

Erl_Docgen

Copyright © 2004-2023 Ericsson AB. All Rights Reserved.
Erl_Docgen 1.4
May 30, 2023

Copyright © 2004-2023 Ericsson AB. All Rights Reserved.
Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License. Ericsson AB. All Rights Reserved..

May 30, 2023

1 Erl_Docgen User's Guide

Erl_Docgen provides functionality for generating HTML/PDF/man documentation for Erlang modules and Erlang/OTP applications from XML source code and/or EDoc comments in Erlang source code.

1.1 How to Build OTP like documentation

1.1.1 Utilities to prepare XML files

Create XML files from code

If there are EDoc comments in a module, the escript `xml_from_edoc.escript` can be used to generate an XML file according to the `erlref` DTD for this module.

Example:

```
1> escript $ERL_TOP/lib/erl_docgen/priv/bin/xml_from_edoc.escript ex1.erl
```

Include code in XML

If there are OTP DTD *codeinclude* tags in the source XML file, the escript `codeline_preprocessing.escript` can be used to include the code and produce an XML file according to the OTP DTDs.

Example:

```
1> escript $ERL_TOP/lib/erl_docgen/priv/bin/codeline_preprocessing.escript \
   ex1.xmlsrc ex1.xml
```

1.1.2 Use xsltproc to generate different output formats

Parameters used in all the the XSL transformations

These parameters to `xsltproc` are used for all the supported output formats.

`docgen`
: Path to erl_docgen's top directory.

`gendate`
: The date string that will be used in the documentation.

`appname`
: The name of the application.>

`appver`
: The version of the application.

Generate HTML output

When generating HTML one also needs these three pramaters to `xsltproc`.

`outdir`
: Output directory for the produced html files.

`topdocdir`
: If one builds a standalone documentation for an application this should be set to ".".

`pdfdir`
: Relative path from the html directory to where the pdf file are placed.

Example:

```
1> xsltproc --noout --stringparam outdir /tmp/myhtmldoc \
      --stringparam docgen $ERL_TOP/lib/erl_docgen \
             --stringparam topdocdir . \
             --stringparam pdfdir $PDFDIR \
             --xinclude \
      --stringparam gendate "December 5 2011" \
             --stringparam appname MyApp \
             --stringparam appver 0.1 \
             -path $ERL_TOP/lib/erl_docgen/priv/dtd \
             -path $ERL_TOP/lib/erl_docgen/priv/dtd_html_entities \
      $ERL_TOP/lib/erl_docgen/priv/xsl/db_html.xsl mybook.xml
```

Generate PDF

The generation of the PDF file is done in two steps. First is `xsltproc` used to generate a `.fo` file which is used as input to the `fop` command to produce th PDF file.

Example:

```
1> xsltproc --output MyApp.fo \
            --stringparam docgen $ERL_TOP/lib/erl_docgen \
     --stringparam gendate  "December 5 2011" \
            --stringparam appname MyApp \
            --stringparam appver 0.1 \
            --xinclude \
            -path $ERL_TOP/lib/erl_docgen/priv/dtd \
            -path $ERL_TOP/lib/erl_docgen/priv/dtd_html_entities \
     $ERL_TOP/lib/erl_docgen/priv/xsl/db_pdf.xsl mybook.xml


       2> fop -fo MyApp.fo -pdf MyApp.pdf
```

Generate man pages

Unix man pages can be generated with `xsltproc` from XML files written according to the different OTP ref type DTDs.

Example:

```
1> xsltproc --output  my_module.3\
      --stringparam docgen $ERL_TOP/lib/erl_docgen \
      --stringparam gendate  "December 5 2011" \
      --stringparam appname MyApp \
      --stringparam appver 0.1 \
      --xinclude -path $ERL_TOP/lib/erl_docgen/priv/dtd  \
             -path $ERL_TOP/lib/erl_docgen/priv/dtd_man_entities \
             $ERL_TOP/lib/erl_docgen/priv/xsl/db_man.xsl my_refpage.xml
```

Upcoming changes

The output from the `erl_docgen` documentation build process is now just the OTP style. But in a near future we will for example add the possibility to change logo, color in the PDF and style sheet for the HTML.

1.2 Overview OTP DTDs

1.2.1 DTD Suite

Input is written as XML according to one of the DTDs and output is corresponding HTML. Documentation for an Erlang/OTP application is usually organized as follows:

User's Guide

(DTD: part) A collection of chapters (chapter).

Reference Manual

(DTD: application A collection of manual pages for modules (erlref), applications (appref), commands (comref), C libraries (cref) and files (fileref).

Release Notes

Same structure as the User's Guide.

In some cases, one or more of the User's Guide, Reference Manual and Release Notes are omitted. Also, it is possible to use either the `application` or `part` DTD to write other types of documentation for the application.

