DocBuilder

version 0.9

Typeset in LaTeX from SGML source using the DocBuilder-0.9.8.4 Document System.

Contents

Chapter 1

DocBuilder User's Guide

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

1.1 Overview

1.1.1 Background

DocBuilder has been used within the OTP project to generate documentation for Erlang/OTP itself for more than ten years. It has now been released as a regular Erlang/OTP application.

The intention with DocBuilder is that it should be as easy to use and maintain as possible and generate adequate documentation for OTP's needs. It uses frames, which can probably be regarded as old-fashioned today. Hopefully, this should be improved in the future.

Originally, DocBuilder input was SGML files and external tools was used for parsing. The internal version used in the OTP project can generate not only HTML code but also LaTeX (for PDF and PostScript) and nroff (for UNIX man pages). (Again, using external tools). Because of this, the parsed source code is transformed into a tree structure before being transformed again into the desired format.

1.1.2 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 [page 3]) A collection of chapters (chapter [page 4]).

Reference Manual (DTD: application [page 5] A collection of manual pages for modules (erlref [page 9]), applications (appref [page 6]), commands (comref [page 7]), C libraries (cref [page 8]) and files (fileref [page 11]).

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.

A special kind of DTD, fascicules [page 14], can be used to specify the differenct parts of the documentation, and which one of those should be shown as default.

1.1.3 Structure of Generated HTML

The generated HTML corresponding to a `part` or `application` document is split into a left frame and a right frame. The left frame contains information about the document and links to the included files, that is chapters or manual pages. The right frame is used to display either the front page for the document, or the selected chapter/manual page.

The left frame also contains links to a bibliography and a glossary, which are automatically generated.

In the case of an `application` document, the left frame also contains a link to an automatically generated index.

1.1.4 Basic Tags

All DTDs in the DocBuilder 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 [page 17] and inline tags [page 23]. 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.

1.1.5 About This Document

In this User's Guide, 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 DocBuilder documentation itself:

- This User's Guide is written using the `part` and `chapter` DTDs.
- The Reference Manual is written using the `application`, `appref` and `erlref` DTDs.

1.1.6 Usage

1. Create the relevant XML files.
   If there are EDoc comments in a module, the function docb_gen:module/1,2 [page 34] can be used to generate an XML file according to the `erlref` DTD for this module.
2. The XML files can be validated using docb_xml_check:validate/1 [page 40].
3. Generate HTML files by using docb_transform:file/1,2 [page 37].

1.2 User's Guide DTDs

1.2.1 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 [page 4] 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>
```

1.2.2 <part>

The top level tag of a `part` DTD.

Contains a <header> [page 15], an optional <description> [page 3], followed by one or more <include> [page 3].

1.2.3 <description>

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

Contains any combination and any number of block tags [page 17] except <image> and <table>.

1.2.4 <include>

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

Example:

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

1.2.5 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>
```

1.2.6 <chapter>

The top level tag of a `chapter` DTD.

Contains a <header> [page 15], an optional introduction consisting of any combination of block tags [page 17], followed by one or more <section> [page 4].

1.2.7 <section>

Subdivision of a chapter.

Contains an optional <marker> [page 24], a <title> [page 5], followed by any combination and any number of block tags [page 17] and <section>.

1.2.8 <title>

Section title, contains plain text.

1.3 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.

1.3.1 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 [page 6], comref [page 7], cref [page 8], erlref [page 9], or fileref [page 11] 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>
```

1.3.2 <application>

The top level tag of an application DTD.

Contains a <header> [page 15], an optional <description> [page 3], followed by one or more <include> [page 3].

1.3.3 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> [page 15], <app> [page 7], <appsummary> [page 7], <description> [page 12], zero or more <section> [page 12] and <funcs> [page 12], followed by zero or more <authors> [page 13].

<app>

The application name. Contains plain text.

<appsummary>

Short summary. Contains plain text.

1.3.4 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> [page 15], <com> [page 8], <comsummary> [page 8], <description> [page 12], zero or more <section> [page 12] and <funcs> [page 12], followed by zero or more <authors> [page 13].

<com>

The command name. Contains plain text.

<comsummary>

Short summary. Contains plain text.

1.3.5 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>
    <p>A longer description of the C library.</p>
  </description>

  <funcs>
    <func>
      <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>
    <title>A title</title>

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


</cref>
```

<cref>

The top level tag for a `cref` DTD.

