Tangle this file: four issues with org-babel-tangle

Christopher Genovese

14 Sep 2011

Contents

1 Four Related Issues with org-babel-tangle

Running org-mode version 7.7 on both Gnu Emacs 23.2.1 and 24.0.50.1 with Mac OS X 10.5.8 (with and without a -Q option), I encountered the following issues/problems/bugs when tangling files with code blocks for which the :comment header argument is org:

1. The subtree associated with the very first code block has its headline tangled without leading stars, but all subsequent sub-trees associated with code blocks have the leading stars included in the comments.

2. If the first code block comes before the first headline, the start of the comment text will be determined by pre-existing match data and thus will likely be incorrect.

3. Org structural elements such as headline stars, `#+` control lines, and drawers are included in the comments.

4. There is no way easy to delimit comment text or transform it that does not also change the structure of the org file (e.g., by adding headlines or source blocks).

Issues 1 and 2 seem to be genuine bugs. Issues 3 and 4 are more subjective, I admit, but seem undesirable. Stars, drawers, and control lines are org structure rather than content, and they are often inappropriate in comments.

To reproduce the behaviors for issues 1 and 3, look at the result of tangling this file. To reproduce issue 2 as well, remove the first two stars from this file and tangle again. Alternatively, within emacs, evaluate the `(buffer-substring ...)` sexp from the original-code code below at the beginning character of a source block. (You can also export this file to PDF for more pleasant reading.)

Below, I give details on these issues and code for two fixes: a simple fix that handles the first two issues and the stars in the third, and a better fix that handles all four issues in a more modular, customizable framework. I'd be interested in hearing feedback on all of this. If the reaction is positive, I will gladly submit a patch. Thanks for your consideration.

2 The Original Code and Details on the Problem

The relevant section of the original code from org-mode version 7.7 is shown below, comprising lines 344 through 357 of ob-tangle.el in function `org-babel-tangle-collect-blocks`. With point is at a `#+begin_src`, it scans back either for a heading line or for the end of the previous code block, whichever comes later. The resulting region becomes the comment text.

```
(comment
 (when (or (string= "both" (cdr (assoc :comments params)))
           (string= "org" (cdr (assoc :comments params))))
   ;; from the previous heading or code-block end
   (buffer-substring
    (max (condition-case nil
             (save-excursion
               (org-back-to-heading t) (point))
           (error 0))
         (save-excursion
           (re-search-backward
            org-babel-src-block-regexp nil t)
           (match-end 0)))
    (point))))
```

Figure 1. Original Code, lines 344–357 in `ob-tangle.el`.

Issue 1. When in the first code block in the file, the second search fails (there is no previous code block), so the (match-end 0) call uses the match data from the (implicit) match during `org-back-to-heading`, which skips the stars. (Not a particularly transparent reference, incidentally.) For subsequent blocks, the `(match-end 0)` gives the end of the previous code block, which in these examples is earlier than the previous headline location.

Issue 2. When the first code block lies before the first headline (say with some text before it), the searches fail in *both* clauses of the max. So, the `match-end` will return an essentially arbitrary result, which is a bug.

Issue 3. `org-back-to-heading` leaves point at the beginning of the stars, so a headline included in the text will have stars, except for the first one.

Issue 4. Control lines at the end of the previous code block and before point are not filtered out and so are included in the comments.

3 A Simple Fix for the First Three Issues

A small change addresses issues 1, 2, and the stars for issue 3: in both cases, simply use the `match-end` and replace 0 values with `(point-min)`. The latter gives a sensible result even if both computed positions are trivial (as when the first code block comes before the first headline) and respects narrowing.

```
(comment
 (when (or (string= "both" (cdr (assoc :comments params)))
           (string= "org" (cdr (assoc :comments params))))
   ;; from the previous heading or code-block end
   (buffer-substring
    (max (condition-case nil
             (save-excursion
               (org-back-to-heading t)  ; sets match data
               (match-end 0))
           (error (point-min)))
         (save-excursion
           (if (re-search-backward
                org-babel-src-block-regexp nil t)
               (match-end 0)
             (point-min))))
    (point))))
```

Figure 2. Simple Fix, replacement for lines 344–357 in `ob-tangle.el`.