The structure of the different documents and the meaning of the tags are explained. There are numerous examples of documentation source code.

For readability and simplicity, the examples have been kept as short as possible. For an example of what the generated HTML will look like, it is recommended to look at the documentation of an OTP application.

1.2.2 Basic Tags

All DTDs in the OTP DTD suite share a basic set of tags. An author can easily switch from one DTD to another and still use the same basic tags. It is furthermore easy to copy pieces of information from one document to another, even though they do not use the same DTD.

The basic set of tags are divided into two categories: block tags and inline tags. Block tags typically define a separate block of information, like a paragraph or a list. Inline tags are typically used within block tags, for example a highlighted word within a paragraph.

1.3 User's Guide DTDs

1.3.1 The part DTD

The `part` DTD is intended for a "normal" document, like the User's Guide or Release Notes. First are some paragraphs introducing the main contents. After that follows chapters, written in separate files with the chapter DTD.

Example:

```
<?xml version="1.0" encoding="latin1" ?>
<!DOCTYPE part SYSTEM "part.dtd">
<part>
  <header>
    <title>The chapter title</title>
    <prepared>The author</prepared>
    <docno/>
    <date/>
    <rev/>
  </header>

  <description>
    <p>Some text..</p>
  </description>

  <include file="file1"></include>
  <include file="file2"></include>
</part>
```

1.3.2 <part>

The top level tag of a `part` DTD.

Contains a <header>, an optional <description>, followed by one or more <include>.

1.3.3 <description>

The introduction after the title and before the bulk of included chapters/manual pages.

Contains any combination and any number of block tags except `<image>` and `<table>`.

1.3.4 <include>

An empty tag. The attribute `file` specifies a file to include. The `.xml` file extension should be omitted.

Example:

```
<include file="notes"></include>
```

1.3.5 The chapter DTD

The `chapter` DTD is intended for a chapter in a User's Guide or similar with text divided into sections, which can be nested.

Example:

```
<?xml version="1.0" encoding="latin1" ?>
<!DOCTYPE chapter SYSTEM "chapter.dtd">
<chapter>
  <header>
    <title>Title on first level</title>
    <prepared/>
    <docno/>
    <date/>
    <rev/>
  </header>

  <p>Introduction...</p>

  <section>
    <title>Title on second level</title>

    <p>First paragraph.</p>

    <p>Second paragraph etc.</p>

    <section>
      <title>Title on third level</title>

      <p>...</p>
    </section>
  </section>

  ...
</chapter>
```

1.3.6 <chapter>

The top level tag of a `chapter` DTD.

Contains a <header>, an optional introduction consisting of any combination of block tags, followed by one or more <section>.

1.3.7 <section>

Subdivision of a chapter.

Contains an optional <marker>, a <title>, followed by any combination and any number of block tags and `<section ghlink="maint/lib/erl_docgen/doc/src/user_guide_dtds.xml#L172">`.

1.3.8 <title>

Section title, contains plain text.

1.4 Reference Manual DTDs

There are five DTDs for writing manual pages about applications, shell commands, C libraries, Erlang modules and files, all with a similar structure:

- A header.
- Name of the application/command/library/module/file.
- Short summary (one line).
- A longer description.

- "Formal" definitions of functions or commands.
- Optional sections of free text.
- Optional section with the name(s) and email(s) of the author(s).

The differences between the DTDs are the tags for the name, the short summary and some tags inside the "formal" definitions.

1.4.1 The application DTD

The `application` DTD is intended for a Reference Manual and groups a set of manual pages into one unit. The structure is similar to the part DTD: first an introduction and then the manual pages, written in separate files with the appref, comref, cref, erlref, or fileref DTD.

Example:

```
<?xml version="1.0" encoding="latin1" ?>
<!DOCTYPE application SYSTEM "application.dtd">
<application>
  <header>
    <title>Application name</title>
    <prepared/>
    <docno/>
    <date/>
    <rev/>
  </header>

  <description>
    <p>Application description...</p>
  </description>

  <include file="module1">
  <include file="module2">
</application>
```

1.4.2 <application>

The top level tag of an `application` DTD.

Contains a <header>, an optional <description>, followed by one or more <include>.

1.4.3 The appref DTD

This is the DTD for writing an application manual page.

Example:

```
<?xml version="1.0" encoding="latin1" ?>
<!DOCTYPE appref SYSTEM "appref.dtd">
<appref>
  <header>
    <title>Application name</title>
    <prepared/>
    <docno/>
    <date/>
    <rev/>
  </header>

  <app>Application name</app>

  <appsummary>A short application summary.</appsummary>

  <description>
    <p>A longer description of the application.</p>
  </description>

  <section>
    <title>Configuration</title>

      <p>...</p>
  </section>

  ...

  <authors>
    <aname>Name of author</aname>
    <email>Email of author</email>
  </authors>
</appref>
```

<appref>

The top level tag of an `appref` DTD.