Contains <header> [page 15], <lib> [page 9], <libsummary> [page 9], <description> [page 12], zero or more <section> [page 12] and <funcs> [page 12], followed by zero or more <authors> [page 13].

<lib>

The C library name or acronym. Contains plain text.

<libsummary>

Short summary. Contains plain text.

1.3.6 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> [page 15], <module> [page 10], <modulesummary> [page 11], <description> [page 12], zero or more <section> [page 12] and <funcs> [page 12], followed by zero or more <authors> [page 13].

<module>

The module name. Contains plain text.

<modulesummary>

Short summary. Contains plain text.

1.3.7 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> [page 15], <file> [page 12], <filesummary> [page 12], <description> [page 12], zero or more <section> [page 12] and <funcs> [page 12], followed by zero or more <authors> [page 13].

<file>

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

<filesummary>

Short summary. Contains plain text.

1.3.8 <description>

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

Contains any combination and any number of block tags [page 17] except <image> and <table>.

1.3.9 <section>

Subdivisions of the document. Contains an optional <marker> [page 24], a <title> [page 5], followed by any combination and any number of block tags [page 17] except <image> and <table>.

1.3.10 <funcs>

A group of "formal" function definitions.

Contains one or more <func> [page 12].

1.3.11 <func>

A "formal" function definition.

Contains one or more <name> [page 12], followed by <fsummary> [page 13], <type> [page 13] (optional) and <desc> [page 13] (optional).

1.3.12 <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, DocBuilder will automatically try to add a marker [page 24], <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>
```

DocBuilder will create a marker <marker id="foo/2"> before the function definition in the generated HTML. That is, referring to the function using <seealso marker="#foo/2">foo/2</seealso> will automatically work.

1.3.13 <fsummary>

Function/command summary. Contains plain text, <c> [page 24] and [page 24].

1.3.14 <type>

Type declarations for the function/command.
Contains one or more pairs of <v> [page 13] and <d> [page 13] (optional).

1.3.15 <v>

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

1.3.16 <d>

Description for an argument or return value. Contains plain text, <c> [page 24] and [page 24].

1.3.17 <desc>

Function/command description. Contains block tags [page 17] except <image> and <table>.

1.3.18 <authors>

Authors of the manual page. The `authors` element is optional.
Contains one or more pairs of <aname> [page 13] and <email> [page 13].

1.3.19 <aname>

Author name. Contains plain text.

1.3.20 <email>

Author email address. Contains plain text.

1.4 Fascicules DTDs

1.4.1 The fascicules DTD

The `fascicules` DTD is a special kind of DTD which can be used to specify the different parts of the documentation, and which one of those should be shown as default.

Example:

```
<?xml version="1.0" encoding="latin1" ?>
<!DOCTYPE fascicules SYSTEM "fascicules.dtd">
<fascicules>
  <fascicule file="part" href="part_frame.html" entry="no">
    User's Guide
  </fascicule>
  <fascicule file="ref_man" href="ref_man_frame.html" entry="yes">
    Reference Manual
  </fascicule>
  <fascicule file="part_notes" href="part_notes_frame.html" entry="no">
    Release Notes
  </fascicule>
</fascicules>
```

In the example, it is specified that the documentation for this application consists of three parts: User's Guide, where the "cover page" (with the two frames) is located in `part_frame.html`, Reference Manual with the cover page `ref_man_frame.html` and Release Notes with the cover page `part_notes_frame.html`.

As a result, at the top of the left frame in the generated HTML documentation, there will be corresponding links to User's Guide, Reference Manual and Release Notes.

The attribute `entry="yes"` specifies that it is the Reference Manual which should be shown as default. This means that when generating the HTML files, `application_frame.html` will be copied to `index.html`.

> **Note:**
> DocBuilder assumes that the XML file written according to the `fascicules` DTD is called `fascicules.xml`.

This file is optional. If it does not exist, there are no links to other parts of the documentation (as they are not known) in the left frame, and no `index.html` is created.

1.4.2 <fascicules>

Top level tag for the `fascicules` DTD.

Contains one or more <fascicule> [page 15].

1.4.3 <fascicule>

Specifies properties for one “part” of the documentation for an application.

Contains plain text, the name of this part.

The `file` attribute should specify the file name for the corresponding `part` or `application`, without the `.xml` extension.

The `href` attribute should specify the file name for the corresponding HTML cover page file, without the `.html` extension.

The optional `entry="yes"|"no"` attribute specfies if the HTML cover page should be copied to `index.html` or not. Default is `"no"`.

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 by DocBuilder. It will likely be removed in the future.

1.5.6 <prepared>

This tag is intended for administrative use and is ignored by DocBuilder.

1.5.7 <responsible>

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

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 by DocBuilder.

1.5.10 <checked>

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

1.5.11 <date>

This tag is intended for administrative use and is ignored by DocBuilder.

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 by DocBuilder.

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 DocBuilder DTD suite: <p> [page 20], <pre> [page 20], <code> [page 17], <list> [page 19], <taglist> [page 21], <codeinclude> [page 18] and <erleval> [page 19].

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 by DocBuilder. Default value is `"none"`

> **Note:**
> No tags are allowed within the tag and no character entities [page 27] 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="gazonk" tag="%% Erlang example"/>
```

results in:

```
-module(gazonk).

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

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

provided there is a file named `gazonk` looking like this:

```
...

%% Erlang example
-module(gazonk).

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 by DocBuilder. Default value is `"none"`

1.6.4 <erleval> - Erlang Evaluation

Include the result from evaluating an Erlang expression. Example:

```
<erleval expr="{A,b,C}={a,b,c}. "/>
```

results in:

```
{a,b,c}
```

Note the '.' and space after the expression.

1.6.5 <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 [page 17] and inline tags [page 23]. Example:

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

results in:

1. Askosal:
    - Nullalisis
    - Facilisis
2. Ankara

1.6.6 <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 [page 24] section.

1.6.7 <p> - Paragraph

Paragraphs contain plain text and inline tags [page 23]. 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.

1.6.8 <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.

1.6.9 <pre> - Pre-formatted Text

Used for documentation of system interaction. Can contain text, seealso [page 25], url [page 25] 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 [page 27] are expanded.

1.6.10 <quote> - Quotation

Highlight quotations from other works, or dialog spoken by characters in a narrative. Contains one or more <p> [page 20] 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.

1.6.11 <taglist> - Definition List

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

<tag> can contain plain text, <c> [page 24], [page 24], <seealso> [page 25] and <url> [page 25] tags.

<item> can contain plain text, the common subset of block tags [page 17] and inline tags [page 23]. 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.

1.6.12 <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.

1.6.13 <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 1.1: A Silly Man

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

1.6.14 <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 horizontically and vertically. Default is “`left`” and “`middle`”.

Each cell contains plain text and inline tags [page 23]. 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 1.1: 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 [page 17] 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 [page 17] tag. All character entities [page 27] 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> [page 24] 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 <seealso> [page 25] tag is used to refer to the marker.

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

1.7.5 <path> - Path

Highlights file paths. The attributes `unix` and `windows` makes it possible to specify different paths for different file path notations. Default for both are "". Example:

```
<p>Look at the <path unix=".profile" windows="win.ini">start-up file</path>
  if you intend to alter the initial behavior.</p>
```

If no `ptype` option is specified when calling docb_transform:file/1,2 [page 37], this simply results in:

"Look at the start-up file if you intend to alter the initial behavior."

If both the options {`ptype`,`unix`} and {`ptype`,`windows`} are specified, the example instead results in:

"Look at the start-up file[1] if you intend to alter the initial behavior."

[1] In Windows: `win.ini`. In UNIX: `.profile`.

