# Configuring and extending Ion3 with Lua

Tuomo Valkonen
tuomov at iki.fi

2007-07-20

Configuring and extending Ion3 with Lua

# Contents

# Chapter 1

# Introduction

This document is an "advanced user" manual for the X11 window manager Ion, version 3. It is an attempt at documenting things that go into Ion's configuration files, how to configure Ion by simple modifications to these files and how to write more complex extensions in Lua, the lightweight configuration and scripting language used by Ion.

Readers unfamiliar with Lua might first want to first glance at some Lua documentation at

`http://www.lua.org/docs.html`, or
`http://lua-users.org/wiki/LuaTutorial`,

although this should not be strictly necessary for basic modifications of configuration files for anyone with at least some familiarity with programming languages.

Back in this document, first in chapter 2 some key concepts and relations are explained. These include the module system, and Ion's object (or "region") and class hierarchies. While it may not be necessary to study the latter for basic copy-paste modifications of configuration files – for that you should not really need this manual either – it is, however, essential to for more extensive customisation, due to the semi-object-oriented nature of most of Ion's scripting interface. Knowing the different object types also helps dealing with the different binding "contexts" (see Section 3.3) that to some extent mirror these classes.

The new user, fed up with the default key bindings and eager to just quickly configure Ion to his liking, may therefore just want to skip to Chapter 3, and attempt to work from therefore. That chapter provides the very basic Ion configuration know-how is provided: all the different configuration files and their locations are explained, instructions are given to allow the reader to configure bindings and so-called "winprops", and the statusbar templates are also explained.

Next, Chapter 4 explains the notion of drawing engines and graphical styles and how to write new looks for Ion. More advanced aspects of Ion's scripting interface are documented in Chapter 5. Finally, most of the functions provided by Ion's scripting interface are listed and documented in the Function reference in Chapter 6. At the end of the document an alphabetical listing of all these functions may be found.

# Chapter 2

# Preliminaries: Key concepts and relations

The purpose of this chapter to explain some of key concepts and relations you need to understand before reading the following chapters. These include modules explained in section 2.1 and the Ion class and object hierarchies, section 2.2.

## 2.1 Modules

Ion has been designed so that the 'ion' executable only implements some basic services on top of which very different kinds of window managers could be build by loading the appropriate 'modules'. On modern system these modules are simply dynamically loaded *.so* libraries. On more primitive systems, or if you want to squeeze total size of the executable and libraries, the modules can optionally be statically linked to the main binary, but must nevertheless be loaded with the `dopath` function. Modules may also include Lua code.

If no modules are loaded, all client windows appear in full screen mode. To get better window management support, one or more workspace modules should be loaded. Currently Ion provides the following modules:

***mod_tiling*** Tilings for workspaces of the original tiled Ion kind.

***mod_query*** Queries (for starting programs and so on) and message boxes.

***mod_menu*** Support for menus, both pull-down and keyboard-operated in-frame menus.

***mod_statusbar*** Module that implements a statusbar that can be adaptively embedded in each workspace's layout.

***mod_dock*** Module for docking Window Maker dock-apps. The dock can both float and be embedded as the statusbar.

***mod_sp*** This module implements a scratchpad frame that can be toggled on/off everywhere. Think of the 'console' in some first-person shooters.

***mod_sm*** Session management support module. *Loaded automatically when needed!*

So-called drawing engines are also implemented as a modules, but they are not discussed here; see chapter 4.

The stock configuration for the *ion3* executable loads all of the modules mentioned above except *mod_dock*. The stock configuration for the *pwm3* executable (which differs from the *ion3* executable in a few configuration details) loads another set of modules.

## 2.2 Class and object hierarchies

While Ion does not not have a truly object-oriented design [1], things that appear on the computer screen are, however, quite naturally expressed as such "objects". Therefore Ion implements a rather primitive OO system for these screen objects and some other things.

It is essential for the module writer to learn this object system, but also people who write their own binding configuration files necessarily come into contact with the class and object hierarchies – you need to know which binding setup routines apply where, and what functions can be used as handlers in which bindings. It is the purpose of this section to attempt to explain these hierarchies. If you do not wish the read the full section, at least read the summary at the end of it, so that you understand the very basic relations.

For simplicity we consider only the essential-for-basic-configuration Ioncore, *mod_tiling* and *mod_query* classes. See Appendix B for the full class hierarchy visible to Lua side.

### 2.2.1 Class hierarchy

One of the most important principles of object-oriented design methodology is inheritance; roughly how classes (objects are instances of classes) extend on others' features. Inheritance gives rise to class hierarchy. In the case of single-inheritance this hierarchy can be expressed as a tree where the class at the root is inherited by all others below it and so on. Figure 2.1 lists out the Ion class hierarchy and below we explain what features of Ion the classes implement.

The core classes:

**Obj** Is the base of Ion's object system.

**WRegion** is the base class for everything corresponding to something on the screen. Each object of type WRegion has a size and position relative to the parent WRegion. While a big part of Ion operates on these instead of more specialised classes, WRegion is a "virtual" base class in that there are no objects of "pure" type WRegion; all concrete regions are objects of some class that inherits WRegion.

**WClientWin** is a class for client window objects, the objects that window managers are supposed to manage.

**WWindow** is the base class for all internal objects having an X window associated to them (WClientWins also have X windows associated to them).

1. the author doesn't like such artificial designs

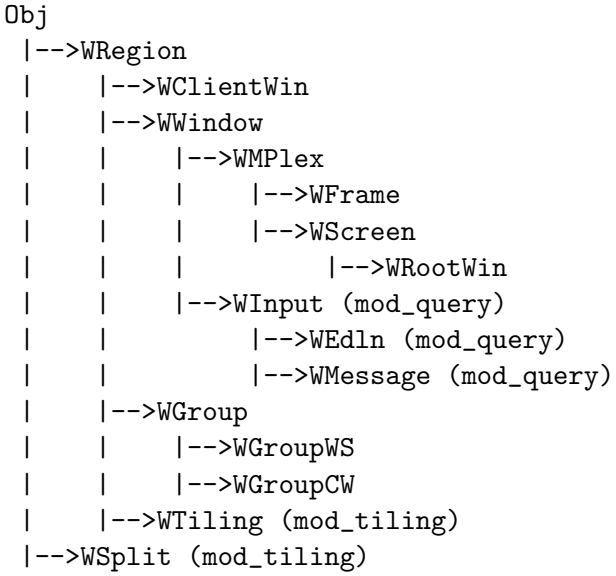


Figure 2.1: Partial Ioncore, *mod_tiling* and *mod_query* class hierarchy.

**WMPlex** is a base class for all regions that "multiplex" other regions. This means that of the regions managed by the multiplexer, only one can be displayed at a time.

**WScreen** is an instance of WMPlex for screens.

**WRootWin** is the class for root windows of X screens. It is an instance of WScreen. Note that an "X screen" or root window is not necessarily a single physical screen as a root window may be split over multiple screens when ugly hacks such as Xinerama are used. (Actually there can be only one root window when Xinerama is used.)

**WFrame** is the class for frames. While most Ion's objects have no graphical presentation, frames basically add to WMPlexes the decorations around client windows (borders, tabs).

**WGroup** is the base class for groups. Particular types of groups are workspaces (WGroupWS) and groups of client windows (WGroupCW).

Classes implemented by the *mod_tiling* module:

**WTiling** is the class for tilings of frames.

**WSplit** (or, more specifically, classes that inherit it) encode the WTiling tree structure.

Classes implemented by the *mod_query* module:

**WInput** is a virtual base class for the two classes below.

**WEdln** is the class for the "queries", the text inputs that usually appear at bottoms of frames and sometimes screens. Queries are the functional equivalent of "mini buffers" in many text editors.

**WMessage** implements the boxes for warning and other messages that Ion may wish to display to the user. These also usually appear at bottoms of frames.

```
WRootWins
 |-->WScreens
      |-->WGroupWSs
      |-->WTilings
      |-->WClientWins in full screen mode
      |-->WFrames
           |-->WGroupCWs
           |-->WClientWins
           |-->WFrames for transients
           |-->a possible WEdln or WMessage
```

Figure 2.2: Most common parent–child relations

There are also some other "proxy" classes that do not refer to objects on the screen. The only important one of these for basic configuration is WMoveresMode that is used for binding callbacks in the move and resize mode.

### 2.2.2 Object hierarchies: WRegion parents and managers

*Parent–child relations*

Each object of type WRegion has a parent and possibly a manager associated to it. The parent for an object is always a WWindow and for WRegion with an X window (WClientWin, WWindow) the parent WWindow is given by the same relation of the X windows. For other WRegions the relation is not as clear. There is generally very few restrictions other than the above on the parent—child relation but the most common is as described in Figure 2.2.

WRegions have very little control over their children as a parent. The manager WRegion has much more control over its managed WRegions. Managers, for example, handle resize requests, focusing and displaying of the managed regions. Indeed the manager—managed relationship gives a better picture of the logical ordering of objects on the screen. Again, there are generally few limits, but the most common hierarchy is given in Figure 2.3. Note that sometimes the parent and manager are the same object and not all objects may have a manager (e.g. the dock in the dock module at the time of writing this) but all have a parent–a screen if not anything else.

*Manager–managed relations*

Note that a workspace can manage another workspace. This can be achieved with the `attach_new` function, and allows you to nest workspaces as deep as you want.

### 2.2.3 Summary

In the standard setup, keeping queries, messages and menus out of consideration:

```
WRootWins
 |-->WScreens
      |-->WGroupCWs for full screen WClientWins
      |    |-->WClientWins
      |    |-->WFrames for transients (dialogs)
      |         |--> WClientWin
      |-->WGroupWSs for workspaces
      |    |-->WTiling
      |    |    |-->WFrames
      |    |    |    |-->WGroupCWs (with contents as above)
      |    |    |-->possibly a WStatusBar or WDock
      |    |-->WFrames for floating content
      |    |-->possibly a WEdln, WMessage or WMenu
      |    |-->possibly a WStatusBar or WDock (if no tiling)
      |-->WFrames for sticky stuff, such as the scratchpad
```

Figure 2.3: Most common manager–managed relations

- The top-level objects that matter are screens and they correspond to physical screens. The class for screens is WScreen.
- Screens contain (multiplex) groups (WGroup) and other objects, such as WFrames. Some of these are mutually exclusive to be viewed at a time.
- Groups of the specific kind WGroupWS often contain a WTiling tiling for tiling frames (WFrame), but groups may also directly contain floating frames.
- Frames are the objects with decorations such as tabs and borders. Frames contain (multiplex) among others (groups of) client windows, to each of which corresponds a tab in the frame's decoration. Only one client window (or other object) can be shown at a time in each frame. The class for client windows is WClientWin.

# Chapter 3

# Basic configuration

This chapter should help your configure Ion to your liking. As the your probably already know, Ion uses Lua as a configuration and extension language. If you're new to it, you might first want to read some Lua documentation as already suggested and pointed to in the Introduction before continuing with this chapter.

Section 3.1 is an overview of the multiple configuration files Ion uses and as a perhaps more understandable introduction to the general layout of the configuration files, a walk-through of the main configuration file *ion.lua* is provided in section 3.2. How keys and mouse action are bound to functions is described in detail in 3.3 and in section 3.5 winprops are explained. Finally, the statusbar is explained in 3.6. For a reference on exported functions, see section 6.

## 3.1 The configuration files

Ion3, to which document applies, stores its stock configuration files in */usr/local/etc/ion3/* unless you, the OS package maintainer or whoever installed the package on the system has modified the variables `PREFIX` or `ETCDIR` in *system.mk* before compiling Ion. In the first case you probably know where to find the files and in the other case the system administrator or the OS package maintainer should have provided documentation to point to the correct location. If these instructions are no help in locating the correct directory, the command `locate cfg_ion.lua` might help provided `updatedb` has been run recently.

User configuration files go in *~/.ion3/*. Ion always searches the user configuration file directory before the stock configuration file directory for files. Therefore, if you want to change some setting, it is advised against that you modify the stock configuration files in-place as subsequent installs of Ion will restore the stock configuration files. Instead you should always make a copy of the stock file in *~/.ion3/* and modify this file. When searching for a file, if no extension or path component is given, compiled *.lc* files are attempted before *.lua* files.

All the configuration files are named *cfg_*.lua* with the "*" part varying. The configuration file for each module *mod_modname* is *cfg_modname.lua*, with *modname* varying by the module in question. The following table summarises these and other configuration files:

| File | Description |
|---|---|
| *cfg_ion.lua* | The main configuration file |
| *cfg_ioncore.lua* | Configuration file for Ion's core library. Most of the bindings and menus are configured here. Bindings that are specific to some module are configured in the module's configuration file. For details, see section 3.3. |
| *cfg_kludges.lua* | Settings to get some applications behave more nicely have been collected here. See section 3.5. |
| *cfg_layouts.lua* | Some workspace layouts are defined here. |
| *cfg_tiling.lua* | Configuration files for different modules. |
| *cfg_query.lua* | |
| *cfg_menu.lua* | |
| *cfg_dock.lua* | |
| *cfg_statusbar.lua* | |
| ... | |

Additionally, there's the file *look.lua* that configures the drawing engine, but it is covered in chapter 4.

## 3.2 A walk through *cfg_ion.lua*

As already mentioned *cfg_ion.lua* is Ion's main configuration file. Some basic 'feel' settings are usually configured there and the necessary modules and other configuration files configuring some more specific aspects of Ion are loaded there. In this section we take a walk through the stock *cfg_ion.lua*. Notice that most of the settings are commented-out (`--` is a line comment in Lua) in the actual file, as they're the defaults nevertheless.

The first thing done in the file, is to set

```
META="Mod1+"
ALTMETA=""
```

These settings cause most of Ion's key bindings to use **Mod1** as the modifier key. If `ALTMETA` is set, it is used as modifier for the keys that don't normally use a modifier. Note that these two are Lua variables used in the configuration files only, and not Ion settings. For details on modifiers and key binding setup in general, see section 3.3.

Next we do some basic feel configuration:

```
ioncore.set{
    dblclick_delay=250,
    kbresize_delay=1500,
}
```

These two will set the delay between button presses in a double click, and the timeout to quit resize mode in milliseconds.

```
ioncore.set{
    opaque_resize=true,
    warp=true
}
```

The first of these two settings enables opaque resize mode: in move/resize move frames and other objects mirror you actions immediately. If opaque resize is disabled, a XOR rubber band is shown during the mode instead. This will, unfortunately, cause Ion to also grab the X server and has some side effects.

There are some other options as well; see the documentation for `ioncore.set` for details.

As a next step, in the actual *cfg_ion.lua* file, we load *cfg_defaults.lua*. However, it is merely a convenience file for doing exactly what we will going through below, and what is commented out in the actual file. If you do not want to load what *cfg_defaults.lua* loads, just comment out the corresponding line, and uncomment the lines for the files that you want:

```
--dopath("cfg_defaults")
dopath("cfg_ioncore")
dopath("cfg_kludges")
dopath("cfg_layouts")
```

Most bindings and menus are defined in *cfg_ioncore.lua*. Details on making such definitions follow in sections 3.3 and 3.4, respectively. some kludges or "winprops" to make some applications behave better under Ion are collected in *cfg_kludges.lua*; see section 3.5 for details. In addition to these, this file lists quite a few statements of the form

```
ioncore.defshortening("[^:]+: (.*)(<[0-9]+>)", "$1$2$|$1$<...$2")
```

These are used to configure how Ion attempts to shorten window titles when they do not fit in a Tab. The first argument is a POSIX regular expression that is used to match against the title and the next is a rule to construct a new title of a match occurs. This particular rule is used to shorten e.g. 'Foo: barbaz<3>' to 'barba...<3>'; for details see the function reference entry for `ioncore.defshortening`. Finally, *cfg_layouts.lua* defines some workspace layouts, available through the **F9** workspace creation query.