Contains <header>, <app>, <appsummary>, <description>, zero or more <section> and <funcs>, followed by zero or more <authors>.

<app>

The application name. Contains plain text.

<appsummary>

Short summary. Contains plain text.

1.4.4 The comref DTD

This is the DTD for writing a command manual page.

Example:

```
<?xml version="1.0" encoding="latin1" ?>
<!DOCTYPE comref SYSTEM "comref.dtd">
<comref>
  <header>
    <title>Command name</title>
    <prepared/>
    <docno/>
    <date/>
    <rev/>
  </header>

  <com>Command name</com>

  <comsummary>A short command summary.</comsummary>

  <description>
    <p>A long description of the command.</p>
  </description>

  <funcs>
    <func>
      <name>command</name>
      <name>command -flag <arg></name>
      <fsummary>A short command summary (max 40 characters).</fsummary>
      <desc>
        <p>An extended command description.
      </desc>
    </func>
  </funcs>

  <section>
    <title>Options</title>

    <p>...</p>
  </section>

  <authors>
    <aname>Name of author</aname>
    <email>Email of author</email>
  </authors>
</comref>
```

<comref>

The top level tag for a `comref` DTD.

Contains <header>, <com>, <comsummary>, <description>, zero or more <section> and <funcs>, followed by zero or more <authors>.

<com>

The command name. Contains plain text.

<comsummary>

Short summary. Contains plain text.

1.4.5 The cref DTD

This is the DTD for writing a C library manual page.

Example:

```
<?xml version="1.0" encoding="latin1" ?>
<!DOCTYPE cref SYSTEM "cref.dtd">
<cref>
  <header>
    <title>C library name</title>
    <prepared/>
    <docno/>
    <date/>
    <rev/>
  </header>

  <lib>C library name</lib>

  <libsummary>A short C library summary.</libsummary>

  <description ghlink="maint/lib/erl_docgen/doc/src/refman_dtds.xml#L285">
    <p>A longer description of the C library.</p>
  </description>

  <funcs>
    <func ghlink="maint/lib/erl_docgen/doc/src/refman_dtds.xml#L290">
      <name><ret>void</ret><nametext>start(bar)</nametext></name>
      <name><ret>void</ret><nametext>start(foo)</nametext></name>
      <fsummary>A short function summary (max 40 characters).</fsummary>
      <type>
        <v>char bar</v>
        <v>int foo</v>
      </type>
      <desc>
        <p>An extended function description.</p>
      </desc>
    </func>

    ...
  </funcs>

  <section ghlink="maint/lib/erl_docgen/doc/src/refman_dtds.xml#L306">
    <title>A title</title>

    <p>Some text...</p>
  </section>


</cref>
```

<cref>

The top level tag for a `cref` DTD.

Contains <header>, <lib>, <libsummary>, <description>, zero or more <section> and <funcs>, followed by zero or more <authors>.

<lib>

The C library name or acronym. Contains plain text.

<libsummary>

Short summary. Contains plain text.

1.4.6 The erlref DTD

This is the DTD for writing Erlang module manual pages.

Example:

```
<?xml version="1.0" encoding="latin1" ?>
<!DOCTYPE erlref SYSTEM "erlref.dtd">
<erlref>
  <header>
    <title>Module name</title>
    <prepared/>
    <docno/>
    <date/>
    <rev/>
  </header>

  <module>Module name</module>

  <modulesummary>A short module summary.</modulesummary>

  <description>
    <p>A longer description of the module.</p>
  </description>

  <funcs>
    <func>
      <name>start() -> Result</name>
      <name>start(N) -> Result</name>
      <fsummary>A short function summary (max 40 characters).</fsummary>
      <type>
        <v>Pid = pid()</v>
        <v>N = int()</v>
        <v>Result = {ok, Pid} | {error, Reason}</v>
        <v>Reason = term()</v>
        <d>A parameter description.</d>
      </type>
      <desc>
        <p>An extended function description.</p>
      </desc>
    </func>

    ...
  </funcs>

  <section>
    <title>Some Title</title>
    <p>Some text...</p>
  </section>

  <authors>
    <aname>Name of author</aname>
    <email>Email of author</email>
  </authors>
</erlref>
```

<erlref>

The top level tag for an `erlref` DTD.

Contains <header>, <module>, <modulesummary>, <description>, zero or more <section> and <funcs>, followed by zero or more <authors>.

<module>

The module name. Contains plain text.

<modulesummary>

Short summary. Contains plain text.

1.4.7 The fileref DTD

This is the DTD for writing file manual pages. In OTP, this DTD is used for defining the format of for example `.rel` and `.app` files.

