

org-special-block-extras

Musa Al-hassy

April 17, 2020

Contents

1	Example Use	1
2	Core Utility	2
3	Colours	4
4	Parallel	5
5	:argument: Extraction	7
6	Editor Comments	8
6.1	Examples	10
6.1.1	No optional arguments	10
6.1.2	Only declaring an :ed: —editor	10
6.1.3	Empty contents, no editor, nothing	10
6.1.4	With a :replacewith: clause	10

1 Example Use

User type something along the lines of the following.

```
#+begin_red org
/This/
    *text*
        _is_
            red!
#+end_red
```

Which generates *red* text when exported to HTML and \LaTeX , **while supporting Org markup.**

This article may be read as a PDF or as HTML or as pure Org!

(Since we're only considering HTML & \LaTeX , the Github rendition of this article may not render things as desired.)

The remaining sections are implementation matter.

<code>#+begin_blue</code>	This text is blue!	This text is black!
This text is blue!		This text is blue!
<code>#+end_blue</code>	This text is brown!	This text is brown!
<code>#+begin_brown</code>	This text is cyan!	This text is cyan!
This text is brown!	This text is darkgray!	This text is darkgray!
<code>#+end_brown</code>		This text is gray!
<code>#+begin_cyan</code>	This text is gray!	This text is green!
This text is cyan!	This text is green!	This text is lightgray!
<code>#+end_cyan</code>		This text is lime!
<code>#+begin_darkgray</code>	This text is lightgray!	This text is magenta!
This text is <u>darkgray</u> !	This text is lime!	This text is olive!
<code>#+end_darkgray</code>		This text is orange!
<code>#+begin_gray</code>	This text is magenta!	This text is pink!
This text is <u>gray</u> !	This text is olive!	This text is purple!
<code>#+end_gray</code>		This text is red!
<code>#+begin_green</code>	This text is orange!	This text is teal!
This text is green!	This text is pink!	This text is violet!
<code>#+end_green</code>		
<code>#+begin_lightgray</code>	This text is purple!	
This text is <u>lightgray</u> !	This text is red!	
<code>#+end_lightgray</code>		
<code>#+begin_lime</code>	This text is teal!	This text is yellow!

Figure 1: Write Org-markup once, generate for many backends ^ _ ^

2 Core Utility

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Core utility
```

```
(defun org-special-block-extras--advice (backend blk contents _)
  "Invoke the appropriate custom block handler, if any."
```

A given custom block BLK has a TYPE extracted from it, then we send the block CONTENTS along with the current export BACKEND to the formatting function ORG-SPECIAL-BLOCK-EXTRAS/TYPE if it is defined, otherwise, we leave the CONTENTS of the block as is.

We also support the seemingly useless blocks that have no contents at all, not even an empty new line."

```
  (let* ((type (nth 1 (nth 1 blk)))
         (handler (intern (format "org-special-block-extras--%s" type))))
    (ignore-errors (apply handler backend (or contents "") nil))))

(advice-add #'org-html-special-block :before-until
  (-partial #'org-special-block-extras--advice 'html))

(advice-add #'org-latex-special-block :before-until
  (-partial #'org-special-block-extras--advice 'latex))
```

Example:

```
#+begin_3parallel org
one

#+latex: \columnbreak
two
```

```
#+latex: \columnbreak
three
#+end_3parallel
```

Yields:

one | two | three

The screenshot shows the Emacs editor interface. The title bar reads "Emacs-x86_64-10_10 — (94 x 24)". The main window displays the source code for the example, with line numbers 243 to 255. The code is as follows:

```
243 Example:
244 <=> org
245 #+begin_3parallel org
246 one
247
248 #+latex: \columnbreak
249 two
250
251 #+latex: \columnbreak
252 three
253 #+end_3parallel|
254 <=
255
```

Below the main window, a buffer window titled "org-special-block-extras.org" is visible. It shows the rendered output of the code, including the "Example:" section and the "Yields:" section with three columns: "one", "two", and "three".

Figure 2: Displaying thoughts side-by-side ^_^

In LaTeX, an `edcomm` appears inline with the text surrounding it.

[**Bobert: Replace:** org-mode is dope, yo! **With:** Org-mode is essentially a path toward enlightenment.]

Unfortunately, in the HTML rendition, the `edcomm` is its own paragraph and thus separated by new lines from its surrounding text.

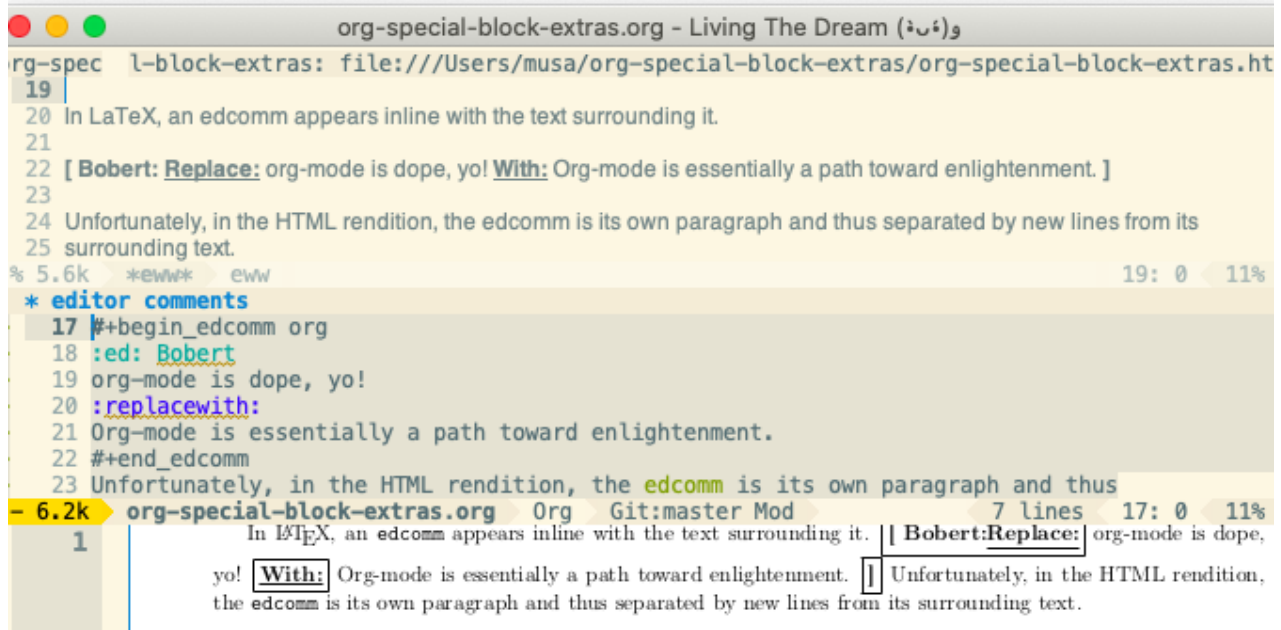


Figure 3: “First-class editor comments” In order: Chrome, Emacs Web Wowser, Org source, PDF

3 Colours

<code>#+begin_blue</code>	This text is blue!
<code>This text is blue!</code>	
<code>#+end_blue</code>	This text is brown!
<code>#+begin_brown</code>	This text is cyan!
<code>This text is brown!</code>	This text is darkgray!
<code>#+end_brown</code>	
<code>#+begin_cyan</code>	This text is gray!
<code>This text is cyan!</code>	This text is green!
<code>#+end_cyan</code>	
<code>#+begin_darkgray</code>	This text is lightgray!
<code>This text is darkgray!</code>	This text is lime!
<code>#+end_darkgray</code>	
<code>#+begin_gray</code>	This text is magenta!
<code>This text is gray!</code>	This text is olive!
<code>#+end_gray</code>	
<code>#+begin_green</code>	This text is orange!
<code>This text is green!</code>	This text is pink!
<code>#+end_green</code>	
<code>#+begin_lightgray</code>	This text is purple!
<code>This text is lightgray!</code>	This text is red!
<code>#+end_lightgray</code>	
<code>#+begin_lime</code>	This text is teal!

This text is black!
This text is blue!
This text is brown!
This text is cyan!
This text is darkgray!
This text is gray!
This text is green!
This text is lightgray!
This text is lime!
This text is magenta!
This text is olive!
This text is orange!
This text is pink!
This text is purple!
This text is red!
This text is teal!
This text is violet!

This text is yellow!

This text is black!
 This text is blue!
 This text is brown!
 This text is cyan!
 This text is darkgray!
 This text is gray!
 This text is green!
 This text is lightgray!
 This text is lime!
 This text is magenta!
 This text is olive!
 This text is orange!
 This text is pink!
 This text is purple!
 This text is red!
 This text is teal!
 This text is violet!

 This text is yellow!

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Load support for 19 colour custom blocks

(defvar org-special-block-extras--colors
  '(black blue brown cyan darkgray gray green lightgray lime
    magenta olive orange pink purple red teal violet white
    yellow)
  "Colours that should be available on all systems.")

(loop for colour in org-special-block-extras--colors
  do (eval (read (format
    "(defun org-special-block-extras--%s (backend contents)
      (format (pcase backend
        ('latex \"\\\\\\begingroup\\\\\\color{%s}%s\\\\\\endgroup\")
        ('html  \"<div style=\\\\"color:%s;\\\\">%s</div>\")
        (t      \"org-special-block-extras: Unsupported backend\")
        contents)))
      colour colour colour))))