To actually be able to do something besides display windows in full screen mode, we must next load some modules:

```
dopath("mod_query")
dopath("mod_menu")
dopath("mod_tiling")
dopath("mod_statusbar")
--dopath("mod_dock")
dopath("mod_sp")
```

## 3.3 Keys and rodents

In the stock configuration file setup, most key and mouse bindings are set from the file *cfg_ioncore.lua* while module-specific bindings are set from the modules' main configuration files (*cfg_modname.lua*). This, however, does not have to be so as long as the module has been loaded prior to defining any module-specific bindings.

Bindings are defined by calling the function `defbindings` with the "context" of the bindings and the a table of new bindings to make. The context is simply string indicating one of the classes of regions (or modes such as WMoveresMode) introduced in section 2.2, and fully listed in appendix B, although not all define a binding map. For example, the following skeleton would be used to define new bindings for all frames:

```
defbindings("WFrame", {
    -- List of bindings to make goes here.
})
```

There has been some confusion among users about the need to define the "context" for each binding, so let me try to explain this design decision here. The thing is that if there was a just a simple 'bind this key to this action' method without knowledge of the context, some limitations would have to be made on the available actions and writing custom handlers would be more complicated. In addition one may want to bind the same function to different key for different types of objects. Indeed, the workspace and frame tab switching functions are the same both classes being based on WMPlex, and in the stock configuration the switch to $n$:th workspaces is bound to **Mod1+n** while the switch to $n$:th tab is bound to the sequence **Mod1+k n**.

Currently known contexts include: '`WScreen`', '`WMPlex`', '`WMPlex.toplevel`', '`WFrame`', '`WFrame.toplevel`', '`WFrame.floating`', '`WFrame.tiled`', '`WFrame.transient`', '`WMoveresMode`', '`WGroup`', '`WGroupCW`', '`WGroupWS`', '`WClientWin`', '`WTiling`', and '`WStatusBar`'. Most of these should be self-explanatory, corresponding to objects of class with the same name. The ones with '`.toplevel`' suffix refer to screens and "toplevel" frames, i.e. frames that are not used for transient windows. Likewise '`.transient`' refers to frames in transient mode, and '`.tiled`' and '`.floating`' to frames in, respectively, tiled and floating modes.

The following subsections describe how to construct elements of the binding table. Note that `defbindings` adds the the newly defined bindings to the previous bindings of the context, overriding duplicates. To unbind an event, set the handler parameter to `nil` for each of the functions to be described in the following subsections.

Also note that when multiple objects want to handle a binding, the innermost (when the root window is considered the outermost) active object in the parent–child hierarchy (see Figure 2.2) of objects gets to handle the action.

### 3.3.1 Binding handlers and special variables

Unlike in Ion2, in Ion3 binding handlers are not normally passed as "anonymous functions", although this is still possible. The preferred method now is to pass the code of

the handler as a string. Two following special variables are available in this code.

| Variable | Description |
|---|---|
| `_` (underscore) | Reference to the object on which the binding was triggered. The object is of the same class as the the context of the `defbindings` call defining the binding. |
| `_sub` | Usually, the currently active *managed object* of the object referred to by `_`, but sometimes (e.g. mouse actions on tabs of frames) something else relevant to the action triggering the binding. |
| `_chld` | Object corresponding to the currently active child window of the object referred to by `_`. This should seldom be needed. |

For example, supposing `_` (underscore) is a WFrame, the following handler should move the active window to the right, if possible:

```
"_:inc_index(_sub)"
```

### 3.3.2 Guards

To suppress error messages, each binding handler may also be accompanied by a "guard" expression that blocks the handler from being called when the guard condition is not met. Currently the following guard expressions are supported (for both `_sub` and `_chld`):

| Guard | Description |
|---|---|
| '`_sub:non-nil`' | The `_sub` parameter must be set. |
| '`_sub:SomeClass`' | The `_sub` parameter must be member of class SomeClass. |

### 3.3.3 Defining the bindings

The descriptions of the individual bindings in the binding table argument to `defbindings` should be constructed with the following functions.

Key presses:

- `kpress`, and `kpress_wait(keyspec, handler [, guard])`.
- `submap(keyspec, { ... more key bindings ... })`.
- `submap_enter`, and `submap_wait(handler [, guard])`.

Mouse actions:

- `mclick`, `mdblclick`, `mpress`, and `mdrag(buttonspec, handler [, guard])`.

The actions that most of these functions correspond to should be clear and as explained in the reference, `kpress_wait` is simply `kpress` with a flag set instructing Ioncore wait for all modifiers to be released before processing any further actions. This is to stop one from accidentally calling e.g. `WRegion.rqclose` multiple times in a row. The `submap` function is used to define submaps or "prefix maps". The second argument to this function is table listing the key press actions (`kpress`) in the submap. The `submap_enter` handler is called when the submap is entered, in which this handler is defined. Likewise, the

`submap_wait` handler is called when all modifiers have been released while waiting for further key presses in the submap.

The parameters `keyspec` and `buttonspec` are explained below in detail. The parameter `handler` is the handler for the binding, and the optional parameter `guard` its guard. These should normally be strings as explained above.

### 3.3.4 Examples

For example, to just bind the key **Mod1+1** to switch to the first workspace and **Mod1+Right** to the next workspace, you would make the following call

```
defbindings("WScreen", {
    kpress("Mod1+Right", "_:switch_next()"),
    kpress("Mod1+1", "_:switch_nth(1)"),
})
```

Note that `_:switch_nth(1)` is the same as calling `WMPlex.switch_next(_, 1)` as WScreen inherits WMPlex and this is where the function is actually defined.

Similarly to the above example, to bind the key sequence **Mod1+k n** switch to the next managed object within a frame, and **Mod1+k 1** to the first, you would issue the following call:

```
defbindings("WFrame", {
    submap("Mod1+K", {
        kpress("Right", "_:switch_next()"),
        kpress("1", "_:switch_nth(1)"),
   }),
})
```

### 3.3.5 Key specifications

As seen above, the functions that create key binding specifications require a `keyspec` argument. This argument should be a string containing the name of a key as listed in the X header file *keysymdef.h*[1] without the `XK_` prefix. Most of the key names are quite intuitive while some are not. For example, the **Enter** key on the main part of the keyboard has the less common name **Return** while the one the numpad is called **KP_Enter**.

The `keyspec` string may optionally have multiple "modifier" names followed by a plus sign (`+`) as a prefix. X defines the following modifiers:

**Shift**, **Control**, **Mod1** to **Mod5**, **AnyModifier** and **Lock**.

X allows binding all of these modifiers to almost any key and while this list of modifiers does not explicitly list keys such as **Alt** that are common on modern keyboards, such

1. This file can usually be found in the directory */usr/X11R6/include/X11/*.

keys are bound to one of the **ModN**. On systems running XFree86 **Alt** is usually **Mod1**. On Suns **Mod1** is the diamond key and **Alt** something else. One of the "flying window" keys on so called Windows-keyboards is probably mapped to **Mod3** if you have such a key. Use the program *xmodmap* to find out what exactly is bound where.

Ion defaults to **AnyModifier** in submaps. This can sometimes lead to unwanted effects when the same key is used with and without explicitly specified modifiers in nested regions. For this reason, Ion recognises **NoModifier** as a special modifier that can be used to reset this default.

Ion ignores the **Lock** modifier and any **ModN** ($N = 1...5$) bound to **NumLock** or **ScrollLock** by default because such[2] locking keys may otherwise cause confusion.

### 3.3.6 Button specifications

Button specifications are similar to key definitions but now instead of specifying modifiers and a key, you specify modifiers and one of the button names **Button1** to **Button5**. Additionally the specification may end with an optional area name following an @-sign. Only frames currently support areas, and the supported values in this case are '`border`', '`tab`', '`empty_tab`', '`client`' and `nil` (for the whole frame).

For example, the following code binds dragging a tab with the first button pressed to initiate tab drag&drop handling:

```
defbindings("WFrame", {
    mdrag("Button1@tab", "_:p_tabdrag()"),
})
```

### 3.3.7 A further note on the default binding configuration

The default binding configuration contains references to the variables `META` and `ALTMETA` instead of directly using the default values of '`Mod1+`' and '' (nothing). As explained in section 3.2, the definitions of these variables appear in *cfg_ion.lua*. This way you can easily change the the modifiers used by all bindings in the default configuration without changing the whole binding configuration. Quite a few people prefer to use the Windows keys as modifiers because many applications already use **Alt**. Nevertheless, **Mod1** is the default as a key bound to it is available virtually everywhere.

## 3.4 Menus

### 3.4.1 Defining menus

In the stock configuration file setup, menus are defined in the file *cfg_menus.lua* as previously mentioned. The *mod_menu* module must be loaded for one to be able to define menus, and this is done with the function `defmenu` provided by it.

---

2. Completely useless keys that should be gotten rid of in the author's opinion.

Here's an example of the definition of a rather simple menu with a submenu:

```
defmenu("exitmenu", {
    menuentry("Restart", "ioncore.restart()"),
    menuentry("Exit", "ioncore.shutdown()"),
})

defmenu("mainmenu", {
    menuentry("Lock screen", "ioncore.exec('xlock')"),
    menuentry("Help", "mod_query.query_man(_)"),
    submenu("Exit", "exitmenu"),
})
```

The `menuentry` function is used to create an entry in the menu with a title and an entry handler to be called when the menu entry is activated. The parameters to the handler are similar to those of binding handlers, and usually the same as those of the binding that opened the menu.

The `submenu` function is used to insert a submenu at that point in the menu. (One could as well just pass a table with the menu entries, but it is not encouraged.)

### 3.4.2 Special menus

The menu module predefines the following special menus. These can be used just like the menus defined as above.

| Menu name | Description |
|---|---|
| '`windowlist`' | List of all client windows. Activating an entry jumps to that window. |
| '`workspacelist`' | List of all workspaces. Activating an entry jumps to that workspaces. |
| '`focuslist`' | List of client windows with recent activity in them, followed by previously focused client windows. |
| '`focuslist_`' | List of previously focused client windows. |
| '`stylemenu`' | List of available *look_*.lua* style files. Activating an entry loads that style and ask to save the selection. |
| '`ctxmenu`' | Context menu for given object. |

### 3.4.3 Defining context menus

The "ctxmenu" is a special menu that is assembled from a defined context menu for the object for which the menu was opened for, but also includes the context menus for the manager objects as submenus.

Context menus for a given region class are defined with the `defctxmenu` function. This is other ways similar to `defmenu`, but the first argument instead being the name of the menu, the name of the region class to define context menu for. For example, here's part of the stock WFrame context menu definition:

```
defctxmenu("WFrame", {
    menuentry("Close", "WRegion.rqclose_propagate(_, _sub)"),
    menuentry("Kill",  "WClientWin.kill(_sub)", "_sub:WClientWin"),
})
```

Some of the same "modes" as were available for some bindings may also be used: '`WFrame.tiled`', '`WFrame.floating`', and '`WFrame.transient`'.

### 3.4.4 Displaying menus

The following functions may be used to display menus from binding handlers (and elsewhere):

| Function | Description |
|---|---|
| `mod_menu.menu` | Keyboard (or mouse) operated menus that open in the bottom-left corner of a screen or frame. |
| `mod_menu.bigmenu` | Same as previous, but uses another graphical style. |
| `mod_menu.pmenu` | Mouse-operated drop-down menus. This function can only be called from a mouse press or drag handler. |
| `mod_menu.grabmenu` | A special version of `mod_menu.menu` that grabs the keyboard and is scrolled with a given key until all modifiers have been released, after which the selected entry is activated. |

The `grabmenu` function takes the extra key parameter, but aside from that each of these functions takes three arguments, which when called from a binding handler, should be the parameters to the handler, and the name of the menu. For example, the following snippet of of code binds the both ways to open a context menu for a frame:

```
defbindings("WFrame", {
    kpress(MOD1.."M", "mod_menu.menu(_, _sub, 'ctxmenu')"),
    mpress("Button3", "mod_menu.pmenu(_, _sub, 'ctxmenu')"),
})
```

## 3.5 Winprops

The so-called "winprops" can be used to change how specific windows are handled and to set up some kludges to deal with badly behaving applications. They are defined by calling the function `defwinprop` with a table containing the properties to set and the necessary information to identify a window. The currently supported winprops are listed below, and the subsequent subsections explain the usual method of identifying windows, and how to obtain this information.

Winprop: `acrobatic` (boolean)

Description: Set this to `true` for Acrobat Reader. It has an annoying habit of trying to manage its dialogs instead of setting them as transients and letting the window manager do its job, causing Ion and acrobat go a window-switching loop when a dialog is opened.

Winprop: `float` (boolean)
Description: Set this to open the window in a floating frame, when in a group.

Winprop: `fullscreen` (boolean)
Description: Should the window be initially in full screen mode?

Winprop: `ignore_cfgrq` (boolean)
Description: Should configure requests on the window be ignored? Only has effect on floating windows.

Winprop: `ignore_net_active_window` (boolean)
Description: Ignore extended WM hints `_NET_ACTIVE_WINDOW` request.

Winprop: `jumpto` (boolean)
Description: Should a newly created client window always be made active, even if the allocated frame isn't.

Winprop: `new_group` (string)
Description: If the region specified by `target` winprop does not exist (or that winprop is not set), create a new workspace using the previously stored layout (see `ioncore.deflayout`) named by this property. After creating the workspace, `target` is attempted to be found again. (If that still fails, the newly created workspace is still asked to manage the client window.)

Winprop: `oneshot` (boolean)
Description: Discard this winprop after first use.

Winprop: `orientation` (string)
Description: The orientation of the window: one of '`vertical`' or '`horizontal`'. This is only useful when using the window as a status display.

Winprop: `statusbar` (string)
Description: Put the window in the statusbar, in the named tray component, (The default tray component is called simply '`systray`', and others you give names to in your custom template, always prefixed by '`systray_`'.

Winprop: `switchto` (boolean)
Description: Should a newly mapped client window be switched to within its frame.

Winprop: `target` (string)
Description: The name of an object (workspace, frame) that should manage windows of this type. See also `new_group`.

Winprop: `transient_mode` (string)
Description: '`normal`': No change in behaviour. '`current`': The window should be thought of as a transient for the current active client window (if any) even if it is not marked as a transient by the application. '`off`': The window should be handled as a normal window even if it is marked as a transient by the application.

Winprop: `transparent` (boolean)
Description: Should frames be made transparent when this window is selected?

### 3.5.1 Sizehint winprops

Additionally, the winprops `max_size`, `min_size`, `aspect`, `resizeinc`, and `ignore_max_size`, `ignore_min_size`, `ignore_aspect`, `ignore_resizeinc`, may be used to override application-supplied size hints. The four first ones are tables with the fields `w` and `h`, indicating the width and height size hints in pixels, and the latter ignore winprop is a boolean.

Finally, the boolean `userpos` option may be used to override the `USPosition` flag of the size hints. Normally, when this flag is set, Ion tries to respect the supplied window position more than when it is not set. Obviously, this makes sense only for floating windows.