Example:

```
<?xml version="1.0" encoding="latin1" ?>
<!DOCTYPE fileref SYSTEM "fileref.dtd">
<fileref>
  <header>
    <title>File name</title>
    <prepared/>
    <docno/>
    <date/>
    <rev/>
  </header>

  <file>fileref</file>

  <filesummary>A short file summary.</filesummary>

  <description>
    <p>A longer description of the file.</p>
  </description>

  <section>
    <title>File format</title>

    <p>...</p>
  </section>

  <authors>
    <aname>Name of author</aname>
    <email>Email of author</email>
  </authors>
</fileref>
```

The file reference manual can also contain function definitions, similar to the `erlref` DTD.

<fileref>

The top level tag for a `fileref` DTD.

Contains <header>, <file>, <filesummary>, <description>, zero or more <section> and <funcs>, followed by zero or more <authors>.

<file>

The name of the file or file type. Contains plain text.

<filesummary>

Short summary. Contains plain text.

1.4.8 <description>

The introduction after the title and before sections and "formal" definitions.

Contains any combination and any number of block tags except `<image>` and `<table>`.

1.4.9 <section>

Subdivisions of the document. Contains an optional <marker>, a <title>, followed by any combination and any number of block tags except `<image>` and `<table>`.

1.4.10 <funcs>

A group of "formal" function definitions.

Contains one or more <func>.

1.4.11 <func>

A "formal" function definition.

Contains one or more <name>, followed by <fsummary>, <type> (optional) and <desc> (optional).

1.4.12 <name>

Function/command signature with name, arguments and return value. Contains plain text, except for the `cref` DTD where it contains a `<ret>` (return type, plain text) and a `<nametext>` (function name and arguments, plain text).

In the case of an `erlref` DTD, it will automatically be added a marker, `<marker id="Name/Arity">` or `<marker id="Name">`, based on the contents of this tag before the function definition.

Example: Consider the following name definition

```
<name>foo(Arg1, Arg2) -> ok | {error, Reason}</name>
```

Then a marker like this will be added `<marker id="foo/2">` before the function definition in the generated HTML. That is, referring to the function using `<seemfa marker="#foo/2">foo/2</seemfa>` will automatically work.

1.4.13 <fsummary>

Function/command summary. Contains plain text, <c> and .

1.4.14 <type>

Type declarations for the function/command.

Contains one or more pairs of <v> and <d> (optional).

1.4.15 <v>

Type declaration for an argument or return value. Contains plain text.

1.4.16 <d>

Description for an argument or return value. Contains plain text, <c> and .

1.4.17 <desc>

Function/command description. Contains block tags except `<image>` and `<table>`.

1.4.18 <authors>

Authors of the manual page. The `authors` element is optional.

Contains one or more pairs of <aname> and <email>.

1.4.19 <aname>

Author name. Contains plain text.

1.4.20 <email>

Author email address. Contains plain text.

1.5 Header Tags

Each document begins with a header part, which looks the same for all DTDs. Here the title of the document is specified, as well as administrative data like who is responsible for the document, which version is it, when was it last changed and such.

An full header looks like:

```
<header>
  <copyright>...</copyright>
  <legalnotice>...</legalnotice>
  <title>...</title>
  <prepared>...</prepared>
  <responsible>...</responsible>
  <docno>...</docno>
  <approved>...</approved>
  <checked>...</checked>
  <date>...</date>
  <rev>...</rev>
  <file>...</file>
</header>
```

1.5.1 <header>

Top level tag for the header part.

1.5.2 <copyright>

The `copyright` element holds information about date(s) and holder(s) of a document copyright. The `copyright` element is optional. The `copyright` element has an inner structure containing one or more `year` elements followed by zero of more `holder` elements.
See example below:

```
    <copyright>
      <year>1997</year>
      <year>2007</year>
      <holder>Ericsson AB</holder>
    </copyright>
```

1.5.3 <legalnotice>

The `legalnotice` element is used to express copyright, trademark, license, and other legal formalities of a document. The element contains only PCDATA in the same manner as `code` and `pre`.

1.5.4 <title>

For `part` and `application` documents, this will be the title of the document, visible in the left frame and on the front page.

For `chapter` documents, this will be the chapter name.

For reference manual documents, this tag is ignored.

1.5.5 <shorttitle>

This optional tag is ignored. It will likely be removed in the future.

1.5.6 <prepared>

This tag is intended for administrative use and is ignored.

1.5.7 <responsible>

This optional tag is intended for administrative use and is ignored.

1.5.8 <docno>

Document number.

For `part` and `application` documents, the document number is visible in the left frame and on the front page.

For other types of documents, this tag is ignored.

1.5.9 <approved>

This optional tag is intended for administrative use and is ignored.