4 A Fix for All Four Issues

A better fix that handles issues 1–4 starts with the region computed as in the simple fix and then processes that text through a user-configurable sequence of functions to derive the final form of the comment text.

The following changes are required.

- Extract Initial Comment Text and State from Org Buffer
  The initial comment text ranges from either the most recent headline at the point after the stars, the beginning of the line after the `#+end_src` of the most recent code block, or the beginning of the buffer, whichever is later, through the line before the source block.[1]

  The code to extract this is given below. (See Figure 3.) It replaces lines 344 through 357 of `ob-tangle.el` from org-mode version 7.7 in the function `org-babel-tangle-collect-blocks`.

- Adjust `org-babel-spec-to-string`
  The commment block collected by the original code (Figure 1) in `org-babel-tangle-collect-blocks` is further processed in `org-babel-spec-to-string` to trim leading and trailing whitespace from string. This was needed because spaces after a source block were included in the comment. In the revised code, however, this space trimming is handled during text transformation, except for removing trailing newlines. (Note: trailing *spaces* are not removed to allow more flexibility in comment processing.) Hence, `org-babel-spec-to-string` needs to be slightly adjusted. See Figure 4.

- Process Comment Text Through Sequence of Transforms
  At the end of the revised comment collection code, the comment text is passed to `org-babel-process-comment-text` which applies a sequence of transformation functions. (See Figure 5.) The list of transformation functions is stored in a customizable variable described below. Several predefined transformations are given below as well.

- Define Customization Variable for Transforms
  A list of nullary functions applied in order to the comment text. The text is inserted in a temporary buffer, so these functions can use the entire Emacs library for operating on buffer text. See Figure 6.

[1]In the original code and in the simple fix above, the comment starts *immediately* after the `#+end_src` rather than at the start of the next line. Starting at the next line seems more natural to me because the comment being constructed relates to the *following* code block. But the original behavior is easily restored if people disagree.

- Define A Collection of Transform Functions
  An advantage of this design is that transformation of the comments is modular and customizable. We can include in `ob-tangle.el` a collection of pre-defined transforms. The default processing stream in `org-babel-comment-processing-functions` is as follows:

  1. Delete a file variables if on the first line of the buffer.
  2. Delete all drawers and their contents.
  3. Delete all org control lines from the comment text.
  4. Trim blank lines from the beginning and end.
  5. Reindent the text by removing the longest common leading string of spaces.

  These and several other useful transforms are given below (e.g., deleting drawer delimiters but not contents).. See Figures 7–13. It is easy to define new transforms; any function that operates on text in the current buffer beginning at point-min will work.

This kind of customization offers some nice possibilities, including controlling indentation, eliminating or transforming org markup, eliminating trailing whitespace, and automating specialized comment formatting (e.g., javadoc). As an additional illustration, consider the transform `org-babel-comment-restrict-comment-range` in Figure 14 below. The idea is that it is sometimes useful to select from the text under a headline a *part* of the text for the comment. We want some org markup that will not affect either the export or the structure of the org file itself. To do this, we use the fact that `#+␣`lines are not exported.[2] So, we can *de facto* use the `#+ TANGLE:` construct to control various aspects of tangling. Here, we use the `#+ TANGLE: start-comment` and `#+ TANGLE: end-comment` to delimit the comment text. (This function needs to come earlier in the function list than the functions that eliminate org control lines. It is sufficient to prepend it to that list.) This is used in the current file, for example.

[2]A feature request: I would propose that the `#+tangle:` construct be recognized as non-exported even with spaces preceding the `#` and no spaces after the `+`. This would enable a variety of interesting customization for tangled comments. Alternatively, a generic construct such as `#+noop:` or `#+generic:` could be a valuable for user-based tags in an org file that serves a similar purpose – allow customized processing without directly being exported.

```
(comment
 (when (or (string= "both" (cdr (assoc :comments params)))
           (string= "org" (cdr (assoc :comments params))))
   (let* ((prev-heading
           (condition-case nil
               (save-excursion
                 (org-back-to-heading t) ; sets match data
                 (match-end 0))
             (error (point-min))))
          (end-of-prev-src-block
           (save-excursion
             (if (null (re-search-backward
                        org-babel-src-block-regexp nil t))
                 (point-min)
               (goto-char (match-end 0))
               (forward-line 1)
               (point))))
          (comment-start
           (max prev-heading end-of-prev-src-block))
          (comment-end
           (save-excursion
             (forward-line 0)
             (point)))
          (state
           (list (cons 'org-drawers
                       org-drawers)
                 (cons 'after-heading
                       (= comment-start prev-heading))
                 (cons 'first-line
                       (= comment-start (point-min))))))
     (org-babel-process-comment-text
      (buffer-substring comment-start comment-end) state))))
```

