

---

# **Workgroups2 Documentation**

***Release 1.2***

**Sergey Pashinin**

August 31, 2014



<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Usage</b>	<b>5</b>
2.1	Basic commands . . . . .	5
2.2	Settings . . . . .	5
<b>3</b>	<b>How does it work?</b>	<b>7</b>
3.1	Serialization / Deserialization of objects . . . . .	7
3.2	Loading a session file . . . . .	8
3.3	Saving session . . . . .	8
3.4	Switching workgroups . . . . .	8
<b>4</b>	<b>Data structures</b>	<b>9</b>
4.1	General info . . . . .	9
4.2	Session . . . . .	10
4.3	Workgroup . . . . .	10
4.4	Wconfig . . . . .	10
4.5	Wtree . . . . .	11
4.6	Win . . . . .	11
4.7	Buffer . . . . .	11
4.8	Parameters . . . . .	11
<b>5</b>	<b>Tests</b>	<b>13</b>
5.1	Serialization tests . . . . .	13
<b>6</b>	<b>Problems</b>	<b>15</b>
6.1	Buffer was not restored . . . . .	15
<b>7</b>	<b>Contribute</b>	<b>17</b>
7.1	Start using the git repo . . . . .	17
7.2	Modify something . . . . .	17
<b>8</b>	<b>Indices and tables</b>	<b>19</b>



Like it: *The original workgroups was already wonderful, the best window configuration manager for Emacs. The new maintainer has lifted the package from merely awesome to wild ecstasy.*

Contents:



---

## Installation

---

Very simple with recent Emacs. Make sure you have these lines:

```
(require 'package)
(add-to-list 'package-archives
  ' ("melpa" . "http://melpa.milkbox.net/packages/") t)
```

before

```
(package-initialize)
```

Then type M-x list-packages

<u>wordsmith-mode</u>	20140203.427	available	melpa	Syntax analysts and NLP text-processors
<u>worf</u>	20140424...	available	melpa	A warrior does not press so many keys
<u>workgroups</u>	20110726.941	available	melpa	workgroups for windows (for Emacs)
I <u>workgroups2</u>	20140830.707	available	melpa	New workspaces for Emacs
<u>world-time-mode</u>	20140627.107	available	melpa	show whole days of world-time diffs
<u>wpuzzle</u>	1.1	available	gnu	find as many word in a given time
<u>writgood-mode</u>	20140605.734	available	melpa	Polish up poor writing on the fly

mark `workgroups2` with `i` and install with `x`.

Then *Configure and activate workgroups-mode*.





---

## Usage

---

The whole config should look like this:

```
(require 'workgroups2)
;; Change some settings
(workgroups-mode 1)      ; put this one at the bottom of .emacs (init.el)
```

Now you activated `workgroups-mode`.

## 2.1 Basic commands

Most commands are bound to both `<prefix> <key>` and `<prefix> C-<key>`.

By default prefix is: `C-c z` (To change it - see settings below)

```
<prefix> <key>
<prefix> c      ; create workgroup
<prefix> A      ; rename workgroup
<prefix> k      ; kill workgroup
<prefix> v      ; switch to workgroup
<prefix> C-s    ; save session
<prefix> C-f    ; load session
```

## 2.2 Settings

```
(require 'workgroups2)
;; Your settings here

;; (setq wg-session-load-on-start t)      ; default: (not (daemonp))

;; Change prefix key (before activating WG)
(setq wg-prefix-key (kbd "C-c z"))

;; Change workgroups session file
(setq wg-session-file "~/ .emacs.d/.emacs_workgroups")

;; Set your own keyboard shortcuts to reload/save/switch WGs:
;; "s" == "Super" or "Win"-key, "S" == Shift, "C" == Control
(global-set-key (kbd "<pause>") 'wg-reload-session)
(global-set-key (kbd "C-S-<pause>") 'wg-save-session)
```

```
(global-set-key (kbd "s-z")      'wg-switch-to-workgroup)
(global-set-key (kbd "s-/")      'wg-switch-to-previous-workgroup)

(workgroups-mode 1)  ; put this one at the bottom of .emacs
```

## 2.2.1 More settings

You can use `M-x customize-group RET workgroups` to see all variables and faces to change.

```
;; What to do on Emacs exit / workgroups-mode exit?
(setq wg-emacs-exit-save-behavior      'save)      ; Options: 'save 'ask nil
(setq wg-workgroups-mode-exit-save-behavior 'save)  ; Options: 'save 'ask nil

;; Mode Line changes
;; Display workgroups in Mode Line?
(setq wg-mode-line-display-on t)        ; Default: (not (featurep 'powerline))
(setq wg-flag-modified t)               ; Display modified flags as well
(setq wg-mode-line-decor-left-brace "["
      wg-mode-line-decor-right-brace "]" ; how to surround it
      wg-mode-line-decor-divider ":")
```