1.5.10 <checked>

This optional tag is intended for administrative use and is ignored.

1.5.11 <date>

This tag is intended for administrative use and is ignored.

1.5.12 <rev>

Document version.

For `part` and `application` documents, the document version is visible in the left frame and on the front page.

For other types of documents, this tag is ignored.

1.5.13 <file>

This optional tag is intended for administrative use and is ignored.

1.6 Block Tags

Block tags typically define a separate block of information, such as a paragraph or a list.

The following subset of block tags are common for all DTDs in the OTP DTD suite: <p>, <pre>, <code>, <list>, <taglist>, <codeinclude> and

1.6.1
 - Line Break

Forces a newline. Example:

```
Eat yourself<br/>senseless!
```

results in:

Eat yourself
senseless!

The `<br>` tag is both a block- and an inline tag.

1.6.2 <code> - Code Example

Highlight code examples. Example:

```
<code>
sum([H|T]) ->
    H + sum(T);
sum([]) ->
    0.
</code>
```

results in:

```
sum([H|T]) ->
    H + sum(T);
sum([]) ->
    0.
```

There is an attribute `type = "erl" | "c" | "none"`, but currently this attribute is ignored. Default value is `"none"`

Note:

No tags are allowed within the tag and no character entities are expanded.

1.6.3 <codeinclude> - Code Inclusion

Include external code snippets. The attribute `file` gives the file name and `tag` defines a string which delimits the code snippet. Example:

```
<codeinclude file="example.txt" tag="%% Erlang example"/>
```

results in:

```
-module(example).

start() ->
    {error,"Pid required!"}.

start(Pid) ->
    spawn(smalltalk,main,[]).
```

provided there is a file named `examples.txt` looking like this:

```
...

%% Erlang example
-module(example).

start() ->
    {error,"Pid required!"}.
start(Pid) ->
    spawn(fun() -> init(Pid) end).
%% Erlang example

...
```

If the `tag` attribute is omitted, the whole file is included.

There is also an attribute `type = "erl" | "c" | "none"`, but currently this attribute is ignored. Default value is `"none"`

1.6.4 <list> - List

The attribute `type = "ordered"|"bulleted"` decides if the list is numbered or bulleted. Default is `"bulleted"`.

Lists contains list items, tag `<item>`, which can contain plain text, the common subset of block tags and inline tags. Example:

```
<list type="ordered">
  <item>Askosal:
    <list>
      <item>Nullalisis</item>
      <item>Facilisis</item>
    </list>
  </item>
  <item>Ankara</item>
</list>
```

results in:

- Askosal:
  - Nullalisis
  - Facilisis
- Ankara

1.6.5 <marker> - Marker

Used as an anchor for hypertext references. The `<marker>` tag is both a block- and an inline tag and is described in the Inline Tags section.

1.6.6 <p> - Paragraph

Paragraphs contain plain text and inline tags. Example:

```
<p>I call specific attention to
  the authority given by the <em>21st Amendment</em>
  to the Constitution to prohibit transportation
  or importation of intoxicating liquors into
  any State in violation of the laws of such
  State.</p>
```

results in:

I call specific attention to the authority given by the **21st Amendment** to the Constitution to prohibit transportation or importation of intoxicating liquors into any State in violation of the laws of such State.

1.6.7 <note> - Note

Highlights a note. Can contain block tags except `<note>`, `<warning>`, `<image>` and `<table>`. Example:

```
<note>
  <p>This function is mainly intended for debugging.</p>
</note>
```

results in:

Note:

This function is mainly intended for debugging.

1.6.8 <pre> - Pre-formatted Text

Used for documentation of system interaction. Can contain text, <see*> tags, <url> and `<input>` tags.

The `<input>` tag is used to highlight user input. Example:

```
<pre>
$ <input>erl</input>
Erlang (BEAM) emulator version 5.5.3 [async-threads:0] [hipe] [kernel-poll:false]

Eshell V5.5.3  (abort with ^G)
1> <input>pwd().</input>
/home/user
2> <input>halt().</input>
</pre>
```

results in:

```
$ erl
Erlang (BEAM) emulator version 5.5.3 [async-threads:0] [hipe] [kernel-poll:false]

Eshell V5.5.3  (abort with ^G)
1> pwd().
/home/user
2> halt().
```

All character entities are expanded.

1.6.9 <quote> - Quotation

Highlight quotations from other works, or dialog spoken by characters in a narrative. Contains one or more <p> tags. Example:

```
<p>Whereas Section 217(a) of the Act of Congress entitled
"An Act ..." approved June 16, 1933, provides as follows:</p>
<quote>
  <p>Section 217(a) The President shall proclaim the law.</p>
</quote>
```

results in:

Whereas Section 217(a) of the Act of Congress entitled "An Act ..." approved June 16, 1933, provides as follows:

Section 217(a) The President shall proclaim the law.

