-h* (left) respectively.

It is also possible to swap adjacent clients among columns. To swap clients leftwards, press *MOD-Control-h*. To swap clients rightwards, press *MOD-Control-l*.

4.3 What about layouts?

Layouts arrange clients in a column. They are related to a single column. Thus it is possible to have different columns in one view, each using another layout.

The default layout arranges each client of the column with equally vertical space. You can enable this layout with *MOD-d* (where the "d" stands for default) explicitly.

The stacking layout can be enabled with *MOD-s*. As you see now, there is only one client using as much space as possible, and only title-bars of the other clients displayed in the column. You can switch between the clients in the column using *MOD-j*.

The max-layout, enabled via *MOD-m*, maximises all clients to the same geometry as the column. Only the focused client is displayed at a time, all other clients are behind it. You can switch between the clients with *MOD-j*.

4.4 Floating layer

To handle clients in the classical way, like in conventional window managers, the so-called "floating layer" is used. Actually there are a bunch of clients which don't fit well into the tiled world, because they have been designed with the conventional window management in mind, for instance clients like the Gimp or xmms.

While *wmii* is a dynamic window manager, which does the window arrangement for you automatically, those old fashioned programs rely on the conventional window managing concept, where all the clients fly around on your desktop and you are forced to constantly order the mess.

To attach such broken clients to the floating layer, you can toggle the focus between floating and managed layer through pressing *MOD-Space*. The *MOD-Shift-Space* shortcut toggles the focused window between floating and managed layer.

Note, the floating layer is addressed as the zeroth-column internally.

4.5 Tags

Up to now all your clients were tagged with “1”, and you only had this single view. But a single view does not scale well, once too many clients appear which are used for different unrelated tasks. Thus you might want to have a view per task, e.g. a view with your editor and your programming tools, another view with your browser, and a third view with your music jukebox.

The good news is, that the tagging concept provides a very dynamic way to achieve such kinds of grouping.

You can give the focused client another tag by pressing *MOD-Shift-Number*, number being one of the numbers 0 to 9.

You can then switch views through pressing *MOD-Number*.

Normally, whenever a new client appears, it automatically inherits the tag of the currently selected view.

Note, there are more powerful uses of tags you will learn about in the next chapter. You will then be able to assign multiple tags to one client and to use proper strings as tags.

4.6 How do I close a window?

Most X-clients have a menu option or button to be closed. For the rare cases they don’t provide a mouse-driven way, like in most terminals, you can press *MOD-Shift-c* to close a window.

4.7 How do I start programs?

You may start programs from a terminal. But *wmii* contains a special keyboard-driven program menu, which is accessible through pressing *MOD-p*. Please note, that the content of this menu is provided by a simple shell-script.

You will see a list of programs. If you start typing, the menu will cut the list and only display items which match the input you entered so far (in that order). Whenever there is only one item left, the menu highlights it and you can start it by pressing *Enter*. You are free to cancel any action by pressing *ESC*.

Thus, if you want to start firefox, just type “fire” and press enter [6].

4.8 How do I quit wmii?

You can quit *wmii*, by using the action’s menu (*MOD-a*) and selecting the action “quit”. That’s all.

5 Looking under the hood

In this chapter you will learn how *wmii* was designed, which ideas the *wmii* developers followed and how it was implemented.

[6] On my system it is sufficient to type “efo” to start firefox ;-)

5.1 Dynamic window management

wmii was designed around the new idea of dynamic window management. Dynamic window management means, that the window manager should make all decisions about window arrangement, thus taking most extra work away from the user and letting him concentrate on his work. This can also be seen as tacit window management.

5.2 Modularity—using distinct tools for distinct tasks

The developers of *wmii* know about the most powerful ideas of Unix. One of them is the idea to use distinct tools for distinct tasks. By carefully designing the window manager, they were able to split the task into several smaller binaries, each with a distinct job.

5.3 The glue that puts it all together—9P

Programs in the Unix world usually communicate via buffers which are addressed by (file) descriptors, one of them are sockets.

To create a lightweight but powerful communication protocol, the *wmii* developers closely looked at the design of Plan9 and chose the 9P protocol.

The basic ideas for configuring and running *wmii* were taken from the Acme user interface for programmers of Plan9. Similar to Acme, *wmii* provides a filesystem-interface, which can be accessed by 9P clients. This allows to interact with any different kind of application through a file system interface, which might be implemented in any programming language of choice. This also avoids to force client programmers to a specific programming language or paradigm.

