Editing SGML with Emacs and PSGML

Lennart Staflin

1 Introduction

PSGML is a major mode for editing SGML and XML documents. It works with GNU Emacs 19.34, 20.3 and later or with XEmacs 19.9 and later. PSGML contains a simple SGML parser and can work with any DTD. Functions provided includes menus and commands for inserting tags with only the contextually valid tags, identification of structural errors, editing of attribute values in a separate window with information about types and defaults, and structure based editing.

SGML, a language for encoding the structure of a document, is an ISO standard: ISO 8879:1986 "Information processing – Text and office systems – Standard Generalized Markup Language (SGML)".

A good introduction to SGML is *A Gentle Introduction to SGML* produced by Text Encoding Initiative (this is really chapter 2 of TEI P3). This can be found on

'`ftp://ftp.ifi.uio.no/pub/SGML/TEI/P3SG.DOC`'.

A SGML document has three major parts, in order:

1. SGML Declaration ('`<!SGML "ISO 8879:1986" ... >`')
2. Document Type Declaration ('`<!DOCTYPE` *name* `... >`')
3. Document Element ('<*name*> ... </*name*>')

The SGML declaration contains general information about character sets, concrete syntax, and SGML features used. PSGML does not use the SGML Declaration, it can be left out, and if included is ignored. Many SGML systems allow the SGML declaration to be defaulted. PSGML always use the Concrete Reference Syntax but without limitations on lengths. Features used has to be indicated with variables (see Chapter 6 [SGML declaration], page 7).

The document type declaration specifies the valid elements and entities and how they can be nested. A document type is usually needed, but can reside in another file (see Chapter 7 [Managing the DTD], page 8).

The system declaration for PSGML:

```
SYSTEM "ISO 8879:1986"
   CHARSET
     BASESET  "ISO 646-1983//CHARSET
               International Reference Version (IRV)//ESC 2/5 4/0"
     DESCSET  0 128 0
     CAPACITY PUBLIC  "ISO 8879:1986//CAPACITY Reference//EN"
   FEATURES
     MINIMIZE DATATAG NO  OMITTAG  YES   RANK     NO  SHORTTAG YES
     LINK     SIMPLE  NO  IMPLICIT NO    EXPLICIT NO
     OTHER    CONCUR  NO  SUBDOC   YES 1 FORMAL   YES
     SCOPE    DOCUMENT
     SYNTAX   PUBLIC  "ISO 8879:1986//SYNTAX Reference//EN"
     VALIDATE
              GENERAL NO  MODEL    NO   EXCLUDE  NO  CAPACITY NO
              NONSGML NO  SGML     NO   FORMAL   NO
     SDIF     PACK    NO  UNPACK   NO
```

2 Installing PSGML

To install PSGML you first need to uncompress and unpack the source archive. This is done with the `gunzip` and `tar` commands.

```
gunzip psgml-1.2.0.tar.gz; tar xf psgml-1.2.0.tar
```

This should create a subdirectory to the current directory with the source code. This directory contains a `configure` command (see the file INSTALL for more information about configure). You can use the `configure` command to configure the package or you can load the file '`psgml-maint`' and execute the `psgml-compile-files` command.

Place the '`*.el`' and the '`*.elc`' files in a directory where Emacs can find it (i.e. one of the directories in the `load-path` variable, you can add a directory to this variable in your '`.emacs`'.)

If you use the `configure` approach, compile psgml with `make` and the you can run `make install` to install it in the system library '`site-lisp`'. The location of '`site-lisp`' is figured out by `configure`, but you can change it in the '`Makefile`'.

Put the following line in your .emacs:

```
(autoload 'sgml-mode "psgml" "Major mode to edit SGML files." t)
(autoload 'xml-mode "psgml" "Major mode to edit XML files." t)
```

You may also want to set up search paths for external entities, See Chapter 4 [Entity manager], page 4.

The '`psgml.info`' is the documentation for PSGML in the info format. You can read this with the Emacs command *C-u C-h i*. You can also install the file in your systems info directory and edit the '`dir`' file to include '`psgml.info`' in the menu.

The info file '`psgml.info`' is created from the texinfo file '`psgml.texi`'. The texinfo file can also be used to create a hard copy of the documentation. To do this you need the TeX program and a copy of '`texinfo.tex`'.

