org-element

Thorsten Jolitz

June 18, 2013

Contents

1 org-element.el — Parser And Applications for Org syntax

Author: Nicolas Goaziou <n.goaziou at gmail dot com> Keywords: outlines, hypermedia, calendar, wp

2 Commentary:

Org syntax can be divided into three categories: "Greater elements", "Elements" and "Objects".

Elements are related to the structure of the document. Indeed, all elements are a cover for the document: each position within belongs to at least one element.

An element always starts and ends at the beginning of a line. With a few exceptions (‘clock’, ‘headline’, ‘inlinetask’, ‘item’, ‘planning’, ‘node-property’, ‘quote-section’ ‘section’ and ‘table-row’ types), it can also accept a fixed set of keywords as attributes. Those are called "affiliated keywords" to distinguish them from other keywords, which are full-fledged elements. Almost all affiliated keywords are referenced in ‘org-element-affiliated-keywords’; the others are export attributes and start with "ATTR_" prefix.

Element containing other elements (and only elements) are called greater elements. Concerned types are: ‘center-block’, ‘drawer’, ‘dynamic-block’, ‘footnote-definition’, ‘headline’, ‘inlinetask’, ‘item’, ‘plain-list’, ‘property-drawer’, ‘quote-block’, ‘section’ and ‘special-block’.

Other element types are: ‘babel-call’, ‘clock’, ‘comment’, ‘comment-block’, ‘diary-sexp’, ‘example-block’, ‘export-block’, ‘fixed-width’, ‘horizontal-rule’, ‘keyword’, ‘latex-environment’, ‘node-property’, ‘paragraph’, ‘planning’, ‘quote-section’, ‘src-block’, ‘table’, ‘table-row’ and ‘verse-block’. Among them, ‘paragraph’ and ‘verse-block’ types can contain Org objects and plain text.

Objects are related to document’s contents. Some of them are recursive. Associated types are of the following: ‘bold’, ‘code’, ‘entity’, ‘export-snippet’, ‘footnote-reference’, ‘inline-babel-call’, ‘inline-src-block’, ‘italic’, ‘latex-fragment’, ‘line-break’, ‘link’, ‘macro’, ‘radio-target’, ‘statistics-cookie’, ‘strike-through’, ‘subscript’, ‘superscript’, ‘table-cell’, ‘target’, ‘timestamp’, ‘underline’ and ‘verbatim’.

Some elements also have special properties whose value can hold objects themselves (i.e. an item tag or a headline name). Such values are called "secondary strings". Any object belongs to either an element or a secondary string.

Notwithstanding affiliated keywords, each greater element, element and object has a fixed set of properties attached to it. Among them, four are shared by all types: ‘:begin’ and ‘:end’, which refer to the beginning and ending buffer positions of the considered element or object, ‘:post-blank’, which holds the number of blank lines, or white spaces, at its end and ‘:parent’ which refers to the element or object containing it. Greater elements, elements and objects containing objects will also have ‘:contents-begin’ and ‘:contents-end’ properties to delimit contents. Eventually, greater elements and elements accepting affiliated keywords will have a ‘:post-affiliated’ property, referring to the buffer position after all such keywords.

At the lowest level, a ‘:parent’ property is also attached to any string, as a text property.

Lisp-wise, an element or an object can be represented as a list. It follows the pattern (TYPE PROPERTIES CONTENTS), where: TYPE is a symbol describing the Org element or object. PROPERTIES is the property list attached to it. See docstring of appropriate parsing function to get an exhaustive list. CONTENTS is a list of elements, objects or raw strings contained in the current element or object, when applicable.

An Org buffer is a nested list of such elements and objects, whose type is ‘org-data’ and properties is nil.

The first part of this file defines Org syntax, while the second one provide accessors and setters functions.

The next part implements a parser and an interpreter for each element and object type in Org syntax.

The following part creates a fully recursive buffer parser. It also provides a tool to map a function to elements or objects matching some criteria in the parse tree. Functions of interest are ‘org-element-parse-buffer’, ‘org-element-map’ and, to a lesser extent, ‘org-element-parse-secondary-string’.

The penultimate part is the cradle of an interpreter for the obtained parse tree: ‘org-element-interpret-data’.

The library ends by furnishing ‘org-element-at-point’ function, and a way to give information about document structure around point with ‘org-element-context’.

3 Code:

```
(eval-when-compile (require 'cl))
(require 'org)
```

4 Definitions And Rules

Define elements, greater elements and specify recursive objects, along with the affiliated keywords recognized. Also set up restrictions on recursive objects combinations.