1.7.6 <seealso> - Local Cross Reference

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. Must contain plain text. Examples:

```
<seealso marker="#marker_example">marker example</seealso>
```

results in: marker example [page 24] (a hypertext link to the marker example above).

```
<seealso marker="block_tags#markerTAG">marker tag</seealso>
```

results in: marker tag [page 19] (a hypertext link to the marker section in the Block Tags chapter).

```
<seealso marker="overview">Overview</seealso>
```

results in: Overview [page 1] (a hypertext link to the Overview chapter).

Note the use of "#" before the name of the marker. Note also that the filename extension `.html` is omitted. This is because the default behavior of DocBuilder is to translate `<seealso marker="File#Marker">text</seealso>` to `<A HREF="File.html#Marker">text</A>`.

The default behaviour can be modified by using the callback module option to `docb_transform:file/1,2` and defining a callback function Module:seealso/1 [page 39]. This possibility is for example used in OTP to resolve cross references between applications.

1.7.7 <url> - Non-Local Cross Reference

A reference to a file outside the documentation, a web adress 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[2]

[2] URL: `http://www.erlang.org`

1.7.8 <term>, <termdef> - Glossary

Used to highlight a term with a local (for this document only) or global definition. The identity of the term is given by the `id` attribute.

For a locally defined term, the tag contains a `<termdef>`, which in turn contains an explanation of the term as plain text. Example:

```
<term id="HTML"><termdef>Hyper-Text Markup Language</termdef></term>
```

For a globally defined term, the tag is empty. Example:

```
<term id="HTML"/>
```

Global definitions are given to DocBuilder in a file, using the docb_transform:file/1,2 [page 37] option `term_defs`. The file should contain a list of tuples, one for each term definition, on the format `{Id,Name,Definition,Owner}`. The `Owner` part is just for administration, if there are several people contributing to a term definition file. Example:

```
[...,
 {"HTML", "HTML", "Hyper-Text Markup Language", "Gunilla"},
 ...].
```

DocBuilder will collect both local and global definitions in a glossary, which can be reached from a link in the left frame of the HTML documentation.

In the generated HTML, it is the term name which will be visible. For locally defined terms, the id and the name are the same. The name has a hypertext link to the definition in the glossary. Example:

```
<term id="HTML"><termdef>Hyper-Text Markup Language</termdef></term>
```

results in: *HTML*

If a term is defined both locally and globally, the global definition takes precedence.

1.7.9 <cite>, <citedef> - Bibliography

Works the same way as `<term>` and `<termdef>`, but for a bibliography list rather than a glossary.

A global bibliography list is given to DocBuilder in a file, using the docb_transform:file/1,2 [page 37] option `cite_defs`. The file should contain a list of tuples, one for each cite, on the format `{Id,Title,Info,Owner}`. The `Owner` part is just for administration, if there are several people contributing to a bibliography file. Example:

```
[...,
 {"erlbook","Concurrent Programming in ERLANG","J. Armstrong, R. Virding, C. Wikstrm, M. Williams, Concu
 ...].
```

1.8 Character Entities

1.8.1 Added Latin 1

The DocBuilder 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 use the entity `&ouml;` and so on.

Character	*Entity*	*Description*
&	&amp;	ampersand
>	&gt;	greater than
<	&lt;	less than
		no-break space
	&iexcl;	inverted exclamation mark
	&cent;	cent sign
	&pound;	pound sterling sign
	&curren;	general currency sign
	&yen;	yen sign
	&brvbar;	broken (vertical) bar
	&sect;	section sign
	&uml;	umlaut (dieresis)
	&copy;	copyright sign
ł	&ordf;	ordinal indicator, feminine
	&laquo;	angle quotation mark, left
	&not;	not sign
	&shy;	soft hyphen
	&reg;	registered sign
	&macr;	macron
	&deg;	degree sign
	&plusmn;	plus-or-minus
	&sup2;	superscript two
	&sup3;	superscript three
	&acute;	acute accent
	&micro;	micro sign
	&para;	pilcrow (paragraph sign)
	&middot;	middle dot
	&cedil;	cedilla
	&sup1;	superscript one
	&ordm;	ordinal indicator, masculine
	&raquo;	angle quotation mark, right
	&frac14;	fraction one-quarter
	&frac12;	fraction one-half

continued ...