### 3.5.2 Classes, roles and instances

The identification information supported are `class`, `role`, `instance`, `name`, `is_transient`, and `is_dockapp`. It is not necessary to specify all of these fields. The first three are strings, and must exactly match the corresponding information obtained from the window's properties. The `name` field is a Lua-style regular expression matched against the window's title. The `is_transient` field is a boolean that can be used to include or exclude transients only, while the `is_dockapp` field is set by Ion for the dock windows of Window Maker dockapp protocol dockapps. Usually this is the only information available for these *icon* windows.

Ion looks for a matching winprop in the order listed by the following table. An 'E' indicates that the field must be set in the winprop and it must match the window's corresponding property exactly or, in case of `name`, the regular expression must match the window title. An asterisk '*' indicates that a winprop where the field is not specified (or is itself an asterisk in case of the first three fields) is tried.

| `class` | `role` | `instance` | other |
|---|---|---|---|
| E | E | E | E |
| E | E | E | * |
| E | E | * | E |
| E | E | * | * |
| E | * | E | E |
| E | * | E | * |
| E | * | * | E |
| ⋮ | ⋮ | ⋮ | etc. |

If there are multiple matching winprops with the same `class`, `role` and `instance`, but other information different, the most recently defined one is used.

### 3.5.3 Finding window identification

The 'Window info' context menu entry (**Mod1+M** or **Button3** on a tab) can be used to list the identification information required to set winprops for a window and all the transient windows managed within it.

Another way to get the identification information is to use `xprop`. Simply run To get class and instance, simply run `xprop WM_CLASS` and click on the particular window of interest. The class is the latter of the strings while the instance is the former. To get the role – few windows have this property – use the command `xprop WM_ROLE`. This method, however, will not work on transients.

So-called "transient windows" are usually short-lived dialogs (although some programs abuse this property) that have a parent window that they are "transient for". On tiled workspaces Ion displays these windows simultaneously with the parent window at the bottom of the same frame. Unfortunately `xprop` is stupid and can't cope with this situation, returning the parent window's properties when the transient is clicked on. For this reason you'll have to do a little extra work to get the properties for that window.[3]

Finally, it should be mentioned that too many authors these days "forget" to set this vital identification to anything meaningful: everything except name is the same for all of the program's windows, for example. Some other programs only set this information after the window has been mapped, i.e. the window manager has been told to start managing it, which is obviously too late. Gtk applications in particular are often guilty on both counts.

### 3.5.4 Some common examples

*Acrobat Reader*

The following is absolutely necessary for Acrobat reader:

```
defwinprop{
    class = "AcroRead",
    instance = "documentShell",
    acrobatic = true,
}
```

*Forcing newly created windows in named frames*

The following winprop should place xterm started with command-line parameter `-name sysmon` and running a system monitoring program in a particular frame:

```
defwinprop{
    class = "XTerm",
```

3. There's a patch to `xprop` to fix this, but nothing seems to be happening with respect to including it in XFree86.

```
    instance = "sysmon",
    target = "sysmonframe",
}
```

For this example to work, we have to somehow create a frame named '`sysmonframe`'. One way to do this is to make the following call in the **Mod1+F3** Lua code query:

```
mod_query.query_renameframe(_)
```

Recall that `_` points to the multiplexer (frame or screen) in which the query was opened. Running this code should open a new query prefilled with the current name of the frame. In our example we would change the name to '`sysmonframe`', but we could just as well have used the default name formed from the frame's class name and an instance number.

## 3.6 The statusbar

The *mod_statusbar* module provides a statusbar that adapts to layouts of tilings, using only the minimal space needed. Ion only supports one adaptive "status display" object per screen, so this statusbar is mutually exclusive with the embedded mode of *mod_dock* docks.

The statusbar is configured in *cfg_statusbar.lua*. Typically, the configuration consists of two steps: creating a statusbar with `mod_statusbar.create`, and then launching the separate `ion-statusd` status daemon process with `mod_statusbar.launch_statusd`. This latter phase is done automatically, if it was not done by the configuration file, but the configuration file may pass extra parameters to `ion-statusd` monitors. (See Section 5.4 for more information on writing `ion-statusd` monitors.)

A typical *cfg_statusbar.lua* configuration might look as follows:

```
-- Create a statusbar
mod_statusbar.create{
    screen = 0,     -- First screen,
    pos = 'bl',     -- bottom left corner
    systray = true, -- Swallow systray windows

    -- The template
    template = "[ %date || load:% %>load || mail:% %>mail_new/%>mail_total ]"
               .. " %filler%systray",
}

-- Launch ion-statusd.
mod_statusbar.launch_statusd{
    -- Date meter
    date={
        -- ISO-8601 date format with additional abbreviated day name
        date_format='%a %Y-%m-%d %H:%M',
```

```
    },
}
```

### 3.6.1 The template

The template specifies what is shown on the statusbar; for information on the other options to `mod_statusbar.create`, see the reference. Strings of the form '`%spec`' tokens specially interpreter by the statusbar; the rest appears verbatim. The `spec` typically consists of the name of the value/meter to display (beginning with a latin alphabet), but may be preceded by an alignment specifier and a number specifying the minimum width. The alignment specifiers are: '`>`' for right, '`<`' for left, and '`|`' for centring. Additionally, space following '`%`' (that is, the string '`% `'), adds "stretchable space" at that point. The special string '`%filler`' may be used to flush the rest of the template to the right end of the statusbar.

The stretchable space works as follows: *mod_statusbar* remembers the widest string (in terms of graphical presentation) that it has seen for each meter, unless the width has been otherwise constrained. If there is stretchable space in the template, it tries to make the meter always take this much space, by stretching any space found in the direction indicated by the alignment specifier: the opposite direction for left or right alignment, and both for centring.

### 3.6.2 The systray

The special '`%systray`' and '`%systray_*`' ('`*`' varying) monitors indicate where to place system tray windows. There may be multiple of these. KDE-protocol system tray icons are placed in '`%systray`' automatically, unless disabled with the `systray` option. Otherwise the `statusbar` winprop may be used to place any window in any particular '`%systray_*`'.

### 3.6.3 Monitors

The part before the first underscore of each monitor name, describes the script/plug-in/module that provides the meter, and any configuration should be passed in the a corresponding sub-table `mod_statusbar.launch_statusd`. Ion comes with date, load and mail (for plain old mbox) `ion-statusd` monitor scripts. More may be obtained from the scripts repository [1]. These included scripts provide the following monitors and their options

*Date*

Options: `date_format`: The date format in as seen above, in the usual `strftime` format. `formats`: table of formats for additional date monitors, the key being the name of the monitor (without the '`date_`' prefix).

Monitors: '`date`' and other user-specified ones with the '`date_`' prefix.

*Load*

Options: `update_interval`: Update interval in milliseconds (default 10s). `important_threshold`: Threshold above which the load is marked as important (default 1.5), so that the drawing engine may be suitably hinted. `critical_threshold`: Threshold above which the load is marked as critical (default 4.0).

Monitors: '`load`' (for all three values), '`load_1min`', '`load_5min`' and '`load_15min`'.

*Mail*

Options: `update_interval`: Update interval in milliseconds (default 1min). `mbox`: mbox-format mailbox location (default `$MAIL`). `files`: list of additional mailboxes, the key giving the name of the monitor.

Monitors: '`mail_new`', '`mail_unread`', '`mail_total`', and corresponding '`mail_*_new`', '`mail_*_unread`', and '`mail_*_total`' for the additional mailboxes ('`*`' varying).

# Chapter 4

# Graphical styles

This chapter first gives in section 4.1 a general outline of how drawing engines are used, of style specifications and then in section 4.2 describes how to specify styles for the default drawing engine. Some additional settings and user attributes are explained in Sections 4.3.

## 4.1 Drawing engines, style specifications and sub-styles

Ion's drawing routines are abstracted into so-called drawing engine modules that can, again depending on the system, be dynamically loaded as needed. The drawing engine modules provide "brushes" that objects can use to draw some high-level primitives such as borders and text boxes (in addition to simple text and rectangle drawing) on their windows and configure e.g. the shape and background of the window. While the drawing engines therefore do not directly implement looks for each possible object (that would hardly be maintainable), different brush styles can be used to give a distinctive look to different objects and engines could interpret some styles as special cases. Style specifications are strings of the form

```
element1-element2-...-elementn
```

An example of such a style specification is '`tab-frame`'; see the table in subsection 4.1.1 for more styles.

When an object asks for a brush of certain style, the selected drawing engine will attempt to find the closest match to this specification. The styles/brushes defined by the drawing engines may have asterisks ('`*`') as some of the elements indicating a match to anything. Exact matches are preferred to asterisk matches and longer matches to shorter. For example, let a brush for style '`foo-bar-baz`' be queried, then the following brushes are in order of preference:

```
foo-bar-baz
foo-*-baz
foo-bar
*
```

Some of the drawing primitives allow extra attributes to be specified, also in the form

`attr1-attr2-...-attrn`

These extra attributes are called *substyles* and allow, for example, the state of the object to be indicated by different colour sets while keeping the interface at an abstract level and the drawing engine completely ignorant of the semantics – only the writer of the drawing engine configuration file has to know them. However the drawing engine can again interpret known substyles as special cases and the default engine indeed does so with frame tab tag and drag states.)

### 4.1.1 Known styles and substyles

*Frames*

| Style name | Description |
|---|---|
| '`frame`' | Style for frames. Substyle attributes: '`active`'/'`inactive`' (mutually exclusive) and '`quasiactive`'/'`not_quasiactive`'. A frame is "quasiactive" when an active region has a back-link to it, such as a detached window. |
| '`frame-tiled`' | A more specific style for tiled frames. Substyle attributes as for '`frame`'. |
| '`frame-tiled-alt`' | An alternative style for tiled frames. Often used to disable the tab-bar. |
| '`frame-floating`' | A more specific style for floating frames. |
| '`frame-transient`' | A more specific style for frames containing transient windows. |

*Tabs and menu entries*

| Style name | Description |
|---|---|
| '`tab`' | Style for frames' tabs and menu entries. Substyle attributes: '`active`'/'`inactive`' and '`selected`'/'`unselected`' |
| '`tab-frame`' | A more specific style for frames' tabs. Additional substyle attributes include: '`tagged`'/'`not_tagged`', '`dragged`'/'`not_dragged`', '`activity`'/'`no_activity`', '`quasiactive`'/'`not_quasiactive`'. |
| '`tab-frame-tiled`', '`tab-frame-tiled-alt`', '`tab-frame-floating`', '`tab-frame-transient`' | More specific styles for frames in the different modes. |
| '`tab-menuentry`' | A more specific style for entries in WMenus. Additional substyle attributes include '`submenu`' and occasionally also '`activity`' is used. |
| '`tab-menuentry-bigmenu`' | An alternate style for entries in WMenus. |

*The rest*

| Style name | Description |
|---|---|
| '`input`' | A style for WInputs. |
| '`input-edln`' | A more specific style for WEdlns. Substyle attributes: '`selection`' for selected text and '`cursor`' for the cursor indicating current editing point. |
| '`input-message`' | A more specific style for WMessages. |
| '`input-menu`' | A more specific style for WMenus. |
| '`input-menu-bigmenu`' | An alternate style for WMenus. |
| '`moveres_display`' | The box displaying position/size when moving or resizing frames. |
| '`stdisp`' | Any status display. |
| '`stdisp-dock`' | The dock. |
| '`stdisp-statusbar`' | The statusbar. Substyles include: the name of any monitor/meter (such as '`date`'), and the supplied hint. Typical hints are: '`normal`', '`important`', and '`critical`'. |

## 4.2 Defining styles for the default drawing engine

Drawing engine style files are usually named *look_foo.lua* where *foo* is the name of the style. The file that Ion loads on startup or when `gr.read_config` is called, however, is *look.lua* and should usually be symlinked to or a copy of of some *look_foo.lua*.

### 4.2.1 The structure of the configuration files

The first thing to do in a style file is to choose the drawing engine, possibly loading the module as well. This is done with the following chunk of code.

```
if not gr.select_engine("de") then
    return
end
```

The `gr.select_engine` function sees if the engine given as argument is registered (the default drawing engine is simply called "de"). If the engine could not be found, it tries to load a module of the same name. If the engine still is not registered, `gr.select_engine` returns '`false`' and in this case we also exit the style setup script. If the engine was found, `gr.select_engine` sees that further requests for brushes are forwarded to that engine and returns '`true`'.

Before defining new styles it may be a good idea to clear old styles from memory so if the old configuration defines more specific styles than the new, the old styles don't override those specified by the new configuration. That can be done by calling

```
de.reset()
```

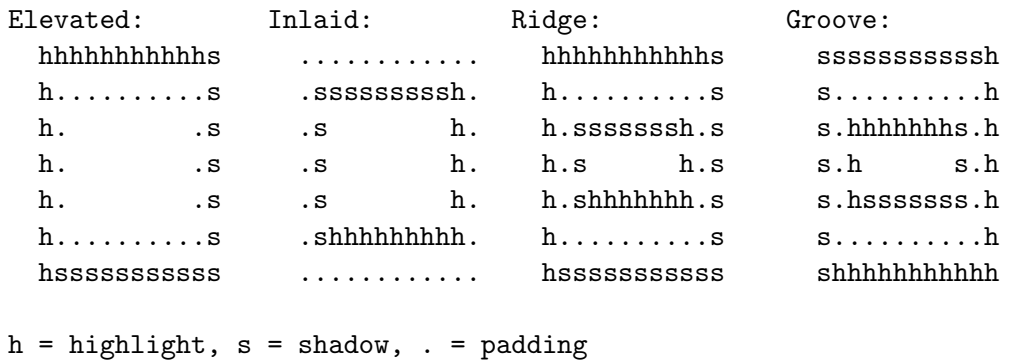


Figure 4.1: Sketch of different border styles and elements

After this the new styles can be defined with `de.defstyle` as explained in the next subsection. Finally, after the styles have been defined we must ask objects on the screen to look up new brushes to reflect the changes in configuration. This is done with

```
gr.refresh()
```

### 4.2.2 Defining the styles

Styles for the default drawing engine are defined with the function `de.defstyle`. It has two arguments the first being a style specification as explained in previous sections and the second a table whose fields describe the style:

```
de.defstyle("some-style", {
    attribute = value,
    ...
})
```

The supported attributes are described in tables below. The different border elements and styles referred to there are explained in Figure 4.1.

*Colours*

Each of these fields a string of the form that can be passed to `XAllocNamedColor`. Valid strings are e.g. hexadecimal RGB specifications of the form `#RRGGBB` and colour names as specified in */usr/X11R6/lib/X11/rgb.txt* (exact path varying).

| Field | Description |
|---|---|
| `highlight_colour` | Colour for the "highlight" part of a border. |
| `shadow_colour` | Colour for the "shadow" part of a border. |
| `foreground_colour` | Colour for the normal drawing operations, e.g. text. |
| `background_colour` | Window background colour (unless transparency is enabled) and background colour boxes. |
| `padding_colour` | Colour for the "padding" part of a border border. Set to `background_colour` if unset. |

*Borders and widths*

All other fields below except `border_style` are non-negative integers indicating a number of pixels.

| Field | Description |
|---|---|
| `border_style` | A string indicating the style of border; one of '`elevated`'/'`inlaid`'/'`ridge`'/'`groove`' as seen in the above sketch. |
| `border_sides` | A string indicating which sides of the border to draw: '`all`'/'`tb`'/'`lr`' for all, top and bottom, and left and right. To control between left/right and top/bottom, use the pixel options below. |
| `highlight_pixels` | Width of the highlight part of the border in pixels. |
| `shadow_pixels` | Width of the shadow part of the border in pixels. |
| `padding_pixels` | Width of the padding part of the border in pixels. |
| `spacing` | Space to be left between all kinds of boxes. |

*Text*

| Field | Description |
|---|---|
| `font` | Font to be used in text-drawing operations; standard X font name. |
| `text_align` | How text is to be aligned in text boxes/tabs; one of the strings '`left`'/'`right`'/'`center`'. |

*Miscellaneous*

| Field | Description |
|---|---|
| `transparent_background` | Should windows' that use this style background be transparent? true/false. |
| `based_on` | The name of a previously defined style that this style should be based on. |

*Substyles*

As discussed in previous sections, styles may have substyles to e.g. indicate different states of the object being drawn. The "de" engine limits what can be configured in substyles to the set of colours in the first table above, but also specifically interprets for the main style '`tab-frame`' the substyles '`*-*-tagged`' and '`*-*-*-dragged`' by, respectively, drawing a right angle shape at the top right corner of a tab and by shading the tab with a stipple pattern. Also for menus the substyles '`*-*-submenu`' are handled as a special case.