These variables really act as a control center for the parsing process.

```
(defconst org-element-paragraph-separate
  (concat "^\\(?:"
          ;; Headlines, inlinetasks.
          org-outline-regexp "\\|"
          ;; Footnote definitions.
          "\\[\\(?:[0-9]+\\|fn:[-_[:word:]]+\\)\\]" "\\|"
          ;; Diary sexps.
          "%%(" "\\|"
          "[ \t]*\\(?:"
          ;; Empty lines.
          "$" "\\|"
          ;; Tables (any type).
          "\\(?:|\\|\\+-[-+]\\)" "\\|"
          ;; Blocks (any type), Babel calls, drawers (any type),
          ;; fixed-width areas and keywords.  Note: this is only an
          ;; indication and need some thorough check.
          "[#:]" "\\|"
          ;; Horizontal rules.
          "-\\{5,\\}[ \t]*$" "\\|"
          ;; LaTeX environments.
          "\\\\begin{\\([A-Za-z0-9]+\\*?\\)}" "\\|"
          ;; Planning and Clock lines.
          (regexp-opt (list org-scheduled-string
                            org-deadline-string
                            org-closed-string
                            org-clock-string))
          "\\|"
          ;; Lists.
          (let ((term (case org-plain-list-ordered-item-terminator
                        (?\) ")") (?. "\\.") (otherwise "[.)]")))
                (alpha (and org-list-allow-alphabetical "\\|[A-Za-z]")))
            (concat "\\(?:[-+*]\\|\\(?:[0-9]+" alpha "\\)" term "\\)"
                    "\\(?:[ \t]\\|$\\)"))
          "\\)\\)")
  "Regexp to separate paragraphs in an Org buffer.
In the case of lines starting with \"#\" and \":\", this regexp
is not sufficient to know if point is at a paragraph ending.  See
`org-element-paragraph-parser' for more information.")

(defconst org-element-all-elements
  '(babel-call center-block clock comment comment-block diary-sexp drawer
               dynamic-block example-block export-block fixed-width
               footnote-definition headline horizontal-rule inlinetask item
               keyword latex-environment node-property paragraph plain-list
               planning property-drawer quote-block quote-section section
               special-block src-block table table-row verse-block)
  "Complete list of element types.")

(defconst org-element-greater-elements
  '(center-block drawer dynamic-block footnote-definition headline inlinetask
                 item plain-list property-drawer quote-block section
                 special-block table)
  "List of recursive element types aka Greater Elements.")

(defconst org-element-all-successors
  '(export-snippet footnote-reference inline-babel-call inline-src-block
                   latex-or-entity line-break link macro plain-link radio-target
                   statistics-cookie sub/superscript table-cell target
                   text-markup timestamp)
  "Complete list of successors.")

(defconst org-element-object-successor-alist
  '((subscript . sub/superscript) (superscript . sub/superscript)
    (bold . text-markup) (code . text-markup) (italic . text-markup)
    (strike-through . text-markup) (underline . text-markup)
    (verbatim . text-markup) (entity . latex-or-entity)
    (latex-fragment . latex-or-entity))
  "Alist of translations between object type and successor name.
```

```
Sharing the same successor comes handy when, for example, the
regexp matching one object can also match the other object.")

(defconst org-element-all-objects
  '(bold code entity export-snippet footnote-reference inline-babel-call
         inline-src-block italic line-break latex-fragment link macro
         radio-target statistics-cookie strike-through subscript superscript
         table-cell target timestamp underline verbatim)
  "Complete list of object types.")

(defconst org-element-recursive-objects
  '(bold italic link subscript radio-target strike-through superscript
         table-cell underline)
  "List of recursive object types.")

(defvar org-element-block-name-alist
  '(("CENTER" . org-element-center-block-parser)
    ("COMMENT" . org-element-comment-block-parser)
    ("EXAMPLE" . org-element-example-block-parser)
    ("QUOTE" . org-element-quote-block-parser)
    ("SRC" . org-element-src-block-parser)
    ("VERSE" . org-element-verse-block-parser))
  "Alist between block names and the associated parsing function.
Names must be uppercase.  Any block whose name has no association
is parsed with ‘org-element-special-block-parser’.")

(defconst org-element-link-type-is-file
  '("file" "file+emacs" "file+sys" "docview")
  "List of link types equivalent to \"file\".
Only these types can accept search options and an explicit
application to open them.")

(defconst org-element-affiliated-keywords
  '("CAPTION" "DATA" "HEADER" "HEADERS" "LABEL" "NAME" "PLOT" "RESNAME" "RESULT"
    "RESULTS" "SOURCE" "SRCNAME" "TBLNAME")
  "List of affiliated keywords as strings.
By default, all keywords setting attributes (i.e. \"ATTR_LATEX\")
are affiliated keywords and need not to be in this list.")

(defconst org-element--affiliated-re
  (format "[ \t]*#\\+%s:"
          ;; Regular affiliated keywords.
          (format "\\(%s\\|ATTR_[-_A-Za-z0-9]+\\)\\(?:\\[\\(.*\\)\\]\\)?"
                  (regexp-opt org-element-affiliated-keywords)))
  "Regexp matching any affiliated keyword.

Keyword name is put in match group 1.  Moreover, if keyword
belongs to ‘org-element-dual-keywords’, put the dual value in
match group 2.

Don’t modify it, set ‘org-element-affiliated-keywords’ instead.")

(defconst org-element-keyword-translation-alist
  '(("DATA" . "NAME")  ("LABEL" . "NAME") ("RESNAME" . "NAME")
    ("SOURCE" . "NAME") ("SRCNAME" . "NAME") ("TBLNAME" . "NAME")
    ("RESULT" . "RESULTS") ("HEADERS" . "HEADER"))
  "Alist of usual translations for keywords.
The key is the old name and the value the new one.  The property
holding their value will be named after the translated name.")

(defconst org-element-multiple-keywords '("CAPTION" "HEADER")
  "List of affiliated keywords that can occur more than once in an element.

Their value will be consed into a list of strings, which will be
returned as the value of the property.

This list is checked after translations have been applied.  See
‘org-element-keyword-translation-alist’.

By default, all keywords setting attributes (i.e. \"ATTR_LATEX\")
allow multiple occurrences and need not to be in this list.")

(defconst org-element-parsed-keywords '("CAPTION")
  "List of affiliated keywords whose value can be parsed.

Their value will be stored as a secondary string: a list of
strings and objects.

This list is checked after translations have been applied.  See
‘org-element-keyword-translation-alist’.")
```

```
(defconst org-element-dual-keywords '("CAPTION" "RESULTS")
  "List of affiliated keywords which can have a secondary value.

In Org syntax, they can be written with optional square brackets
before the colons.  For example, RESULTS keyword can be
associated to a hash value with the following:

  #+RESULTS[hash-string]: some-source

This list is checked after translations have been applied.  See
`org-element-keyword-translation-alist'.")

(defconst org-element-document-properties '("AUTHOR" "DATE" "TITLE")
  "List of properties associated to the whole document.
Any keyword in this list will have its value parsed and stored as
a secondary string.")

(defconst org-element-object-restrictions
  (let* ((standard-set
          (remq 'plain-link (remq 'table-cell org-element-all-successors)))
         (standard-set-no-line-break (remq 'line-break standard-set)))
    `((bold ,@standard-set)
      (footnote-reference ,@standard-set)
      (headline ,@standard-set-no-line-break)
      (inlinetask ,@standard-set-no-line-break)
      (italic ,@standard-set)
      (item ,@standard-set-no-line-break)
      (keyword ,@standard-set)
      ;; Ignore all links excepted plain links in a link description.
      ;; Also ignore radio-targets and line breaks.
      (link export-snippet inline-babel-call inline-src-block latex-or-entity
            macro plain-link statistics-cookie sub/superscript text-markup)
      (paragraph ,@standard-set)
      ;; Remove any variable object from radio target as it would
      ;; prevent it from being properly recognized.
      (radio-target latex-or-entity sub/superscript)
      (strike-through ,@standard-set)
      (subscript ,@standard-set)
      (superscript ,@standard-set)
      ;; Ignore inline babel call and inline src block as formulas are
      ;; possible.  Also ignore line breaks and statistics cookies.
      (table-cell export-snippet footnote-reference latex-or-entity link macro
                  radio-target sub/superscript target text-markup timestamp)
      (table-row table-cell)
      (underline ,@standard-set)
      (verse-block ,@standard-set)))
  "Alist of objects restrictions.

CAR is an element or object type containing objects and CDR is
a list of successors that will be called within an element or
object of such type.

For example, in a ‘radio-target’ object, one can only find
entities, latex-fragments, subscript and superscript.

This alist also applies to secondary string.  For example, an
‘headline’ type element doesn’t directly contain objects, but
still has an entry since one of its properties (‘:title’) does.")

(defconst org-element-secondary-value-alist
  '((headline . :title)
    (inlinetask . :title)
    (item . :tag)
    (footnote-reference . :inline-definition))
  "Alist between element types and location of secondary value.")

(defconst org-element-object-variables '(org-link-abbrev-alist-local)
  "List of buffer-local variables used when parsing objects.
These variables are copied to the temporary buffer created by
‘org-export-secondary-string’.")
```

5 Accessors and Setters

Provide four accessors: ‘org-element-type’, ‘org-element-property’ ‘org-element-contents’ and ‘org-element-restriction’.

Setter functions allow to modify elements by side effect. There is ‘org-element-put-property’, ‘org-element-set-contents’, ‘org-element-set-element’ and ‘org-element-adopt-element’. Note that ‘org-element-set-element’ and ‘org-element-adopt-elements’ are higher level functions since also update ‘:parent’ property.

```
(defsubst org-element-type (element)
  "Return type of ELEMENT.

The function returns the type of the element or object provided.
It can also return the following special value:
  ‘plain-text’       for a string
  ‘org-data’         for a complete document
  nil                in any other case."


(defsubst org-element-property (property element)
  "Extract the value from the PROPERTY of an ELEMENT."


(defsubst org-element-contents (element)
  "Extract contents from an ELEMENT."


(defsubst org-element-restriction (element)
  "Return restriction associated to ELEMENT.
ELEMENT can be an element, an object or a symbol representing an
element or object type."


(defsubst org-element-put-property (element property value)
  "In ELEMENT set PROPERTY to VALUE.
Return modified element."


(defsubst org-element-set-contents (element &rest contents)
  "Set ELEMENT contents to CONTENTS.
Return modified element."


(defsubst org-element-set-element (old new)
  "Replace element or object OLD with element or object NEW.
The function takes care of setting ‘:parent’ property for NEW."
```

```
  ;; Since OLD is going to be changed into NEW by side-effect, first
  ;; make sure that every element or object within NEW has OLD as
  ;; parent.

(defsubst org-element-adopt-elements (parent &rest children)
  "Append elements to the contents of another element.

PARENT is an element or object.  CHILDREN can be elements,
objects, or a strings.

The function takes care of setting ‘:parent’ property for CHILD.
Return parent element."
```

6 Greater elements

For each greater element type, we define a parser and an interpreter.