The interface of *wmii* can be compared to the *procfs* file system of the Linux kernel.

If you want to interact with a running *wmii* process, you can access its 9P file-system service through either using the bundled tool *wmiir* or the 9P2000 kernel module of a Linux kernel later than 2.4.16+. Using the kernel module has the advantage to mount the filesystem of *wmii* into your file system hierarchy directly, though it has the drawback that this need *root* privileges.

5.4 Tools

This section provides a basic overview about the tools which are bundled with *wmii*. But for a more detailed description, you should read the associated man page of the specific tool.

wmiir is a small tool we is used to access the virtual file-system service of *wmii* remotely. It basically supports four operations:

- read
- write
- remove
- create

wmiir needs to know the address of the file-system service to work with. On startup, *wmii* exports the `WMII_ADDRESS` environment variable, which points to the address of the file-system service of *wmii*. This address can be:

- a local unix socket address like `unix!/path/to/socket`
- a tcp-capable socket address like `tcp!hostname:port`

If you want to work on a different filesystem, you may specify it manually with the *-a address* command line option. A sample invocation looks like:

```
wmiir read /
```

This command actually prints the contents of the root directory of the virtual file-system of *wmii*.

wmiimenu is a generic keyboard-driven menu, which matches items through providing patterns. You may want to learn more about it by reading the man-page.

wmiiwarp is a tiny tool to warp the mouse pointer at specific coordinates on your screen.

wmiiwm is the main window manager binary. You may interact with its virtual file-system only.

wmiipsel prints the contents of the current X selection to STDOUT. This is useful for scripts.

5.5 Conclusion

The virtual file-system service of *wmii* and the tools presented enable you to fully control the window manager from scripts. You will see some examples in the next section 6.

6 Scripting wmii

In this chapter you will see how to control *wmii* through scripts. I will give you some pointers, that you can start scripting on your own.

6.1 Language

As mentioned earlier, the only requirement for interacting with *wmii* is to access its file-system service. The easiest way to do this, is by using the wmiir tool. Thus shell scripts are the easiest way of adapting *wmii* too fit your needs.

Hence, you can control *wmii* in any programming language you want. However, *wmii*'s default scripts are written in a subset of "sh" that is POSIX compliant, to keep *wmii* as *portable* as possible.

To keep simplicity, the following examples will stick to "sh" as well, and don't depend on python, tcl, ruby, ...

6.2 Actions

In *wmii* you may group certain tasks into *actions*. Actions are nothing more than simple scripts which are located either in your local or in the default *wmii* configuration directory [7]. Through pressing *MOD-a* you can open the actions menu. It works similar to the program menu, but only displays actions.

You might want to add your own actions through writing shell scripts in the default *wmii* configuration directory or in the `$HOME/.wmii-3`.

[7] `$CONFPREFIX` is set in *config.mk* which points to `/usr/local/etc` or `$HOME/.wmii-3` by default

This works, because in the *wmii* controlling script exports the variable `$PATH` as `$PATH=~/.wmii-3:$CONFPREFIX/wmii:$PATH` before launching the wmiiwm, this way local user actions under `~/.wmii-3` take precedence over the defaults from `$CONFPREFIX/wmii` of the default actions.

You may edit this file on the fly, which means you don't need to stop *wmii* before editing. After you've finished editing, you may simply run wmiirc and the changes will take effect immediately. To do so, just open the actions menu (with pressing *MOD-a*) and choose the *wmiirc* action. It's also possible to launch *wmii* actions directly from a terminal, which is a neat side effect that results from exporting `$PATH` in the *wmii* startup script.

6.3 wmiirc

wmiirc is a special "sh"-script which is launched on startup of *wmii* to take care of configuring and controlling *wmii*.

It does so through writing data to several files of the virtual *wmii* file-system, and through reading events reported by *wmii* during runtime. Events are mostly shortcut presses, mouse clicks or user-defined. The events are processed in a loop in the script.

Thus, for the basic configuration of *wmii*, like changing the default modifier key *MOD=Mod1* or the navigation keys this script is the place to look at.

The name *wmiirc* means *wmii run command*, because "rc" is an old Unix abbreviation for "run command".