3 How to invoke PSGML

PSGML defines major modes called `sgml-mode` and `xml-mode`. Files with extensions '`.sgml`', '`.sgm`' or '`.dtd`' will automatically be edited in SGML mode. To edit some other file in SGML mode, type *`M-x sgml-mode`* ⟨RET⟩ after finding the file. To edit XML files, type *`M-x xml-mode`* ⟨RET⟩.

If you can modify the file you can add a *Local Variables* list (see section "Local Variables in Files" in *The Emacs Editor*) to the end of the file. This can make Emacs automatically set sgml mode and user options when the file is loaded. The simplest Local Variables list would look like:

```
<!--
Local Variables:
mode: sgml
End:
-->
```

You can also put a line at the top of the file to tell emacs to use sgml mode:

```
<!-- -*- sgml -*- -->
```

For XML replace '`sgml`' with '`xml`' in the above examples. But remember that you can't have a comment before the *SGML declaration* or the *XML declaration*.

4 The Entity Manager

SGML can refer to an external file (really entity) with an *external identifier*, this is a *public identifier* or a *system identifier*, or both.

A typical public identifier looks like

```
PUBLIC "ISO 8879:1986//ENTITIES Added Latin 1//EN"
```

where "ISO 8879:1986" is the owner, "ENTITIES" is the text class and "Added Latin 1" is the text description (and "EN" is language).

A system identifier looks like

```
SYSTEM "htmlplus.dtd"
```

where "htmlplus.dtd" is a system-specific identifier.

To map external identifiers to file names, PSGML first searches entity catalog files and then search the list of file name templates in the variable `sgml-public-map`.

The catalog format is according to SGML/Opens resolution on entity management. The catalog consists of a series of entries and comments. A comment is delimited by '`--`' like in a markup declaration. The entry types recognized are described in the following table.

'`public` *pubid file*'
: The *file* will be used for the entity text of an entity with the public identifier *pubid*.

'`entity` *name file*'
: The *file* will be used for the entity text of an entity with the name *name*. If the *name* starts with a '`%`' the rest of the name will be matched against parameter entities.

'`doctype` *name file*'
: The *file* will be used for the entity text of an entity used as external subset of a document declaration with *name* as document type name.

'`sgmldecl` *file*'
: Used to specify a default SGML declaration. Recognized but not used by PSGML other than to pass to an external validation command (`sgml-validate-command`).

When PSGML is looking for the file containing an external entity, the following things will be tried in order:

1. Try the system identifier, as a file name, if there is a system identifier and the variable `sgml-system-identifiers-are-preferred` is non-`nil` and there is no elements containing '`%s`' in `sgml-public-map`. If the system identifier is a relative file name it will be relative to the directory containing the defining entity.
2. Look thru each catalog in `sgml-local-catalogs` and `sgml-catalog-files` in order. For each catalog look first for entries matching the public identifier, if any. Then look for other matching entries in the order they appear in the catalog.

   Currently an entry will be ignored if it is matching but its file is non-existent or unreadable. (This is under reconsideration, perhaps it should signal error instead).

3. Try the system identifier, if any, as a file name. If `sgml-system-identifiers-are-preferred` is `nil` and there is no elements containing '`%s`' in `sgml-public-map`.
4. Try the entries in `sgml-public-map`. Using the catalogs are preferred. The `sgml-public-map` may disappear in a future version of PSGML (not soon though).

The `sgml-public-map` variable can contain a list of file name templates where '`%P`' will be substituted with the whole public identifier, owner is substituted for '`%O`', public text class for '`%C`', and public text description for '`%D`'. The text class will be converted to lower case and the owner and description will be transliterated according to the variable `sgml-public-transliterations`. The templates in the list is tried in order until an existing file is found. The `sgml-public-map` is modeled after '`sgmls`' environment variable `SGML_PATH` and psgml understand the following substitution characters: %%, %N, %P, %S, %Y, %C, %L, %O, %T, and %V. The default value of `sgml-public-map` is taken from the environment variable `SGML_PATH`.