A parser returns the element or object as the list described above. Most of them accepts no argument. Though, exceptions exist. Hence every element containing a secondary string (see ‘org-element-secondary-value-alist’) will accept an optional argument to toggle parsing of that secondary string. Moreover, ‘item’ parser requires current list’s structure as its first element.

An interpreter accepts two arguments: the list representation of the element or object, and its contents. The latter may be nil, depending on the element or object considered. It returns the appropriate Org syntax, as a string.

Parsing functions must follow the naming convention: org-element-TYPE-parser, where TYPE is greater element’s type, as defined in ‘org-element-greater-elements’.

Similarly, interpreting functions must follow the naming convention: org-element-TYPE-interpreter.

With the exception of ‘headline’ and ‘item’ types, greater elements cannot contain other greater elements of their own type.

Beside implementing a parser and an interpreter, adding a new greater element requires to tweak ‘org-element–current-element’. Moreover, the newly defined type must be added to both ‘org-element-all-elements’ and ‘org-element-greater-elements’.

6.1 Center Block

```
(defun org-element-center-block-parser (limit affiliated)
  "Parse a center block.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘center-block’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:hiddenp’, ‘:contents-begin’,
‘:contents-end’, ‘:post-blank’ and ‘:post-affiliated’ keywords.

Assume point is at the beginning of the block."
```

```
(defun org-element-center-block-interpreter (center-block contents)
  "Interpret CENTER-BLOCK element as Org syntax.
CONTENTS is the contents of the element."
```

6.2 Drawer

```
(defun org-element-drawer-parser (limit affiliated)
  "Parse a drawer.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘drawer’ and CDR is a plist containing
‘:drawer-name’, ‘:begin’, ‘:end’, ‘:hiddenp’, ‘:contents-begin’,
‘:contents-end’, ‘:post-blank’ and ‘:post-affiliated’ keywords.

Assume point is at beginning of drawer."

(defun org-element-drawer-interpreter (drawer contents)
  "Interpret DRAWER element as Org syntax.
CONTENTS is the contents of the element."
```

6.3 Dynamic Block

```
(defun org-element-dynamic-block-parser (limit affiliated)
  "Parse a dynamic block.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘dynamic-block’ and CDR is a plist
containing ‘:block-name’, ‘:begin’, ‘:end’, ‘:hiddenp’,
‘:contents-begin’, ‘:contents-end’, ‘:arguments’, ‘:post-blank’
and ‘:post-affiliated’ keywords.

Assume point is at beginning of dynamic block."

(defun org-element-dynamic-block-interpreter (dynamic-block contents)
  "Interpret DYNAMIC-BLOCK element as Org syntax.
CONTENTS is the contents of the element."
```

6.4 Footnote Definition

```
(defun org-element-footnote-definition-parser (limit affiliated)
  "Parse a footnote definition.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘footnote-definition’ and CDR is
a plist containing ‘:label’, ‘:begin’ ‘:end’, ‘:contents-begin’,
‘:contents-end’, ‘:post-blank’ and ‘:post-affiliated’ keywords.

Assume point is at the beginning of the footnote definition."


(defun org-element-footnote-definition-interpreter (footnote-definition contents)
  "Interpret FOOTNOTE-DEFINITION element as Org syntax.
CONTENTS is the contents of the footnote-definition."
```

6.5 Headline

```
(defun org-element-headline-parser (limit &optional raw-secondary-p)
  "Parse a headline.

Return a list whose CAR is ‘headline’ and CDR is a plist
containing ‘:raw-value’, ‘:title’, ‘:alt-title’, ‘:begin’,
‘:end’, ‘:pre-blank’, ‘:hiddenp’, ‘:contents-begin’ and
‘:contents-end’, ‘:level’, ‘:priority’, ‘:tags’,
‘:todo-keyword’,‘:todo-type’, ‘:scheduled’, ‘:deadline’,
‘:closed’, ‘:quotedp’, ‘:archivedp’, ‘:commentedp’ and
‘:footnote-section-p’ keywords.

The plist also contains any property set in the property drawer,
with its name in upper cases and colons added at the
beginning (i.e. ‘:CUSTOM_ID’).

When RAW-SECONDARY-P is non-nil, headline’s title will not be
parsed as a secondary string, but as a plain string instead.

Assume point is at beginning of the headline."

(defun org-element-headline-interpreter (headline contents)
  "Interpret HEADLINE element as Org syntax.
CONTENTS is the contents of the element."
```

6.6 Inlinetask

```
(defun org-element-inlinetask-parser (limit &optional raw-secondary-p)
  "Parse an inline task.

Return a list whose CAR is ‘inlinetask’ and CDR is a plist
containing ‘:title’, ‘:begin’, ‘:end’, ‘:hiddenp’,
‘:contents-begin’ and ‘:contents-end’, ‘:level’, ‘:priority’,
‘:raw-value’, ‘:tags’, ‘:todo-keyword’, ‘:todo-type’,
‘:scheduled’, ‘:deadline’, ‘:closed’ and ‘:post-blank’ keywords.

The plist also contains any property set in the property drawer,
with its name in upper cases and colons added at the
beginning (i.e. ‘:CUSTOM_ID’).

When optional argument RAW-SECONDARY-P is non-nil, inline-task’s
title will not be parsed as a secondary string, but as a plain
string instead.

Assume point is at beginning of the inline task."


(defun org-element-inlinetask-interpreter (inlinetask contents)
  "Interpret INLINETASK element as Org syntax.
CONTENTS is the contents of inlinetask."
```

6.7 Item

```
(defun org-element-item-parser (limit struct &optional raw-secondary-p)
  "Parse an item.

STRUCT is the structure of the plain list.

Return a list whose CAR is ‘item’ and CDR is a plist containing
‘:bullet’, ‘:begin’, ‘:end’, ‘:contents-begin’, ‘:contents-end’,
‘:checkbox’, ‘:counter’, ‘:tag’, ‘:structure’, ‘:hiddenp’ and
‘:post-blank’ keywords.

When optional argument RAW-SECONDARY-P is non-nil, item’s tag, if
any, will not be parsed as a secondary string, but as a plain
string instead.

Assume point is at the beginning of the item."


(defun org-element-item-interpreter (item contents)
  "Interpret ITEM element as Org syntax.
CONTENTS is the contents of the element."
```

6.8 Plain List

```
(defun org-element-plain-list-parser (limit affiliated structure)
  "Parse a plain list.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.  STRUCTURE is the structure of the plain list being
parsed.

Return a list whose CAR is ‘plain-list’ and CDR is a plist
containing ‘:type’, ‘:begin’, ‘:end’, ‘:contents-begin’ and
‘:contents-end’, ‘:structure’, ‘:post-blank’ and
‘:post-affiliated’ keywords.

Assume point is at the beginning of the list."
```

```
(defun org-element-plain-list-interpreter (plain-list contents)
  "Interpret PLAIN-LIST element as Org syntax.
CONTENTS is the contents of the element."
```

6.9 Property Drawer

```
(defun org-element-property-drawer-parser (limit affiliated)
  "Parse a property drawer.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘property-drawer’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:hiddenp’, ‘:contents-begin’,
‘:contents-end’, ‘:post-blank’ and ‘:post-affiliated’ keywords.
```

```
Assume point is at the beginning of the property drawer."

(defun org-element-property-drawer-interpreter (property-drawer contents)
  "Interpret PROPERTY-DRAWER element as Org syntax.
CONTENTS is the properties within the drawer."
```