6.4 Changing the style

The style of *wmii*-3 is defined through font and colour values, which are unobtrusively exported with the following *environment variables*.

```
WMII_SELCOLORS='#000000 #eaffff #8888cc'
WMII_NORMCOLORS='#000000 #ffffea #bdb76b'
WMII_FONT=static
```

`WMII_SELCOLORS` defines the colours of the selected client's window title and border, whereas `WMII_NORMCOLORS` defines the colours of all unselected clients. The numbers are hexadecimal rgb tuple-values, which you might know from HTML. You can grab them with the Gimps colour-chooser for instance.

The first colour defines the text colour of strings in bars and menus. The second colour defines the background colour of bars and clients, and the third colour defines the 1px borders surrounding bars and clients.

`WMII_FONT` defines the font which should be used for drawing text. in title-bars, the status-bar, and the wmiimenu. You can query for different fonts using *xfontsel* for instance.

6.5 Filling the status-bar

The status bar of *wmii* has its own `/bar` directory with a subdirectory for each of the labels created. So while editing this document my status-bar looked like:

```
$ wmiir read /bar
d-r-x------ salva salva     0 Mon Apr 17 14:19:51 2006 1
```

```
d-r-x------ salva salva      0 Mon Apr 17 14:19:51 2006 2
d-r-x------ salva salva      0 Mon Apr 17 14:19:51 2006 status
```

At the same time each of the subdirectories contains two files,

```
$ wmiir read /bar/status
--rw------- salva salva     23 Mon Apr 17 14:22:14 2006 colors
--rw------- salva salva     23 Mon Apr 17 14:22:14 2006 data
```

The first file contains the colour definitions that control how the bar will be drawn, while the second contains the data which is displayed.

Now you can start your own experiments by creating a new label, and exploring and modifying it by reading & writing values to its `colors` & `data` files. A nice feature of the bar (and clients) is that they generate events corresponding to mouse clicks on them. You can open a terminal and run `wmiir read /event` to see how the events are generated when you click onto the status-bar. This is a mechanism that allows controlling applications directly from the bar. If you've finished and you want to get rid of your label, a `wmiir remove /bar/foo` command.

If you want to learn more, take a look at the status script and visit http://wmii.de for good examples, like the following:

- *status*: monitoring remaining battery, temperature, ... on laptops
- *status-mpd*: controlling the running mpd
- *status-load*: show the machine load
- *status-net*: monitoring wireless network signal

Now open the default status script and try to understand yourself, how it works . The first line is a handy `xwrite` function declaration, to prevent you from several verbosity when issuing a write to some file. The following 3 lines take care of creating and setting up the `status` label. The last section is a `while` loop that *tries* to write the system load and date information to the bar.

The tricky bit is, that it *tries*. So what could make the write fail? If `xwrite` tried to write to a non existent (removed) label, then it would fail, thus the condition of the loop would be false, and the status script exits cleanly, which makes sense because nobody wants a program that updates a nonexistent label.

Now go back to the first lines of the script, and note that there is a `sleep delay` between the removal of the label and its creation.

This ensures that the `status` label will not exist, hence all the writes made from any previously running `status` script will fail, thus they will finish. This way we make sure that we only run one status script at each time. And thus we keep the one-to-one correspondence between label and status script.

6.6 Assigning new tags

As mentioned before, you can achieve more powerful things with tags, than with the standard key-bindings. You might use any string as a tag. You may even use more than one tag per client. To do so, you have to separate the tags with a "+".

```
echo -n web+code | wmiir write /view/sel/sel/tags
```

This command would give the current focused client the tags “web” and “code”.
You can now view the “web” tag by executing the following:

```
echo -n view web | wmiir write /ctl
```

As the development of *wmii*-3 progressed, it became clear that this action is so common, that it got its own keybinding. By default *MOD-t* brings up a menu to choose a view and *MOD-Shift-t* brings up a menu enabling you to assign new tags to the focused client.

7 The End

We hope this helps you getting used to *wmii*. If you’ve seen something that you think is wrong, confusing or missing in this document, feel free to drop us a note:
Contact information is to be found here: direct mail, [wmii] mailing-list, #wmii irc channel or even with smoke signals [8].
Also remember that *wmii* is written by people with taste, so most of the decisions made have strong reasons supporting them, so if you think something doesn’t make sense or doesn’t fit into the picture, just try to understand it by yourself first before asking, probably you’ll end up learning a lot and if it’s really wrong in the end, you’ll provide us with much better feedback to solve the issue.