Substyles are defined with the function `de.substyle` within the table defining the main style. The parameters to this function are similar to those of `de.defstyle`.

```
de.defstyle("some-style", {
   ...
   de.substyle("some-substyle", {
      ...
   }),
   ...
})
```

### 4.2.3 An example

The following shortened segment from *look_cleanviolet.lua* should help to clarify the matters discussed in the previous subsection.

```
de.defstyle("*", {
    -- Gray background
    highlight_colour = "#eeeeee",
    shadow_colour = "#eeeeee",
    background_colour = "#aaaaaa",
    foreground_colour = "#000000",

    shadow_pixels = 1,
    highlight_pixels = 1,
    padding_pixels = 1,
    spacing = 0,
    border_style = "elevated",

    font = "-*-helvetica-medium-r-normal-*-12-*-*-*-*-*-*-*",
    text_align = "center",
})

de.defstyle("tab-frame", {
    based_on = "*",

    de.substyle("active-selected", {
        -- Violet tab
        highlight_colour = "#aaaacc",
        shadow_colour = "#aaaacc",
        background_colour = "#666699",
        foreground_colour = "#eeeeee",
    }),

    --  More substyles would follow ...
})
```

## 4.3 Miscellaneous settings

### 4.3.1 Frame user attributes

The function `WFrame.set_grattr` may be used to give frames (and their tabs) arbitrary extra attributes to be passed to the drawing engine. Hence, by configuring such substyles in the style configuration files, and turning on the attribute when needed, scripts may display visual cues related to the frame. There is also one extra attribute specially interpreted by the default drawing engine: the '`numbered`' attribute, which causes numbers to be displayed on the tabs.

### 4.3.2 Extra fields for style '`frame`'

The following style fields are independent of the drawing engine used, but are related to objects' styles and therefore configured in the drawing engine configuration file.

| Field | Description |
|---|---|
| `bar` | Controls the style of the tab-bar. Possible values are the strings '`none`', '`inside`', '`outside`' and '`shaped`', with the last providing the PWM-style tab-bars for floating frames. |
| `floatframe_tab_min_w` | Minimum tab width in pixels for the shaped style, given that this number times number of tabs doesn't exceed frame width. |
| `floatframe_bar_max_w_q` | Maximum tab-bar width quotient of frame width for the shaped styles. A number in the interval $(0, 1]$. |

### 4.3.3 Extra fields for style '`dock`'

| Field | Description |
|---|---|
| `outline_style` | How borders are drawn: '`none`' – no border, '`all`' – border around whole dock, '`each`' – border around each dockapp. |
| `tile_size` | A table with entries '`width`' and '`height`', indicating the width and height of tiles in pixels. |

Hopefully that's enough to get you started in writing new style configuration files for Ion. When in doubt, study the existing style configuration files.

# Chapter 5

# Scripting

This chapter documents some additional features of the Ion configuration and scripting interface that can be used for more advanced scripting than the basic configuration explained in chapter 3.

## 5.1 Hooks

Hooks are lists of functions to be called when a certain event occurs. There are two types of them; normal and "alternative" hooks. Normal hooks do not return anything, but alt-hooks should return a boolean indicating whether it handled its assigned task successfully. In the case that `true` is returned, remaining handlers are not called.

Hook handlers are registered by first finding the hook with `ioncore.get_hook` and then calling `WHook.add` on the (successful) result with the handler as parameter. Similarly handlers are unregistered with `WHook.remove`. For example:

```
ioncore.get_hook("ioncore_snapshot_hook"):add(
    function() print("Snapshot hook called.") end
)
```

In this example the hook handler has no parameters, but many hook handlers do. The types of parameters for each hook are listed in the hook reference, section 6.9.

Note that many of the hooks are called in "protected mode" and can not use any functions that modify Ion's internal state.

## 5.2 Referring to regions

### 5.2.1 Direct object references

All Ion objects are passed to Lua scripts as 'userdatas', and you may safely store such object references for future use. The C-side object may be destroyed while Lua still refers to the object. All exported functions gracefully fail in such a case, but if you need to explicitly test that the C-side object still exists, use `obj_exists`.

As an example, the following short piece of code implements bookmarking:

```
local bookmarks={}

-- Set bookmark bm point to the region reg
function set_bookmark(bm, reg)
    bookmarks[bm]=reg
end

-- Go to bookmark bm
function goto_bookmark(bm)
    if bookmarks[bm] then
        -- We could check that bookmarks[bm] still exists, if we
        -- wanted to avoid an error message.
        bookmarks[bm]:goto()
    end
end
```

### 5.2.2 Name-based lookups

If you want to a single non-WClientWin region with an exact known name, use `ioncore.lookup_region`. If you want a list of all regions, use `ioncore.region_list`. Both functions accept an optional argument that can be used to specify that the returned region(s) must be of a more specific type. Client windows live in a different namespace and for them you should use the equivalent functions `ioncore.lookup_clientwin` and `ioncore.clientwin_list`.

To get the name of an object, use `WRegion.name`. Please be aware, that the names of client windows reflect their titles and are subject to changes. To change the name of a non-client window region, use `WRegion.set_name`.

## 5.3 Alternative winprop selection criteria

It is possible to write more complex winprop selection routines than those described in section 3.5. To match a particular winprop using whatever way you want to, just set the `match` field of the winprop to a function that receives the client window as its sole parameter, and that returns `true` if the winprop matches, and `false` otherwise.

The class, instance and role properties can be obtained with `WClientWin.get_ident`, and the title with `WRegion.name`. If you want to match against (almost) arbitrary window properties, have a look at the documentation for the following functions, and their standard Xlib counterparts: `ioncore.x_intern_atom` (XInternAtom), `ioncore.x_get_window_property` (XGetWindowProperty), and `ioncore.x_get_text_property` (XGetTextProperty).

## 5.4 Writing `ion-statusd` monitors

All statusbar meters that do not monitor the internal state of Ion should go in the separate `ion-statusd` program.

Whenever the user requests a meter '`%foo`' or '`%foo_bar`' to be inserted in a statusbar, *mod_statusbar* asks `ion-statusd` to load `statusd_foo.lua` on its search path (same as that for Ion-side scripts). This script should then supply all meters with the initial part '`foo`'.

To provide this value, the script should simply call `statusd.inform` with the name of the meter and the value as a string. Additionally the script should provide a 'template' for the meter to facilitate expected width calculation by *mod_statusbar*, and may provide a 'hint' for colour-coding the value. The interpretation of hints depends on the graphical style in use, and currently the stock styles support the '`normal`', '`important`' and '`critical`' hints.

In our example of the 'foo monitor', at script initialisation we might broadcast the template as follows:

```
statusd.inform("foo_template", "000")
```

To inform *mod_statusbar* of the actual value of the meter and indicate that the value is critical if above 100, we might write the following function:

```
local function inform_foo(foo)
    statusd.inform("foo", tostring(foo))
    if foo>100 then
        statusd.inform("foo_hint", "critical")
    else
        statusd.inform("foo_hint", "normal")
    end
end
```

To periodically update the value of the meter, we must use timers. First we must create one:

```
local foo_timer=statusd.create_timer()
```

Then we write a function to be called whenever the timer expires. This function must also restart the timer.

```
local function update_foo()
    local foo= ... measure foo somehow ...
    inform_foo(foo)
    foo_timer:set(settings.update_interval, update_foo)
end
```

Finally, at the end of our script we want to do the initial measurement, and set up timer for further measurements:

```
update_foo()
```

If our scripts supports configurable parameters, the following code (at the beginning of the script) will allow them to be configured in *cfg_statusbar.lua* and passed to the status daemon and our script:

```
local defaults={
    update_interval=10*1000, -- 10 seconds
}

local settings=table.join(statusd.get_config("foo"), defaults)
```

# Chapter 6

# Function reference

## 6.1 Functions defined in *ioncore*

Synopsis: `ioncore.TR(s, ...)`
Description: gettext+string.format

Synopsis: `ioncore.bdoc(text)`
Description: Used to enter documentation among bindings so that other programs can read it. Does nothing.

Synopsis: `ioncore.chdir_for(reg, dir)`
Description: Change default working directory for new programs started in `reg`.

Synopsis: `ioncore.compile_cmd(cmd, guard)`
Description: Compile string `cmd` into a bindable function. Within `cmd`, the variable "`_`" (underscore) can be used to refer to the object that was selecting for the bound action and chosen to handle it. The variable "`_sub`" refers to a "currently active" sub-object of `_`, or a sub-object where the action loading to the binding being called actually occured.
The string `guard` maybe set to pose limits on `_sub`. Currently supported guards are `_sub:non-nil` and `_sub:WFoobar`, where WFoobar is a class.

Synopsis: `WTimer ioncore.create_timer()`
Description: Create a new timer.

Synopsis: `ioncore.create_ws(scr, tmpl, layout)`
Description: Create new workspace on screen `scr`. The table `tmpl` may be used to override parts of the layout named with `layout`. If no `layout` is given, "default" is used.

Synopsis: `ioncore.defbindings(context, bindings)`
Description: Define bindings for context `context`. Here `binding` is a table composed of entries created with `ioncore.kpress`, etc.; see Section 3.3 for details.

Synopsis: `ioncore.defctxmenu(ctx, ...)`
Description: Define context menu for context `ctx`, `tab` being a table of menu entries.

Synopsis: `ioncore.deflayout(name, tab)`
Description: Define a new workspace layout with name `name`, and attach/creation parameters given in `tab`. The layout "empty" may not be defined.

Synopsis: `ioncore.defmenu(name, tab)`
Description: Define a new menu with `name` being the menu's name and `tab` being a table of menu entries. If `tab.append` is set, the entries are appended to previously-defined ones, if possible.

Synopsis: `ioncore.defwinprop(list)`
Description: Define a winprop. For more information, see section 3.5.

Synopsis: `ioncore.exec_on(reg, cmd, merr_internal)`
Description: Run `cmd` with the environment variable DISPLAY set to point to the root window of the X screen `reg` is on. If `cmd` is prefixed by a colon (`:`), the following command is executed in an xterm (or other terminal emulator) with the help of the `ion-runinxterm` script. If the command is prefixed by two colons, `ion-runinxterm` will ask you to press enter after the command is finished, even if it returns succesfully.

Synopsis: `table ioncore.read_savefile(string basename)`
Description: Read a savefile.

Synopsis: `string ioncore.get_savefile(string basename)`
Description: Get a file name to save (session) data in. The string `basename` should contain no path or extension components.

Synopsis: `string ioncore.lookup_script(string file, string sp)`
Description: Lookup script `file`. If `try_in_dir` is set, it is tried before the standard search path.

Synopsis: `bool ioncore.write_savefile(string basename, table tab)`
Description: Write `tab` in file with basename `basename` in the session directory.

Synopsis: `ioncore.find_manager(obj, t)`
Description: Find an object with type name `t` managing `obj` or one of its managers.

Synopsis: `ioncore.get_dir_for(reg)`
Description: Get default working directory for new programs started in `reg`.

Synopsis: `ioncore.getbindings(maybe_context)`
Description: Get a table of all bindings.

Synopsis: `ioncore.getctxmenu(name)`
Description: Returns a context menu defined with `ioncore.defctxmenu`.

Synopsis: `ioncore.getlayout(name, all)`
Description: Get named layout (or all of the latter parameter is set, but this is for internal use only).

Synopsis: `ioncore.getmenu(name)`
Description: Returns a menu defined with `ioncore.defmenu`.

Synopsis: `ioncore.getwinprop(cwin)`
Description: Find winprop table for `cwin`.

Synopsis: `string ioncore.aboutmsg()`
Description: Returns an about message (version, author, copyright notice).

Synopsis: `WRegion ioncore.activity_first()`
Description: Returns first region on activity list.

Synopsis: `bool ioncore.activity_i(function iterfn)`
Description: Iterate over activity list until `iterfn` returns `false`. The function itself returns `true` if it reaches the end of list without this happening.

Synopsis: `bool ioncore.clientwin_i(function fn)`
Description: Iterate over client windows until `iterfn` returns `false`. The function itself returns `true` if it reaches the end of list without this happening.

Synopsis: `WRegion ioncore.current()`
Description: Returns the currently focused region, if any.

Synopsis: `bool ioncore.defshortening(string rx, string rule, bool always)`
Description: Add a rule describing how too long titles should be shortened to fit in tabs. The regular expression `rx` (POSIX, not Lua!) is used to match titles and when `rx` matches, `rule` is attempted to use as a replacement for title. If `always` is set, the rule is used even if no shortening is necessary.
Similarly to sed's 's' command, `rule` may contain characters that are inserted in the resulting string and specials as follows:

| Special | Description |
|---|---|
| $0 | Place the original string here. |
| $1 to $9 | Insert n:th capture here (as usual,captures are surrounded by parentheses in the regex). |
| $\| | Alternative shortening separator. The shortening described before the first this kind of separator is tried first and if it fails to make the string short enough, the next is tried, and so on. |
| $< | Remove characters on the left of this marker to shorten the string. |
| $> | Remove characters on the right of this marker to shorten the string. Only the first $< or $> within an alternative shortening is used. |

Synopsis: `bool ioncore.detach(WRegion reg, string how)`
Description: Detach or reattach `reg`, depending on whether `how` is '`set`', '`unset`' or '`toggle`'. (Detaching means making `reg` managed by its nearest ancestor

WGroup, framed if `reg` is not itself WFrame. Reattaching means making it managed where it used to be managed, if a return-placeholder exists.) If `reg` is the 'bottom' of some group, the whole group is detached. If `reg` is a WWindow, it is put into a frame.

Synopsis: `integer ioncore.exec(string cmd)`
Description: Run `cmd` with the environment variable DISPLAY set to point to the X display the WM is running on. No specific screen is set unlike with `WRootWin.exec_on`. The PID of the (shell executing the) new process is returned.

Synopsis: `WScreen ioncore.find_screen_id(integer id)`
Description: Find the screen with numerical id `id`.

Synopsis: `bool ioncore.focushistory_i(function iterfn)`
Description: Iterate over focus history until `iterfn` returns `false`. The function itself returns `true` if it reaches the end of list without this happening.

Synopsis: `table ioncore.get()`
Description: Get ioncore basic settings. For details see `ioncore.set`.

Synopsis: `table ioncore.get_paths(table tab)`
Description: Get important directories (the fields `userdir`, `sessiondir`, `searchpath` in the returned table).