6.10 Quote Block

```
(defun org-element-quote-block-parser (limit affiliated)
  "Parse a quote block.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is `quote-block' and CDR is a plist
containing `:begin', `:end', `:hiddenp', `:contents-begin',
`:contents-end', `:post-blank' and `:post-affiliated' keywords.

Assume point is at the beginning of the block."

(defun org-element-quote-block-interpreter (quote-block contents)
  "Interpret QUOTE-BLOCK element as Org syntax.
CONTENTS is the contents of the element."
```

6.11 Section

```
(defun org-element-section-parser (limit)
  "Parse a section.

LIMIT bounds the search.

Return a list whose CAR is `section' and CDR is a plist
containing `:begin', `:end', `:contents-begin', `contents-end'
and `:post-blank' keywords."
```

```
(defun org-element-section-interpreter (section contents)
  "Interpret SECTION element as Org syntax.
CONTENTS is the contents of the element."
  contents)
```

6.12 Special Block

```
(defun org-element-clock-interpreter (clock contents)
  "Interpret CLOCK element as Org syntax.
CONTENTS is nil."
```

6.13 Comment

```
(defun org-element-comment-parser (limit affiliated)
  "Parse a comment.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘comment’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:value’, ‘:post-blank’,
‘:post-affiliated’ keywords.

Assume point is at comment beginning."

(defun org-element-comment-interpreter (comment contents)
  "Interpret COMMENT element as Org syntax.
CONTENTS is nil."
```

6.14 Comment Block

```
(defun org-element-comment-block-parser (limit affiliated)
  "Parse an export block.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
```

```
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘comment-block’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:hiddenp’, ‘:value’, ‘:post-blank’
and ‘:post-affiliated’ keywords.

Assume point is at comment block beginning."

(defun org-element-comment-block-interpreter (comment-block contents)
  "Interpret COMMENT-BLOCK element as Org syntax.
CONTENTS is nil."
```

6.15 Diary Sexp

```
(defun org-element-diary-sexp-parser (limit affiliated)
  "Parse a diary sexp.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘diary-sexp’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:value’, ‘:post-blank’ and
‘:post-affiliated’ keywords."

(defun org-element-example-block-interpreter (example-block contents)
  "Interpret EXAMPLE-BLOCK element as Org syntax.
CONTENTS is nil."
```

6.16 Export Block

```
(defun org-element-export-block-parser (limit affiliated)
  "Parse an export block.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
```

```
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘export-block’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:type’, ‘:hiddenp’, ‘:value’,
‘:post-blank’ and ‘:post-affiliated’ keywords.

Assume point is at export-block beginning."

(defun org-element-export-block-interpreter (export-block contents)
  "Interpret EXPORT-BLOCK element as Org syntax.
CONTENTS is nil."
```

6.17 Fixed-width

```
(defun org-element-fixed-width-parser (limit affiliated)
  "Parse a fixed-width section.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘fixed-width’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:value’, ‘:post-blank’ and
‘:post-affiliated’ keywords.

Assume point is at the beginning of the fixed-width area."

(defun org-element-fixed-width-interpreter (fixed-width contents)
  "Interpret FIXED-WIDTH element as Org syntax.
CONTENTS is nil."
```

6.18 Horizontal Rule

```
(defun org-element-horizontal-rule-parser (limit affiliated)
  "Parse an horizontal rule.
```

```
LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘horizontal-rule’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:post-blank’ and ‘:post-affiliated’
keywords."


(defun org-element-horizontal-rule-interpreter (horizontal-rule contents)
  "Interpret HORIZONTAL-RULE element as Org syntax.
CONTENTS is nil."
  "-----")
```

6.19 Keyword

```
(defun org-element-keyword-interpreter (keyword contents)
  "Interpret KEYWORD element as Org syntax.
CONTENTS is nil."
```

6.20 Latex Environment

```
(defun org-element-latex-environment-parser (limit affiliated)
  "Parse a LaTeX environment.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘latex-environment’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:value’, ‘:post-blank’ and
‘:post-affiliated’ keywords.

Assume point is at the beginning of the latex environment."

(defun org-element-latex-environment-interpreter (latex-environment contents)
  "Interpret LATEX-ENVIRONMENT element as Org syntax.
CONTENTS is nil."
```

6.21 Node Property

```
(defun org-element-node-property-parser (limit)
  "Parse a node-property at point.

LIMIT bounds the search.

Return a list whose CAR is ‘node-property’ and CDR is a plist
containing ‘:key’, ‘:value’, ‘:begin’, ‘:end’ and ‘:post-blank’
keywords."

(defun org-element-node-property-interpreter (node-property contents)
  "Interpret NODE-PROPERTY element as Org syntax.
CONTENTS is nil."
```

6.22 Paragraph

```
(defun org-element-paragraph-parser (limit affiliated)
  "Parse a paragraph.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘paragraph’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:contents-begin’ and
‘:contents-end’, ‘:post-blank’ and ‘:post-affiliated’ keywords.

Assume point is at the beginning of the paragraph."


(defun org-element-paragraph-interpreter (paragraph contents)
  "Interpret PARAGRAPH element as Org syntax.
CONTENTS is the contents of the element."
```

```
  contents)
```

6.23 Planning

```
(defun org-element-planning-interpreter (planning contents)
  "Interpret PLANNING element as Org syntax.
CONTENTS is nil."
```

6.24 Quote Section

```
(defun org-element-quote-section-parser (limit)
  "Parse a quote section.

LIMIT bounds the search.

Return a list whose CAR is ‘quote-section’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:value’ and ‘:post-blank’ keywords.

Assume point is at beginning of the section."


(defun org-element-quote-section-interpreter (quote-section contents)
  "Interpret QUOTE-SECTION element as Org syntax.
CONTENTS is nil."
```

6.25 Src Block

```
(defun org-element-src-block-parser (limit affiliated)
  "Parse a src block.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘src-block’ and CDR is a plist
containing ‘:language’, ‘:switches’, ‘:parameters’, ‘:begin’,
‘:end’, ‘:hiddenp’, ‘:number-lines’, ‘:retain-labels’,
‘:use-labels’, ‘:label-fmt’, ‘:preserve-indent’, ‘:value’,
‘:post-blank’ and ‘:post-affiliated’ keywords.

Assume point is at the beginning of the block."


(defun org-element-src-block-interpreter (src-block contents)
  "Interpret SRC-BLOCK element as Org syntax.
CONTENTS is nil."
```

6.26 Table

```
(defun org-element-table-parser (limit affiliated)
  "Parse a table at point.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘table’ and CDR is a plist containing
‘:begin’, ‘:end’, ‘:tblfm’, ‘:type’, ‘:contents-begin’,
‘:contents-end’, ‘:value’, ‘:post-blank’ and ‘:post-affiliated’
keywords.

Assume point is at the beginning of the table."

(defun org-element-table-interpreter (table contents)
  "Interpret TABLE element as Org syntax.
CONTENTS is nil."
```

6.27 Table Row

```
(defun org-element-table-row-parser (limit)
  "Parse table row at point.

LIMIT bounds the search.
```

```
Return a list whose CAR is ‘table-row’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:contents-begin’, ‘:contents-end’,
‘:type’ and ‘:post-blank’ keywords."


(defun org-element-table-row-interpreter (table-row contents)
  "Interpret TABLE-ROW element as Org syntax.
CONTENTS is the contents of the table row."
```

6.28 Verse Block

```
(defun org-element-verse-block-parser (limit affiliated)
  "Parse a verse block.

LIMIT bounds the search.  AFFILIATED is a list of which CAR is
the buffer position at the beginning of the first affiliated
keyword and CDR is a plist of affiliated keywords along with
their value.

Return a list whose CAR is ‘verse-block’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:contents-begin’, ‘:contents-end’,
‘:hiddenp’, ‘:post-blank’ and ‘:post-affiliated’ keywords.

Assume point is at beginning of the block."

(defun org-element-verse-block-interpreter (verse-block contents)
  "Interpret VERSE-BLOCK element as Org syntax.
CONTENTS is verse block contents."
```

7 Objects

Unlike to elements, interstices can be found between objects. That’s why, along with the parser, successor functions are provided for each object. Some objects share the same successor (i.e. ‘code’ and ‘verbatim’ objects).

