AsciiDoc is a text document format for writing notes, documentation, articles, books, ebooks, slideshows, web pages, blogs and UNIX man pages. AsciiDoc files can be translated to many formats including HTML, PDF, EPUB, man page. AsciiDoc is highly configurable: both the AsciiDoc source file syntax and the backend output markups (which can be almost any type of SGML/XML markup) can be customized and extended by the user.

This document

This is an overly large document, it probably needs to be refactored into a Tutorial, Quick Reference and Formal Reference.

If you're new to AsciiDoc read this section and the Getting Started section and take a look at the example AsciiDoc (`*.txt`) source files in the distribution `doc` directory.

AsciiDoc is a plain text human readable/writable document format that can be translated to DocBook or HTML using the asciidoc(1) command. You can then either use asciidoc(1) generated HTML directly or run asciidoc(1) DocBook output through your favorite DocBook toolchain or use the AsciiDoc a2x(1) toolchain wrapper to produce PDF, EPUB, DVI, LaTeX, PostScript, man page, HTML and text formats.

The AsciiDoc format is a useful presentation format in its own right: AsciiDoc markup is simple, intuitive and as such is easily proofed and edited.

AsciiDoc is light weight: it consists of a single Python script and a bunch of configuration files. Apart from asciidoc(1) and a Python interpreter, no other programs are required to convert AsciiDoc text files to DocBook or HTML. See Example AsciiDoc Documents below.

Text markup conventions tend to be a matter of (often strong) personal preference: if the default syntax is not to your liking you can define your own by editing the text based asciidoc(1) configuration files. You can also create configuration files to translate AsciiDoc documents to almost any SGML/XML markup.

asciidoc(1) comes with a set of configuration files to translate AsciiDoc articles, books and man pages to HTML or DocBook backend formats.

My AsciiDoc Itch

DocBook has emerged as the de facto standard Open Source documentation format. But DocBook is a complex language, the markup is difficult to read and even more difficult to write directly — I found I was spending more time typing markup tags, consulting reference manuals and fixing syntax errors, than I was writing the documentation.

See the `README` and `INSTALL` files for install prerequisites and procedures. Packagers take a look at Packager Notes.

The best way to quickly get a feel for AsciiDoc is to view the AsciiDoc web site and/or distributed examples:

- Take a look at the linked examples on the AsciiDoc web site home page http://asciidoc.org/. Press the *Page Source* sidebar menu item to view corresponding AsciiDoc source.
- Read the `*.txt` source files in the distribution `./doc` directory along with the corresponding HTML and DocBook XML files.

There are three types of AsciiDoc documents: article, book and manpage. All document types share the same AsciiDoc format with some minor variations. If you are familiar with DocBook you will have noticed that AsciiDoc document types correspond to the same-named DocBook document types.

Use the asciidoc(1) `-d` (`--doctype`) option to specify the AsciiDoc document type — the default document type is *article*.

By convention the `.txt` file extension is used for AsciiDoc document source files.

Used for short documents, articles and general documentation. See the AsciiDoc distribution `./doc/article.txt` example.

AsciiDoc defines standard DocBook article frontmatter and backmatter section markup templates (appendix, abstract, bibliography, glossary, index).

Books share the same format as articles, with the following differences:

- The part titles in multi-part books are top level titles (same level as book title).
- Some sections are book specific e.g. preface and colophon.

Book documents will normally be used to produce DocBook output since DocBook processors can automatically generate footnotes, table of contents, list of tables, list of figures, list of examples and indexes.

AsciiDoc defines standard DocBook book frontmatter and backmatter section markup templates (appendix, dedication, preface, bibliography, glossary, index, colophon).

Example book documents

Book

The `./doc/book.txt` file in the AsciiDoc distribution.

Multi-part book

The `./doc/book-multi.txt` file in the AsciiDoc distribution.

Used to generate roff format UNIX manual pages. AsciiDoc manpage documents observe special header title and section naming conventions — see the Manpage Documents section for details.

AsciiDoc defines the *synopsis* section markup template to generate the DocBook `refsynopsisdiv` section.

See also the asciidoc(1) man page source (`./doc/asciidoc.1.txt`) from the AsciiDoc distribution.

The asciidoc(1) command translates an AsciiDoc formatted file to the backend format specified by the `-b` (`--backend`) command-line option. asciidoc(1) itself has little intrinsic knowledge of backend formats, all translation rules are contained in customizable cascading configuration files. Backend specific attributes are listed in the Backend Attributes section.

docbook45

    Outputs DocBook XML 4.5 markup.

html4

    This backend generates plain HTML 4.01 Transitional markup.

xhtml11

    This backend generates XHTML 1.1 markup styled with CSS2. Output files have an `.html` extension.

html5

    This backend generates HTML 5 markup, apart from the inclusion of audio and video block macros it is functionally identical to the *xhtml11* backend.

slidy

    Use this backend to generate self-contained Slidy HTML slideshows for your web browser from AsciiDoc documents. The Slidy backend is documented in the distribution `doc/slidy.txt` file and online.

wordpress

    A minor variant of the *html4* backend to support blogpost.

latex

    Experimental LaTeX backend.

Backend aliases are alternative names for AsciiDoc backends. AsciiDoc comes with two backend aliases: *html* (aliased to *xhtml11*) and *docbook* (aliased to *docbook45*).

You can assign (or reassign) backend aliases by setting an AsciiDoc attribute named like `backend-alias-<alias>` to an AsciiDoc backend name. For example, the following backend alias attribute definitions appear in the `[attributes]` section of the global `asciidoc.conf` configuration file:

```
backend-alias-html=xhtml11
backend-alias-docbook=docbook45
```

The asciidoc(1) `--backend` option is also used to install and manage backend plugins.

- A backend plugin is used just like the built-in backends.
- Backend plugins take precedence over built-in backends with the same name.
- You can use the `{asciidoc-confdir}` intrinsic attribute to refer to the built-in backend configuration file location from backend plugin configuration files.
- You can use the `{backend-confdir}` intrinsic attribute to refer to the backend plugin configuration file location.
- By default backends plugins are installed in `$HOME/.asciidoc/backends/<backend>` where `<backend>` is the backend name.