1.6.10 <taglist> - Definition List

Definition lists contains pairs of tags, `<tag>`, and list items, `<item>`.

`<tag>` can contain plain text, <c>, , <see*> tags and <url> tags.

`<item>` can contain plain text, the common subset of block tags and inline tags. Example:

```
<taglist>
  <tag><c>eacces</c></tag>
  <item>Permission denied.</item>
  <tag><c>enoent</c></tag>
  <item>No such file or directory.</item>
</taglist>
```

results in:

`eacces`
: Permission denied.

`enoent`
: No such file or directory.

1.6.11 <warning> - Warning

Highlights a warning. Can contain block tags except `<note>`, `<warning>`, `<image>` and `<table>`. Example:

```
<warning>
  <p>This function might be removed in a future version without
    prior warning.</p>
</warning>
```

results in:

Warning:

This function might be removed in a future version without prior warning.

1.6.12 <image> - Image

Graphics is imported using the `<image>` tag. An image caption `<icaption>`, containing plain text, must be supplied. Example:

```
<image file="man">
  <icaption>A Silly Man</icaption>
</image>
```

results in:



Figure 6.1: A Silly Man

This assumes that `man.gif` exists in the current directory.

1.6.13 <table> - Table

The table format is similar to how tables are described in HTML 3.2. A table contains one or more rows, `<row>`, and a table caption `<tcaption>`, containing plain text.

Each row contains one or more cells, `<cell>`. The attributes `align = "left"|"center"|"right"` and `valign = "top"|"middle"|"bottom"` decides how text is aligned in the cell horizontally and vertically. Default is "`left`" and "`middle`".

Each cell contains plain text and inline tags. Example:

```
<table>
  <row>
    <cell align="left" valign="top"><em>Boys</em></cell>
    <cell align="center" valign="middle"><em>Girls</em></cell>
  </row>
  <row>
    <cell align="left" valign="middle">Juda</cell>
    <cell align="right" valign="bottom">Susy</cell>
  </row>
  <row>
    <cell align="left" valign="middle">Anders</cell>
    <cell align="left" valign="middle">Victoria</cell>
  </row>
  <tcaption>A table caption</tcaption>
</table>
```

results in:

Boys	**Girls**
Juda	Susy
Anders	Victoria

Table 6.1: A table caption

1.7 Inline Tags

Inline tags are typically used within block tags, for example to highlight a word within a paragraph.

1.7.1
 - Line Break

Forces a newline. The `<br>` tag is both a block- and an inline tag and is described in the Block Tags section.

1.7.2 <c> - Code

Highlights things like variables and file names in a text flow. Can contain plain text only. Newlines and tabs are ignored as opposed to the code tag. All character entities are expanded. Example:

```
<p>Returns <c>true</c> if <c>Term</c> is an integer.</p>
```

results in:

Returns `true` if `Term` is an integer.

1.7.3 - Emphasis

Highlights words which are important within a text flow. Example:

```
<p>The application <em>must</em> be up and running.</p>
```

results in:

The application **must** be up and running.

Contains plain text or a <c> tag.

1.7.4 <marker> - Marker

Used as an anchor for hypertext references. The `id` attribute defines the name of the marker. Example:

```
<marker id="marker_example"/>
```

The <see*> tags are used to refer to the marker.

The `<marker>` tag is both a block- and an inline tag.

1.7.5 <see*> - See tags

A cross reference (hypertext link) to a marker in the same file, a marker in another file, or (the top of) another file, given by the `marker` attribute. The syntax used within the `marker` attribute is `application:file#anchor` for the general case. `application` and `file` can be omitted if the link target is the current application or file.

There are several different see tags that are to be used depending on what it is that they point to.

<seemfa>

Points to an MFA using the syntax `application:module#function/arity`. These links must point to functions documented in a <funcs> section. Examples:

```
<seemfa marker="stdlib:string#length/1">string:length/1</seemfa>
<seemfa marker="string#length/1">string:length/1</seemfa>
<seemfa marker="#length/1">string:length/1</seemfa>
```

results in: string:length/1.

<seeerl>

Points to an Erlang module or a custom marker within a module. Example:

```
<seeerl marker="stdlib:string">string(3)</seeerl>,
<seeerl marker="stdlib:string#oldapi">Old API in string</seeerl>
```

results in: string(3),Old API in string.

<seetype>

Points to a type using the syntax `application:module#type`. These links must point to types documented in a <datatypes> section. Example:

```
<seetype marker="stdlib:string#grapheme_cluster">string::grapheme_cluster()</seetype>
```

results in: string::grapheme_cluster().