A successor must accept a single argument bounding the search. It will return either a cons cell whose CAR is the object’s type, as a symbol, and CDR the position of its next occurrence, or nil.

Successors follow the naming convention: org-element-NAME-successor, where NAME is the name of the successor, as defined in ‘org-element-all-successors’.

Some object types (i.e. ‘italic’) are recursive. Restrictions on object types they can contain will be specified in ‘org-element-object-restrictions’.

Adding a new type of object is simple. Implement a successor, a parser, and an interpreter for it, all following the naming convention. Register type in ‘org-element-all-objects’ and successor in ‘org-element-all-successors’. Maybe tweak restrictions about it, and that’s it.

7.1 Bold

```
(defun org-element-bold-parser ()
  "Parse bold object at point.

Return a list whose CAR is ‘bold’ and CDR is a plist with
‘:begin’, ‘:end’, ‘:contents-begin’ and ‘:contents-end’ and
‘:post-blank’ keywords.

Assume point is at the first star marker."


(defun org-element-bold-interpreter (bold contents)
  "Interpret BOLD object as Org syntax.
CONTENTS is the contents of the object."


(defun org-element-text-markup-successor (limit)
  "Search for the next text-markup object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is a symbol among ‘bold’,
‘italic’, ‘underline’, ‘strike-through’, ‘code’ and ‘verbatim’
and CDR is beginning position."
```

7.2 Code

```
(defun org-element-code-parser ()
  "Parse code object at point.

Return a list whose CAR is ‘code’ and CDR is a plist with
‘:value’, ‘:begin’, ‘:end’ and ‘:post-blank’ keywords.

Assume point is at the first tilde marker."


(defun org-element-code-interpreter (code contents)
  "Interpret CODE object as Org syntax.
CONTENTS is nil."
```

7.3 Entity

```
(defun org-element-entity-parser ()
  "Parse entity at point.

Return a list whose CAR is ‘entity’ and CDR a plist with
‘:begin’, ‘:end’, ‘:latex’, ‘:latex-math-p’, ‘:html’, ‘:latin1’,
‘:utf-8’, ‘:ascii’, ‘:use-brackets-p’ and ‘:post-blank’ as
keywords.

Assume point is at the beginning of the entity."

(defun org-element-entity-interpreter (entity contents)
  "Interpret ENTITY object as Org syntax.
CONTENTS is nil."


(defun org-element-latex-or-entity-successor (limit)
  "Search for the next latex-fragment or entity object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘entity’ or
‘latex-fragment’ and CDR is beginning position."
```

7.4 Export Snippet

```
(defun org-element-export-snippet-parser ()
  "Parse export snippet at point.

Return a list whose CAR is ‘export-snippet’ and CDR a plist with
‘:begin’, ‘:end’, ‘:back-end’, ‘:value’ and ‘:post-blank’ as
keywords.

Assume point is at the beginning of the snippet."
```

```
(defun org-element-export-snippet-interpreter (export-snippet contents)
  "Interpret EXPORT-SNIPPET object as Org syntax.
CONTENTS is nil."
```

```
(defun org-element-export-snippet-successor (limit)
  "Search for the next export-snippet object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘export-snippet’ and CDR
its beginning position."
```

7.5 Footnote Reference

```
(defun org-element-footnote-reference-parser ()
  "Parse footnote reference at point.

Return a list whose CAR is ‘footnote-reference’ and CDR a plist
with ‘:label’, ‘:type’, ‘:inline-definition’, ‘:begin’, ‘:end’
and ‘:post-blank’ as keywords."

(defun org-element-footnote-reference-interpreter (footnote-reference contents)
  "Interpret FOOTNOTE-REFERENCE object as Org syntax.
CONTENTS is nil."
```

```
(defun org-element-footnote-reference-successor (limit)
  "Search for the next footnote-reference object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘footnote-reference’ and
CDR is beginning position."
```

7.6 Inline Babel Call

```
(defun org-element-inline-babel-call-parser ()
  "Parse inline babel call at point.

Return a list whose CAR is ‘inline-babel-call’ and CDR a plist
with ‘:begin’, ‘:end’, ‘:info’ and ‘:post-blank’ as keywords.

Assume point is at the beginning of the babel call."


(defun org-element-inline-babel-call-interpreter (inline-babel-call contents)
  "Interpret INLINE-BABEL-CALL object as Org syntax.
CONTENTS is nil."


(defun org-element-inline-babel-call-successor (limit)
  "Search for the next inline-babel-call object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘inline-babel-call’ and
CDR is beginning position."
```

7.7 Inline Src Block

```
(defun org-element-inline-src-block-parser ()
  "Parse inline source block at point.

LIMIT bounds the search.
```

```
Return a list whose CAR is ‘inline-src-block’ and CDR a plist
with ‘:begin’, ‘:end’, ‘:language’, ‘:value’, ‘:parameters’ and
‘:post-blank’ as keywords.

Assume point is at the beginning of the inline src block."


(defun org-element-inline-src-block-interpreter (inline-src-block contents)
  "Interpret INLINE-SRC-BLOCK object as Org syntax.
CONTENTS is nil."


(defun org-element-inline-src-block-successor (limit)
  "Search for the next inline-babel-call element.

LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘inline-babel-call’ and
CDR is beginning position."
```

7.8 Italic

```
(defun org-element-italic-parser ()
  "Parse italic object at point.

Return a list whose CAR is ‘italic’ and CDR is a plist with
‘:begin’, ‘:end’, ‘:contents-begin’ and ‘:contents-end’ and
‘:post-blank’ keywords.

Assume point is at the first slash marker."


(defun org-element-italic-interpreter (italic contents)
  "Interpret ITALIC object as Org syntax.
CONTENTS is the contents of the object."
```

7.9 Latex Fragment

```
(defun org-element-latex-fragment-parser ()
  "Parse latex fragment at point.

Return a list whose CAR is ‘latex-fragment’ and CDR a plist with
‘:value’, ‘:begin’, ‘:end’, and ‘:post-blank’ as keywords.

Assume point is at the beginning of the latex fragment."


(defun org-element-latex-fragment-interpreter (latex-fragment contents)
  "Interpret LATEX-FRAGMENT object as Org syntax.
CONTENTS is nil."
```

7.10 Line Break

```
(defun org-element-line-break-parser ()
  "Parse line break at point.

Return a list whose CAR is ‘line-break’, and CDR a plist with
‘:begin’, ‘:end’ and ‘:post-blank’ keywords.

Assume point is at the beginning of the line break."


(defun org-element-line-break-interpreter (line-break contents)
  "Interpret LINE-BREAK object as Org syntax.
CONTENTS is nil."
  "\\\\\n")


(defun org-element-link-interpreter (link contents)
  "Interpret LINK object as Org syntax.
CONTENTS is the contents of the object, or nil."


(defun org-element-link-successor (limit)
  "Search for the next link object.
```

```
LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘link’ and CDR is
beginning position."


(defun org-element-plain-link-successor (limit)
  "Search for the next plain link object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘link’ and CDR is
beginning position."
```

7.11 Macro

```
(defun org-element-macro-parser ()
  "Parse macro at point.

Return a list whose CAR is ‘macro’ and CDR a plist with ‘:key’,
‘:args’, ‘:begin’, ‘:end’, ‘:value’ and ‘:post-blank’ as
keywords.

Assume point is at the macro."


(defun org-element-macro-interpreter (macro contents)
  "Interpret MACRO object as Org syntax.
CONTENTS is nil."


(defun org-element-macro-successor (limit)
  "Search for the next macro object.

LIMIT bounds the search.

Return value is cons cell whose CAR is ‘macro’ and CDR is
beginning position."
```