Given the public identifier above and the file name template '`/usr/local/lib/sgml/%o/%c/%d`', the resulting file name is

```
/usr/local/lib/sgml/ISO_8879:1986/entities/Added_Latin_1
```

Note: blanks are transliterated to '`_`' (and also '`/`' to '`%`') and the text class is down cased.

sgml-catalog-files User Option

This is a list of catalog entry files. The files are in the format defined in the SGML Open Draft Technical Resolution on Entity Management. The Emacs variable is initialized from the environment variable `SGML_CATALOG_FILES` or if this variable is undefined the default is

```
("CATALOG" "/usr/local/lib/sgml/CATALOG")
```

sgml-local-catalogs User Option

A list of SGML entity catalogs to be searched first when parsing the buffer. This is used in addition to `sgml-catalog-files`, and `sgml-public-map`. This variable is automatically local to the buffer.

sgml-system-identifiers-are-preferred User Option

If `nil`, PSGML will look up external entities by searching the catalogs in `sgml-local-catalogs` and `sgml-catalog-files` and only if the entity is not found in the catalogs will a given system identifier be used. If the variable is non-nil and a system identifier is given, the system identifier will be used for the entity. If no system identifier is given the catalogs will searched.

sgml-public-map User Option

This should be a list of file name templates. This variable is initialized from the environment variable `SGML_PATH`. This is the same environment variable that '`sgmls`' uses. If the environment variable is undefined the default is

```
("%S" "/usr/local/lib/sgml/%o/%c/%d")
```

5 Running an external SGML parser

PSGML can not validate an SGML document (see below what it can and can't do). If you have a validating SGML parser, like '`sgmls`', you can run the parser on your file with the command *C-c C-v* (`sgml-validate`).

Some variables control this function:

sgml-validate-command User Option

The shell command to validate an SGML document.

This is a `format` control string that by default should contain two `%s` conversion specifications: the first will be replaced by the value of `sgml-declaration` (or the empty string, if nil); the second will be replaced by the current buffer's file name (or the empty string, if nil).

If `sgml-validate-files` is non-nil, the format string should contain one `%s` conversion specification for each element of its result.

If sgml-validate-command is a list, then every element should be a string. The strings will be tried in order and '`%`'-sequences in the string will be replaced according to the list below, if the string contains '`%`'-sequences with no replacement value the next string will be tried.

`%b` means the visited file of the current buffer

`%s` means the SGML declaration specified in the sgml-declaration variable

`%d` means the file containing the DOCTYPE declaration, if not in the buffer

The default value is `nsgmls -s %s %s`.

sgml-validate-files User Option

If non-nil, a function of no arguments that returns a list of file names. These file names will serve as the arguments to the `sgml-validate-command` format control string instead of the defaults.

sgml-declaration User Option

The name of the SGML declaration file.

sgml-offer-save User Option

If non-nil, *C-c C-v* (`sgml-validate`) will ask about saving modified buffers before running the validate command. The default value is `t`.

The built-in parser can find some markup errors. The command *C-c C-o* (`sgml-next-trouble-spot`) is the best way to use the built-in parser for this. To check the whole file go to the beginning of the buffer and use *C-c C-o*.

Some of the markup errors not found are:

- Errors in the SGML declaration.
- Errors in attribute specifications.
- Omitted start-tags for empty elements.

6 SGML Declaration

PSGML does not understand the SGML declaration, it accepts one in the file but it is ignored. If you have the SGML declaration in another file you can make '`sgmls`' use it when you use the *C-c C-v* (`sgml-validate`) command (see Chapter 5 [Validate], page 6).

PSGML has some options in what features it uses and what markup it creates. You have to set these options to make PSGML's behavior consistent with your SGML declaration and personal preferences.

sgml-omittag User Option
: Set this to `t` if the SGML declaration has '`OMITTAG YES`' and to `nil` otherwise.

sgml-shorttag User Option
: Set this to `t` if the SGML declaration has '`SHORTTAG YES`' and to `nil` otherwise.

sgml-always-quote-attributes User Option
: If non-nil, quote all attribute values inserted after finishing edit attributes. If this variable is `nil` and `sgml-shorttag` is non-`nil`, attribute values that consists of only name characters will not be quoted.

sgml-minimize-attributes User Option
: Determines minimization of attributes inserted by edit-attributes. If non-nil, omit attribute name if the attribute value is from a token group. If `max`, omit attributes with default value. Minimization will only be done if they produce legal SGML (assuming `sgml-omittag` and `sgml-shorttag` are set correctly).

7 Document Type Declaration

PSGML needs to know about the DTD you are using for many of its commands. If you do not have a '`DOCTYPE`' declaration in your file, PSGML will try assume that there is one of the form

`<!DOCTYPE` *name* `SYSTEM>`

where *name* is the value of `sgml-default-doctype-name`, if the value is non-`nil`, else the GI of the first tag will be used.

PSGML will try to parse the document type declaration the first time you do something that needs to parse the document or immediately if the variable `sgml-auto-activate-dtd` is `t`. You can also initiate the parsing of the document type declaration with the command `sgml-parse-prolog`. Big DTDs take some time to parse.