## 2.2.2 Hooks

Hooks' names can tell when they are executed

```
workgroups-mode-hook          ; when 'workgroups-mode' is turned on
workgroups-mode-exit-hook     ; 'workgroups-mode' is turned off
wg-before-switch-to-workgroup-hook
wg-after-switch-to-workgroup-hook
```

---

## How does it work?

---



---

**Note:** The most important part to understand is *Data structures*. After that it's easy to write code in other parts.

---

### 3.1 Serialization / Deserialization of objects

In Emacs we have many types of objects like:

- `#<buffer tests.el>`
- `#<marker at 3427 in tests.el>`
- simple "string"
- integers 123
- ... and other

And we have to represent them as text to save. This is done using `wg-pickel` and functions defined in this var:

```
(defvar wg-pickel-object-serializers
  '( (integer      . identity)
      (float       . identity)
      (string      . identity)
      (symbol      . wg-pickel-symbol-serializer)
      (cons        . wg-pickel-cons-serializer)
      (vector      . wg-pickel-vector-serializer)
      (hash-table  . wg-pickel-hash-table-serializer)
      (buffer      . wg-pickel-buffer-serializer)
      (marker      . wg-pickel-marker-serializer))
  "Alist mapping types to object serialization functions.")
```

So when you meet an object that cannot be represented as text - you:

1. Add it's type in this variable
2. Write mentioned "serializer" function itself

For example for "buffer" objects:

```
(defun wg-pickel-buffer-serializer (buffer)
  "Return BUFFER's UID in workgroups buffer list."
  (list 'b (wg-add-buffer-to-buf-list buffer)))
```

'b' - is just a marker that will tell to run `wg-pickel-deserialize-buffer` when restoring a buffer.

Last element is buffer UID and it is enough to restore the buffer with `(wg-restore-buffer (wg-find-buf-by-uid uid))`

## 3.2 Loading a session file

It is done in `wg-open-session`. First you read a *Session object* from file in this line:

```
(let ((session (read (f-read-text filename))))  
  ...
```

Then you just switch to 1 of the saved workgroups in this object according to settings.

## 3.3 Saving session

Writing objects to file is done in... (function stack):

**wg-write-sexp-to-file**

**wg-pickel-all-session-parameters**

**wg-pickel-workgroup-parameters** `wg-pickel` ← main function

So the main function to transform Lisp objects to strings is `wg-pickel`.

## 3.4 Switching workgroups

---

## Data structures

---

Let's look at ~/.workgroups file:

```
[cl-struct-wg-session "0G3A08BU1E35GEA0-18GPMY" ...
  ([cl-struct-wg-workgroup "0G3A08D8APKR11T4-1C1G10" "Tasks" ...
    [cl-struct-wg-wconfig "0GGI0JY4B3HD0WEO-86RSR3" ...
      [cl-struct-wg-wtree ...
        ([cl-struct-wg-win ...
          [cl-struct-wg-win ...
```

### 4.1 General info

All these structs (better to say functions to work with these objects) are created with `wg-defstruct` macro. For example for:

```
(wg-defstruct wg session
  (uid (wg-generate-uid))
  (field-2)
  ...
```

`wg-defstruct` creates functions like `wg-make-session`, `wg-copy-session` and `wg-session-...`, (to manipulate structures). Then you will have `(wg-session-field-2 obj)` and other defined fields to read properties of this object.

To set values `(setf ...) function` is used.

Example for current session object:

```
;; Read
(wg-session-file-name (wg-current-session)) ; Get a filename of current session
(wg-workgroup-parameters (wg-current-workgroup)) ; Get workgroup parameters

;; Write (used just before saving session to file)
(setf (wg-session-file-name (wg-current-session)) filename) ; Set session filename
(setf (wg-session-version (wg-current-session)) wg-version) ; Write workgroups version
```

**Warning:** Changing these defstructs themselves may break everyone's session files. That's why many of them have *Parameters* field. This one is exactly for extending saved information.

### 4.1.1 How to work with these structures?

Ok, we define a session structure, and you can get the value of it with (wg-current-session)

wg-defstruct creates functions like wg-session-..., wg-make-session (to manipulate structures). So if you have (wg-defstruct wg session ...) - then you have wg-session-file-name and other defined fields.

## 4.2 Session

The session object is the top level “class” that has workgroups in it.

```
(wg-defstruct wg session
  (uid (wg-generate-uid))
  (name)
  (modified)
  (parameters)
  (file-name)
  (version wg-version)
  (workgroup-list)
  (buf-list))
```

---

**Note:** List of buffers is a common pool for all workgroups. When you open a file (doesn't matter in which workgroup) - the corresponding *Buffer* object will be added in wg-session-buf-list

---

## 4.3 Workgroup

workgroups contain frame states (that includes window configuration)

```
(wg-defstruct wg workgroup
  (uid (wg-generate-uid))
  (name)
  (modified)
  (parameters)
  (base-wconfig)
  (selected-frame-wconfig)
  (saved-wconfigs)
  (strong-buf-uids)
  (weak-buf-uids))
```