7.12 Radio-target

```
(defun org-element-radio-target-parser ()
  "Parse radio target at point.

Return a list whose CAR is ‘radio-target’ and CDR a plist with
‘:begin’, ‘:end’, ‘:contents-begin’, ‘:contents-end’, ‘:value’
and ‘:post-blank’ as keywords.

Assume point is at the radio target."


(defun org-element-radio-target-interpreter (target contents)
  "Interpret TARGET object as Org syntax.
CONTENTS is the contents of the object."


(defun org-element-radio-target-successor (limit)
  "Search for the next radio-target object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘radio-target’ and CDR
is beginning position."
```

7.13 Statistics Cookie

```
(defun org-element-statistics-cookie-parser ()
  "Parse statistics cookie at point.

Return a list whose CAR is ‘statistics-cookie’, and CDR a plist
with ‘:begin’, ‘:end’, ‘:value’ and ‘:post-blank’ keywords.

Assume point is at the beginning of the statistics-cookie."


(defun org-element-statistics-cookie-interpreter (statistics-cookie contents)
  "Interpret STATISTICS-COOKIE object as Org syntax.
CONTENTS is nil."
```

```
(defun org-element-statistics-cookie-successor (limit)
  "Search for the next statistics cookie object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘statistics-cookie’ and
CDR is beginning position."
```

7.14 Strike-Through

```
(defun org-element-strike-through-parser ()
  "Parse strike-through object at point.

Return a list whose CAR is ‘strike-through’ and CDR is a plist
with ‘:begin’, ‘:end’, ‘:contents-begin’ and ‘:contents-end’ and
‘:post-blank’ keywords.

Assume point is at the first plus sign marker."
```

```
(defun org-element-strike-through-interpreter (strike-through contents)
  "Interpret STRIKE-THROUGH object as Org syntax.
CONTENTS is the contents of the object."
```

7.15 Subscript

```
(defun org-element-subscript-parser ()
  "Parse subscript at point.

Return a list whose CAR is ‘subscript’ and CDR a plist with
‘:begin’, ‘:end’, ‘:contents-begin’, ‘:contents-end’,
‘:use-brackets-p’ and ‘:post-blank’ as keywords.

Assume point is at the underscore."
```

```
(defun org-element-subscript-interpreter (subscript contents)
  "Interpret SUBSCRIPT object as Org syntax.
CONTENTS is the contents of the object."
```

```
(defun org-element-sub/superscript-successor  (limit)
  "Search for the next sub/superscript object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is either ‘subscript’ or
‘superscript’ and CDR is beginning position."
```

7.16 Superscript

```
(defun org-element-superscript-parser ()
  "Parse superscript at point.

Return a list whose CAR is ‘superscript’ and CDR a plist with
‘:begin’, ‘:end’, ‘:contents-begin’, ‘:contents-end’,
‘:use-brackets-p’ and ‘:post-blank’ as keywords.

Assume point is at the caret."
```

```
(defun org-element-superscript-interpreter (superscript contents)
  "Interpret SUPERSCRIPT object as Org syntax.
CONTENTS is the contents of the object."
```

7.17 Table Cell

```
(defun org-element-table-cell-parser ()
  "Parse table cell at point.

Return a list whose CAR is ‘table-cell’ and CDR is a plist
containing ‘:begin’, ‘:end’, ‘:contents-begin’, ‘:contents-end’
```

```
and ‘:post-blank’ keywords."
    (let* ((begin (match-beginning 0))
         (end (match-end 0))
         (contents-begin (match-beginning 1))
         (contents-end (match-end 1)))
    (list 'table-cell
          (list :begin begin
                :end end
                :contents-begin contents-begin
                :contents-end contents-end
                :post-blank 0))))

(defun org-element-table-cell-interpreter (table-cell contents)
  "Interpret TABLE-CELL element as Org syntax.
CONTENTS is the contents of the cell, or nil."


(defun org-element-table-cell-successor (limit)
  "Search for the next table-cell object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘table-cell’ and CDR is
beginning position."
```

7.18 Target

```
(defun org-element-target-parser ()
  "Parse target at point.

Return a list whose CAR is ‘target’ and CDR a plist with
‘:begin’, ‘:end’, ‘:value’ and ‘:post-blank’ as keywords.

Assume point is at the target."


(defun org-element-target-interpreter (target contents)
  "Interpret TARGET object as Org syntax.
CONTENTS is nil."
```

```
(defun org-element-target-successor (limit)
  "Search for the next target object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘target’ and CDR is
beginning position."
```

7.19 Timestamp

```
(defun org-element-timestamp-parser ()
  "Parse time stamp at point.

Return a list whose CAR is ‘timestamp’, and CDR a plist with
‘:type’, ‘:begin’, ‘:end’, ‘:value’ and ‘:post-blank’ keywords.

Assume point is at the beginning of the timestamp."


(defun org-element-timestamp-interpreter (timestamp contents)
  "Interpret TIMESTAMP object as Org syntax.
CONTENTS is nil."
  ;; Use ‘:raw-value’ if specified.


(defun org-element-timestamp-successor (limit)
  "Search for the next timestamp object.

LIMIT bounds the search.

Return value is a cons cell whose CAR is ‘timestamp’ and CDR is
beginning position."
```

7.20 Underline

```
(defun org-element-underline-parser ()
  "Parse underline object at point.

Return a list whose CAR is ‘underline’ and CDR is a plist with
‘:begin’, ‘:end’, ‘:contents-begin’ and ‘:contents-end’ and
‘:post-blank’ keywords.

Assume point is at the first underscore marker."


(defun org-element-underline-interpreter (underline contents)
  "Interpret UNDERLINE object as Org syntax.
CONTENTS is the contents of the object."
```

7.21 Verbatim

```
(defun org-element-verbatim-parser ()
  "Parse verbatim object at point.

Return a list whose CAR is ‘verbatim’ and CDR is a plist with
‘:value’, ‘:begin’, ‘:end’ and ‘:post-blank’ keywords.

Assume point is at the first equal sign marker."


(defun org-element-verbatim-interpreter (verbatim contents)
  "Interpret VERBATIM object as Org syntax.
CONTENTS is nil."
```

8 Parsing Element Starting At Point

‘org-element–current-element’ is the core function of this section. It returns the Lisp representation of the element starting at point.

‘org-element–current-element’ makes use of special modes. They are activated for fixed element chaining (i.e. ‘plain-list’ > ‘item’) or fixed conditional element chaining (i.e. ‘headline’ > ‘section’). Special modes are: ‘first-section’, ‘item’, ‘node-property’, ‘quote-section’, ‘section’ and ‘table-row’.

```
(defun org-element--current-element
  (limit &optional granularity special structure)
  "Parse the element starting at point.

LIMIT bounds the search.

Return value is a list like (TYPE PROPS) where TYPE is the type
of the element and PROPS a plist of properties associated to the
element.

Possible types are defined in ‘org-element-all-elements’.

Optional argument GRANULARITY determines the depth of the
recursion.  Allowed values are ‘headline’, ‘greater-element’,
‘element’, ‘object’ or nil.  When it is broader than ‘object’ (or
nil), secondary values will not be parsed, since they only
contain objects.

Optional argument SPECIAL, when non-nil, can be either
‘first-section’, ‘item’, ‘node-property’, ‘quote-section’,
‘section’, and ‘table-row’.

If STRUCTURE isn’t provided but SPECIAL is set to ‘item’, it will
be computed.

This function assumes point is always at the beginning of the
element it has to parse."
```

Most elements can have affiliated keywords. When looking for an element beginning, we want to move before them, as they belong to that element, and, in the meantime, collect information they give into appropriate properties. Hence the following function.

```
(defun org-element--collect-affiliated-keywords (limit)
  "Collect affiliated keywords from point down to LIMIT.

Return a list whose CAR is the position at the first of them and
CDR a plist of keywords and values and move point to the
beginning of the first line after them.
```

```
As a special case, if element doesn't start at the beginning of
the line (i.e. a paragraph starting an item), CAR is current
position of point and CDR is nil."
```

9 The Org Parser

The two major functions here are ‘org-element-parse-buffer’, which parses Org syntax inside the current buffer, taking into account region, narrowing, or even visibility if specified, and ‘org-element-parse-secondary-string’, which parses objects within a given string.