AsciiDoc generates *article*, *book* and *refentry* DocBook documents (corresponding to the AsciiDoc *article*, *book* and *manpage* document types).

Most Linux distributions come with conversion tools (collectively called a toolchain) for converting DocBook files to presentation formats such as Postscript, HTML, PDF, EPUB, DVI, PostScript, LaTeX, roff (the native man page format), HTMLHelp, JavaHelp and text. There are also programs that allow you to view DocBook files directly, for example Yelp (the GNOME help viewer).

DocBook files are validated, parsed and translated various presentation file formats using a combination of applications collectively called a DocBook *tool chain*. The function of a tool chain is to read the DocBook markup (produced by AsciiDoc) and transform it to a presentation format (for example HTML, PDF, HTML Help, EPUB, DVI, PostScript, LaTeX).

A wide range of user output format requirements coupled with a choice of available tools and stylesheets results in many valid tool chain combinations.

One of the biggest hurdles for new users is installing, configuring and using a DocBook XML toolchain. `a2x(1)` can help — it's a toolchain wrapper command that will generate XHTML (chunked and unchunked), PDF, EPUB, DVI, PS, LaTeX, man page, HTML Help and text file outputs from an AsciiDoc text file. `a2x(1)` does all the grunt work associated with generating and sequencing the toolchain commands and managing intermediate and output files. `a2x(1)` also optionally deploys admonition and navigation icons and a CSS stylesheet. See the `a2x(1)` man page for more details. In addition to `asciidoc(1)` you also need xsltproc(1), DocBook XSL Stylesheets and optionally: dblatex or FOP (to generate PDF); `w3m(1)` or `lynx(1)` (to generate text).

The following examples generate `doc/source-highlight-filter.pdf` from the AsciiDoc `doc/source-highlight-filter.txt` source file. The first example uses `dblatex(1)` (the default PDF generator) the second example forces FOP to be used:

```
$ a2x -f pdf doc/source-highlight-filter.txt
$ a2x -f pdf --fop doc/source-highlight-filter.txt
```

See the `a2x(1)` man page for details.

Tip Use the `--verbose` command-line option to view executed toolchain commands.

AsciiDoc produces nicely styled HTML directly without requiring a DocBook toolchain but there are also advantages in going the DocBook route:

- HTML from DocBook can optionally include automatically generated indexes, tables of contents, footnotes, lists of figures and tables.
- DocBook toolchains can also (optionally) generate separate (chunked) linked HTML pages for each document section.
- Toolchain processing performs link and document validity checks.
- If the DocBook *lang* attribute is set then things like table of contents, figure and table captions and admonition captions will be output in the specified language (setting the AsciiDoc *lang* attribute sets the DocBook *lang* attribute).

On the other hand, HTML output directly from AsciiDoc is much faster, is easily customized and can be used in situations where there is no suitable DocBook toolchain (for example, see the AsciiDoc website).

There are two commonly used tools to generate PDFs from DocBook, dblatex and FOP.

dblatex or FOP?

- *dblatex* is easier to install, there's zero configuration required and no Java VM to install — it just works out of the box.
- *dblatex* source code highlighting and numbering is superb.
- *dblatex* is easier to use as it converts DocBook directly to PDF whereas before using *FOP* you have to convert DocBook to XML-FO using DocBook XSL Stylesheets.
- *FOP* is more feature complete (for example, callouts are processed inside literal layouts) and arguably produces nicer looking output.

1. Convert DocBook XML documents to HTML Help compiler source files using DocBook XSL Stylesheets and xsltproc(1).
2. Convert the HTML Help source (`.hhp` and `.html`) files to HTML Help (`.chm`) files using the Microsoft HTML Help Compiler.

AsciiDoc

    Converts AsciiDoc (`.txt`) files to DocBook XML (`.xml`) files.

DocBook XSL Stylesheets

    These are a set of XSL stylesheets containing rules for converting DocBook XML documents to HTML, XSL-FO, manpage and HTML Help files. The stylesheets are used in conjunction with an XML parser such as xsltproc(1).

xsltproc

    An XML parser for applying XSLT stylesheets (in our case the DocBook XSL Stylesheets) to XML documents.

dblatex

    Generates PDF, DVI, PostScript and LaTeX formats directly from DocBook source via the intermediate LaTeX typesetting language — uses DocBook XSL Stylesheets, xsltproc(1) and `latex(1)`.

FOP

    The Apache Formatting Objects Processor converts XSL-FO (`.fo`) files to PDF files. The XSL-FO files are generated from DocBook source files using DocBook XSL Stylesheets and xsltproc(1).

Microsoft Help Compiler

The Microsoft HTML Help Compiler (`hhc.exe`) is a command-line tool that converts HTML Help source files to a single HTML Help (`.chm`) file. It runs on MS Windows platforms and can be downloaded from http://www.microsoft.com.

The AsciiDoc distribution `./dblatex` directory contains `asciidoc-dblatex.xsl` (customized XSL parameter settings) and `asciidoc-dblatex.sty` (customized LaTeX settings). These are examples of optional dblatex output customization and are used by a2x(1).

You will have noticed that the distributed HTML and HTML Help documentation files (for example `./doc/asciidoc.html`) are not the plain outputs produced using the default *DocBook XSL Stylesheets* configuration. This is because they have been processed using customized DocBook XSL Stylesheets along with (in the case of HTML outputs) the custom `./stylesheets/docbook-xsl.css` CSS stylesheet.

You'll find the customized DocBook XSL drivers along with additional documentation in the distribution `./docbook-xsl` directory. The examples that follow are executed from the distribution documentation (`./doc`) directory. These drivers are also used by a2x(1).

`common.xsl`

Shared driver parameters. This file is not used directly but is included in all the following drivers.

`chunked.xsl`

Generate chunked XHTML (separate HTML pages for each document section) in the `./doc/chunked` directory. For example:

```
$ python ../asciidoc.py -b docbook asciidoc.txt
$ xsltproc --nonet ../docbook-xsl/chunked.xsl asciidoc.xml
```

`epub.xsl`

Used by a2x(1) to generate EPUB formatted documents.