```

4 Parallel

Example:

```

#+begin_3parallel org
one

#+latex: \columnbreak
two

#+latex: \columnbreak
three
#+end_3parallel

```

Yields:

one

two

three

Example:

```
#+begin_3parallel org
one

#+latex: \columnbreak
two
```

```
#+latex: \columnbreak
three
#+end_3parallel
```

Yields:

one | two | three

The screenshot shows the Emacs editor interface. The title bar reads "Emacs-x86_64-10_10 — (94 x 24)". The main window displays the source code for the example, with line numbers 243 to 255. The code is as follows:

```
243 Example:
244 <=> org
245 #+begin_3parallel org
246 one
247
248 #+latex: \columnbreak
249 two
250
251 #+latex: \columnbreak
252 three
253 #+end_3parallel|
254 <=
255
```

Below the main window, a buffer window titled "org-special-block-extras.org" is visible. It shows the rendered output of the code, which is a table with three columns: "one", "two", and "three". The table is created using the `#+begin_3parallel` and `#+end_3parallel` directives.

Figure 4: Displaying thoughts side-by-side ^_^

I initially used the names `parallel<n>` but names ending with a number did not inherit highlighting, so I shifted the number to being a prefix instead.

- For L^AT_EX, new lines are used to suggest opportunities for column breaks and are needed even if explicit columnbreaks are declared.

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Parallel blocks: parallel<n>[NB] for n:2..5, optionally with 'N'o 'b'ar
;; in-between the columns.
;;
;; Common case is to have three columns, and we want to avoid invoking the
;; attribute via org, so making this.

(loop for cols in '("1" "2" "3" "4" "5")
      do (loop for rule in '("solid" "none")
              do (eval (read (concat
"(defun org-special-block-extras--" cols "parallel"
(if (equal rule "solid") "" "NB")
"(backend contents)"
"(format (pcase backend"
"('html \"<div style=\\\"\\\"column-rule-style:\" rule \";column-count:\" cols \";\\\"\\\"%s</div>\")"
"('latex \"\\\\\\par \\\\setlength{\\\\\\columnseprule}{\" (if (equal rule \"solid\") \"2\" \"0\") \"pt}\""
"          \\\\begin{minipage}[t]{\\\\\\linewidth}\"
"          \\\\begin{multicols}{\" cols \"}\"
"          %s\"
"          \\\\end{multicols}\\\\\\end{minipage}\\\\\\)) contents))))))

(defalias #'org-special-block-extras--parallel #'org-special-block-extras--2parallel)
(defalias #'org-special-block-extras--parallelNB #'org-special-block-extras--2parallelNB)

```

(The Emacs Web Wowser, M-x eww, does not display parallel environments as desired.)

5 :argument: Extraction

```

(defun org-special-block-extras--extract-arguments (contents &rest args)
"Return list of CONTENTS string with ARGS lines stripped out and values of ARGS.

```

Example usage:

```

(-let [(contents' . (&alist 'k0 ... 'kn))
      (...extract-arguments contents 'k0 ... 'kn)]
  body)

```

Within 'body', each 'k_i' refers to the 'value' of argument ':k_i:' in the CONTENTS text and 'contents' is CONTENTS with all ':k_i:' lines stripped out.

- + If ':k:' is not an argument in CONTENTS, then it is assigned value 'nil'.
- + If ':k:' is an argument in CONTENTS but is not given a value in CONTENTS, then it has value the empty string."