## 4.4 Wconfig

```
(wg-defstruct wg wconfig
  (uid (wg-generate-uid))
  (name)
  (parameters)
  (left)
  (top)
  (width)
  (height)
  (vertical-scroll-bars))
```

```
(scroll-bar-width)
(wtree))
```

What's the difference between wconfig and wtree? Well a workgroup can have several wconfigs (buffer layouts). But to keep it simple let's say each workgroup has only 1 wconfig.

wconfig = wtree + additional parameters

## 4.5 Wtree

```
(wg-defstruct wg wtree
  (uid)
  (dir)
  (edges)
  (wlist))
```

## 4.6 Win

```
(wg-defstruct wg win
  (uid)
  (parameters)
  (edges)
  (point)
  (start)
  (hscroll)
  (dedicated)
  (selected)
  (minibuffer-scroll)
  (buf-uid))
```

## 4.7 Buffer

## 4.8 Parameters

Changing main structures may lead to huge problems in compatibility. That's why there are parameters for *Session*, *Workgroup*, *Wconfig* and *Win* objects. They allow you to save your custom data.

For example to set (key, value) pair for current workgroup:

```
;; Write (key, value)
(wg-set-workgroup-parameter
 'ecb ; name
 (and (boundp 'ecb-minor-mode) ecb-minor-mode)) ; value
```

Usually these functions are called like:

```
wg-<object>-parameter ; to read
wg-set-<object>-parameter ; to set
wg-remove-<object>-parameter ; to remove parameter
```

For session: wg-session-parameter, wg-set-session-parameter, wg-remove-session-parameter For workgroup: wg-workgroup-parameter, wg-set-workgroup-parameter, wg-remove-workgroup-parameter





---

## Tests

---

Tests are cool now. To run them just use:

```
make deps
make testgui
```

Tests also run automatically on [Travis-CI](#) using the GUI version of Emacs. So you can tests any frames as on your desktop.

Tests themselves are in `tests/workgroups2-tests.el`

### 5.1 Serialization tests

If you see an error like this:

```
wg-add-buffer-to-buf-list (nil)
wg-pickel-marker-serializer(#<marker in no buffer>)
#[ (obj id) "... " [id obj result wg-pickel-object-serializer] 3] (#<marker in no buffer> 18)
maphash(#[ (obj id) "... " [id obj result wg-pickel-object-serializer] 3] #s(hash-table size 65 te
wg-pickel-serialize-objects(#s(hash-table size 65 test eq rehash-size 1.5 rehash-threshold 0.8 d
wg-pickel(((#<buffer todo-orgx.org> #<marker at 1 in todo-orgx.org> #<marker at 158366 in todo-o
...

```

then we have a problem in `wg-pickel` function. More precisely object `#<marker in no buffer>` cannot be serialized. And that was a bug.

To create a test in `workgroups2-tests.el` for such situation find this:

```
(defmacro test-pickel (value)
  "Test 'wg-pickel' 'wg-unpickel' on VALUE."
  `(eq (wg-unpickel (wg-pickel ,value)) ,value))

(ert-deftest 110-wg-pickel ()
  (test-pickel 123)
  (test-pickel "str")
  (test-pickel 'symbol)
  (test-pickel (current-buffer)) ; #<buffer tests.el>
  (test-pickel (point-marker))   ; #<marker at 3427 in tests.el>
  (test-pickel (make-marker))    ; #<marker in no buffer>
  (test-pickel (list 'describe-variable 'help-xref-stack-item (get-buffer "*Help*"))))
)
```

And pass an object that cannot be serialized and should be checked. Then you need to fix something in `wg-pickel`, see *Serialization / Deserialization of objects*.

---

## Problems

---

**You do have problems, right?**

- Buffer was not restored
  - Restored, but not the way I want

### 6.1 Buffer was not restored

I doubt it was a simple file buffer (or [report a bug](#)).

**Warning:** You know `major-mode` you use **better than me**. So please if you ask to add support for any particular `major-mode` - write how you install, configure and run yours.

Such complex buffers are called “special buffers”. A simple way to restore them is to use `wg-support` macro:

```
(wg-support 'inferior-emacs-lisp-mode 'ielm
  `((deserialize . , (lambda (buffer vars)
                        (ielm) (get-buffer "*ielm*")))))
```

To understand how this works - see `special-buffers`

#### 6.1.1 Restored, but not the way I want

Discuss it



## 7.1 Start using the git repo

1. Remove workgroups2 package you installed from Melpa
2. Clone the repo from Github (or make a submodule in your .emacs repo)

```
cd ~/some/path
git clone https://github.com/pashinin/workgroups2.git

cd ~/.emacs.d
git submodule add git://github.com/pashinin/workgroups2.git workgroups2
```

3. Add repo's src/ directory to load-path and then use a simple (require ...)

```
(add-to-list 'load-path "~/emacs.d/workgroups2/src")
(require 'workgroups2)

;; your existing settings...
(workgroups-mode 1)
```

Then to make changes I think you need to understand *How this extension work*.

## 7.2 Modify something



---

## Indices and tables

---

- *genindex*
- *modindex*
- *search*