`fo.xsl`

Generate XSL Formatting Object (`.fo`) files for subsequent PDF file generation using FOP. For example:

```
$ python ../asciidoc.py -b docbook article.txt
$ xsltproc --nonet ../docbook-xsl/fo.xsl article.xml > article.fo
$ fop article.fo article.pdf
```

`htmlhelp.xsl`

Generate Microsoft HTML Help source files for the MS HTML Help Compiler in the `./doc/htmlhelp` directory. This example is run on MS Windows from a Cygwin shell prompt:

```
$ python ../asciidoc.py -b docbook asciidoc.txt
$ xsltproc --nonet ../docbook-xsl/htmlhelp.xsl asciidoc.xml
$ c:/Program\ Files/HTML\ Help\ Workshop/hhc.exe htmlhelp.hhp
```

`manpage.xsl`

Generate a `roff(1)` format UNIX man page from a DocBook XML *refentry* document. This example generates an `asciidoc.1` man page file:

```
$ python ../asciidoc.py -d manpage -b docbook asciidoc.1.txt
$ xsltproc --nonet ../docbook-xsl/manpage.xsl asciidoc.1.xml
```

`xhtml.xsl`

Convert a DocBook XML file to a single XHTML file. For example:

```
$ python ../asciidoc.py -b docbook asciidoc.txt
$ xsltproc --nonet ../docbook-xsl/xhtml.xsl asciidoc.xml > asciidoc.html
```

If you want to see how the complete documentation set is processed take a look at the A-A-P script `./doc/main.aap`.

AsciiDoc does not have a text backend (for most purposes AsciiDoc source text is fine), however you can convert AsciiDoc text files to formatted text using the AsciiDoc a2x(1) toolchain wrapper utility.

The *xhtml11* and *html5* backends embed or link CSS and JavaScript files in their outputs, there is also a themes plugin framework.

- If the AsciiDoc *linkcss* attribute is defined then CSS and JavaScript files are linked to the output document, otherwise they are embedded (the default behavior).
- The default locations for CSS and JavaScript files can be changed by setting the AsciiDoc *stylesdir* and *scriptsdir* attributes respectively.
- The default locations for embedded and linked files differ and are calculated at different times — embedded files are loaded when asciidoc(1) generates the output document, linked files are loaded by the browser when the user views the output document.
- Embedded files are automatically inserted in the output files but you need to manually copy linked CSS and Javascript files from AsciiDoc configuration directories to the correct location relative to the output document.

Table 1. Stylesheet file locations

***stylesdir* attribute**	**Linked location (*linkcss* attribute defined)**	**Embedded location (*linkcss* attribute undefined)**
Undefined (default).	Same directory as the output document.	`stylesheets` subdirectory in the AsciiDoc configuration directory (the directory containing the backend conf file).
Absolute or relative directory name.	Absolute or relative to the output document.	Absolute or relative to the AsciiDoc configuration directory (the directory containing the backend conf file).

Table 2. JavaScript file locations

***scriptsdir* attribute**	**Linked location (*linkcss* attribute defined)**	**Embedded location (*linkcss* attribute undefined)**
Undefined (default).	Same directory as the output document.	`javascripts` subdirectory in the AsciiDoc configuration directory (the directory containing the backend conf file).
Absolute or relative directory name.	Absolute or relative to the output document.	Absolute or relative to the AsciiDoc configuration directory (the directory containing the backend conf file).

The AsciiDoc *theme* attribute is used to select an alternative CSS stylesheet and to optionally include additional JavaScript code.

- Theme files reside in an AsciiDoc configuration directory named `themes/<theme>/` (where `<theme>` is the the theme name set by the *theme* attribute). asciidoc(1) sets the *themedir* attribute to the theme directory path name.
- The *theme* attribute can also be set using the asciidoc(1) `--theme` option, the `--theme` option can also be used to manage theme plugins.
- AsciiDoc ships with two themes: *flask* and *volnitsky*.
- The `<theme>.css` file replaces the default `asciidoc.css` CSS file.
- The `<theme>.js` file is included in addition to the default `asciidoc.js` JavaScript file.
- If the data-uri attribute is defined then icons are loaded from the theme `icons` sub-directory if it exists (i.e. the *iconsdir* attribute is set to theme `icons` sub-directory path).
- Embedded theme files are automatically inserted in the output files but you need to manually copy linked CSS and Javascript files to the location of the output documents.
- Linked CSS and JavaScript theme files are linked to the same linked locations as other CSS and JavaScript files.

For example, the command-line option `--theme foo` (or `--attribute theme=foo`) will cause asciidoc(1) to search configuration file locations 1, 2 and 3 for a sub-directory called `themes/foo` containing the stylesheet `foo.css` and optionally a JavaScript file name `foo.js`.

An AsciiDoc document consists of a series of block elements starting with an optional document Header, followed by an optional Preamble, followed by zero or more document Sections.

Almost any combination of zero or more elements constitutes a valid AsciiDoc document: documents can range from a single sentence to a multi-part book.

Block elements consist of one or more lines of text and may contain other block elements.

The AsciiDoc block structure can be informally summarized as follows [1]:

```
Document      ::= (Header?,Preamble?,Section*)
Header        ::= (Title,(AuthorInfo,RevisionInfo?)?)
AuthorInfo    ::= (FirstName,(MiddleName?,LastName)?,EmailAddress?)
RevisionInfo  ::= (RevisionNumber?,RevisionDate,RevisionRemark?)
Preamble      ::= (SectionBody)
Section       ::= (Title,SectionBody?,(Section)*)
SectionBody   ::= ((BlockTitle?,Block)|BlockMacro)+
Block         ::= (Paragraph|DelimitedBlock|List|Table)
List          ::= (BulletedList|NumberedList|LabeledList|CalloutList)
BulletedList  ::= (ListItem)+
NumberedList  ::= (ListItem)+
CalloutList   ::= (ListItem)+
LabeledList   ::= (ListEntry)+
ListEntry      ::= (ListLabel,ListItem)
ListLabel      ::= (ListTerm+)
ListItem       ::= (ItemText,(List|ListParagraph|ListContinuation)*)
```

Where:

- *?* implies zero or one occurrence, + implies one or more occurrences, * implies zero or more occurrences.
- All block elements are separated by line boundaries.
- `BlockId`, `AttributeEntry` and `AttributeList` block elements (not shown) can occur almost anywhere.
- There are a number of document type and backend specific restrictions imposed on the block syntax.
- The following elements cannot contain blank lines: Header, Title, Paragraph, ItemText.
- A ListParagraph is a Paragraph with its *listelement* option set.
- A ListContinuation is a list continuation element.