```

(let ((ctnts contents)
      (values (loop for a in args

```

```

      for regex = (format ":%s:\\(.*\\)" a)
      for v = (cadr (s-match regex contents))
      collect (cons a v)))
(loop for a in args
      for regex = (format ":%s:\\(.*\\)" a)
      do (setq cntns (s-replace-regexp regex "" cntns)))
(cons cntns values)))

```

6 Editor Comments

“Editor Comments” are intended to be top-level first-class comments in an article that are inline with the surrounding text and are delimited in such a way that they are visible but drawing attention. I first learned about this idea from Wolfram Kahl—who introduced me to Emacs many years ago.

In \LaTeX , an `edcomm` appears inline with the text surrounding it. [Bobert:Replace: org-mode is dope, yo! With: Org-mode is essentially a path toward enlightenment.] Unfortunately, in the HTML rendition, the `edcomm` is its own paragraph and thus separated by new lines from its surrounding text.

In \LaTeX , an `edcomm` appears inline with the text surrounding it.

[Bobert: Replace: org-mode is dope, yo! With: Org-mode is essentially a path toward enlightenment.]

Unfortunately, in the HTML rendition, the `edcomm` is its own paragraph and thus separated by new lines from its surrounding text.

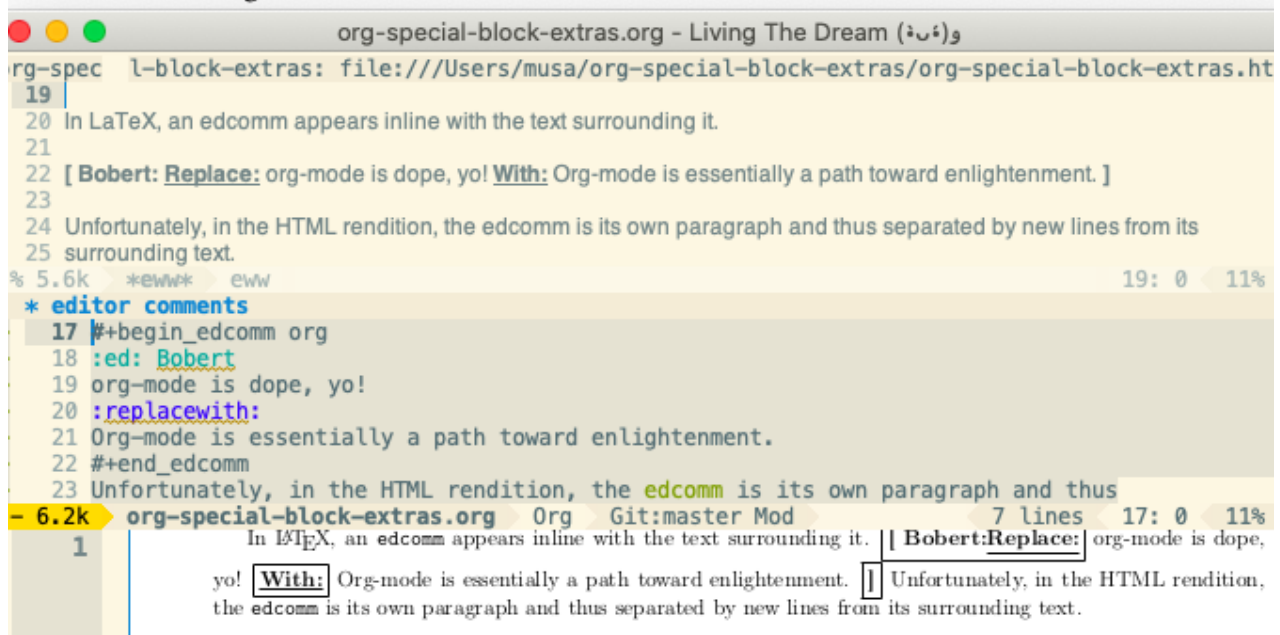


Figure 5: In order: Chrome, Emacs Web Wowser, Org source, PDF

Any new —possibly empty— inner lines in the `edcomm` are desirably preserved

```

1 (defvar org-special-block-extras-hide-editor-comments nil
2   "Should editor comments be shown in the output or not.")
3
4 (defun org-special-block-extras--edcomm (backend contents)
5   "Format CONTENTS as an first-class editor comment according to BACKEND."

```