When the DTD has been parsed or loaded the name of the document element will be displayed in the mode line inside brackets. If there was an error parsing the DTD or there is no DTD, the mode line will display '`[ANY]`' (*** this is not really correct! a DTD will be established even if there are missing entities, it may even be empty).

7.1 Precompiled DTD Subsets

If parsing the DTD takes too long time you can arrange to have PSGML cache an internal complied version of the DTD. Caching can be done of DTD fragments in favourable situations. It is possible to have an external DTD subset cached but still have an internal DTD subset as long as the internal subset does not define parameter entities that affect the parsing of the external subset (*** what is the exact conditions?, probably you can't use the cached external subset if the internal subset defines parameter entities that are also defined in the external subset with another value).

To enable caching you have to create special catalog files, hereafter called ECAT files due to (temporary) lack of imagination. These catalogs have similar syntax to the entity catalogs and there are two variables containing lists of catalogs to search: `sgml-ecat-files` and `sgml-local-ecat-files`. The ECAT files can contain the following types of entries:

'`file` *dtdfile entitydef cfile*'

The *dtdfile* is the name of a file containing a DTD subset that should be cached in *cfile*. The *entitydef* is optional and if given have the following syntax:

`[` *name1 literal1 name2 literal2* `...` `]`

Using *entitydef* will modify the DTD subset by defining the parameter entity with name *name1* to be *literal1*, ... The cached version of the subset will be created with those entity definitions, and when PSGML search for a matching cached subset will check that the parameter entities in *entitydef* has been defined with those values before trying to use '`cfile`'.

'`public` *pubid entitydef cfile*'

Cache the DTD subset with public identifier *pubid* in file *cfile*.

sgml-recompile-out-of-date-cdtd User Option

If non-`nil`, out of date compiled DTDs will be automatically recompiled. If the value is `ask`, PSGML will ask before recompiling. A `nil` value will cause PSGML to silently load an out of date compiled DTD. A DTD that refers to undefined external entities is always out of date, thus in such case it can be useful to set this variable to `nil`.

Previous versions of PSGML have had another way of speeding up DTD parsing. This code remains in this version of PSGML, but is not actively maintained and may disappear in the future.

You can save the parsed DTD in a file using the command *M-x sgml-save-dtd*. Next time PSGML can load that file instead of parsing the DTD. For PSGML to find the saved DTD you must either save the DTD using the default name or do a *M-x sgml-save-options* after saving the DTD. To directly use an already parsed and saved DTD, load the file containing the saved DTD with the command *M-x sgml-load-dtd*.

sgml-default-dtd-file User Option

This is the default file name for saved DTD. This is set by `sgml-mode` to the buffer file name less extension plus the extension `.ced`, if that file exists. Can be changed in the Local variables section of the file.

7.2 Using a Split Document

You can have the '`DOCTYPE`' declaration in another file by setting `sgml-doctype` to the other file.

sgml-parent-document User Option

Used when the current file is part of a bigger document.

The variable describes how the current file's content fit into the element hierarchy. The variable should have the form

`(`*parent-file context-element*`*` *top-element* `(`*has-seen-element*`*)?)`

parent-file is a string, the name of the file containing the document entity.

context-element
is a string, that is the name of an element type. It can occur 0 or more times and is used to set up exceptions and short reference map. Good candidates for these elements are the elements open when the entity pointing to the current file is used.

top-element
is a string that is the name of the element type of the top level element in the current file. The file should contain one instance of this element, unless the last (lisp) element of sgml-parent-document is a list. If it is a list, the top level of the file should follow the content model of top-element.

has-seen-element
is a string that is the name of an element type. This element is satisfied in the content model of top-element.

7.3 Inserting a DOCTYPE

*** Describe the DTD menu in general. Describe customized entries for special DTDs. Mention `C-c C-u C-d` for inserting a DOCTYPE from keyboard.

If you change the doctype you must execute `sgml-parse-prolog`, changes in the doctype are not automatically recognized.

sgml-custom-dtd User Option

Menu entries to be added to the DTD menu. The value should be a list of entries to be added to the DTD menu.

Every entry should be a list. The first element of the entry is a string used as the menu entry. The second element is a string containing a doctype declaration (this can be nil if no doctype). The rest of the list should be a list of variables and values. For backward compatibility a single string instead of a variable is assigned to `sgml-default-dtd-file`. All variables are made buffer local and are also added to the buffers local variables list.