The (almost) almighty ‘org-element-map’ allows to apply a function on elements or objects matching some type, and accumulate the resulting values. In an export situation, it also skips unneeded parts of the parse tree.

```
(defun org-element-parse-buffer (&optional granularity visible-only)
  "Recursively parse the buffer and return structure.
If narrowing is in effect, only parse the visible part of the
buffer.

Optional argument GRANULARITY determines the depth of the
recursion.  It can be set to the following symbols:

‘headline’          Only parse headlines.
‘greater-element’   Don't recurse into greater elements excepted
                    headlines and sections.  Thus, elements
                    parsed are the top-level ones.
‘element’           Parse everything but objects and plain text.
‘object’            Parse the complete buffer (default).

When VISIBLE-ONLY is non-nil, don't parse contents of hidden
elements.

An element or an objects is represented as a list with the
pattern (TYPE PROPERTIES CONTENTS), where :

  TYPE is a symbol describing the element or object.  See
  ‘org-element-all-elements’ and ‘org-element-all-objects’ for an
  exhaustive list of such symbols.  One can retrieve it with
  ‘org-element-type’ function.

  PROPERTIES is the list of attributes attached to the element or
  object, as a plist.  Although most of them are specific to the
  element or object type, all types share ‘:begin’, ‘:end’,
  ‘:post-blank’ and ‘:parent’ properties, which respectively
  refer to buffer position where the element or object starts,
  ends, the number of white spaces or blank lines after it, and
  the element or object containing it.  Properties values can be
  obtained by using ‘org-element-property’ function.

  CONTENTS is a list of elements, objects or raw strings
  contained in the current element or object, when applicable.
  One can access them with ‘org-element-contents’ function.

The Org buffer has ‘org-data’ as type and nil as properties.
‘org-element-map’ function can be used to find specific elements
or objects within the parse tree.

This function assumes that current major mode is ‘org-mode’."


(defun org-element-parse-secondary-string (string restriction &optional parent)
  "Recursively parse objects in STRING and return structure.

RESTRICTION is a symbol limiting the object types that will be
looked after.

Optional argument PARENT, when non-nil, is the element or object
containing the secondary string.  It is used to set correctly
‘:parent’ property within the string."
  ;; Copy buffer-local variables listed in
  ;; ‘org-element-object-variables’ into temporary buffer.  This is
  ;; required since object parsing is dependent on these variables.


(defun org-element-map
  (data types fun &optional info first-match no-recursion with-affiliated)
  "Map a function on selected elements or objects.
DATA is a parse tree, an element, an object, a string, or a list
of such constructs.  TYPES is a symbol or list of symbols of
elements or objects types (see ‘org-element-all-elements’ and
‘org-element-all-objects’ for a complete list of types).  FUN is
the function called on the matching element or object.  It has to
accept one argument: the element or object itself.

When optional argument INFO is non-nil, it should be a plist
holding export options.  In that case, parts of the parse tree
not exportable according to that property list will be skipped.

When optional argument FIRST-MATCH is non-nil, stop at the first
match for which FUN doesn’t return nil, and return that value.

Optional argument NO-RECURSION is a symbol or a list of symbols
representing elements or objects types.  ‘org-element-map’ won’t
enter any recursive element or object whose type belongs to that
list.  Though, FUN can still be applied on them.

When optional argument WITH-AFFILIATED is non-nil, FUN will also
apply to matching objects within parsed affiliated keywords (see
‘org-element-parsed-keywords’).

Nil values returned from FUN do not appear in the results.


Examples:
---------

Assuming TREE is a variable containing an Org buffer parse tree,
the following example will return a flat list of all ‘src-block’
and ‘example-block’ elements in it:

  \(org-element-map tree ’(example-block src-block) ’identity)

The following snippet will find the first headline with a level
of 1 and a \"phone\" tag, and will return its beginning position:

  \          \(org-element-property :begin hl)))
   nil t)
```

```
The next example will return a flat list of all ‘plain-list’ type
elements in TREE that are not a sub-list themselves:

  \(org-element-map tree 'plain-list 'identity nil nil 'plain-list)

Eventually, this example will return a flat list of all ‘bold’
type objects containing a ‘latex-snippet’ type object, even
looking into captions:

  \(org-element-map tree 'bold
   \(lambda (b)
     \(and (org-element-map b 'latex-snippet 'identity nil t) b))
   nil nil nil t)"
  ;; Ensure TYPES and NO-RECURSION are a list, even of one element.
  (unless (listp types) (setq types (list types)))
  (unless (listp no-recursion) (setq no-recursion (list no-recursion)))
  ;; Recursion depth is determined by --CATEGORY.
  (let* ((--category
          (catch 'found
            (let ((category 'greater-elements))
              (mapc (lambda (type)
                      (cond ((or (memq type org-element-all-objects)
                                 (eq type 'plain-text))
                             ;; If one object is found, the function
                             ;; has to recurse into every object.
                             (throw 'found 'objects))
                            ((not (memq type org-element-greater-elements))
                             ;; If one regular element is found, the
                             ;; function has to recurse, at least,
                             ;; into every element it encounters.
                             (and (not (eq category 'elements))
                                  (setq category 'elements)))))
                    types)
              category)))
         ;; Compute properties for affiliated keywords if necessary.
         (--affiliated-alist
          (and with-affiliated
               (mapcar (lambda (kwd)
                         (cons kwd (intern (concat ":" (downcase kwd)))))
                       org-element-affiliated-keywords)))
         --acc
         --walk-tree
         (--walk-tree
          (function
           (lambda (--data)
             ;; Recursively walk DATA.  INFO, if non-nil, is a plist
             ;; holding contextual information.
             (let ((--type (org-element-type --data)))
               (cond
                ((not --data))
                ;; Ignored element in an export context.
                ((and info (memq --data (plist-get info :ignore-list))))
                ;; List of elements or objects.
                ((not --type) (mapc --walk-tree --data))
                ;; Unconditionally enter parse trees.
                ((eq --type 'org-data)
                 (mapc --walk-tree (org-element-contents --data)))
                (t
                 ;; Check if TYPE is matching among TYPES.  If so,
                 ;; apply FUN to --DATA and accumulate return value
                 ;; into --ACC (or exit if FIRST-MATCH is non-nil).
                 (when (memq --type types)
                   (let ((result (funcall fun --data)))
                     (cond ((not result))
                           (first-match (throw '--map-first-match result))
                           (t (push result --acc)))))
                 ;; If --DATA has a secondary string that can contain
                 ;; objects with their type among TYPES, look into it.
                 (when (and (eq --category 'objects) (not (stringp --data)))
                   (let ((sec-prop
                          (assq --type org-element-secondary-value-alist)))
                     (when sec-prop
                       (funcall --walk-tree
                                (org-element-property (cdr sec-prop) --data)))))
                 ;; If --DATA has any affiliated keywords and
                 ;; WITH-AFFILIATED is non-nil, look for objects in
                 ;; them.
                 (when (and with-affiliated
                            (eq --category 'objects)
```

```
                        (memq --type org-element-all-elements))
               (mapc (lambda (kwd-pair)
                       (let ((kwd (car kwd-pair))
                             (value (org-element-property
                                     (cdr kwd-pair) --data)))
                         ;; Pay attention to the type of value.
                         ;; Preserve order for multiple keywords.
                         (cond
                          ((not value))
                          ((and (member kwd org-element-multiple-keywords)
                                (member kwd org-element-dual-keywords))
                           (mapc (lambda (line)
                                   (funcall --walk-tree (cdr line))
                                   (funcall --walk-tree (car line)))
                                 (reverse value)))
                          ((member kwd org-element-multiple-keywords)
                           (mapc (lambda (line) (funcall --walk-tree line))
                                 (reverse value)))
                          ((member kwd org-element-dual-keywords)
                           (funcall --walk-tree (cdr value))
                           (funcall --walk-tree (car value)))
                          (t (funcall --walk-tree value)))))
                     --affiliated-alist))
             ;; Determine if a recursion into --DATA is possible.
             (cond
              ;; --TYPE is explicitly removed from recursion.
              ((memq --type no-recursion))
              ;; --DATA has no contents.
              ((not (org-element-contents --data)))
              ;; Looking for greater elements but --DATA is simply
              ;; an element or an object.
              ((and (eq --category 'greater-elements)
                    (not (memq --type org-element-greater-elements))))
              ;; Looking for elements but --DATA is an object.
              ((and (eq --category 'elements)
                    (memq --type org-element-all-objects)))
              ;; In any other case, map contents.
              (t (mapc --walk-tree (org-element-contents --data))))))))))
(catch '--map-first-match
  (funcall --walk-tree data)
```

```
      ;; Return value in a proper order.
      (nreverse --acc))))
(put 'org-element-map 'lisp-indent-function 2)
```