<seeapp>

Points to the application documentation. `index` can be used as the target file. Example:

```
<seeapp marker="stdlib:STDLIB_app">STDLIB app</seeapp>,
<seeapp marker="stdlib:index">STDLIB index</seeapp>
```

results in: STDLIB, STDLIB.

<seecom>

Points to the documentation of any command line utility. Example:

```
<seecom marker="erts:epmd">epmd</seecom>
```

results in: epmd.

<seecref>

Points to the documentation of any C reference. Example:

```
<seecref marker="erts:erl_nif">erl_nif</seecref>
```

results in: erl_nif.

<seefile>

Points to the documentation of a file format. Example:

```
<seefile marker="kernel:config">config(3)</seefile>
```

results in: config(3).

<seeguide>

Points to the User"s Guide of any application. `index` can be used as the target file. Example:

```
<seeguide marker="kernel:index">Kernel User's Guide Index</seeguide>,
<seeguide marker="kernel:logger_chapter">Logging in the Kernel User's Guide</seeguide>
```

results in: Kernel User's Guide Index, Logging in the Kernel User's Guide.

1.7.6 <url> - Non-Local Cross Reference

A reference to a file outside the documentation, a web address or similar, given by the `href` attribute. Must contain plain text. Example:

```
<url href="http://www.erlang.org">erlang.org</url>
```

results in: **erlang.org**

1.8 Character Entities

1.8.1 Added Latin 1

The OTP DTD suite uses the same character entities as defined in HTML 3.2 (`ISO 8879-1986//ENTITIES Added Latin 1//EN//HTML`). That is: for an & (ampersand), use the entity: `&`, for ö use the entity `ö` and so on.

Character	Entity	Description
&	&	ampersand
>	>	greater than

<	<	less than
		no-break space
¡	¡	inverted exclamation mark
¢	¢	cent sign
£	£	pound sterling sign
¤	¤	general currency sign
¥	¥	yen sign
¦	¦	broken (vertical) bar
§	§	section sign
¨	¨	umlaut (dieresis)
©	©	copyright sign
ª	ª	ordinal indicator, feminine
«	«	angle quotation mark, left
¬	¬	not sign
		soft hyphen
®	®	registered sign
¯	¯	macron
°	°	degree sign
±	±	plus-or-minus
²	²	superscript two
³	³	superscript three
´	´	acute accent
µ	µ	micro sign
¶	¶	pilcrow (paragraph sign)
·	·	middle dot
¸	¸	cedilla
¹	¹	superscript one

º	º	ordinal indicator, masculine
»	»	angle quotation mark, right
¼	¼	fraction one-quarter
½	½	fraction one-half
¾	¾	fraction three-quarters
¿	¿	inverted question mark
À	À	capital A, grave accent
Á	Á	capital A, acute accent
Â	Â	capital A, circumflex accent
Ã	Ã	capital A, tilde
Ä	Ä	capital A, dieresis or umlaut mark
Å	Å	capital A, ring
Æ	Æ	capital AE diphthong (ligature)
Ç	Ç	capital C, cedilla
È	È	capital E, grave accent
É	É	capital E, acute accen
Ê	Ê	capital E, circumflex accent
Ë	Ë	capital E, dieresis or umlaut mark
Ì	Ì	capital I, grave accent
Í	Í	capital I, acute accent
Î	Î	capital I, circumflex accent
Ï	Ï	capital I, dieresis or umlaut mark
Ð	Ð	capital Eth, Icelandic
Ñ	Ñ	capital N, tilde
Ò	Ò	capital O, grave accent
Ó	Ó	capital O, acute accent
Ô	Ô	capital O, circumflex accent

Õ	Õ	capital O, tilde
Ö	Ö	capital O, dieresis or umlaut mark
×	×	multiply sign
Ø	Ø	capital O, slash
Ù	Ù	capital U, grave accent
Ú	Ú	capital U, acute accent
Û	Û	capital U, circumflex accent
Ü	Ü	capital U, dieresis or umlaut mark
Ý	Ý	capital Y, acute accent
Þ	Þ	capital THORN, Icelandic
ß	ß	small sharp s, German (sz ligature)
à	à	small a, grave accent
á	á	small a, acute accent
â	â	small a, circumflex accent
ã	ã	small a, tilde
ä	ä	small a, dieresis or umlaut mark
å	å	small a, ring
æ	æ	small ae diphthong (ligature)
ç	ç	small c, cedilla
è	è	small e, grave accent
é	é	small e, acute accent
ê	ê	small e, circumflex accent
ë	ë	small e, dieresis or umlaut mark
ì	ì	small i, grave accent
í	í	small i, acute accent
î	î	small i, circumflex accent
ï	ï	small i, dieresis or umlaut mark

ð	ð	small eth, Icelandic
ñ	ñ	small n, tilde
ò	ò	small o, grave accent
ó	ó	small o, acute accent
ô	ô	small o, circumflex accent
õ	õ	small o, tilde
ö	ö	small o, dieresis or umlaut mark
÷	÷	divide sign
ø	ø	small o, slash
ù	ù	small u, grave accent
ú	ú	small u, acute accent
û	û	small u, circumflex accent
ü	ü	small u, dieresis or umlaut mark
ý	ý	small y, acute accent
þ	þ	small thorn, Icelandic
ÿ	ÿ	small y, dieresis or umlaut mark

Table 8.1: Accented Latin-1 alphabetic characters.