... continued

	¾	fraction three-quarters
	¿	inverted question mark
	À	capital A, grave accent
	Á	capital A, acute accent
	Â	capital A, circumflex accent
	Ã	capital A, tilde
	Ä	capital A, dieresis or umlaut mark
	Å	capital A, ring
	Æ	capital AE diphthong (ligature)
	Ç	capital C, cedilla
	È	capital E, grave accent
	É	capital E, acute accen
	Ê	capital E, circumflex accent
	Ë	capital E, dieresis or umlaut mark
	Ì	capital I, grave accent
	Í	capital I, acute accent
	Î	capital I, circumflex accent
	Ï	capital I, dieresis or umlaut mark
	Ð	capital Eth, Icelandic
	Ñ	capital N, tilde
	Ò	capital O, grave accent
	Ó	capital O, acute accent
	Ô	capital O, circumflex accent
	Õ	capital O, tilde
	Ö	capital O, dieresis or umlaut mark
	×	multiply sign
	Ø	capital O, slash
	Ù	capital U, grave accent
	Ú	capital U, acute accent
	Û	capital U, circumflex accent
	Ü	capital U, dieresis or umlaut mark
	Ý	capital Y, acute accent
	Þ	capital THORN, Icelandic
	ß	small sharp s, German (sz ligature)
	à	small a, grave accent
	á	small a, acute accent
	â	small a, circumflex accent
	ã	small a, tilde
	ä	small a, dieresis or umlaut mark

continued ...

... continued

	&aring;	small a, ring
	&aelig;	small ae diphthong (ligature)
	&ccedil;	small c, cedilla
	&egrave;	small e, grave accent
	&eacute;	small e, acute accent
	&ecirc;	small e, circumflex accent
	&euml;	small e, dieresis or umlaut mark
	&igrave;	small i, grave accent
	&iacute;	small i, acute accent
	&icirc;	small i, circumflex accent
	&iuml;	small i, dieresis or umlaut mark
	&eth;	small eth, Icelandic
	&ntilde;	small n, tilde
	&ograve;	small o, grave accent
	&oacute;	small o, acute accent
	&ocirc;	small o, circumflex accent
	&otilde;	small o, tilde
	&ouml;	small o, dieresis or umlaut mark
	&divide;	divide sign
	&oslash;	small o, slash
	&ugrave;	small u, grave accent
	&uacute;	small u, acute accent
	&ucirc;	small u, circumflex accent
	&uuml;	small u, dieresis or umlaut mark
	&yacute;	small y, acute accent
	&thorn;	small thorn, Icelandic
	&yuml;	small y, dieresis or umlaut mark

Table 1.2: Accented Latin-1 alphabetic characters.

DocBuilder Reference Manual

Short Summaries

- Application docbuilder [page 33] – The DocBuilder Application
- Erlang Module docb_gen [page 34] – Generate XML from EDoc comments in Erlang source code.
- Erlang Module docb_transform [page 37] – Transform XML to HTML
- Erlang Module docb_xml_check [page 40] – Validate XML documentation source code

docbuilder

No functions are exported.

docb_gen

The following functions are exported:

- `module(File) -> ok | {error, Reason}`
  [page 34] Generate XML from EDoc comments in Erlang source code.
- `module(File, Options) -> ok | {error, Reason}`
  [page 34] Generate XML from EDoc comments in Erlang source code.
- `users_guide(File) -> ok | {error, Reason}`
  [page 34] Generate XML from EDoc comments in Erlang source code
- `users_guide(File, Options) -> ok | {error, Reason}`
  [page 34] Generate XML from EDoc comments in Erlang source code

docb_transform

The following functions are exported:

- `file(File) -> ok | {error, Reason}`
  [page 37] Transform XML to HTML
- `file(File, Options) -> ok | {error, Reason}`
  [page 37] Transform XML to HTML
- `Module:head() -> string()`
  [page 38] Snippet to be included in head of a document.