The Header contains document meta-data, typically title plus optional authorship and revision information:

- The Header is optional, but if it is used it must start with a document title.
- Optional Author and Revision information immediately follows the header title.
- The document header must be separated from the remainder of the document by one or more blank lines and cannot contain blank lines.
- The header can include comments.
- The header can include attribute entries, typically *doctype*, *lang*, *encoding*, *icons*, *data-uri*, *toc*, *numbered*.
- Header attributes are overridden by command-line attributes.
- If the header contains non-UTF-8 characters then the *encoding* must precede the header (either in the document or on the command-line).

Here's an example AsciiDoc document header:

```
Writing Documentation using AsciiDoc
====================================
Joe Bloggs <jbloggs@mymail.com>
v2.0, February 2003:
Rewritten for version 2 release.
```

The author information line contains the author's name optionally followed by the author's email address. The author's name is formatted like:

```
firstname[ [middlename ]lastname][ <email>]]
```

i.e. a first name followed by optional middle and last names followed by an email address in that order. Multi-word first, middle and last names can be entered using the underscore as a word separator. The email address comes last and must be enclosed in angle <> brackets. Here a some examples of author information lines:

```
Joe Bloggs <jbloggs@mymail.com>
Joe Bloggs
Vincent Willem van_Gogh
```

If the author line does not match the above specification then the entire author line is treated as the first name.

The optional revision information line follows the author information line. The revision information can be one of two formats:

1. An optional document revision number followed by an optional revision date followed by an optional revision remark:
   - If the revision number is specified it must be followed by a comma.
   - The revision number must contain at least one numeric character.
   - Any non-numeric characters preceding the first numeric character will be dropped.
   - If a revision remark is specified it must be preceded by a colon. The revision remark extends from the colon up to the next blank line, attribute entry or comment and is subject to normal text substitutions.
   - If a revision number or remark has been set but the revision date has not been set then the revision date is set to the value of the *docdate* attribute.

   Examples:

```
v2.0, February 2003
February 2003
v2.0,
v2.0, February 2003: Rewritten for version 2 release.
February 2003: Rewritten for version 2 release.
v2.0,: Rewritten for version 2 release.
:Rewritten for version 2 release.
```

2. The revision information line can also be an RCS/CVS/SVN Id marker:
   - AsciiDoc extracts the *revnumber*, *revdate*, and *author* attributes from the Id revision marker and displays them in the document header.
   - If an Id revision marker is used the header author line can be omitted.

   Example:

   ```
   $Id: mydoc.txt,v 1.5 2009/05/17 17:58:44 jbloggs Exp $
   ```

You can override or set header parameters by passing *revnumber*, *revremark*, *revdate*, *email*, *author*, *authorinitials*, *firstname* and *lastname* attributes using the asciidoc(1) `-a` (`--attribute`) command-line option. For example:

```
$ asciidoc -a revdate=2004/07/27 article.txt
```

Attribute entries can also be added to the header for substitution in the header template with Attribute Entry elements.

The *title* element in HTML outputs is set to the AsciiDoc document title, you can set it to a different value by including a *title* attribute entry in the document header.

AsciiDoc has two mechanisms for optionally including additional meta-data in the header of the output document:

docinfo configuration file sections

  If a configuration file section named *docinfo* has been loaded then it will be included in the document header. Typically the *docinfo* section name will be prefixed with a + character so that it is appended to (rather than replace) other *docinfo* sections.

docinfo files

  Two docinfo files are recognized: one named `docinfo` and a second named like the AsciiDoc source file with a `-docinfo` suffix. For example, if the source document is called `mydoc.txt` then the document information files would be `docinfo.xml` and `mydoc-docinfo.xml` (for DocBook outputs) and `docinfo.html` and `mydoc-docinfo.html` (for HTML outputs). The docinfo, docinfo1 and docinfo2 attributes control which docinfo files are included in the output files.

The contents docinfo templates and files is dependent on the type of output:

HTML

  Valid *head* child elements. Typically *style* and *script* elements for CSS and JavaScript inclusion.

DocBook

  Valid *articleinfo* or *bookinfo* child elements. DocBook defines numerous elements for document meta-data, for example: copyrights, document history and authorship information. See the DocBook `./doc/article-docinfo.xml` example that comes with the AsciiDoc distribution. The rendering of meta-data elements (or not) is DocBook processor dependent.

The Preamble is an optional untitled section body between the document Header and the first Section title.

In addition to the document title (level 0), AsciiDoc supports four section levels: 1 (top) to 4 (bottom). Section levels are delimited by section titles. Sections are translated using configuration file section markup templates. AsciiDoc generates the following intrinsic attributes specifically for use in section markup templates:

level

  The `level` attribute is the section level number, it is normally just the title level number (1..4). However, if the `leveloffset` attribute is defined it will be added to the `level` attribute. The `leveloffset` attribute is useful for combining documents.

sectnum

  The `-n` (`--section-numbers`) command-line option generates the `sectnum` (section number) attribute. The `sectnum` attribute is used for section numbers in HTML outputs (DocBook section numbering are handled automatically by the DocBook toolchain commands).