Figure 3. Better Fix, replacement for lines 344–357 in `ob-tangle.el`.

```
--- ob-tangle.el          2011-09-14 11:48:26.000000000 -0400
+++ new-ob-tangle.el      2011-09-14 11:55:56.000000000 -0400
@@ -398,3 +398,3 @@
     (flet ((insert-comment (text)
-            (let ((text (org-babel-trim text)))
+            (let ((text (org-babel-chomp text "[\f\t\n\r\v]")))
          (when (and comments (not (string= comments "no"))
```

Figure 4. Changes to `org-spec-to-string` in `ob-tangle.el`, unified diff, one line of context

```
(defun org-babel-process-comment-text (text &optional state)
  "Apply list of transforms to comment TEXT assuming bindings in alist STATE.
Returns the modified text string, which may have text properties.
See `org-babel-comment-processing-functions' for the transforms to be
applied and details on the allowed keys in the STATE alist."
  (let ((funcs org-babel-comment-processing-functions))
    (with-temp-buffer
      (insert text)
      (let ((org-drawers
             (or (cdr (assoc 'org-drawers state))
                 org-drawers))
            (after-heading
             (cdr (assoc 'after-heading state)))
            (first-line
             (cdr (assoc 'first-line state))))
        (while funcs
          (goto-char (point-min))
          (funcall (car funcs))
          (setq funcs (cdr funcs))))
      (buffer-substring (point-min) (point-max)))))
```

Figure 5. Better Fix, comment transformation driver.

```
(defcustom org-babel-comment-processing-functions
  '(org-babel-comment-delete-file-variables-line
    org-babel-comment-delete-org-control-lines
    org-babel-comment-delete-drawers
    org-babel-comment-trim-blank-lines
    org-babel-comment-trim-indent-prefix)
  "List of functions to transform source-block comment text before insertion.
Each function will be called with no arguments with point at the
beginning of a buffer containing only the comment text. Each
function can modify the text at will and leave point anywhere,
but it should *not* modify the narrowing state of the buffer.
Several dynamic state variables are set prior to execution that
each function can reference. These currently include:

   + org-drawers:   names of drawers in the original org buffer.
   + from-heading:  t if comment starts at an org heading line,
                    nil otherwise.
   + first-line:    t if initial comment starts on first line
                    of the original org buffer, nil otherwise.

If a function changes the value of these state variables, the new
value will be seen by all following functions in the list, but
this is not generally recommended.

The functions in this list are called *in order*, and this order
can influence the form of the resulting comment text."
  :group 'org-babel
  :type 'list)
```

Figure 6. Better Fix, customizable transform list.

```
(defun org-babel-comment-delete-file-variables-line ()
  "Delete file variables comment line if at beginning of buffer.
This only checks the first line of the buffer, and so should be
placed first (or at least early enough) in the list
‘org-babel-comment-processing-functions’ to ensure that the no
other text has been inserted earlier."
  (when (and first-line
             (looking-at ; file-variables line
              "^#[ \t]*-\\*-.*:.*;[ \t]**-\\*-[ \t]*$"))
    (let ((kill-whole-line t))
      (kill-line))))
```

Figure 7. Comment Transform.

```
(defun org-babel-comment-delete-org-control-lines ()
  "Remove all org #+ control lines from comment."
  (let ((control-regexp "^[ \t]*#\\+.*$"))
    (delete-matching-lines control-regexp)))
```

Figure 8. Comment Transform.

```
(defun org-babel-comment-delete-org-in-buffer-settings ()
  "Remove all org #+ in-buffer setting lines, leaving other control lines.
In-buffer setting lines begin with #+ and have all caps keyword
names."
  (let ((setting-regexp "^#\\+[ \t]*[A-Z_]+:.*$"))
    (delete-matching-lines setting-regexp)))
```