- `Module:top() -> string()`
  [page 38] Snippet to be included at the top of a document.
- `Module:bottom() -> string()`
  [page 39] Snippet to be included at the bottom of a document.
- `Module:seealso(SeeAlso) -> Href`
  [page 39]

docb_xml_check

The following functions are exported:

- `validate(File) -> ok | error | {error, badfile}`
  [page 40] Validate XML source code.

docbuilder

Application

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

Limitations

DocBuilder is primarily intended for generating documentation for Erlang/OTP itself. That is, no attempt has been made to create a tool suitable for generating documentation in general.

See Also

DocBuilder User's Guide, docb_gen(3) [page 34], docb_transform(3) [page 37] [page 40]

docb_gen

Erlang Module

docb_gen contains functions for generating XML documentation source code according to the erlref or chapter DTD from [EDoc] comments in Erlang source code or an overview.edoc file, using EDoc.

Exports

```
module(File) -> ok | {error, Reason}
module(File, Options) -> ok | {error, Reason}
```

Types:

- File = string()
- Options = [Opt]
- Opt = {def,Defs} | {includes,Dirs} | {preprocess,Bool} | {sort_functions,Bool}
- Defs = [{atom(),string()}]
- Dirs = [string()]
- Bool = bool()
- Reason = badfile | {badopt,term()} | term()

Generates XML documentation source code according to the erlref DTD from EDoc comments File, using the EDoc application.

File is an Erlang source file, given with or without the .erl extension as Name.erl or Name. The resulting XML file is created in the current working directory and named Name.xml.

Options is a list of options, see below.

Returns ok if successful, and an error tuple otherwise.

```
users_guide(File) -> ok | {error, Reason}
users_guide(File, Options) -> ok | {error, Reason}
```

Types:

- File – see module/1,2
- Options – see module/1,2
- Reason – see module/1,2

Like module/1,2 but generates XML source code according to the chapter DTD from an overview.edoc or similar file.

The resulting file is named chapter.xml.

Options

{def, [{Name,Text}]} Specifies EDoc macro definitions. See [edoc:get_doc/2].

{includes, [Dir]} Specifies directories where EDoc should search for include files. See [edoc:read_source/2].

{preprocess, true|false} Specifies if EDoc should read the source file via the Erlang preprocessor. Default is false. See [edoc:read_source/2].

{sort_functions, true|false} Specifies if the functions in the resulting XML file should be sorted alphabetically. Default is true.

Limitations

The mapping from the EDoc XHTML output to valid Erlang/OTP XML is not complete. An attempt has been made to cover the most commonly used XHTML constructs, but there will still be cases where XML generation fails or where the resulting XML is inadequate. This is especially true for users_guide/1,2.

Known limitations for some XHTML tags:

<a> All attributes except the first href or name attribute are ignored.
A href attribute means the <a> tag will be transformed to a <seealso> or <url> tag and an attempt is made to resolve the reference if necessary.
A name attribute means the <a> tag will be transformed to a <marker> tag.

<b>, , <pre> Cannot contain other tags in Erlang/OTP XML, content is converted to plain text.

 No corresponding Erlang/OTP XML tag, converted to plain text.

<font> No corresponding Erlang/OTP XML tag, converted to plain text.

<h1>, <h2>, ... There is no tag corresponding to a header in Erlang/OTP XML, so these are converted to plain text instead, with the exception of <h3> and <h4> tags within overview.edoc, see part about "chapter DTD" below.

<sup> There is no tag corresponding to superscript in Erlang/OTP XML, so this is converted to plain text within brackets "(..)".

References The markers automatically inserted by EDoc at each heading and function will override the markers automatically inserted by DocBuilder, with the unfortunate result that the links in the left-hand frame of the User's Guide will not work, and also that cross referencing a function in a module the usual Erlang/OTP way "<seealso marker="edoc:edoc#run/3...>" does not work. (But "<seealso marker="edoc:edoc#run-3...>" does.)

erlref DTD

Tables Tables are not allowed. The contents of a table is converted to text instead, each row corresponding to one line of text.

chapter DTD

Sections Only two levels of sections. <h3> (equivalent to EDoc headings "== Heading ==") is interpreted as start of top-level section, or if there is no <h3> tag, the entire document is made into one top-level section. <h4> (equivalent to EDoc sub-headings ("=== Sub-heading ===") is interpreted as start of second-level section.