When an entry is selected from the DTD menu, the doctype declaration will be inserted, the variables will be set to the values in the entry and a local variables list will be created in the buffer.

Example:

```
(("HTML" nil
  sgml-default-dtd-file "~/sgml/html.ced"
  sgml-omittag nil sgml-shorttag nil)
 ("HTML+" "<!doctype htmlplus system 'htmlplus.dtd'>"
  "~/sgml/htmlplus.ced"
  sgml-omittag t sgml-shorttag nil)
 ("DOCBOOK" "<!doctype docbook system 'docbook.dtd'>"
  "~/sgml/docbook.ced"
  sgml-omittag nil sgml-shorttag t)))
```

7.4 Information from the DTD

PSGML can list various information about the current DTD. The following commands can be used via `M-x` and can also be found in the DTD menu.

`sgml-general-dtd-info`
: Display information about the current DTD.

`sgml-describe-element-type`
: Describe the properties of an element type as declared in the current DTD.

`sgml-describe-entity`
: Describe the properties of an entity as declared in the current DTD.

`sgml-list-elements`
: Will list all elements and the attributes declared for the element.

`sgml-list-attributes`
: Will list all attributes declared and the elements that use them.

`sgml-list-terminals`
: Will list all elements that can contain data.

`sgml-list-occur-in-elements`
: Will list all element types and where it can occur.

`sgml-list-content-elements`
: Will list all element types and the element types that can occur in its content.

8 Commands for editing

8.1 Inserting Markup

The commands that insert start-tags works only if the document has an associated DTD.

Keyboard commands for inserting:

C-c <
: Will ask, for the tag to insert, in the mini-buffer with completion on the tags that are valid at point (`sgml-insert-tag`).

  If the option `sgml-balanced-tag-edit` is non-nil, inserting a start-tag will also insert the corresponding end-tag. If, in addition, `sgml-auto-insert-required-elements` is non-nil, tags for elements required between the inserted tags will also be inserted.

  The list of valid tags, computed for a position in the buffer, will contain:

  1. The end-tag for the current element, if it can be ended at the position and `sgml-balanced-tag-edit` is nil. Furthermore it will contain end-tags for enclosing elements if the necessary omissible end-tag declarations have been made in the DTD.
  2. The start-tags of all elements that could occur after point. If `sgml-omittag-transparent` is nil, the above will be limited to the elements that can occur within the current element.

C-c C-e
: Insert start and end-tags for an element (`sgml-insert-element`). The name of the element is read from the mini-buffer with completion on valid elements. If `sgml-insert-end-tag-on-new-line` is non-nil or the element has element content, the end-tag will be inserted on a new line after the start-tag.

  If `sgml-omittag-transparent` is nil, the list of valid elements will only contain the elements that can be in the content of the current element.

  Required elements in the content will be automatically inserted if the option `sgml-auto-insert-required-elements` is non-nil. When the content model demands an element but there is more than one to choose from, a comment can be inserted with the available choices if the option `sgml-insert-missing-element-comment` is non-nil.

C-c C-i
: Inserts a new element in the current element where it is legal. Prompts for element name with completion. The completion list contains all elements that could be added to the current element somewhere, without making the content invalid. This assumes that the content is valid to begin with. Currently this list only has regular elements, not inclusions. The new element will be inserted as late as possible in the current element (unless prefix argument is given, then as early as possible.)

C-c C-r
: Makes the region into a new element (`sgml-tag-region`). Reads element name from mini-buffer with completion as for *C-c C-e*.

C-c /
: Inserts an end-tag for the current element (`sgml-insert-end-tag`).

C-c RET Split the current element at point. If repeated, the containing element will be split before the beginning of then current element.

Typical use is to start a new paragraph element when inside a paragraph.

C-c + Read attribute name and value from mini-buffer and insert attribute specification (`sgml-insert-attribute`). If point is immediately after a start-tag, this command operates on that start-tag. Otherwise the command will operate on the element after point.

The attribute name will be read with completion. If the attribute has a token list as declared value the attribute value will also be read with completion. The prompt for attribute value will typically look like:

`Value for` *attribute* `(`*type* `Default:` *current value*`):`

C-c C-u C-m

Give keyboard access to the customized part of the Markup menu. Emacs will prompt for the markup to insert using the menu line as selector. (See *sgml-custom-markup* below.)

Menu bar:

'`Markup`' Selecting from this menu will insert markup. The menu contains sub menus with tags and with entities, some other markup and a user defined section.