```

6
7 The CONTENTS string has two optional argument switches:
8 1. :ed: ⇒ To declare an editor of the comment.
9 2. :replacewith: ⇒ [Nullary] The text preceding this clause
10    should be replaced by the text after it."
11 (-let* (
12     ;; Get arguments
13     ((contents1 . (&alist 'ed))
14     (org-special-block-extras--extract-arguments contents 'ed))
15
16     ;; Strip out any <p> tags
17     (_ (setq contents1 (s-replace-regexp "<p>" "" contents1)))
18     (_ (setq contents1 (s-replace-regexp "</p>" "" contents1)))
19
20     ;; Are we in the html backend?
21     (html? (equal backend 'html))
22
23     ;; fancy display style
24     (boxed (lambda (x)
25              (if html?
26                  (concat "<span style=\"border-width:1px"
27                          ";border-style:solid;padding:5px\">"
28                          "<strong> x </strong></span>")
29                  (concat "\\fbbox{\\bf " x "}")))))
30
31     ;; Is this a replacement clause?
32     ((this that) (s-split ":replacewith:" contents1))
33     (replacement-clause? that) ;; There is a 'that'
34     (replace-keyword (if html? "&nbsp;<u>Replace:</u>"
35                          "\\underline{Replace:}"))
36     (with-keyword (if html? "<u>With:</u>"
37                       "\\underline{With:}"))
38     (editor (format "[%s:%s"
39                     (if (s-blank? ed) "Editor Comment" ed)
40                     (if replacement-clause?
41                         replace-keyword
42                         "")))
43     (contents2 (if replacement-clause?
44                    (format "%s %s %s" this
45                            (funcall boxed with-keyword)
46                            that)
47                    contents1))
48
49     ;; "[Editor Comment:"
50     (edcomm-begin (funcall boxed editor))
51     ;; "]"
52     (edcomm-end (funcall boxed "]")))
53
54 (setq org-export-allow-bind-keywords t) ;; So users can use "#+bind" immediately
55 (if org-special-block-extras-hide-editor-comments
56     ""
57     (format (pcase backend

```

```

58      ('html "<p> %s %s %s</p>" )
59      ('latex "%s %s %s" ))
60      edcomm-begin contents2 edcomm-end))))

```

In the HTML export, the `edcomm` special block is *not* in-line with the text surrounding it — ideally, it would be inline so that existing paragraphs are not split into multiple paragraphs but instead have an editor’s comment indicating suggested alterations; see Line 16 above.

6.1 Examples

Org-markup is supported, as expected.

All editor comments are disabled by declaring, in your Org file:

```
#+bind: org-special-block-extras-hide-editor-comments t
```

The `#+bind:` keyword makes Emacs variables buffer-local during export — it is evaluated *after* any `src` blocks. To use it, one must declare in their Emacs init file the following line, which our `edcomm` utility ensures is true.

```
(setq org-export-allow-bind-keywords t)
```

(Remember to `C-c C-c` the `#+bind` to activate it, the first time it is written.)

6.1.1 No optional arguments

[Editor Comment: *Please change this section to be more, ya know, professional.* **]**

6.1.2 Only declaring an `:ed:` —editor

[Bobert: *Please change this section to be more, ya know, professional.* **]**

Possibly with no contents: **[Bobert:** **]**

6.1.3 Empty contents, no editor, nothing

[Editor Comment: **]**

Possibly with an empty new line: **[Editor Comment:** **]**

6.1.4 With a `:replacewith:` clause

[Editor Comment:Replace: The two-dimensional notation; e.g., $\sum_{i=0}^n i^2$ **With:** A linear one-dimensional notation; e.g., $(\sum i : 0..n \bullet i^2)$ **]**

Possibly “malformed” replacement clauses.

1. Forget the thing to be replaced. **[Editor Comment:Replace:** **With:** A linear one-dimensional notation; e.g., $(\sum i : 0..n \bullet i^2)$ **]**
2. Forget the new replacement thing. **[Editor Comment:Replace:** The two-dimensional notation; e.g., $\sum_{i=0}^n i^2$ **With:** **]**
3. Completely lost one’s train of thought. **[Editor Comment:Replace:** **With:** **]**