Specification of a Simple Document Language

Table of Contents

1 Abstract

The `document` format is a **plain text** input language designed to express simple documents. It supports basic text formatting which consists of **paragraphs**, **lists**, **bold**, *italic* and **fixed width** text as well as **hyperlinks**. It allows structuring documents by means of **sections** with headlines. It also supports embedding of **tables**, **preformatted plain text** and media such as but not limited to **images**, **audio** and **video** data.

Its implementation features an `HTML5` compatible output generator. The resulting *HTML* output can be combined with *CSS style sheets* to produce **rich web documents** and **print media** (see http://www.princexml.com/).

This specification of the `document` format itself is written in the `document` language and serves also as a tutorial and as an example for its usage.

2 Formatting text

The abstract of this document already uses most text formatting functions, namely paragraphs, bold, italic and fixed width text and a hyperlink.

Type	Control character(s)	Example	Rendering
Bold	*	`*Bold text*`	**Bold text**
Italic	_	`_Italic text_`	*Italic text*
Fixed width	{ and }	`{Fixed width text}`	`Fixed width text`
Hyperlink	[and]	`[http://google.de]`	http://google.de

Figure 2.1 Typesetting modifiers and their control characters.

Pragraphs are set by seperating text passages with a blank line. Typesetting modifiers like bold text are expressed by surrounding parts of text with *control characters* as seen in figure 2.1.

```
The {document} format is a *plain text* input language designed to
express simple documents. It supports basic text formatting which
consists of *paragraphs*, *lists*, *bold*, *italic* and *fixed width*
text as well as *hyperlinks*. It allows structuring documents by means
of *sections* with headlines. It also supports embedding of *tables*,
*pre formatted plain text* and media such as but not limited to
*images*, *audio* and *video* data.

Its implementation features an {HTML5} compatible output generator. The
resulting _HTML_ output can be combined with _CSS style sheets_ to
produce *rich web documents* and *print media*
(see [http://www.princexml.com/]).
```

Figure 2.2 Source input for the first two paragraphs of the abstract section.

Figure 2.2 shows an example input source demonstrating paragraphs and all available typesetting modifiers.

```
+ List item one.
+ A list item with
  a line break.
+ List item three.
```

Figure 2.3 Source input for a list.

Another supported form of text formatting is the list. Users can create lists by prepending the text passages that make up list items with the `+` character. For instance the source input from figure 2.3 expresses a list with three items and could be rendered like this:

- List item one
- A list item with a line break
- List item three

3 Quoting control characters

Sometimes input text will contain control characters that invoke a syntax rule when it is not desired. In those cases users will need to *quote* the offending character so that it is not interpreted as a control character.

For example if a user wants to input the word "foo*bar" he needs to quote the * control character with the quote character \. The quote character itself can be quoted with a quote character.

```
For example if a user wants to input the word "foo\*bar" he needs to
quote the {*} control character with the quote character {\\}.
```

Figure 3.1 Source input with quoted control characters.

The interpreter of the `document` input language is designed to reduce the need for quoting as much as possible.

4 Structuring documents

Documents can be structured by dividing them into *sections*. Each section consists of a headline and a body. A sections headline is a regular text passage. The body of a section may contain any elements of a document including subsections.

```
< Section headline

 Section body...

 < Subsection headline

  Subsection body...

 >

>
```

Figure 4.1 Source input for a section.

Sections are opened with the < character followed by whitespace and a text passage that will be the section header. A blank line signifies the end of the section header and the start of the section body wich itself is terminated by the > character.

The indenting of the section body in figure 4.1 is not mandated by the language but merely a convention to improve source input file readability.

5 Embedding figures

Documents can include three types of non-text objects. Namely **tables**, **media objects** such as but not restricted to images, audio or video files and **preformatted plain text**.

Embedding type	Keyword
Table	`TABLE`
Media object	`MEDIA`
Preformatted plain text	`CODE`

Figure 5.1 Embedding keywords.

All three types of embedding share the same header syntax. An embedding starts with the # character followed by a *keyword* (see figure 5.2). The keyword is followed by whitespace and a text passage terminated by another # character. The text passage will be the embeddings caption. The header is followed by the actual embedding wich varies for the different kinds of embedding types.

```
#KEYWORD HEADLINE#
...
```

Figure 5.2 Generic syntax for embedding headers.

5.1 Tables

A table body consists of one row per line. Rows consist of columns wich start with the | (vertical bar) character followed by a text passage. A blank line signifies the end of the table.

```
#table *Figure 5.2* Embedding keywords.#
| Embedding type           | Keyword
| Table                    | {TABLE}
| Media object             | {MEDIA}
| Preformatted plain text  | {CODE}
```

Figure 5.3 Source input for the table from figure 5.1.

Alignment of the table columns as seen in figure 5.3 is not not mandated by the language but merely a convention to improve source input file readability.

5.2 Media

Embedding arbitrary media is as simple as specifying a *URL* after the emdedding header. A blank line after the url signifies its end. How given media is rendered depends on the output generator. The *HTML* output generator supports images, audio and video data.

```
#media Media caption.#
http://example.org/media
```

Figure 5.4 Source input for a media embedding.

5.3 Preformatted plain text

After the `CODE` header arbitrary plain text can be entered. The plain text passage is terminated by another # character. The plain text will remain its formatting while the interpreter takes into account the indenting of the embedding.

```
#code Code caption.#
arbitrary plain text
which will remain unchanged
#
```

Figure 5.5 Source input for a preformatted plain text embedding.