Figure 9. Comment Transform.

```
(defun org-babel-comment-delete-drawers ()
  "Delete drawer delimiters and contents from comment.
Drawer names are restricted to those in the ‘org-drawers’ state."
  (let ((drawer-start-regexp
         (format "^[ \t]*:\\(?:%s\\):[ \t]*$"
                 (mapconcat 'identity
                            org-drawers
                            "\\|")))
        (drawer-end-regexp "^[ \t]*:END:[ \t]*$"))
    (while (re-search-forward drawer-start-regexp nil t)
      (let ((beg (save-excursion
                   (forward-line 0)
                   (point)))
            (end (save-excursion
                   (re-search-forward drawer-end-regexp nil t)
                   (forward-line 1)
                   (point))))
        (goto-char end)
        (delete-region beg end)))))
```

Figure 10. Comment Transform.

```
(defun org-babel-comment-delete-drawer-delimiters ()
  "Delete drawer delimiters from comment leaving content.
Drawer names are restricted to those given by the ‘org-drawers’
state."
  (let ((drawer-delim-regexp
         (format "^[ \t]*:\\(?:%s\\)"
                 (mapconcat 'identity
                            (cons "END" org-drawers)
                            "\\|"))))
    (delete-matching-lines drawer-delim-regexp)))
```

Figure 11. Comment Transform.

```
(defun org-babel-comment-trim-blank-lines ()
  "Trim whitespace-only lines from beginning and end of text."
  (while (and (looking-at "^[ \t\f]*$")
              (< (point) (point-max)))
    (forward-line 1))
  (delete-region (point-min) (point))
  (when (< (point) (point-max))
    (goto-char (point-max))
    (let ((last-point (point)))
      (forward-line 0)
      (while (and (looking-at "^[ \t\f]*$")
                  (> (point) (point-min)))
        (setq last-point (point))
        (forward-line -1))
      (delete-region last-point (point-max)))))
```

Figure 12. Comment Transform.

```
(defun org-babel-comment-trim-indent-prefix ()
  "Remove longest common leading prefix of spaces from each line of TEXT.
Prefix is computed from the initial whitespace on each line with
tabs converted to spaces, preserving indentation."
  (let* ((common-indent nil)
         (common-length (1+ (- (point-max) (point-min))))
         (current-indent "")                  ; enter first loop
         (current-length common-length))      ; skip first assignment
    (goto-char (point-min))
    (while current-indent
      (when (< current-length common-length)
        (setq common-indent current-indent
              common-length current-length))
      (setq current-indent
            (let* ((found (re-search-forward "^\\([ \t]*\\)\\S-" nil t))
                   (bol (match-beginning 0))
                   (eos (match-end 1))
                   (space-str (match-string 1))
                   (indent-tabs-mode nil))
              (cond
               ((not found)
                nil)
               ((not (string-match "\t" space-str))
                space-str)
               (t                      ; detabify indent string
                (goto-char eos)
                (let ((col (current-column)))
                  (delete-region bol eos)
                  (indent-to col))
                (buffer-substring-no-properties bol (point))))))
      (setq current-length (length current-indent)))
    (when (and common-indent (> common-length 0))
      (let ((indent-re (concat "^" common-indent)))
        (goto-char (point-min))
        (while (re-search-forward indent-re nil t)
          (replace-match "" nil nil))))))
```

Figure 13. Comment Transform.

```
(defun org-babel-comment-restrict-comment-range ()
  "Remove all comment text outside start-comment and end-comment delimiters.
Comment delimiters are #+TANGLE lines with respective keywords
start-comment and end-comment. THE #+TANGLE lines are also
deleted. To be effective, this function should be positioned in
the list ‘org-babel-comment-processing-functions’ before any
functions that remove org control lines or process other
co-occuring attributes of #+TANGLE lines."
  (when (re-search-forward "^[ \t]*#\\+[ \t]*TANGLE:.*start-comment.*$" nil t)
    (forward-line 1)
    (delete-region (point-min) (point)))
  (when (re-search-forward "^[ \t]*#\\+[ \t]*TANGLE:.*end-comment.*$" nil t)
    (forward-line 0)
    (delete-region (point) (point-max))))
```

Figure 14. Transform to illustrate some customization possibilities.