Sub menus:

'`Insert element`'

Pops up a menu of valid elements and insert start and end-tags for the selected element. Selections from the menu works like the *C-c C-e* command.

'`Insert start-tag`'

Pops up a menu of valid start-tags and insert the selected tag. The menu has the same start-tags as the completion list for *C-c* `<`.

'`Insert end-tag`'

Pops up a menu of valid end-tags and insert the selected tag.

'`Tag region`'

Pops up a menu of valid elements and tag the region with the selection. Selections from the menu works like the *C-c C-r* command.

'`Insert entity`'

Menu of all general entities defined in the DTD.

'`Add Element to Element`'

Pops up a menu of all elements valid somewhere in the current element. The menu contains all elements that could be added to the current element somewhere, without making the content invalid. The new element will be inserted as late as possible in the current element.

'`Insert attribute`'

Pops up a menu with all the attributes of an element. The element is either the one which start-tag is immediately before point or the element after point. Selecting from this menu edits the attribute specification list for the element.

The menu has a sub menu for every attribute which declared value is a token list. The rest of the attributes are collected in one sub menu. For the token list attributes, selecting a value will insert that attribute-value pair. Selecting some other attribute reads the attribute-value from the mini-buffer and inserts the attribute value pair.

A menu is also available directly with a mouse button click in the buffer. In GNU Emacs it is the first mouse button combined with shift (`S-`⟨mouse-1⟩). In Lucid Emacs it is bound to the third mouse button. The mouse button click will pop-up a menu of valid tags or a menu of attributes if the point is in a start-tag. The attributes menu works as the "Insert attribute" menu from the menu-bar. The tags list is the list of valid tags described above for command `C-c <`. Selection from the tags menu works like the `C-c <` command, with the following exception:

You can tag a region, with start and end-tag. There are two ways to indicate the region to mark:

1. Use the normal mouse commands to mark region.

   For this to work you must either use *transient mark mode* (see section "Transient Mark Mode" in *The Emacs Editor*) or set the option `sgml-tag-region-if-active` to non-nil (don't set this unless you are sure that you want it).

2. Alternatively make a secondary selection, this is done by holding down the meta key and using the mouse buttons. See section "Secondary selection" in *The Emacs Editor*. Some window managers intercept these events, which makes it hard use the secondary selection in Emacs.

sgml-balanced-tag-edit User Option

If non-nil, inserting a start-tag will also insert the corresponding end-tag.

sgml-auto-insert-required-elements User Option

If non-nil, automatically inserts required elements in the content of an inserted element.

sgml-omittag-transparent User Option

If non-nil, will show legal tags inside elements with omissible start-tags and legal tags beyond omissible end-tags.

sgml-tag-region-if-active User Option

If non-nil, the '`Insert tags`' menu will tag a region if the region is considered active by emacs. If nil, region must be active and `transient-mark-mode` must be on for the region to be tagged.

sgml-custom-markup User Option

Menu entries to be added to the Markup menu. The value should be a list of lists of two strings. The first string is the menu line and the second string is the text inserted when the menu item is selected. The second string can contain a '`\r`' where the cursor should be left. Also, if a selection is made according to the same rules as for the *S-mouse-1* menu, the selection is replaced with the second string and '`\r`' is replaced with the selection.

Example:

```
(("Version1" "<![%Version1[\r]]>")
 ("New page"  "<?NewPage>"))
```

sgml-insert-missing-element-comment User Option

If non-nil, and sgml-auto-insert-required-elements also true, `sgml-insert-element` will insert a comment if there is an element required but there is more than one to choose from.

sgml-insert-end-tag-on-new-line User Option

If non-nil, `sgml-insert-element` will put the end-tag on a new line after the start-tag. Useful on slow terminals if you find the end-tag after the cursor irritating.

8.2 Markup completion

If you are typing in markup directly, *M-TAB* will help you by completing a tag name, an entity name or a markup declaration name. If you type *M-TAB* after a plain word, `ispell-complete-word` will be invoked instead.

If you have typed (⋆ marks the position of point)

```
&At⋆
```

and type *M-TAB* (assuming you use the '`ISOLat1`' entity set) you get:

```
&Atilde⋆
```

8.3 Showing information

Commands for showing information obtained by parsing the buffer.

C-c C-c Shows in the message area: context at point, if in a tag or in mixed content and the open elements (`sgml-show-context`).

C-c C-w Shows what element the character after point (under the cursor) belongs to; also shows context of element (`sgml-what-element`).