Synopsis: `bool ioncore.goto_activity()`
Description: Go to first region on activity list.

Synopsis: `WRegion ioncore.goto_first(WRegion reg, string dirstr, table param)`
Description: Go to first region within `reg` in direction `dirstr`. For information on `param`, see `ioncore.navi_next`. Additionally this function supports the boolean `nofront` field, for not bringing the object to front.

Synopsis: `WRegion ioncore.goto_next(WRegion reg, string dirstr, table param)`
Description: Go to region next from `reg` in direction `dirstr`. For information on `param`, see `ioncore.navi_next`. Additionally this function supports the boolean `nofront` field, for not bringing the object to front.

Synopsis: `WScreen ioncore.goto_next_screen()`
Description: Switch focus to the next screen and return it.
Note that this function is asynchronous; the screen will not actually have received the focus when this function returns.

Synopsis: `WScreen ioncore.goto_nth_screen(integer id)`
Description: Switch focus to the screen with id `id` and return it.
Note that this function is asynchronous; the screen will not actually have received the focus when this function returns.

Synopsis: `WScreen ioncore.goto_prev_screen()`
Description: Switch focus to the previous screen and return it.
Note that this function is asynchronous; the screen will not actually have received the focus when this function returns.

Synopsis: `WRegion ioncore.goto_previous()`
Description: Go to and return to a previously active region (if any).
Note that this function is asynchronous; the region will not actually have received the focus when this function returns.

Synopsis: `bool ioncore.is_i18n()`
Description: Is Ion supporting locale-specifically multibyte-encoded strings?

Synopsis: `bool ioncore.load_module(string modname)`
Description: Attempt to load a C-side module.

Synopsis: `WClientWin ioncore.lookup_clientwin(string name)`
Description: Attempt to find a client window with name `name`.

Synopsis: `WRegion ioncore.lookup_region(string name, string typenam)`
Description: Attempt to find a non-client window region with name `name` and type inheriting `typenam`.

Synopsis: `WRegion ioncore.navi_first(WRegion reg, string dirstr, table param)`
Description: Find first region within `reg` in direction `dirstr`. For information on `param`, see `ioncore.navi_next`.

Synopsis: `WRegion ioncore.navi_next(WRegion reg, string dirstr, table param)`
Description: Find region next from `reg` in direction `dirstr` ('`up`', '`down`', '`left`', '`right`', '`next`', '`prev`', or '`any`'). The table `param` may contain the boolean field `nowrap`, instructing not to wrap around, and the WRegions `no_ascend` and `no_descend`, and boolean functions `ascend_filter` and `descend_filter` on `WRegion` pairs (`to`, `from`), are used to decide when to descend or ascend into another region.

Synopsis: `integer ioncore.popen_bgread(string cmd, function h, function errh)`
Description: Run `cmd` with a read pipe connected to its stdout and stderr. When data is received through one of these pipes, `h` or `errh` is called with that data. When the pipe is closed, the handler is called with `nil` argument. The PID of the new process is returned, or -1 on error.

Synopsis: `string ioncore.progname()`
Description: Returns the name of program using Ioncore.

Synopsis: `bool ioncore.region_i(function fn, string typenam)`

Description: Iterate over all non-client window regions with (inherited) class `typenam` until `iterfn` returns `false`. The function itself returns `true` if it reaches the end of list without this happening.

Synopsis: `void ioncore.request_selection(function fn)`
Description: Request (string) selection. The function `fn` will be called with the selection when and if it is received.

Synopsis: `void ioncore.resign()`
Description: Causes the window manager to simply exit without saving state/session.

Synopsis: `void ioncore.restart()`
Description: Restart, saving session first.

Synopsis: `void ioncore.restart_other(string cmd)`
Description: Attempt to restart another window manager `cmd`.

Synopsis: `void ioncore.set(table tab)`
Description: Set ioncore basic settings. The table `tab` may contain the following fields.

| Field | Description |
|---|---|
| `opaque_resize` | (boolean) Controls whether interactive move and resize operations simply draw a rubberband during the operation (false) or immediately affect the object in question at every step (true). |
| `warp` | (boolean) Should focusing operations move the pointer to the object to be focused? |
| `switchto` | (boolean) Should a managing WMPlex switch to a newly mapped client window? |
| `screen_notify` | (boolean) Should notification tooltips be displayed for hidden workspaces with activity? |
| `frame_default_index` | (string) Specifies where to add new regions on the mutually exclusive list of a frame. One of '`last`', '`next`', (for after current), or '`next-act`' (for after current and anything with activity right after it). |
| `dblclick_delay` | (integer) Delay between clicks of a double click. |
| `kbresize_delay` | (integer) Delay in milliseconds for ending keyboard resize mode after inactivity. |
| `kbresize_t_max` | (integer) Controls keyboard resize acceleration. See description below for details. |
| `kbresize_t_min` | (integer) See below. |
| `kbresize_step` | (floating point) See below. |
| `kbresize_maxacc` | (floating point) See below. |
| `framed_transients` | (boolean) Put transients in nested frames. |
| `float_placement_method` | (string) How to place floating frames. One of '`udlr`' (up-down, then left-right), '`lrud`' (left-right, then up-down), or '`random`'. |
| `mousefocus` | (string) Mouse focus mode: '`disable`' or '`sloppy`'. |
| `unsqueeze` | (boolean) Auto-unsqueeze transients/-menus/queries/etc. |
| `autoraise` | (boolean) Autoraise regions in groups on goto. |

When a keyboard resize function is called, and at most `kbresize_t_max` milliseconds has passed from a previous call, acceleration factor is reset to 1.0. Otherwise, if at least `kbresize_t_min` milliseconds have passed from the from previous acceleration update or reset the squere root of the acceleration factor is incremented by `kbresize_step`. The maximum acceleration factor (pixels/call modulo size hints) is given by `kbresize_`

maxacc. The default values are (200, 50, 30, 100).

Synopsis: `bool ioncore.set_paths(table tab)`
Description: Set important directories (the fields `sessiondir`, `searchpath` of `tab`).

Synopsis: `void ioncore.set_selection(string p)`
Description: Set primary selection and cutbuffer0 to `p`.

Synopsis: `void ioncore.shutdown()`
Description: End session saving it first.

Synopsis: `void ioncore.snapshot()`
Description: Save session.

Synopsis: `void ioncore.tagged_clear()`
Description: Untag all regions.

Synopsis: `WRegion ioncore.tagged_first(bool untag)`
Description: Returns first tagged object, untagging it as well if `untag` is set.

Synopsis: `bool ioncore.tagged_i(function iterfn)`
Description: Iterate over tagged regions until `iterfn` returns `false`. The function itself returns `true` if it reaches the end of list without this happening.

Synopsis: `void ioncore.unsqueeze(WRegion reg, bool override)`
Description: Try to detach `reg` if it fits poorly in its current location. This function does not do anything, unless `override` is set or the `unsqueeze` option of `ioncore.set` is set.

Synopsis: `string ioncore.version()`
Description: Returns Ioncore version string.

Synopsis: `void ioncore.warn(string str)`
Description: Issue a warning. How the message is displayed depends on the current warning handler.

Synopsis: `void ioncore.warn_traced(string str)`
Description: Similar to `ioncore.warn`, but also print Lua stack trace.

Synopsis: `void ioncore.x_change_property(integer win, integer atom, integer atom_type, integer format, string mode, table tab)`
Description: Modify a window property. The `mode` is one of '`replace`', '`prepend`' or '`append`', and format is either 8, 16 or 32. Also see `ioncore.x_get_window_property` and the `XChangeProperty`(3) manual page.

Synopsis: `void ioncore.x_delete_property(integer win, integer atom)`
Description: Delete a window property.

Synopsis: `string ioncore.x_get_atom_name(integer atom)`
Description: Get the name of an atom. See `XGetAtomName`(3) manual page for details.

Synopsis: `table ioncore.x_get_text_property(integer win, integer atom)`
Description: Get a text property for a window. The fields in the returned table (starting from 1) are the null-separated parts of the property. See the `XGetTextProperty` (3) manual page for more information.

Synopsis: `table ioncore.x_get_window_property(integer win, integer atom, integer atom_type, integer n32expected, bool more)`
Description: Get a property `atom` of type `atom_type` for window `win`. The `n32expected` parameter indicates the expected number of 32bit words, and `more` indicates whether all or just this amount of data should be fetched. Each 8, 16 or 32bit element of the property, as deciphered from `atom_type` is a field in the returned table. See `XGetWindowProperty`(3) manual page for more information.

Synopsis: `integer ioncore.x_intern_atom(string name, bool only_if_exists)`
Description: Create a new atom. See `XInternAtom`(3) manual page for details.

Synopsis: `void ioncore.x_set_text_property(integer win, integer atom, table tab)`
Description: Set a text property for a window. The fields of `tab` starting from 1 should be the different null-separated parts of the property. See the `XSetTextProperty` (3) manual page for more information.

Synopsis: `ioncore.kpress(keyspec, cmd, guard)`
Description: Creates a binding description table for the action of pressing a key given by `keyspec` (with possible modifiers) to the function `cmd`. The `guard` controls when the binding can be called. For more informationp see Section 3.3.

Synopsis: `ioncore.kpress_wait(keyspec, cmd, guard)`
Description: This is similar to `ioncore.kpress` but after calling `cmd`, Ioncore waits for all modifiers to be released before processing any further actions. For more information on bindings, see Section 3.3.

Synopsis: `bool ioncore.defer(function fn)`
Description: Defer execution of `fn` until the main loop.

Synopsis: `WHook ioncore.get_hook(string name)`
Description: Find named hook `name`.

Synopsis: `ioncore.match_winprop_dflt(prop, cwin, id)`
Description: The basic name-based winprop matching criteria.

Synopsis: `ioncore.mclick(buttonspec, cmd, guard)`
Description: Creates a binding description table for the action of clicking a mouse button while possible modifier keys are pressed, both given by `buttonspec`, to the function `cmd`. For more information, see Section 3.3.

Synopsis: `ioncore.mdblclick(buttonspec, cmd, guard)`
Description: Similar to `ioncore.mclick` but for double-click. Also see Section 3.3.

Synopsis: `ioncore.mdrag(buttonspec, cmd, guard)`
Description: Creates a binding description table for the action of moving the mouse (or other pointing device) while the button given by `buttonspec` is held pressed and the modifiers given by `buttonspec` were pressed when the button was initially pressed. Also see section 3.3.

Synopsis: `ioncore.menuentry(name, cmd, guard_or_opts)`
Description: Use this function to define normal menu entries. The string `name` is the string shown in the visual representation of menu. The parameter `cmd` and `guard_or_opts` (when string) are similar to those of `ioncore.defbindings` . If `guard_or_opts` is a table, it may contains the `guard` field, and the `priority` field, for controlling positioning of entries in context menus. (The default priority is 1 for most entries, and -1 for auto-generated submenus.)

Synopsis: `ioncore.mpress(buttonspec, cmd, guard)`
Description: Similar to `ioncore.mclick` but for just pressing the mouse button. Also see Section 3.3.

Synopsis: `ioncore.refresh_stylelist()`
Description: Refresh list of known style files.

Synopsis: `ioncore.submap(keyspec, list)`
Description: Returns a function that creates a submap binding description table. When the key press action `keyspec` occurs, Ioncore will wait for a further key presse and act according to the submap. For details, see Section 3.3.

Synopsis: `ioncore.submap_enter(cmd, guard)`
Description: Submap enter event for bindings.

Synopsis: `ioncore.submap_wait(cmd, guard)`
Description: Submap modifier release event for bindings.

Synopsis: `ioncore.submenu(name, sub_or_name, options)`
Description: Use this function to define menu entries for submenus. The parameter `sub_or_name` is either a table of menu entries or the name of an already defined menu. The initial menu entry to highlight can be specified by `options.initial` as either an integer starting from 1, or a function that returns such a number. Another option supported is `options.noautoexpand` that will cause `mod_query.query_menu` to not automatically expand this submenu.

Synopsis: `ioncore.tabnum.clear()`
Description: Clear all tab numbers set by `ioncore.tabnum.show`.

Synopsis: `ioncore.tabnum.show(frame, delay)`
Description: Show tab numbers on `frame`, clearing them when submap grab is released

the next time. If `delay` is given, in milliseconds, the numbers are not actually displayed until this time has passed.

Synopsis: `ioncore.tagged_attach(reg, param)`
Description: Attach tagged regions to `reg`. The method of attach depends on the types of attached regions and whether `reg` implements `attach_framed` and `attach`. If `param` is not set, the default of `{switchto=true}` is used. The function returns `true` if all tagged regions were succesfully attached, and `false` otherwisse.

### 6.1.1 WClientWin functions

Synopsis: `table WClientWin.get_ident(WClientWin cwin)`
Description: Returns a table containing the properties `WM_CLASS` (table entries `instance` and `class`) and `WM_WINDOW_ROLE` (`role`) properties for `cwin`. If a property is not set, the corresponding field(s) are unset in the table.

Synopsis: `void WClientWin.kill(WClientWin cwin)`
Description: Attempt to kill (with `XKillWindow`) the client that owns the X window correspoding to `cwin`.

Synopsis: `void WClientWin.nudge(WClientWin cwin)`
Description: Attempts to fix window size problems with non-ICCCM compliant programs.

Synopsis: `void WClientWin.quote_next(WClientWin cwin)`
Description: Send next key press directly to `cwin`.

Synopsis: `double WClientWin.xid(WClientWin cwin)`
Description: Return the X window id for the client window.

### 6.1.2 WFrame functions

Synopsis: `bool WFrame.is_shaded(WFrame frame)`
Description: Is `frame` shaded?

Synopsis: `void WFrame.maximize_horiz(WFrame frame)`
Description: Attempt to toggle horizontal maximisation of `frame`.

Synopsis: `void WFrame.maximize_vert(WFrame frame)`
Description: Attempt to toggle vertical maximisation of `frame`.

Synopsis: `string WFrame.mode(WFrame frame)`
Description: Get frame mode.

Synopsis: `void WFrame.p_switch_tab(WFrame frame)`
Description: Display the region corresponding to the tab that the user pressed on. This function should only be used by binding it to a mouse action.

Synopsis: `void WFrame.p_tabdrag(WFrame frame)`
Description: Start dragging the tab that the user pressed on with the pointing device. This function should only be used by binding it to *mpress* or *mdrag* action with area '`tab`'.

Synopsis: `bool WFrame.set_grattr(WFrame frame, string attr, string how)`
Description: Set extra drawing engine attributes for the frame. The parameter `attr` is the attribute, and `how` is one of '`set`', '`unset`', or '`toggle`'.

Synopsis: `bool WFrame.set_mode(WFrame frame, string modestr)`
Description: Set frame mode.

Synopsis: `bool WFrame.set_shaded(WFrame frame, string how)`
Description: Set shading state according to the parameter `how` ('`set`', '`unset`', or '`toggle`'). Resulting state is returned, which may not be what was requested.

### 6.1.3 WGroup functions

Synopsis: `WRegion WGroup.attach(WGroup ws, WRegion reg, table param)`
Description: Attach and reparent existing region `reg` to `ws`. The table `param` may contain the fields `index` and `switchto` that are interpreted as for `WMPlex.attach_new`.