[8] but don’t ask us for advice, here you’re on your own `;-P`.

8 Appendix

8.1 filesystem

/bar
- the bar namespace
- to add a label, create /bar/`label` (this automatically creates the `colors` and `data` files.
- to delete it, remove /bar/`label`

/bar/label
- each bar has it's own namespace which is named according to its label
- label can be any arbitrary string

/bar/label/data
- the data written to the label `label`

/bar/label/colors
- the colours of the bar

/client
- the clients namespace

/client/n
- every client, including ones not shown in the view has a namespace here
- `n` is a non-negative integer and clients are numbered oldest first

/client/n/class
- a file containing the class:instance information for client `n`

/client/n/ctl
- control file for client `n`
- accepted commands:
  - kill (removes client)
  - sendto `area|prev|next|new`
  - swap `up|down|prev|next`

/client/n/geom
- the window geometry of client `n`
- displayed as four blank separated integers (x y width height?)

/client/n/index contains the client's index in the /client namespace, in this case n

/client/n/name
- the name of client `n` as it's seen by *wmii* (displayed in the tagbar)

/client/n/tags
- a plus(+)-separated list of tags to which client `n` is related

/ctl
- the *wmii* control file and command interface
- accepted commands:
  - quit
  - retag
  - view `tag`

/def

/def/border width of the border around clients

/def/font the font used by *wmii* (an xlib font name)

/def/keys a newline separated list of the keys to be grabbed by *wmii*

/def/rules
- a newline separated list of rules to specify how to automatically tag clients
- matches against the class.instance values of a client

- syntax: `/$regex/ -> $tag` (tag might be ˜ to indicate floating mode

/def/colwidth with of newly created columns (in px)

/def/colmode mode of newly created columns

/view
- a manifestation of the collection of clients associated with a specific tag

/view/area
- each area has its own namespace in the current view
- the areas are numbered starting with 0
- 0 is always the floating area, the others are columns

/view/area/ctl
- accepted commands:
  - select `client|prev|next`
  - client refers to the client number relative to the number and age of the clients in `area`

/view/area/mode the current layout for the area, equal, stack or max

/view/area/sel the selected client of the area

/view/area/client the namespace for each client in `area`

/view/ctl accepted commands: select `area|prev|next|toggle`

/view/sel the selected area in workspace

/view/tag the tag which is common to all clients in the workspace

8.2 keybindings

Here are the default keybindings. `$MODKEY` is a placeholder, which is usually mapped to Mod1 or Alt.

Keybinding	Action
Moving Focus	
`$MODKEY`-h	move focus to prev column
`$MODKEY`-l	move focus to next column
`$MODKEY`-j	move focus to next client in column
`$MODKEY`-k	move focus to prev client in column
`$MODKEY`-space	toggle to/from floating column 0
`$MODKEY`-[0-9]	select tag/view [0-9]
Moving Clients	
`$MODKEY`-Control-h	swap client with last client of prev column
`$MODKEY`-Control-l	swap client with last client of next column
`$MODKEY`-Control-j	swap client down in the column
`$MODKEY`-Control-k	swap client up in the column
`$MODKEY`-Shift-h	send client to prev column
`$MODKEY`-Shift-l	send client to next column
`$MODKEY`-Shift-space	send client to/from floating column 0
`$MODKEY`-Shift-[0-9]	send client to tag/view [0-9]
Layouts	
`$MODKEY`-d	select default layout
`$MODKEY`-s	select stacked layout
`$MODKEY`-m	select max layout
Menu Bar Functions	
`$MODKEY`-a	choose action from menu bar
`$MODKEY`-p	choose program from menu bar
`$MODKEY`-t	choose view from menu bar
`$MODKEY`-Shift-t	assign tag(s) to client from menu bar
Clients and Applications	
`$MODKEY`-Return	open terminal client
`$MODKEY`-Shift-c	kill client

Table 1: Default keybindings of *wmii*

9 Credits

Author Steffen Liebergeld

Main critic and inquisitor Salvador Peiró

Helpers Anselm Garbe
Denis Grelich
Ross Mohn
Neptun Florin
Jochen Schwartz
Adrian Ratnapala
R. Clayton

10 Copyright notice