Tables Tables without borders are converted to text in the same manner as for the `erlref` DTD.

docb_transform

Erlang Module

docb_transform contains functions for transforming XML documentation source code to HTML.

Exports

```
file(File) -> ok | {error, Reason}
file(File, Options) -> ok | {error, Reason}
```

Types:

- File = string()
- Options = [Opt]
- Opt – see below

Transforms XML documentation source code to HTML.

File is a documentation source file, given with or without the .xml extension as Name.xml or Name.

If File contains XML code according to a basic DTD (chapter, erlref, ...), the resulting HTML file is named Name.html.

If File contains XML code according to a compound DTD (application or part), several files are created:

- A cover page for the application with two frames, Name_frame.html.
- The contents of the left frame and a front page, Name.html and Name_first.html.
- A bibliography and a glossary, Name_cite.html and Name_term.html.
- In the case of an application DTD an index is created, Name.kwc and Name_index.html.
- One HTML file for each file included from File.
- Also, if there exists a fascicules.xml file where the value of the entry attribute for File is "yes", the cover page is copied to index.html.

Options

{html_mod, Module}, Module=atom() A callback module can be used for specifying HTML snippets that should be included in the generated HTML files, see below.

{outdir, Dir}, Dir=string() Destination for generated files. Default is current working directory.

{number, Number}, Number=int() First chapter number when transforming a chapter file. Default is 1.

{ptype, unix|windows} For path elements, the specified file path should be presented.

silent Silent - no warnings, only error information is printed.

{top, Index}, Index=string() Specifies the value of "Top" in the left frame of a front page, which normally should be some kind of top index file for the documentation.

{vsn, Vsn}, Vsn=string() Application version number. Overrides a version number defined in the XML document. Visible in the left frame and on the front page.

{term_defs, File}, File=string() Use the global glossary definitions in File, which should contain a list of tuples {Id, Name, Definition, Owner}. See the section <term>, <termdef> - Glossary [page 25] in the User's Guide.

{cite_defs, File}, File=string() Use the global bibliography definitions in File, which should contain a list of tuples {Id, Title, Info, Owner}. See the section <cite>, <citedef> - Bibliography [page 26] in the User's Guide.

Callback Module

A html_mod callback module can include the functions specified below. Note that there is no check that the resulting HTML code is valid. All functions are optional.

Exports

Module:head() -> string()

Defines a HTML snippet to be included in the head of a document, after the <HEAD> start tag and <TITLE> tag:

```
<HTML>
<HEAD>
  <TITLE>...</TITLE>
  - snippet is included here -
  ...
</HEAD>
...
</HTML>
```

Module:top() -> string()

Defines a HTML snippet to be included at the top of a document, after the <BODY> start tag.

`Module:bottom() -> string()`

Defines a HTML snippet to be included at the bottom of a document, before the </BODY> end tag.

`Module:seealso(SeeAlso) -> Href`

Types:

- SeeAlso = Href = string()

When referring to another part of the document, or another document, the XML tag <seealso> is used: <seealso marker="File#Marker">...text...</seealso>. By default, this is translated to <A HREF="File.html#Marker>...text...</A>.

This function makes it possible to specify an alternative translation `Href` of the `marker` attribute value `SeeAlso`. For example, in OTP this is used to resolve cross references between applications.

docb_xml_check

Erlang Module

docb_xml_check contains functions for validating XML documentation source code.

Exports

validate(File) -> ok | error | {error, badfile}

Types:

- File = string()

Validates the XML documentation source code in File. The .xml extension can be omitted.

Returns ok if successful, otherwise error information is printed and the function returns error. If File does not exist, {error, badfile} is returned.

List of Figures

List of Tables

Glossary

HTML

Hyper-Text Markup Language
Local for chapter 1.

HTML

Hypertext Markup Language.

Index of Modules and Functions

Modules are typed in *this way*.
Functions are typed in `this way`.