Section markup templates specify output markup and are defined in AsciiDoc configuration files. Section markup template names are derived as follows (in order of precedence):

1. From the title’s first positional attribute or *template* attribute. For example, the following three section titles are functionally equivalent:

```
[[terms]]
[glossary]
List of Terms
-------------

["glossary",id="terms"]
List of Terms
-------------

[template="glossary",id="terms"]
List of Terms
-------------
```

2. When the title text matches a configuration file `[specialsections]` entry.

3. If neither of the above the default `sect<level>` template is used (where `<level>` is a number from 1 to 4).

In addition to the normal section template names (*sect1*, *sect2*, *sect3*, *sect4*) AsciiDoc has the following templates for frontmatter, backmatter and other special sections: *abstract*, *preface*, *colophon*, *dedication*, *glossary*, *bibliography*, *synopsis*, *appendix*, *index*. These special section templates generate the corresponding Docbook elements; for HTML outputs they default to the *sect1* section template.

If no explicit section ID is specified an ID will be synthesised from the section title. The primary purpose of this feature is to ensure persistence of table of contents links (permalinks): the missing section IDs are generated dynamically by the JavaScript TOC generator **after** the page is loaded. If you link to a dynamically generated TOC address the page will load but the browser will ignore the (as yet ungenerated) section ID.

The IDs are generated by the following algorithm:

- Replace all non-alphanumeric title characters with underscores.
- Strip leading or trailing underscores.
- Convert to lowercase.
- Prepend the `idprefix` attribute (so there's no possibility of name clashes with existing document IDs). Prepend an underscore if the `idprefix` attribute is not defined.
- A numbered suffix (`_2`, `_3` …) is added if a same named auto-generated section ID exists.
- If the `ascii-ids` attribute is defined then non-ASCII characters are replaced with ASCII equivalents. This attribute may be deprecated in future releases and **should be avoided**, it's sole purpose is to accommodate deficient downstream applications that cannot process non-ASCII ID attributes.

Example: the title *Jim's House* would generate the ID `_jim_s_house`.

Section ID synthesis can be disabled by undefining the `sectids` attribute.

AsciiDoc has a mechanism for mapping predefined section titles auto-magically to specific markup templates. For example a title *Appendix A: Code Reference* will automatically use the *appendix* section markup template. The mappings from title to template name are specified in `[specialsections]` sections in the Asciidoc language configuration files (`lang-*.conf`). Section entries are formatted like:

```
<title>=<template>
```

`<title>` is a Python regular expression and `<template>` is the name of a configuration file markup template section. If the `<title>` matches an AsciiDoc document section title then the backend output is marked up using the `<template>` markup template (instead of the default `sect<level>` section template). The `{title}` attribute value is set to the value of the matched regular expression group named *title*, if there is no *title* group `{title}` defaults to the whole of the AsciiDoc section title. If `<template>` is blank then any existing entry with the same `<title>` will be deleted.

Special section titles vs. explicit template names

AsciiDoc has two mechanisms for specifying non-default section markup templates: you can specify the template name explicitly (using the *template* attribute) or indirectly (using *special section titles*). Specifying a section template attribute explicitly is preferred. Auto-magical *special section titles* have the following drawbacks:

- They are non-obvious, you have to know the exact matching title for each special section on a language by language basis.
- Section titles are predefined and can only be customised with a configuration change.
- The implementation is complicated by multiple languages: every special section title has to be defined for each language (in each of the `lang-*.conf` files).

Specifying special section template names explicitly does add more noise to the source document (the *template* attribute declaration), but the intention is obvious and the syntax is consistent with other AsciiDoc elements c.f. bibliographic, Q&A and glossary lists.

Special section titles have been deprecated but are retained for backward compatibility.

Inline document elements are used to format text and to perform various types of text substitution. Inline elements and inline element syntax is defined in the asciidoc(1) configuration files.

Here is a list of AsciiDoc inline elements in the (default) order in which they are processed:

Special characters

: These character sequences escape special characters used by the backend markup (typically `<`, `>`, and `&` characters). See `[specialcharacters]` configuration file sections.

Quotes

: Elements that markup words and phrases; usually for character formatting. See `[quotes]` configuration file sections.

Special Words

: Word or word phrase patterns singled out for markup without the need for further annotation. See `[specialwords]` configuration file sections.

Replacements

: Each replacement defines a word or word phrase pattern to search for along with corresponding replacement text. See `[replacements]` configuration file sections.

Attribute references

: Document attribute names enclosed in braces are replaced by the corresponding attribute value.

Inline Macros

: Inline macros are replaced by the contents of parametrized configuration file sections.

The AsciiDoc source document is read and processed as follows:

1. The document *Header* is parsed, header parameter values are substituted into the configuration file `[header]` template section which is then written to the output file.
2. Each document *Section* is processed and its constituent elements translated to the output file.
3. The configuration file `[footer]` template section is substituted and written to the output file.

When a block element is encountered asciidoc(1) determines the type of block by checking in the following order (first to last): (section) Titles, BlockMacros, Lists, DelimitedBlocks, Tables, AttributeEntrys, AttributeLists, BlockTitles, Paragraphs.

The default paragraph definition `[paradef-default]` is last element to be checked.

Knowing the parsing order will help you devise unambiguous macro, list and block syntax rules.

Inline substitutions within block elements are performed in the following default order:

1. Special characters
2. Quotes
3. Special words
4. Replacements
5. Attributes
6. Inline Macros
7. Replacements2

The substitutions and substitution order performed on Title, Paragraph and DelimitedBlock elements is determined by configuration file parameters.

Words and phrases can be formatted by enclosing inline text with quote characters:

Emphasized text

: Word phrases 'enclosed in single quote characters' (acute accents) or _underline characters_ are emphasized.

Strong text

: Word phrases *enclosed in asterisk characters* are rendered in a strong font (usually bold).

`Monospaced text`

Word phrases +enclosed in plus characters+ are rendered in a monospaced font. Word phrases `enclosed in backtick characters` (grave accents) are also rendered in a monospaced font but in this case the enclosed text is rendered literally and is not subject to further expansion (see inline literal passthrough).

‘Single quoted text’

Phrases enclosed with a `single grave accent to the left and a single acute accent to the right' are rendered in single quotation marks.

“Double quoted text”

Phrases enclosed with ``two grave accents to the left and two acute accents to the right'' are rendered in quotation marks.