1.9 EEP-48: Implementation in Erlang/OTP

1.9.1 EEP-48: Documentation storage and format

EEP-48 defines a common documentation storage format for module documentation in the Erlang/OTP ecosystem. Erl_Docgen can generate documentation in this format from XML files following the DTD's described in the other User's Guides in this application.

Some special considerations have to be taken when writing documentation that should also be available through EEP-48 style storage.

- The `#PCDATA` within `<name>` tags must be parseable to figure out the arity of the function.
- It is not allowed to mix `<name>` tags with #PCDATA and attributes.
- All `<name>` tags within `<func>` has to have a `since` attribute.
- All callback function documentations have to start with a `Module` prefix.

1.9.2 Erlang Documentation Format

When generating documentation for EEP-48 Erl_Docgen uses the format mime type <<"application/erlang+html">>. The documentation content is an Erlang term that represents an HTML like structure.

```
-type chunk_elements() :: [chunk_element()].
-type chunk_element() :: {chunk_element_type(),chunk_element_attrs(),
                          chunk_elements()} | unicode:unicode_binary().
-type chunk_element_attrs() :: [chunk_element_attr()].
-type chunk_element_attr() :: {atom(),unicode:unicode_binary()}.
-type chunk_element_type() :: chunk_element_inline_type() | chunk_element_block_type().
-type chunk_element_inline_type() :: a | code | strong | b | em | i.
-type chunk_element_block_type() :: p | 'div' | br | pre | ul |
                                    ol | li | dl | dt | dd |
                                    h1 | h2 | h3 | h4 | h5 | h6.
```

The different element types follow their HTML meaning when rendered. The following are some general rules for how the chunk elements are allowed to be generated.

- Inline and `pre` elements are not allowed to contain block elements.
- `p` elements are not allowed to be nested.

The attributes on some elements have a special meaning.

`{'div',[{class,unicode:unicode_binary()}],_}`
: The class name will be used to provide styling to the content in the div. The types of classes used by Erlang/OTP are: `warning`, `note`, `do`, `dont` and `quote`.

`{ul,[{class,<<"types">>}],_}`
: This is a list containing type documentation.

`{li,[{name,TypeName :: unicode:unicode_binary()}],_}`
: A list item with a type specification located in the metadata of this modules EEP-48 documentation. The implementation should look for the AST representation of the type under the `types` key. This attribute is only valid under a `ul` with class <<"types">>.

`{li,[{class,<<"type">>}],_}`
: A list item with the type described in the Erlang Documentation Format. This attribute is only valid under a `ul` with class <<"types">>.

`{li,[{class,<<"description">>}],_}`
: A list item with the description of the type previous in the list. This attribute is only valid under a `ul` with class <<"types">>.

The `shell_docs:validate/1` function can be used to do a validation of the Erlang Documentation Format.

1.9.3 Erlang Documentation extra Metadata

Erlang/OTP uses some extra metadata fields to embed more information into the EEP-48 docs.

- Fields on module level:

  `otp_doc_vsn := {non_neg_integer(),non_neg_integer(),non_neg_integer()}`
  : Describes the version of the Erlang Documentation Format used within this module

  `types := #{ TypeName :: unicode:unicode_binary() => TypeAST }`
  : A map containing the AST of the types that are part of this module. This map is used to by functions and callbacks to render the types inline into their documentation.

- Fields on functions and types:

  `signature := SpecAST`
  : The spec AST associated with this function. It is used to render a more descriptive slogan for the documentation entry.

`equiv := {Type,Name,Arity}`
: The current function/type shares documentation with another function/type. This means that if this and the target function/type are to be shown at the same time only the prototype of this function/type should will be displayed and the documentation will use a common body of text.

1.9.4 See Also

`shell_docs(3)`, `code:get_doc(3)`

2 Reference Manual

The **erl_docgen** supports the OTP documentation build.

erl_docgen

Application

The application consists of the following parts:

XSL

A number of XSL files that is used to transform the xml files to html, pdf or man pages.

DTDs

The DTDs used for the OTP documentation.

escripts

Some scripts that is used to produce xml files according to OTP DTDs from some different input.

misc

Erlang logo, javascripts and css stylesheets used in the documentation.