C-c C-t List contextually valid tags (`sgml-list-valid-tags`). Displays information about current element, all valid end-tags, valid start-tags in current element, and start-tags valid at this point but in other elements together with the tags omitted.

You can make the mode-line display the name of the current open element by setting the `sgml-live-element-indicator` variable. Setting this will make all commands slower due to the work needed to keep the mode-line up to date.

sgml-live-element-indicator User Option

If non-nil, indicate current element in mode line.

NOTE: Setting this implies that every command can cause a parse.

8.4 Indentation according to structure

You can indent a line according to the depth of element nesting at the beginning of the line. To indent the current line use ⟨TAB⟩. You can also use ⟨LFD⟩ (`newline-and-indent`) to start a new line with correct indentation.

sgml-indent-step User Option

How much to increment indent for every element level. If nil, no indentation.

If this is nil, ⟨TAB⟩ will insert a tab instead of indenting.

sgml-indent-data User Option

If non-nil, indent in data/mixed context also.

8.5 Move in the element structure

These commands move in the element structure. The commands uses knowledge of SGML syntax, and if available the specific DTD.

C-M-a Move to the (content) beginning of the current element (`sgml-beginning-of-element`).

C-M-e Move to the (content) end of the current element (`sgml-end-of-element`).

C-M-f Move forward by element (`sgml-forward-element`).

C-M-b Move backward by element (`sgml-backward-element`).

C-M-u Move up to before current element (`sgml-backward-up-element`).

C-c C-n Move up to after current element (`sgml-up-element`).

C-M-d Move down to the (content) beginning of the next element (`sgml-down-element`).

C-c C-d Move to the next place where data is allowed (`sgml-next-data-field`).

You can also move to the next place where there is some structural error with *C-c C-o* (see Chapter 5 [Validate], page 6).

8.6 Editing attributes

If you want to change the attributes of a start-tag you can simply edit them directly in the buffer. Or you can place the cursor at or after the start-tag and use the `sgml-edit-attributes` command, available from the '`SGML`'-menu or on *C-c C-a*. This will create a new Emacs window with all possible attributes listed in the form

attribute name `=` *current value*.

The *current value* may be shown as '`#DEFAULT`' if the attribute has not been given a value in the start-tag. The list also contains the attributes declaration as a comment. Note also that the *current value* is show without eventual quotes.

It is now possible to edit the attribute values. You can move to the next attribute with ⟨TAB⟩. If you want to let an attribute have its default value use *C-c C-d*, this will insert a '#DEFAULT' in the value field.

If Emacs is running in an X window, the '#DEFAULT' will be underlined to distinguish it from normal values.

Finish the editing with *C-c C-c*; this will replace the attribute values in the main buffer with those edited. Note that values will be quoted as needed.

If you want to abort the editing, you can remove the window with *C-x 0* or if you want it neat, kill the buffer and remove the window.

Some other keys are:

C-a Go to the beginning of the value field (`sgml-edit-attrib-field-start`).

C-e Go to the end of the value field (`sgml-edit-attrib-field-end`).

C-c C-k Clear the value field (`sgml-edit-attrib-clear`).

C-c C-d Set the value field to '#DEFAULT' (`sgml-edit-attrib-default`). This is a special value that will make the attribute be implied.

8.7 Changing and deleting markup

C-c = Change the name of the current element (`sgml-change-element-name`). Tries to translate attribute specifications. An attribute will be translated to an attribute with the same name. If the new element has no attribute with the same name, the attribute will be ignored. If there is an attribute with the same name but different declared content, a warning is given.

ID attributes are handled specially, an attribute with declared value ID will always be translated to the attribute with declared value ID.

C-c C-k Kill next tag, markup declaration or process instruction (`sgml-kill-markup`).

C-M-k Kill the element following the cursor (`sgml-kill-element`).

C-c - Remove tags from current element (`sgml-untag-element`).

C-c # Convert character after point to a character reference (`sgml-make-character-reference`). If called with a numeric argument, convert a character reference back to a normal character.

C-c C-q Fills an element as a paragraph (`sgml-fill-element`). This is a substitute for the normal `fill-paragraph`. The command uses heuristics to decide what should be a paragraph.

1. If point is in an element content, recursively fill the sub-elements.
2. Find the biggest element with mixed content containing point.
3. If the above element is mixed but contains elements with pure element content then fill what is between the pure elements as paragraphs and fill the pure elements recursively.