Unquoted text

Placing #hashes around text# does nothing, it is a mechanism to allow inline attributes to be applied to otherwise unformatted text.

New quote types can be defined by editing asciidoc(1) configuration files. See the Configuration Files section for details.

Quoted text behavior

- Quoting cannot be overlapped.
- Different quoting types can be nested.
- To suppress quoted text formatting place a backslash character immediately in front of the leading quote character(s). In the case of ambiguity between escaped and non-escaped text you will need to escape both leading and trailing quotes, in the case of multi-character quotes you may even need to escape individual characters.

Quoted text can be prefixed with an attribute list. The first positional attribute (*role* attribute) is translated by AsciiDoc to an HTML *span* element *class* attribute or a DocBook *phrase* element *role* attribute.

DocBook XSL Stylesheets translate DocBook *phrase* elements with *role* attributes to corresponding HTML *span* elements with the same *class* attributes; CSS can then be used to style the generated HTML. Thus CSS styling can be applied to both DocBook and AsciiDoc generated HTML outputs. You can also specify multiple class names separated by spaces.

CSS rules for text color, text background color, text size and text decorators are included in the distributed AsciiDoc CSS files and are used in conjunction with AsciiDoc *xhtml11*, *html5* and *docbook* outputs. The CSS class names are:

- *<color>* (text foreground color).
- *<color>-background* (text background color).
- *big* and *small* (text size).
- *underline*, *overline* and *line-through* (strike through) text decorators.

Where *<color>* can be any of the sixteen HTML color names. Examples:

```
[red]#Obvious# and [big red yellow-background]*very obvious*.
```

```
[underline]#Underline text#, [overline]#overline text# and
[blue line-through]*bold blue and line-through*.
```

is rendered as:

Obvious and **very obvious**.

Underline text, overline text and **bold blue and line-through**.

Note Color and text decorator attributes are rendered for XHTML and HTML 5 outputs using CSS stylesheets. The mechanism to implement color and text decorator attributes is provided for DocBook toolchains via the DocBook *phrase* element *role* attribute, but the actual rendering is toolchain specific and is not part of the AsciiDoc distribution.

There are actually two types of quotes:

Quoted must be bounded by white space or commonly adjoining punctuation characters. These are the most commonly used type of quote.

Unconstrained quotes have no boundary constraints and can be placed anywhere within inline text. For consistency and to make them easier to remember unconstrained quotes are double-ups of the _, `*`, `+` and `#` constrained quotes:

```
__unconstrained emphasized text__
**unconstrained strong text**
```

```
++unconstrained monospaced text++
##unconstrained unquoted text##
```

The following example emboldens the letter F:

```
**F**ile Open...
```

Put \^carets on either^ side of the text to be superscripted, put \~tildes on either side~ of text to be subscripted. For example, the following line:

```
e^&#960;i^+1 = 0. H~2~O and x^10^. Some ^super text^
and ~some sub text~
```

Is rendered like:

$e^{\pi i}$+1 = 0. H_2O and x^{10}. Some ^super text^ and ~some sub text~

Superscripts and subscripts are implemented as unconstrained quotes and they can be escaped with a leading backslash and prefixed with with an attribute list.

A plus character preceded by at least one space character at the end of a non-blank line forces a line break. It generates a line break (`br`) tag for HTML outputs and a custom XML `asciidoc-br` processing instruction for DocBook outputs. The `asciidoc-br` processing instruction is handled by a2x(1).

A line of three or more less-than (<<<) characters will generate a hard page break in DocBook and printed HTML outputs. It uses the CSS `page-break-after` property for HTML outputs and a custom XML `asciidoc-pagebreak` processing instruction for DocBook outputs. The `asciidoc-pagebreak` processing instruction is handled by a2x(1). Hard page breaks are sometimes handy but as a general rule you should let your page processor generate page breaks for you.

A line of three or more apostrophe characters will generate a ruler line. It generates a ruler (`hr`) tag for HTML outputs and a custom XML `asciidoc-hr` processing instruction for DocBook outputs. The `asciidoc-hr` processing instruction is handled by a2x(1).

By default tab characters input files will translated to 8 spaces. Tab expansion is set with the *tabsize* entry in the configuration file `[miscellaneous]` section and can be overridden in included files by setting a *tabsize* attribute in the `include` macro's attribute list. For example:

```
include::addendum.txt[tabsize=2]
```

The tab size can also be set using the attribute command-line option, for example `--attribute tabsize=4`

The following replacements are defined in the default AsciiDoc configuration:

```
(C) copyright, (TM) trademark, (R) registered trademark,
-- em dash, ... ellipsis, -> right arrow, <- left arrow, => right
double arrow, <= left double arrow.
```

Which are rendered as:

© copyright, ™ trademark, ® registered trademark, — em dash, … ellipsis, → right arrow, ← left arrow, ⇒ right double arrow, ⇐ left double arrow.

You can also include arbitrary entity references in the AsciiDoc source. Examples:

```
&#x278a; &#182;
```

renders:

➊ ¶

To render a replacement literally escape it with a leading back-slash.

The Configuration Files section explains how to configure your own replacements.

Words defined in `[specialwords]` configuration file sections are automatically marked up without having to be explicitly notated.

The Configuration Files section explains how to add and replace special words.

Document and section titles can be in either of two formats:

A two line title consists of a title line, starting hard against the left margin, and an underline. Section underlines consist a repeated character pairs spanning the width of the preceding title (give or take up to two characters):

The default title underlines for each of the document levels are:

```
Level 0 (top level):     ======================
Level 1:                 ----------------------
Level 2:                 ~~~~~~~~~~~~~~~~~~~~~~
Level 3:                 ^^^^^^^^^^^^^^^^^^^^^^
Level 4 (bottom level):  ++++++++++++++++++++++
```

Examples:

```
Level One Section Title
-----------------------

Level 2 Subsection Title
~~~~~~~~~~~~~~~~~~~~~~~~
```

One line titles consist of a single line delimited on either side by one or more equals characters (the number of equals characters corresponds to the section level minus one). Here are some examples:

```
= Document Title (level 0) =
== Section title (level 1) ==
=== Section title (level 2) ===
==== Section title (level 3) ====
===== Section title (level 4) =====
```

Note

- One or more spaces must fall between the title and the delimiters.
- The trailing title delimiter is optional.
- The one-line title syntax can be changed by editing the configuration file `[titles]` section `sect0`…`sect4` entries.

Setting the title's first positional attribute or *style* attribute to *float* generates a free-floating title. A free-floating title is rendered just like a normal section title but is not formally associated with a text body and is not part of the regular section hierarchy so the normal ordering rules do not apply. Floating titles can also be used in contexts where section titles are illegal: for example sidebar and admonition blocks. Example:

```
[float]
The second day
~~~~~~~~~~~~~~
```

Floating titles do not appear in a document's table of contents.