Synopsis: `WRegion WGroup.attach_new(WGroup ws, table param)`
Description: Create a new region to be managed by `ws`. At least the following fields in `param` are understood:

| Field | Description |
|---|---|
| `type` | (string) Class of the object to be created. Mandatory. |
| `name` | (string) Name of the object to be created. |
| `switchto` | (boolean) Should the region be switched to? |
| `level` | (integer) Stacking level; default is 1. |
| `modal` | (boolean) Make object modal; ignored if level is set. |
| `sizepolicy` | (string) Size policy; see Section 6.10.1. |
| `bottom` | (boolean) Mark the attached region as the "bottom" of `ws`. |

In addition parameters to the region to be created are passed in this same table.

Synopsis: `WRegion WGroup.bottom(WGroup ws)`
Description: Returns the 'bottom' of `ws`.

Synopsis: `bool WGroup.is_fullscreen(WGroup grp)`
Description: Is `reg` in full screen mode?

Synopsis: `bool WGroup.managed_i(WGroup ws, function iterfn)`
Description: Iterate over managed regions of `ws` until `iterfn` returns `false`. The function itself returns `true` if it reaches the end of list without this happening.

Synopsis: `bool WGroup.set_bottom(WGroup ws, WRegion reg)`
Description: Sets the 'bottom' of `ws`. The region `reg` must already be managed by `ws`, unless `nil`.

Synopsis: `bool WGroup.set_fullscreen(WGroup grp, string how)`
Description: Set client window `reg` full screen state according to the parameter `how` (one of '`set`', '`unset`', or '`toggle`'). Resulting state is returned, which may not be what was requested.

### 6.1.4 WGroupCW functions

### 6.1.5 WGroupWS functions

Synopsis: `bool WGroupWS.attach_framed(WGroupWS ws, WRegion reg, table t)`

Description: Attach region `reg` on `ws`. At least the following fields in `t` are supported:

| Field | Description |
|---|---|
| `switchto` | Should the region be switched to (boolean)? Optional. |
| `geom` | Geometry; `x` and `y`, if set, indicates top-left of the frame to be created while `width` and `height`, if set, indicate the size of the client window within that frame. Optional. |

### 6.1.6 WHook functions

Synopsis: `bool WHook.add(WHook hk, function efn)`
Description: Add `efn` to the list of functions to be called when the hook `hk` is triggered.

Synopsis: `bool WHook.listed(WHook hk, function efn)`
Description: Is `fn` hooked to hook `hk`?

Synopsis: `bool WHook.remove(WHook hk, function efn)`
Description: Remove `efn` from the list of functions to be called when the hook `hk` is triggered.

### 6.1.7 WInfoWin functions

Synopsis: `void WInfoWin.set_text(WInfoWin p, string str, integer maxw)`
Description: Set contents of the info window.

### 6.1.8 WMPlex functions

Synopsis: `WRegion WMPlex.attach(WMPlex mplex, WRegion reg, table param)`
Description: Attach and reparent existing region `reg` to `mplex`. The table `param` may contain the fields `index` and `switchto` that are interpreted as for `WMPlex.attach_new`.

Synopsis: `WRegion WMPlex.attach_new(WMPlex mplex, table param)`
Description: Create a new region to be managed by `mplex`. At least the following fields in `param` are understood (all but `type` are optional).

| Field | Description |
|---|---|
| `type` | (string) Class name (a string) of the object to be created. |
| `name` | (string) Name of the object to be created (a string). |
| `switchto` | (boolean) Should the region be switched to (boolean)? |
| `unnumbered` | (boolean) Do not put on the numbered mutually exclusive list. |
| `index` | (integer) Index on this list, same as for `WMPlex.set_index`. |
| `level` | (integer) Stacking level. |
| `modal` | (boolean) Shortcut for modal stacking level. |
| `hidden` | (boolean) Attach hidden, if not prevented by e.g. the mutually exclusive list being empty. This option overrides `switchto`. |
| `passive` | (boolean) Skip in certain focusing operations. |
| `pseudomodal` | (boolean) The attached region is "pseudomodal" if the stacking level dictates it to be modal. This means that the region may be hidden to display regions with lesser stacking levels. |
| `sizepolicy` | (string) Size policy; see Section 6.10.1. |
| `geom` | (table) Geometry specification. |

In addition parameters to the region to be created are passed in this same table.

Synopsis: `void WMPlex.dec_index(WMPlex mplex, WRegion r)`
Description: Move `r` "left" within objects managed by `mplex` on list 1.

Synopsis: `integer WMPlex.get_index(WMPlex mplex, WRegion reg)`
Description: Get index of `reg` on the mutually exclusive list of `mplex`. The indices begin from zero.. If `reg` is not on the list, -1 is returned.

Synopsis: `table WMPlex.get_stdisp(WMPlex mplex)`
Description: Get status display information. See `WMPlex.get_stdisp` for information on the fields.

Synopsis: `void WMPlex.inc_index(WMPlex mplex, WRegion r)`
Description: Move `r` "right" within objects managed by `mplex` on list 1.

Synopsis: `bool WMPlex.is_hidden(WMPlex mplex, WRegion reg)`
Description: Is `reg` on within `mplex` and hidden?

Synopsis: `bool WMPlex.managed_i(WMPlex mplex, function iterfn)`
Description: Iterate over managed regions of `mplex` until `iterfn` returns `false`. The function itself returns `true` if it reaches the end of list without this happening.

Synopsis: `integer WMPlex.mx_count(WMPlex mplex)`
Description: Returns the number of objects on the mutually exclusive list of `mplex`.

Synopsis: `WRegion WMPlex.mx_current(WMPlex mplex)`
Description: Returns the managed object currently active within the mutually exclusive list of `mplex`.

Synopsis: `bool WMPlex.mx_i(WMPlex mplex, function iterfn)`
Description: Iterate over numbered/mutually exclusive region list of `mplex` until `iterfn` returns `false`. The function itself returns `true` if it reaches the end of list without this happening.

Synopsis: `WRegion WMPlex.mx_nth(WMPlex mplex, integer n)`
Description: Returns the `n`:th object on the mutually exclusive list of `mplex`.

Synopsis: `bool WMPlex.set_hidden(WMPlex mplex, WRegion reg, string how)`
Description: Set the visibility of the region `reg` on `mplex` as specified with the parameter `how` (one of '`set`', '`unset`', or '`toggle`'). The resulting state is returned.

Synopsis: `void WMPlex.set_index(WMPlex mplex, WRegion reg, integer index )`
Description: Set index of `reg` to `index` within the mutually exclusive list of `mplex`. Special values for `index` are:

| | |
|---|---|
| $-1$ | Last. |
| $-2$ | After `WMPlex.mx_current`. |

Synopsis: `WRegion WMPlex.set_stdisp(WMPlex mplex, table t)`
Description: Set/create status display for `mplex`. Table is a standard description of the object to be created (as passed to e.g. `WMPlex.attach_new`). In addition, the following fields are recognised:

| Field | Description |
|---|---|
| `pos` | (string) The corner of the screen to place the status display in: one of '`tl`', '`tr`', '`bl`' or '`br`'. |
| `fullsize` | (boolean) Waste all available space. |
| `action` | (string) If this field is set to '`keep`', `pos` and `fullsize` are changed for the existing status display. If this field is set to '`remove`', the existing status display is removed. If this field is not set or is set to '`replace`', a new status display is created and the old, if any, removed. |

Synopsis: `void WMPlex.switch_next(WMPlex mplex)`
Description: Have `mplex` display next (wrt. currently selected) object managed by it.

Synopsis: `void WMPlex.switch_nth(WMPlex mplex, integer n)`
Description: Have `mplex` display the `n`:th object managed by it.

Synopsis: `void WMPlex.switch_prev(WMPlex mplex)`
Description: Have `mplex` display previous (wrt. currently selected) object managed by it.

### 6.1.9 WMoveresMode functions

Synopsis: `void WMoveresMode.cancel(WMoveresMode mode)`
Description: Return from move/resize cancelling changes if opaque move/resize has not been enabled.

Synopsis: `void WMoveresMode.finish(WMoveresMode mode)`
Description: Return from move/resize mode and apply changes unless opaque move/resize is enabled.

Synopsis: `table WMoveresMode.geom(WMoveresMode mode)`
Description: Returns current geometry.

Synopsis: `void WMoveresMode.move(WMoveresMode mode, integer horizmul, integer vertmul)`
Description: Move resize mode target one step:

| `horizmul/vertmul` | effect |
|---|---|
| -1 | Move left/up |
| 0 | No effect |
| 1 | Move right/down |

Synopsis: `void WMoveresMode.resize(WMoveresMode mode, integer left, integer right, integer top, integer bottom)`
Description: Shrink or grow resize mode target one step in each direction. Acceptable values for the parameters `left`, `right`, `top` and `bottom` are as follows: -1: shrink along, 0: do not change, 1: grow along corresponding border.

Synopsis: `table WMoveresMode.rqgeom(WMoveresMode mode, table g)`
Description: Request exact geometry in move/resize mode. For details on parameters, see `WRegion.rqgeom`.

### 6.1.10 WRegion functions

Synopsis: `WMoveresMode WRegion.begin_kbresize(WRegion reg)`
Description: Enter move/resize mode for `reg`. The bindings set with `ioncore.set_bindings` for WMoveresMode are used in this mode. Of the functions exported by the Ion C core, only `WMoveresMode.resize`, `WMoveresMode.move`, `WMoveresMode.cancel` and `WMoveresMode.end` are allowed to be called while in this mode.

Synopsis: `WRegion WRegion.current(WRegion mgr)`
Description: Return the object, if any, that is considered "currently active" within the objects managed by `mplex`.

Synopsis: `table WRegion.geom(WRegion reg)`
Description: Returns the geometry of `reg` within its parent; a table with fields `x`, `y`, `w` and `h`.

Synopsis: `table WRegion.get_configuration(WRegion reg, bool clientwins)`
Description: Get configuration tree. If `clientwins` is unset, client windows are filtered out.

Synopsis: `bool WRegion.goto(WRegion reg)`
Description: Attempt to display `reg`, save region activity status and then warp to (or simply set focus to if warping is disabled) `reg`.
Note that this function is asynchronous; the region will not actually have received the focus when this function returns.

Synopsis: `WRegion WRegion.groupleader_of(WRegion reg)`
Description: Returns the group of `reg`, if `reg` is its bottom, and `reg` itself otherwise.

Synopsis: `bool WRegion.is_active(WRegion reg)`
Description: Is `reg` active/does it or one of it's children of focus?

Synopsis: `bool WRegion.is_activity(WRegion reg)`
Description: Is activity notification set on `reg`.

Synopsis: `bool WRegion.is_mapped(WRegion reg)`
Description: Is `reg` visible/is it and all it's ancestors mapped?

Synopsis: `bool WRegion.is_tagged(WRegion reg)`
Description: Is `reg` tagged?

Synopsis: `WRegion WRegion.manager(WRegion reg)`
Description: Returns the region that manages `reg`.

Synopsis: `string WRegion.name(WRegion reg)`
Description: Returns the name for `reg`.

Synopsis: `WWindow WRegion.parent(WRegion reg)`
Description: Returns the parent region of `reg`.

Synopsis: `WRootWin WRegion.rootwin_of(WRegion reg)`
Description: Returns the root window `reg` is on.

Synopsis: `void WRegion.rqclose(WRegion reg, bool relocate)`
Description: Attempt to close/destroy `reg`. Whether this operation works depends on whether the particular type of region in question has implemented the feature and, in case of client windows, whether the client supports the `WM_DELETE` protocol (see also `WClientWin.kill`). The region will not be destroyed when this function returns. To find out if and when it is destroyed, use the '`deinit`' notification. If `relocate` is not set, and `reg` manages other regions, it will not be closed. Otherwise the managed regions will be attempted to be relocated.

Synopsis: `WRegion WRegion.rqclose_propagate(WRegion reg, WRegion maybe_sub)`

Description: Recursively attempt to close a region or one of the regions managed by it. If `sub` is set, it will be used as the managed region, otherwise `WRegion.current(reg)`. The object to be closed is returned, or NULL if nothing can be closed. For further details, see notes for `WRegion.rqclose`.

Synopsis: `table WRegion.rqgeom(WRegion reg, table g)`
Description: Attempt to resize and/or move `reg`. The table `g` is a usual geometry specification (fields `x`, `y`, `w` and `h`), but may contain missing fields, in which case, `reg`'s manager may attempt to leave that attribute unchanged.

Synopsis: `bool WRegion.rqorder(WRegion reg, string ord)`
Description: Request ordering. Currently supported values for `ord` are '`front`' and '`back`'.

Synopsis: `WScreen WRegion.screen_of(WRegion reg)`
Description: Returns the screen `reg` is on.

Synopsis: `bool WRegion.set_activity(WRegion reg, string how)`
Description: Set activity flag of `reg`. The `how` parameter must be one of '`set`', '`unset`' or '`toggle`'.

Synopsis: `bool WRegion.set_name(WRegion reg, string p)`
Description: Set the name of `reg` to `p`. If the name is already in use, an instance number suffix '`<n>`' will be attempted. If `p` has such a suffix, it will be modified, otherwise such a suffix will be added. Setting `p` to nil will cause current name to be removed.

Synopsis: `bool WRegion.set_name_exact(WRegion reg, string p)`
Description: Similar to `WRegion.set_name` except if the name is already in use, other instance numbers will not be attempted. The string `p` should not contain a '`<n>`' suffix or this function will fail.

Synopsis: `bool WRegion.set_tagged(WRegion reg, string how)`
Description: Change tagging state of `reg` as defined by `how` (one of '`set`', '`unset`', or '`toggle`'). The resulting state is returned.

Synopsis: `table WRegion.size_hints(WRegion reg)`
Description: Returns size hints for `reg`. The returned table always contains the fields `min_?`, `base_?` and sometimes the fields `max_?`, `base_?` and `inc_?`, where `?`=`w`, `h`.

### 6.1.11 WRootWin functions

Synopsis: `WScreen WRootWin.current_scr(WRootWin rootwin)`
Description: Returns previously active screen on root window `rootwin`.

### 6.1.12 WScreen functions

Synopsis: `integer WScreen.id(WScreen scr)`
Description: Return the numerical id for screen `scr`.

Synopsis: `bool WScreen.set_managed_offset(WScreen scr, table offset)`
Description: Set offset of objects managed by the screen from actual screen geometry. The table `offset` should contain the entries `x`, `y`, `w` and `h` indicating offsets of that component of screen geometry.

### 6.1.13 WTimer functions

Synopsis: `bool WTimer.is_set(WTimer timer)`
Description: Is timer set?

Synopsis: `void WTimer.reset(WTimer timer)`
Description: Reset timer.

Synopsis: `void WTimer.set(WTimer timer, integer msecs, function fn)`
Description: Set `timer` to call `fn` in `msecs` milliseconds.

### 6.1.14 WWindow functions

Synopsis: `void WWindow.p_move(WWindow wwin)`
Description: Start moving `wwin` with the mouse or other pointing device. This function should only be used by binding it to *mpress* or *mdrag* action.

Synopsis: `void WWindow.p_resize(WWindow wwin)`
Description: Start resizing `wwin` with the mouse or other pointing device. This function should only be used by binding it to *mpress* or *mdrag* action.

Synopsis: `double WWindow.xid(WWindow wwin)`
Description: Return the X window id for `wwin`.

### 6.1.15 global functions

Synopsis: `export(lib, ...)`
Description: Export a list of functions from `lib` into global namespace.

### 6.1.16 gr functions

Synopsis: `void gr.read_config()`
Description: Read drawing engine configuration file *look.lua*.

Synopsis: `void gr.refresh()`
Description: Refresh objects' brushes to update them to use newly loaded style.