The following functions are internal parts of the parser.

The first one, ‘org-element–parse-elements’ acts at the element’s level.

The second one, ‘org-element–parse-objects’ applies on all objects of a paragraph or a secondary string. It uses ‘org-element–get-next-object-candidates’ to optimize the search of the next object in the buffer.

More precisely, that function looks for every allowed object type first. Then, it discards failed searches, keeps further matches, and searches again types matched behind point, for subsequent calls. Thus, searching for a given type fails only once, and every object is searched only once at top level (but sometimes more for nested types).

```
(defun org-element--parse-elements
  (beg end special structure granularity visible-only acc)
  "Parse elements between BEG and END positions.

SPECIAL prioritize some elements over the others.  It can be set
to ‘first-section’, ‘quote-section’, ‘section’ ‘item’ or
‘table-row’.

When value is ‘item’, STRUCTURE will be used as the current list
structure.

GRANULARITY determines the depth of the recursion.  See
‘org-element-parse-buffer’ for more information.

When VISIBLE-ONLY is non-nil, don’t parse contents of hidden
elements.

Elements are accumulated into ACC."


(defun org-element--parse-objects (beg end acc restriction)
  "Parse objects between BEG and END and return recursive structure.

Objects are accumulated in ACC.
```

```
RESTRICTION is a list of object successors which are allowed in
the current object."


(defun org-element--get-next-object-candidates (limit restriction objects)
  "Return an alist of candidates for the next object.

LIMIT bounds the search, and RESTRICTION narrows candidates to
some object successors.

OBJECTS is the previous candidates alist.  If it is set to
‘initial’, no search has been done before, and all symbols in
RESTRICTION should be looked after.

Return value is an alist whose CAR is the object type and CDR its
beginning position."
```

10 Towards A Bijective Process

The parse tree obtained with ‘org-element-parse-buffer’ is really a snapshot of the corresponding Org buffer. Therefore, it can be interpreted and expanded into a string with canonical Org syntax. Hence ‘org-element-interpret-data’.

The function relies internally on ‘org-element–interpret-affiliated-keywords’.

###autoload

```
(defun org-element-interpret-data (data &optional parent)
  "Interpret DATA as Org syntax.

DATA is a parse tree, an element, an object or a secondary string
to interpret.

Optional argument PARENT is used for recursive calls.  It contains
the element or object containing data, or nil.

Return Org syntax as a string."


(defun org-element--interpret-affiliated-keywords (element)
  "Return ELEMENT’s affiliated keywords as Org syntax.
```

```
If there is no affiliated keyword, return the empty string."
```

Because interpretation of the parse tree must return the same number of blank lines between elements and the same number of white space after objects, some special care must be given to white spaces.

The first function, ‘org-element-normalize-string’, ensures any string different from the empty string will end with a single newline character.

The second function, ‘org-element-normalize-contents’, removes global indentation from the contents of the current element.

```
(defun org-element-normalize-string (s)
  "Ensure string S ends with a single newline character.

If S isn’t a string return it unchanged.  If S is the empty
string, return it.  Otherwise, return a new string with a single
newline character at its end."


(defun org-element-normalize-contents (element &optional ignore-first)
  "Normalize plain text in ELEMENT’s contents.

ELEMENT must only contain plain text and objects.

If optional argument IGNORE-FIRST is non-nil, ignore first line’s
indentation to compute maximal common indentation.

Return the normalized element that is element with global
indentation removed from its contents.  The function assumes that
indentation is not done with TAB characters."
```

11 The Toolbox

The first move is to implement a way to obtain the smallest element containing point. This is the job of ‘org-element-at-point’. It basically jumps back to the beginning of section containing point and moves, element after element, with ‘org-element–current-element’ until the container is found. Note: When using ‘org-element-at-point’, secondary values are never parsed since the function focuses on elements, not on objects.

At a deeper level, 'org-element-context' lists all elements and objects containing point.

'org-element-nested-p' and 'org-element-swap-A-B' may be used internally by navigation and manipulation tools.

###autoload

```
(defun org-element-at-point (&optional keep-trail)
  "Determine closest element around point.

Return value is a list like (TYPE PROPS) where TYPE is the type
of the element and PROPS a plist of properties associated to the
element.

Possible types are defined in 'org-element-all-elements'.
Properties depend on element or object type, but always include
':begin', ':end', ':parent' and ':post-blank' properties.

As a special case, if point is at the very beginning of a list or
sub-list, returned element will be that list instead of the first
item.  In the same way, if point is at the beginning of the first
row of a table, returned element will be the table instead of the
first row.

If optional argument KEEP-TRAIL is non-nil, the function returns
a list of elements leading to element at point.  The list's CAR
is always the element at point.  The following positions contain
element's siblings, then parents, siblings of parents, until the
first element of current section."
```

###autoload

```
(defun org-element-context (&optional element)
  "Return closest element or object around point.

Return value is a list like (TYPE PROPS) where TYPE is the type
of the element or object and PROPS a plist of properties
associated to it.

Possible types are defined in 'org-element-all-elements' and
```

```
‘org-element-all-objects’.  Properties depend on element or
object type, but always include ‘:begin’, ‘:end’, ‘:parent’ and
‘:post-blank’.

Optional argument ELEMENT, when non-nil, is the closest element
containing point, as returned by ‘org-element-at-point’.
Providing it allows for quicker computation."
```

```
(defun org-element-nested-p (elem-A elem-B)
  "Non-nil when elements ELEM-A and ELEM-B are nested."
```

```
(defun org-element-swap-A-B (elem-A elem-B)
  "Swap elements ELEM-A and ELEM-B.
Assume ELEM-B is after ELEM-A in the buffer.  Leave point at the
end of ELEM-A."
    ;; There are two special cases when an element doesn’t start at bol:
  ;; the first paragraph in an item or in a footnote definition.
```

```
(provide ’org-element)
```

Local variables: generated-autoload-file: "org-loaddefs.el" End:

12 org-element.el ends here