A *BlockTitle* element is a single line beginning with a period followed by the title text. A BlockTitle is applied to the immediately following Paragraph, DelimitedBlock, List, Table or BlockMacro. For example:

```
.Notes
- Note 1.
- Note 2.
```

is rendered as:

Notes

- Note 1.
- Note 2.

A *BlockId* is a single line block element containing a unique identifier enclosed in double square brackets. It is used to assign an identifier to the ensuing block element. For example:

```
[[chapter-titles]]
Chapter titles can be ...
```

The preceding example identifies the ensuing paragraph so it can be referenced from other locations, for example with `<<chapter-titles,chapter titles>>`.

BlockId elements can be applied to Title, Paragraph, List, DelimitedBlock, Table and BlockMacro elements. The BlockId element sets the `{id}` attribute for substitution in the subsequent block's markup template. If a second positional argument is supplied it sets the `{reftext}` attribute which is used to set the DocBook `xreflabel` attribute.

The *BlockId* element has the same syntax and serves the same function to the anchor inline macro.

An *AttributeList* block element is an attribute list on a line by itself:

- *AttributeList* attributes are only applied to the immediately following block element — the attributes are made available to the block's markup template.
- Multiple contiguous *AttributeList* elements are additively combined in the order they appear.
- The first positional attribute in the list is often used to specify the ensuing element's style.

By default, only substitutions that take place inside attribute list values are attribute references, this is because not all attributes are destined to be marked up and rendered as text (for example the table *cols* attribute). To perform normal inline text substitutions (special characters, quotes, macros, replacements) on an attribute value you need to enclose it in single quotes. In the following quote block the second attribute value in the AttributeList is quoted to ensure the *http* macro is expanded to a hyperlink.

```
[quote,'http://en.wikipedia.org/wiki/Samuel_Johnson[Samuel Johnson]']
_____________________________________________________________________
Sir, a woman's preaching is like a dog's walking on his hind legs. It
is not done well; but you are surprised to find it done at all.
_____________________________________________________________________
```

Most block elements support the following attributes:

Name	Backends	Description
id	html4, html5, xhtml11, docbook	Unique identifier typically serve as link targets. Can also be set by the *BlockId* element.
role	html4, html5, xhtml11, docbook	Role contains a string used to classify or subclassify an element and can be applied to AsciiDoc block elements. The AsciiDoc *role* attribute is translated to the *role* attribute in DocBook outputs and is included in the *class* attribute in HTML outputs, in this respect it behaves like the quoted text role attribute. DocBook XSL Stylesheets translate DocBook *role* attributes to HTML *class* attributes; CSS can then be used to style the generated HTML.
reftext	docbook	*reftext* is used to set the DocBook *xreflabel* attribute. The *reftext* attribute can an also be set by the *BlockId* element.

Paragraphs are blocks of text terminated by a blank line, the end of file, or the start of a delimited block or a list. There are three paragraph syntaxes: normal, indented (literal) and admonition which are rendered, by default, with the corresponding paragraph style.

Each syntax has a default style, but you can explicitly apply any paragraph style to any paragraph syntax. You can also apply delimited block styles to single paragraphs.

The built-in paragraph styles are: *normal*, *literal*, *verse*, *quote*, *listing*, *TIP*, *NOTE*, *IMPORTANT*, *WARNING*, *CAUTION*, *abstract*, *partintro*, *comment*, *example*, *sidebar*, *source*, *music*, *latex*, *graphviz*.

Normal paragraph syntax consists of one or more non-blank lines of text. The first line must start hard against the left margin (no intervening white space). The default processing expectation is that of a normal paragraph of text.

Literal paragraphs are rendered verbatim in a monospaced font without any distinguishing background or border. By default there is no text formatting or substitutions within Literal paragraphs apart from Special Characters and Callouts.

The *literal* style is applied implicitly to indented paragraphs i.e. where the first line of the paragraph is indented by one or more space or tab characters. For example:

```
  Consul *necessitatibus* per id,
  consetetur, eu pro everti postulant
  homero verear ea mea, qui.
```

Renders:

```
Consul *necessitatibus* per id,
consetetur, eu pro everti postulant
homero verear ea mea, qui.
```

Note Because lists can be indented it's possible for your indented paragraph to be misinterpreted as a list — in situations like this apply the *literal* style to a normal paragraph.

Instead of using a paragraph indent you could apply the *literal* style explicitly, for example:

```
[literal]
Consul *necessitatibus* per id,
consetetur, eu pro everti postulant
homero verear ea mea, qui.
```

Renders:

```
Consul *necessitatibus* per id,
consetetur, eu pro everti postulant
homero verear ea mea, qui.
```

The optional *attribution* and *citetitle* attributes (positional attributes 2 and 3) specify the author and source respectively.

The *verse* style retains the line breaks, for example:

```
[verse, William Blake, from Auguries of Innocence]
To see a world in a grain of sand,
And a heaven in a wild flower,
Hold infinity in the palm of your hand,
And eternity in an hour.
```

Which is rendered as:

```
To see a world in a grain of sand,
And a heaven in a wild flower,
Hold infinity in the palm of your hand,
And eternity in an hour.
```

— William Blake
from Auguries of Innocence

The *quote* style flows the text at left and right margins, for example:

```
[quote, Bertrand Russell, The World of Mathematics (1956)]
A good notation has subtlety and suggestiveness which at times makes
it almost seem like a live teacher.
```

Which is rendered as:

> A good notation has subtlety and suggestiveness which at times makes it almost seem like a live teacher.