Synopsis: `bool gr.select_engine(string engine)`
Description: Future requests for "brushes" are to be forwarded to the drawing engine `engine`. If no engine of such name is known, a module with that name is attempted to be loaded. This function is only intended to be called from colour scheme etc. configuration files and can not be used to change the look of existing objects; for that use `gr.read_config`.

### 6.1.17 string functions

Synopsis: `string.shell_safe(str)`
Description: Make `str` shell-safe.

### 6.1.18 table functions

Synopsis: `table.append(t1, t2)`
Description: Add entries that do not exist in `t1` from `t2` to `t1`.

Synopsis: `table.copy(t, deep)`
Description: Make copy of `table`. If `deep` is unset, shallow one-level copy is made, otherwise a deep copy is made.

Synopsis: `table.icat(t1, t2)`
Description: Insert all positive integer entries from t2 into t1.

Synopsis: `table.join(t1, t2)`
Description: Create a table containing all entries from `t1` and those from `t2` that are missing from `t1`.

Synopsis: `table.map(f, t)`
Description: Map all entries of `t` by `f`.

## 6.2 Functions defined in *mod_tiling*

Synopsis: `table mod_tiling.get()`
Description: Get parameters. For details see `mod_tiling.set`.

Synopsis: `bool mod_tiling.mkbottom(WRegion reg)`
Description: Create a new WTiling 'bottom' for the group of `reg`, consisting of `reg`.

Synopsis: `void mod_tiling.set(table tab)`
Description: Set parameters. Currently only `raise_delay` (in milliseconds) is supported.

Synopsis: `bool mod_tiling.untile(WTiling tiling)`
Description: If `tiling` is managed by some group, float the frames in the tiling in that group, and dispose of `tiling`.

### 6.2.1 WSplit functions

Synopsis: `table WSplit.geom(WSplit split)`
Description: Returns the area of workspace used by the regions under `split`.

Synopsis: `WSplitInner WSplit.parent(WSplit split)`
Description: Return parent split for `split`.

Synopsis: `table WSplit.rqgeom(WSplit node, table g)`
Description: Attempt to resize and/or move the split tree starting at `node`. Behaviour and the `g` parameter are as for `WRegion.rqgeom` operating on `node` (if it were a WRegion).

Synopsis: `void WSplit.transpose(WSplit node)`
Description: Transpose contents of `node`.

### 6.2.2 WSplitInner functions

Synopsis: `WSplit WSplitInner.current(WSplitInner node)`
Description: Returns the most previously active child node of `split`.

### 6.2.3 WSplitRegion functions

Synopsis: `WRegion WSplitRegion.reg(WSplitRegion node)`
Description: Returns the region contained in `node`.

### 6.2.4 WSplitSplit functions

Synopsis: `WSplit WSplitSplit.br(WSplitSplit split)`
Description: Returns the bottom or right child node of `split` depending on the direction of the split.

Synopsis: `string WSplitSplit.dir(WSplitSplit split)`
Description: Returns the direction of `split`; either '`vertical`' or '`horizontal`'.

Synopsis: `void WSplitSplit.flip(WSplitSplit split)`
Description: Flip contents of `split`.

Synopsis: `WSplit WSplitSplit.tl(WSplitSplit split)`
Description: Returns the top or left child node of `split` depending on the direction of the split.

### 6.2.5 WTiling functions

Synopsis: `bool WTiling.flip_at(WTiling ws, WRegion reg)`
Description: Flip `ws` at `reg` or root if nil.

Synopsis: `bool WTiling.transpose_at(WTiling ws, WRegion reg)`
Description: Transpose `ws` at `reg` or root if nil.

Synopsis: `WRegion WTiling.farthest(WTiling ws, string dirstr, bool any)`
Description: Return the most previously active region on `ws` with no other regions next to it in direction `dirstr` ('`left`', '`right`', '`up`', or '`down`'). If `any` is not set, the status display is not considered.

Synopsis: `bool WTiling.managed_i(WTiling ws, function iterfn)`
Description: Iterate over managed regions of `ws` until `iterfn` returns `false`. The function itself returns `true` if it reaches the end of list without this happening.

Synopsis: `WRegion WTiling.nextto(WTiling ws, WRegion reg, string dirstr, bool any)`
Description: Return the most previously active region next to `reg` in direction `dirstr` ('`left`', '`right`', '`up`', or '`down`'). The region `reg` must be managed by `ws`. If `any` is not set, the status display is not considered.

Synopsis: `WSplitRegion WTiling.node_of(WTiling ws, WRegion reg)`
Description: For region `reg` managed by `ws` return the WSplit a leaf of which `reg` is.

Synopsis: `bool WTiling.set_floating_at(WTiling ws, WRegion reg, string how, string dirstr)`
Description: Toggle floating of the sides of a split containin `reg` as indicated by the parameters `how` ('`set`', '`unset`', or '`toggle`') and `dirstr` ('`left`', '`right`', '`up`', or '`down`'). The new status is returned (and `false` also on error).

Synopsis: `WSplitSplit WTiling.set_floating(WTiling ws, WSplitSplit split, string how)`
Description: Toggle floating of a split's sides at `split` as indicated by the parameter `how` ('`set`', '`unset`', or '`toggle`'). A split of the appropriate is returned, if there was a change.

Synopsis: `WFrame WTiling.split(WTiling ws, WSplit node, string dirstr)`
Description: Create a new frame on `ws` '`above`', '`below`' '`left`' of, or '`right`' of `node` as indicated by `dirstr`. If `dirstr` is prefixed with '`floating:`' a floating split is created.

Synopsis: `WFrame WTiling.split_at(WTiling ws, WFrame frame, string dirstr, bool attach_current)`
Description: Split `frame` creating a new frame to direction `dirstr` (one of '`left`', '`right`', '`top`' or '`bottom`') of `frame`. If `attach_current` is set, the region currently displayed in `frame`, if any, is moved to thenew frame. If `dirstr` is prefixed with '`floating:`', a floating split is created.

Synopsis: `WFrame WTiling.split_top(WTiling ws, string dirstr)`
Description: Same as `WTiling.split` at the root of the split tree.

Synopsis: `WSplit WTiling.split_tree(WTiling ws)`
Description: Returns the root of the split tree.

Synopsis: `void WTiling.unsplit_at(WTiling ws, WRegion reg)`
Description: Try to relocate regions managed by `reg` to another frame and, if possible, destroy it.

## 6.3 Functions defined in *mod_query*

Synopsis: `mod_query.defcmd(cmd, fn)`
Description: Define a command override for the `query_exec` query.

Synopsis: `mod_query.message(mplex, str)`
Description: Display a message in `mplex`.

Synopsis: `table mod_query.get()`
Description: Get module configuration. For more information see `mod_query.set`.

Synopsis: `void mod_query.history_clear()`
Description: Clear line editor history.

Synopsis: `string mod_query.history_get(integer n)`
Description: Get entry at index `n` in line editor history, 0 being the latest.

Synopsis: `bool mod_query.history_push(string str)`
Description: Push an entry into line editor history.

Synopsis: `integer mod_query.history_search(string s, integer from, bool bwd, bool exact)`
Description: Try to find matching history entry. Returns -1 if none was found. The parameter `from` specifies where to start searching from, and `bwd` causes backward search from that point. If `exact` is not set, `s` only required to be a prefix of the match.

Synopsis: `table mod_query.history_table()`
Description: Return table of history entries.

Synopsis: `void mod_query.set(table tab)`
Description: Set module configuration. The following are supported:

| Field | Description |
|---|---|
| `autoshowcompl` | (boolean) Is auto-show-completions enabled? (default: true). |
| `autoshowcompl_delay` | (integer) auto-show-completions delay in milliseconds (default: 250). |
| `caseicompl` | (boolean) Turn some completions case-insensitive (default: false). |

Synopsis: `mod_query.popen_completions(cp, cmd, fn, reshnd)`

Description: This function can be used to read completions from an external source. The parameter `cp` is the completion proxy to be used, and the string `cmd` the shell command to be executed. To its stdout, the command should on the first line write the `common_beg` parameter of `WComplProxy.set_completions` (which `fn` maybe used to override) and a single actual completion on each of the successive lines. The function `reshnd` may be used to override a result table building routine.

Synopsis: `mod_query.query(mplex, prompt, initvalue, handler, completor, context)`

Description: Low-level query routine. `mplex` is the WMPlex to display the query in, `prompt` the prompt string, and `initvalue` the initial contents of the query box. `handler` is a function that receives (`mplex`, result string) as parameter when the query has been succesfully completed, `completor` the completor routine which receives a (`cp`, `str`, `point`) as parameters. The parameter `str` is the string to be completed and `point` cursor's location within it. Completions should be eventually, possibly asynchronously, set with `WComplProxy.set_completions` on `cp`.

Synopsis: `mod_query.query_attachclient(mplex)`

Description: This query asks for the name of a client window and attaches it to the frame the query was opened in. It uses the completion function `ioncore.complete_clientwin`.

Synopsis: `mod_query.query_editfile(mplex, script, prompt)`

Description: Asks for a file to be edited. This script uses `run-mailcap --mode=edit` by default, but you may provide an alternative script to use. The default prompt is "Edit file:" (translated).

Synopsis: `mod_query.query_exec(mplex)`

Description: This function asks for a command to execute with */bin/sh*. If the command is prefixed with a colon (':'), the command will be run in an XTerm (or other terminal emulator) using the script *ion-runinxterm*. Two colons ('::') will ask you to press enter after the command has finished.

Synopsis: `mod_query.query_gotoclient(mplex)`

Description: This query asks for the name of a client window and switches focus to the one entered. It uses the completion function `ioncore.complete_clientwin`.

Synopsis: `mod_query.query_lua(mplex)`

Description: This query asks for Lua code to execute. It sets the variable '`_`' in the local environment of the string to point to the mplex where the query was created. It also sets the table `arg` in the local environment to `{_, _:current()}`.

Synopsis: `mod_query.query_man(mplex, prog)`
Description: This query asks for a manual page to display. By default it runs the `man` command in an `xterm` using `ion-runinxterm`, but it is possible to pass another program as the `prog` argument.

Synopsis: `mod_query.query_menu(mplex, sub, themenu, prompt)`
Description: This query can be used to create a query of a defined menu.

Synopsis: `mod_query.query_renameframe(frame)`
Description: This function asks for a name new for the frame where the query was created.

Synopsis: `mod_query.query_renameworkspace(mplex, ws)`
Description: This function asks for a name new for the workspace `ws`, or the one on which `mplex` resides, if it is not set. If `mplex` is not set, one is looked for.

Synopsis: `mod_query.query_restart(mplex)`
Description: This query asks whether the user wants restart Ioncore. If the answer is 'y', 'Y' or 'yes', so will happen.

Synopsis: `mod_query.query_runfile(mplex, script, prompt)`
Description: Asks for a file to be viewed. This script uses `run-mailcap --action=view` by default, but you may provide an alternative script to use. The default prompt is "View file:" (translated).

Synopsis: `mod_query.query_shutdown(mplex)`
Description: This query asks whether the user wants to exit Ion (no session manager) or close the session (running under a session manager that supports such requests). If the answer is 'y', 'Y' or 'yes', so will happen.

Synopsis: `mod_query.query_ssh(mplex, ssh)`
Description: This query asks for a host to connect to with SSH. Hosts to tab-complete are read from *~/.ssh/known_hosts*.

Synopsis: `mod_query.query_workspace(mplex)`
Description: This query asks for the name of a workspace. If a workspace (an object inheriting WGroupWS) with such a name exists, it will be switched to. Otherwise a new workspace with the entered name will be created and the user will be queried for the type of the workspace.

Synopsis: `mod_query.query_yesno(mplex, prompt, handler)`
Description: This function query will display a query with prompt `prompt` in `mplex` and if the user answers affirmately, call `handler` with `mplex` as parameter.

Synopsis: `mod_query.show_about_ion(mplex)`
Description: Display an "About Ion" message in `mplex`.

Synopsis: `mod_query.show_tree(mplex, reg, max_depth)`
Description: Show information about a region tree

Synopsis: `mod_query.warn(mplex, str)`
Description: Display an error message box in the multiplexer `mplex`.

### 6.3.1 WComplProxy functions

Synopsis: `bool WComplProxy.set_completions(WComplProxy proxy, table compls)`
Description: Set completion list of the WEdln that `proxy` refers to to `compls`, if it is still waiting for this completion run. The numerical indexes of `compls` list the found completions. If the entry `common_beg` (`common_end`) exists, it gives an extra common prefix (suffix) of all found completions.

### 6.3.2 WEdln functions

Synopsis: `void WEdln.back(WEdln wedln)`
Description: Move backward one character.

Synopsis: `void WEdln.backspace(WEdln wedln)`
Description: Delete previous character.

Synopsis: `void WEdln.bkill_word(WEdln wedln)`
Description: Starting from the previous characters, delete possible whitespace and preceding alphanumeric characters until previous non-alphanumeric character.

Synopsis: `void WEdln.bol(WEdln wedln)`
Description: Go to the beginning of line.

Synopsis: `void WEdln.bskip_word(WEdln wedln)`
Description: Go to to beginning of current sequence of alphanumeric characters followed by whitespace.

Synopsis: `void WEdln.clear_mark(WEdln wedln)`
Description: Clear *mark*.

Synopsis: `void WEdln.complete(WEdln wedln, string cycle, string mode)`
Description: Call completion handler with the text between the beginning of line and current cursor position, or select next/previous completion from list if in auto-show-completions mode and `cycle` is set to '`next`' or '`prev`', respectively. The `mode` may be '`history`' or '`normal`'. If it is not set, the previous mode is used. Normally next entry is not cycled to despite the setting of `cycle` if mode switch occurs. To override this, use '`next-always`' and '`prev-always`' for `cycle`.

Synopsis: `string WEdln.contents(WEdln wedln)`
Description: Get line editor contents.

Synopsis: `string WEdln.context(WEdln wedln)`
Description: Get history context for `wedln`.

Synopsis: `void WEdln.copy(WEdln wedln)`
Description: Copy text between *mark* and current cursor position to clipboard.

Synopsis: `void WEdln.cut(WEdln wedln)`
Description: Copy text between *mark* and current cursor position to clipboard and then delete that sequence.

Synopsis: `void WEdln.delete(WEdln wedln)`
Description: Delete current character.

Synopsis: `void WEdln.eol(WEdln wedln)`
Description: Go to the end of line.

Synopsis: `void WEdln.finish(WEdln wedln)`
Description: Close `wedln` and call any handlers.

Synopsis: `void WEdln.forward(WEdln wedln)`
Description: Move forward one character.

Synopsis: `void WEdln.history_next(WEdln wedln, bool match)`
Description: Replace line editor contents with next entry in history if one exists. If `match` is `true`, the initial part of the history entry must match the current line from beginning to point.

Synopsis: `void WEdln.history_prev(WEdln wedln, bool match)`
Description: Replace line editor contents with previous in history if one exists. If `match` is `true`, the initial part of the history entry must match the current line from beginning to point.

Synopsis: `void WEdln.insstr(WEdln wedln, string str)`
Description: Input `str` in wedln at current editing point.

Synopsis: `bool WEdln.is_histcompl(WEdln wedln)`
Description: Get history completion mode.

Synopsis: `void WEdln.kill_line(WEdln wedln)`
Description: Delete the whole line.

Synopsis: `void WEdln.kill_to_bol(WEdln wedln)`
Description: Delete all characters from previous to beginning of line.