M-x sgml-expand-all-shortrefs

Short references to text entities are expanded to the replacement text of the entity other short references are expanded into general entity references. If argument, *to-entity*, is non-`nil`, or if called interactive with numeric prefix argument, all short references are replaced by generally entity references.

M-x sgml-normalize

Normalize the document in the buffer. This will

1. expand short references,
2. insert missing tags,
3. replace minimized tags with full tags,
4. fix attribute specification lists according to options set.

There is one argument, *to-entity*, with the same meaning as for `sgml-expand-all-shortrefs`.

There is one option for the normalize command. With its default value, normalize may actually change the data content of some elements. But only by removing some white-space from the end of elements with omitted end-tags.

sgml-normalize-trims User Option

If non-nil, `sgml-normalize` will trim off white space from end of element when adding end-tag.

Default: `t`.

8.8 Translating between characters and entity references

Set the variable `sgml-display-char-list-filename` to a file that contains mappings between all characters present in the presentation character set, and their "standard replacement text" names, e.g. "" -> "[aring]", e.t.c.

The default value for this variable is 'iso88591.map'.

Then use the functions (also in the Modify menu)

`sgml-charent-to-display-char`

`sgml-display-char-to-charent`

to translate between entities and characters.

9 Appearance of text in the buffer

9.1 Folding editing

With these commands you can make parts of the text temporarily invisible to make it easier to see the overall structure of your text.

When folding a region all the lines but the first will be invisible. The first line of the region will still be visible with an ellipsis at the end.

See section "Outline Mode" in *The Emacs Editor*.

`C-c C-f C-r`
: The region between point and mark will be folded (`sgml-fold-region`).

`C-c C-f C-e`
: The region between the start and end of the current element will be folded (`sgml-fold-element`).

  This command can also fold the SGML declaration or the DOCTYPE declaration.

`C-c C-f C-s`
: Fold all the sub elements of the current element (`sgml-fold-subelement`).

`C-c C-s`
`C-c C-u C-l`
: Unfold the current line, assuming it is the first line of a folded region (`sgml-unfold-line`).

`C-c C-u C-e`
: Make all lines in current element visible (`sgml-unfold-element`).

`C-c C-u C-a`
: Make all lines in current buffer visible (`sgml-unfold-all`).

`C-c C-f C-x`
: Unfold current element and then fold the subelements (`sgml-expand-element`). If the current element is folded this expands what is visible.

9.2 Hiding markup

*** Describe hide-tags

9.3 Highlighting markup

PSGML can highlight the markup giving the markup a different *face* (see section "Using Multiple Typefaces" in *The Emacs Editor*). The highlighting will only be done if the variable `sgml-set-face` is non-`nil`. The default settings make tags bold and comments italic, but this can be modified with the variable `sgml-markup-faces`. When highlighting is on PSGML will parse after every command until the whole buffer has been parsed or user event occurs.

To remove the highlighting type *`M-x sgml-clear-faces`*.

sgml-set-face User Option

If non-nil, psgml will set the face of parsed markup.

sgml-markup-faces User Option

A list of markup to face mappings. Each element looks like `(`*markup-type* `.` *face*`)`. Possible values for *markup-type* is:

`comment` comment declaration

`doctype` doctype declaration

`end-tag` end-tag

`ignored` ignored marked section

`ms-start` marked section end, if not ignored

`ms-end` marked section start, if not ignored

`pi` processing instruction

`sgml` SGML declaration

`start-tag`
start-tag

`entity` entity reference

`shortref` short reference

10 Miscellaneous options

*** describe sgml-save-options

sgml-ignore-undefined-elements User Option

Start-tags for undefined elements will either be ignored, if `sgml-ignore-undefined-elements` is `t`, or assumed to be acceptable in the current element and defined with `O O ANY`

sgml-range-indicator-max-length User Option

Maximum number of characters used from the first and last entry of a sub-menu to indicate the range of that menu.

This is used for long menus of elements, tags or entities that are split into `sgml-max-menu-size` big sub-menus.

11 Bugs

If you encounter something that you think is a bug, please report it. Try to include a clear description of the undesired behaviour. A test case that exhibits the bug, would also be useful.

You can report a bug with the command *M-x sgml-submit-bug-report*.

When PSGML needs contextual information it parses the document up to the point. During the parsing, it builds a parse tree. The parse tree is used to initialize the next parse, to avoid having to parse things already parsed. Changes to the buffer is supposed to prune the tree of all outdated information. But if you get strange complaints from the parser, try and back up a bit and use *C-c C-o* (`sgml-next-trouble-spot`).

12 Index

Table of Contents