Synopsis: `void WEdln.kill_to_eol(WEdln wedln)`
Description: Delete all characters from current to end of line.

Synopsis: `void WEdln.kill_word(WEdln wedln)`
Description: Starting from the current point, delete possible whitespace and following alphanumeric characters until next non-alphanumeric character.

Synopsis: `integer WEdln.mark(WEdln wedln)`
Description: Get current mark (start of selection) for `wedln`. Return value of -1 indicates that there is no mark, and 0 is the beginning of the line.

Synopsis: `bool WEdln.next_completion(WEdln wedln)`
Description: Select next completion.

Synopsis: `void WEdln.paste(WEdln wedln)`
Description: Request selection from application holding such.
Note that this function is asynchronous; the selection will not actually be inserted before Ion receives it. This will be no earlier than Ion return to its main loop.

Synopsis: `integer WEdln.point(WEdln wedln)`
Description: Get current editing point. Beginning of the edited line is point 0.

Synopsis: `bool WEdln.prev_completion(WEdln wedln)`
Description: Select previous completion.

Synopsis: `void WEdln.set_context(WEdln wedln, string context)`
Description: Set history context for `wedln`.

Synopsis: `void WEdln.set_mark(WEdln wedln)`
Description: Set *mark* to current cursor position.

Synopsis: `void WEdln.skip_word(WEdln wedln)`
Description: Go to to end of current sequence of whitespace followed by alphanumeric characters..

Synopsis: `void WEdln.transpose_chars(WEdln wedln)`
Description: Transpose characters.

Synopsis: `void WEdln.transpose_words(WEdln wedln)`
Description: Transpose words.

### 6.3.3 WInput functions

Synopsis: `void WInput.cancel(WInput input)`
Description: Close input not calling any possible finish handlers.

Synopsis: `void WInput.scrolldown(WInput input)`
Description: Scroll input `input` text contents down.

Synopsis: `void WInput.scrollup(WInput input)`
Description: Scroll input `input` text contents up.

## 6.4 Functions defined in *mod_menu*

Synopsis: `mod_menu.grabmenu(mplex, sub, menu_or_name, param)`
Description: This function is similar to `mod_menu.menu`, but input is grabbed and the key used to active the menu can be used to cycle through menu entries.

Synopsis: `mod_menu.menu(mplex, sub, menu_or_name, param)`
Description: Display a menu in the lower-left corner of `mplex`. The variable `menu_or_name` is either the name of a menu defined with `mod_menu.defmenu` or directly a table similar to ones passesd to this function. When this function is called from a binding handler, `sub` should be set to the second argument of to the binding handler (`_sub`) so that the menu handler will get the same parameters as the binding handler. Extra options can be passed in the table `param`. The initial entry can be specified as the field `initial` as an integer starting from 1. Menus can be made to use a bigger style by setting the field `big` to `true`.

Synopsis: `table mod_menu.get()`
Description: Get module basic settings. For details, see `mod_menu.set`.

Synopsis: `void mod_menu.set(table tab)`
Description: Set module basic settings. The parameter table may contain the following fields:

| Field | Description |
|---|---|
| `scroll_amount` | Number of pixels to scroll at a time pointer-controlled menus when one extends beyond a border of the screen and the pointer touches that border. |
| `scroll_delay` | Time between such scrolling events in milliseconds. |

Synopsis: `mod_menu.pmenu(win, sub, menu_or_name)`
Description: This function displays a drop-down menu and should only be called from a mouse press handler. The parameters are similar to those of `mod_menu.menu`.

### 6.4.1 WMenu functions

Synopsis: `void WMenu.cancel(WMenu menu)`
Description: Close `menu` not calling any possible finish handlers.

Synopsis: `void WMenu.finish(WMenu menu)`
Description: If selected entry is a submenu, display that. Otherwise destroy the menu and call handler for selected entry.

Synopsis: `void WMenu.select_next(WMenu menu)`
Description: Select next entry in menu.

Synopsis: `void WMenu.select_nth(WMenu menu, integer n)`
Description: Select `n`:th entry in menu.

Synopsis: `void WMenu.select_prev(WMenu menu)`
Description: Select previous entry in menu.

Synopsis: `void WMenu.typeahead_clear(WMenu menu)`
Description: Clear typeahead buffer.

## 6.5 Functions defined in *mod_dock*

Synopsis: `void mod_dock.set_floating_shown_on(WMPlex mplex, string how)`
Description: Toggle floating docks on `mplex`.

### 6.5.1 WDock functions

Synopsis: `bool WDock.attach(WDock dock, WRegion reg)`
Description: Attach `reg` to `dock`.

Synopsis: `table WDock.get(WDock dock)`
Description: Get `dock`'s configuration table. See `WDock.set` for a description of the table.

Synopsis: `void WDock.resize(WDock dock)`
Description: Resizes and refreshes `dock`.

Synopsis: `void WDock.set(WDock dock, table conftab)`
Description: Configure `dock`. `conftab` is a table of key/value pairs:

| Key | Values | Description |
|---|---|---|
| `name` | string | Name of dock |
| `pos` | string in $\{t, m, b\} \times \{t, c, b\}$ | Dock position. Can only be used in floating mode. |
| `grow` | up/down/left/right | Growth direction where new dockapps are added. Also sets orientation for dock when working as WMPlex status display (see `WMPlex.set_stdisp`). |
| `is_auto` | bool | Should `dock` automatically manage new dockapps? |

Any parameters not explicitly set in `conftab` will be left unchanged.

## 6.6 Functions defined in *mod_sp*

Synopsis: `bool mod_sp.set_shown(WFrame sp, string how)`
Description: Toggle displayed status of `sp`. The parameter `how` is one of '`set`', '`unset`', or '`toggle`'. The resulting status is returned.

Synopsis: `bool mod_sp.set_shown_on(WMPlex mplex, string how)`
Description: Change displayed status of some scratchpad on `mplex` if one is found. The parameter `how` is one of '`set`', '`unset`', or '`toggle`'. The resulting status is returned.

## 6.7 Functions defined in *mod_statusbar*

Synopsis: `mod_statusbar.create(param)`
Description: Create a statusbar. The possible parameters in the table `param` are:

| Variable | Type | Description |
|---|---|---|
| `template` | string | The template; see Section 3.6. |
| `pos` | string | Position: '`tl`', '`tr`', '`bl`' or '`br`' (for the obvious combinations of top/left/bottom/right). |
| `screen` | integer | Screen number to create the statusbar on. |
| `fullsize` | boolean | If set, the statusbar will waste space instead of adapting to layout. |
| `systray` | boolaen | Swallow (KDE protocol) systray icons. |

Synopsis: `mod_statusbar.inform(name, value)`
Description: Inform of a value.

Synopsis: `mod_statusbar.launch_statusd(cfg)`
Description: Load modules and launch *ion-statusd* with configuration table `cfg`. The options for each *ion-statusd* monitor script should be contained in the corresponding sub-table of `cfg`.

Synopsis: `table mod_statusbar.statusbars()`
Description: Returns a list of all statusbars.

Synopsis: `mod_statusbar.update(update_templates)`
Description: Update statusbar contents. To be called after series of `mod_statusbar.inform` calls.

### 6.7.1 WStatusBar functions

Synopsis: `table WStatusBar.get_template_table(WStatusBar sb)`
Description: Get statusbar template as table.

Synopsis: `bool WStatusBar.is_systray(WStatusBar sb)`
Description: Is `sb` used as a systray?

Synopsis: `bool WStatusBar.set_systray(WStatusBar sb, string how)`
Description: Enable or disable use of `sb` as systray. The parameter `how` can be one of '`set`', '`unset`', or '`toggle`'. Resulting state is returned.

Synopsis: `void WStatusBar.set_template(WStatusBar sb, string tmpl)`
Description: Set statusbar template.

Synopsis: `void WStatusBar.set_template_table(WStatusBar sb, table t)`
Description: Set statusbar template as table.

Synopsis: `void WStatusBar.update(WStatusBar sb, table t)`
Description: Set statusbar template.

## 6.8 Functions defined in *de*

Synopsis: `bool de.defstyle(string name, table tab)`
Description: Define a style.

Synopsis: `bool de.defstyle_rootwin(WRootWin rootwin, string name, table tab)`
Description: Define a style for the root window `rootwin`.

Synopsis: `void de.reset()`
Description: Clear all styles from drawing engine memory.

Synopsis: `table de.substyle(string pattern, table tab)`
Description: Define a substyle.

## 6.9 Hooks

Hook name: `clientwin_do_manage_alt`
Parameters: `(WClientWin, table)`
Description: Called when we want to manage a new client window. The table argument contains the following fields:

| Field | Type | Description |
|---|---|---|
| `switchto` | bool | Do we want to switch to the client window. |
| `jumpto` | bool | Do we want to jump to the client window. |
| `userpos` | bool | Geometry set by user. |
| `dockapp` | bool | Client window is a dock-app. |
| `maprq` | bool | Map request (and not initialisation scan). |
| `gravity` | number | Window gravity. |
| `geom` | table | Requested geometry; `x`, `y`, `w`, `h`. |
| `tfor` | WClientWin | Transient for window. |

This hook is not called in protected mode and can be used for arbitrary placement policies (deciding in which workspace a new WClientWin should go). In this case, you can call
`reg:attach(cwin)`
where `reg` is the region where the window should go, and `cwin` is the first argument of the function added to the hook.

Hook name: `clientwin_mapped_hook`
Parameters: `WClientWin`
Description: Called when we have started to manage a client window.

Hook name: `clientwin_property_change_hook`
Parameters: `(WClientWin, integer)`
Description: Called when the property identified by the parameter atom id (integer) has changed on a client window.

Hook name: `clientwin_unmapped_hook`
Parameters: `number`
Description: Called when we no longer manage a client window. The parameter is the X ID of the window; see `WClientWin.xid`.

Hook name: `frame_managed_changed_hook`
Parameters: `table`
Description: Called when there are changes in the objects managed by a frame or their order. The table parameter has the following fields:

| Field | Type | Description |
|---|---|---|
| `reg` | WFrame | The frame in question |
| `mode` | string | '`switchonly`', '`reorder`', '`add`' or '`remove`' |
| `sw` | bool | Switch occurred |
| `sub` | WRegion | The managed region (primarily) affected |

Hook name: `ioncore_sigchld_hook`
Parameters: `integer`
Description: Called when a child process has exited. The parameter is the PID of the process.

Hook name: `ioncore_deinit_hook`
Parameters: `()`
Description: Called when Ion is deinitialising and about to quit.

Hook name: `ioncore_post_layout_setup_hook`
Parameters: `()`
Description: Called when Ion has done all initialisation and is almost ready to enter the main-loop, except no windows are yet being managed.

Hook name: `ioncore_snapshot_hook`
Parameters: `()`
Description: Called to signal scripts and modules to save their state (if any).

Hook name: `ioncore_submap_ungrab_hook`
Parameters: `()`
Description: This hook is used to signal whenever Ion leaves the submap grab mode.

Hook name: `tiling_placement_alt`
Parameters: `table`
Description: Called when a client window is about to be managed by a WTiling to allow for alternative placement policies. The table has the following fields:

| Field | Type | Description |
|---|---|---|
| `tiling` | WTiling | The tiling |
| `reg` | WRegion | The region (always a WClientWin at the moment) to be placed |
| `mp` | table | This table contains the same fields as the parameter of `clientwin_do_manage_alt` |
| `res_frame` | WFrame | A successful handler should return the target frame here. |

This hook is just for placing within a given workspace after the workspace has been decided by the default workspace selection policy. It is called in protected mode. For arbitrary placement policies, `clientwin_do_manage_alt` should be used; it isn't called in protected mode,

Hook name: `region_do_warp_alt`
Parameters: `WRegion`
Description: This alt-hook exist to allow for alternative pointer warping implementations.

Hook name: `screen_managed_changed_hook`
Parameters: `table`
Description: Called when there are changes in the objects managed by a screen or their order. The table parameter is similar to that of `frame_managed_changed_hook`.

Hook name: `region_notify_hook`
Parameters: `(WRegion, string)`
Description: Signalled when something (minor) has changed in relation to the first parameter region. The string argument gives the change:

| String | Description |
|---|---|
| `deinit` | The region is about to be deinitialised. |
| `activated` | The region has received focus. |
| `inactivated` | The region has lost focus. |
| `activity` | There's been activity in the region itself. |
| `sub_activity` | There's been activity in some sub-region. |
| `name` | The name of the region has changed. |
| `unset_manager` | The region no longer has a manager. |
| `set_manager` | The region now has a manager. |
| `tag` | Tagging state has changed. |
| `pseudoactivated` | The region has become pseudo-active (see below). |
| `pseudoinactivated` | The region is no longer pseudo-active. |

A region is pseudo-active, when a) it is itself not active (does not not have the focus, and may not even have a window that could have it), but b) some region managed by it is active.

## 6.10 Miscellaneous

### 6.10.1 Size policies

Some functions accept a `sizepolicy` parameter. The possible values are:

'`default`', '`full`', '`full_bounds`', '`free`', '`free_glue`', '`northwest`', '`north`', '`northeast`', '`west`', '`center`', '`east`', '`southwest`', '`south`', '`southeast`', '`stretch_top`', '`stretch_bottom`', '`stretch_left`', '`stretch_right`', '`free_glue_northwest`', '`free_glue_north`', '`free_glue_northeast`', '`free_glue_west`', '`free_glue_center`', '`free_glue_east`', '`free_glue_southwest`', '`free_glue_south`', and '`free_glue_southeast`'.

The "free" policies allow the managed object to be moved around, whereas the other versions do not. The "glue" policies glue the object to some border, while allowing it to be moved away from it by user action, but not automatically. The "stretch" policies stretch the object along the given border, while the coordinate-based policies simply place the object along that border.

# Appendix A

# The GNU General Public License

Version 2, June 1991



59 Temple Place - Suite 330, Boston, MA 02111-1307, USA



## Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".
   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.
1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.
   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.
2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

(b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

(c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance

of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

## End of Terms and Conditions

### Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> one line to give the program's name and a brief idea of what it does.
> Copyright (C) yyyy name of author
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) yyyy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
> type 'show w'.
> This is free software, and you are welcome to redistribute it under certain
> conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program
> 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> signature of Ty Coon, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

# Appendix B

## Full class hierarchy visible to Lua-side

```
Obj
 |-->WHook
 |-->WTimer
 |-->WMoveresMode
 |-->WRegion
 |    |-->WClientWin
 |    |-->WWindow
 |    |    |-->WMPlex
 |    |    |    |-->WFrame
 |    |    |    |-->WScreen
 |    |    |         |->WRootWin
 |    |    |-->WInfoWin
 |    |    |    |-->WStatusBar (mod_statusbar)
 |    |    |-->WMenu (mod_menu)
 |    |    |-->WInput (mod_query)
 |    |         |-->WEdln (mod_query)
 |    |         |-->WMessage (mod_query)
 |    |-->WGroup
 |    |    |-->WGroupWS
 |    |    |-->WGroupCW
 |    |-->WTiling (mod_tiling)
 |-->WSplit (mod_tiling)
      |-->WSplitInner (mod_tiling)
      |    |-->WSplitSplit (mod_tiling)
      |         |-->WSplitFloat (mod_tiling)
      |-->WSplitRegion (mod_tiling)
           |-->WSplitST (mod_tiling)
```

## List of functions

# Index

# Bibliography

[1] The Ion 3 scripts repository, `http://iki.fi/tuomov/repos/ion-scripts-3/`.