— Bertrand Russell
The World of Mathematics (1956)

TIP, *NOTE*, *IMPORTANT*, *WARNING* and *CAUTION* admonishment paragraph styles are generated by placing `NOTE:`, `TIP:`, `IMPORTANT:`, `WARNING:` or `CAUTION:` as the first word of the paragraph. For example:

```
NOTE: This is an example note.
```

Alternatively, you can specify the paragraph admonition style explicitly using an AttributeList element. For example:

```
[NOTE]
This is an example note.
```

Renders:

Note This is an example note.

Tip If your admonition requires more than a single paragraph use an admonition block instead.

Note Admonition customization with `icons`, `iconsdir`, `icon` and `caption` attributes does not apply when generating DocBook output. If you are going the DocBook route then the a2x(1) `--no-icons` and `--icons-dir` options can be used to set the appropriate XSL Stylesheets parameters.

By default the asciidoc(1) HTML backends generate text captions instead of admonition icon image links. To generate links to icon images define the `icons` attribute, for example using the `-a icons` command-line option.

The `iconsdir` attribute sets the location of linked icon images.

You can override the default icon image using the `icon` attribute to specify the path of the linked image. For example:

```
[icon="./images/icons/wink.png"]
NOTE: What lovely war.
```

Use the `caption` attribute to customize the admonition captions (not applicable to `docbook` backend). The following example suppresses the icon image and customizes the caption of a *NOTE* admonition (undefining the `icons` attribute with `icons=None` is only necessary if admonition icons have been enabled):

```
[icons=None, caption="My Special Note"]
NOTE: This is my special note.
```

This subsection also applies to Admonition Blocks.

Delimited blocks are blocks of text enveloped by leading and trailing delimiter lines (normally a series of four or more repeated characters). The behavior of Delimited Blocks is specified by entries in configuration file `[blockdef-*]` sections.

AsciiDoc ships with a number of predefined DelimitedBlocks (see the `asciidoc.conf` configuration file in the asciidoc(1) program directory):

Predefined delimited block underlines:

```
CommentBlock:     //////////////////////////
PassthroughBlock: ++++++++++++++++++++++++++
ListingBlock:      --------------------------
LiteralBlock:      ..........................
SidebarBlock:      **************************
QuoteBlock:        __________________________
ExampleBlock:      ==========================
OpenBlock:         --
```

Table 3. Default DelimitedBlock substitutions

	Attributes	Callouts	Macros	Quotes	Replacements	Special chars	Special words
PassthroughBlock	Yes	No	Yes	No	No	No	No
ListingBlock	No	Yes	No	No	No	Yes	No
LiteralBlock	No	Yes	No	No	No	Yes	No
SidebarBlock	Yes	No	Yes	Yes	Yes	Yes	Yes
QuoteBlock	Yes	No	Yes	Yes	Yes	Yes	Yes
ExampleBlock	Yes	No	Yes	Yes	Yes	Yes	Yes
OpenBlock	Yes	No	Yes	Yes	Yes	Yes	Yes

ListingBlocks are rendered verbatim in a monospaced font, they retain line and whitespace formatting and are often distinguished by a background or border. There is no text formatting or substitutions within Listing blocks apart from Special Characters and Callouts. Listing blocks are often used for computer output and file listings.

Here's an example:

```
--------------------------------------
#include <stdio.h>

int main() {
   printf("Hello World!\n");
   exit(0);
}
--------------------------------------
```

Which will be rendered like:

```
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    exit(0);
}
```

By convention filter blocks use the listing block syntax and are implemented as distinct listing block styles.

LiteralBlocks are rendered just like literal paragraphs. Example:

```
...................................
Consul *necessitatibus* per id,
consetetur, eu pro everti postulant
homero verear ea mea, qui.
...................................
```

Renders:

```
Consul *necessitatibus* per id,
consetetur, eu pro everti postulant
homero verear ea mea, qui.
```

If the *listing* style is applied to a LiteralBlock it will be rendered as a ListingBlock (this is handy if you have a listing containing a ListingBlock).

A sidebar is a short piece of text presented outside the narrative flow of the main text. The sidebar is normally presented inside a bordered box to set it apart from the main text.

The sidebar body is treated like a normal section body.

Here’s an example:

```
.An Example Sidebar
************************************************
Any AsciiDoc SectionBody element (apart from
SidebarBlocks) can be placed inside a sidebar.
************************************************
```

Which will be rendered like:

An Example Sidebar

Any AsciiDoc SectionBody element (apart from SidebarBlocks) can be placed inside a sidebar.

The contents of *CommentBlocks* are not processed; they are useful for annotations and for excluding new or outdated content that you don’t want displayed. CommentBlocks are never written to output files. Example:

```
//////////////////////////////////////////
CommentBlock contents are not processed by
asciidoc(1).
//////////////////////////////////////////
```

See also Comment Lines.

Note System macros are executed inside comment blocks.

By default the block contents is subject only to *attributes* and *macros* substitutions (use an explicit *subs* attribute to apply different substitutions). PassthroughBlock content will often be backend specific. Here’s an example:

```
[subs="quotes"]
++++++++++++++++++++++++++++++++++++++
<table border="1"><tr>
  <td>*Cell 1*</td>
  <td>*Cell 2*</td>
</tr></table>
++++++++++++++++++++++++++++++++++++++
```

The following styles can be applied to passthrough blocks:

pass

  No substitutions are performed. This is equivalent to `subs="none"`.

asciimath, latexmath

  By default no substitutions are performed, the contents are rendered as mathematical formulas.

QuoteBlocks are used for quoted passages of text. There are two styles: *quote* and *verse*. The style behavior is identical to quote and verse paragraphs except that blocks can contain multiple paragraphs and, in the case of the *quote* style, other section elements. The first positional attribute sets the style, if no attributes are specified the *quote* style is used. The optional *attribution* and *citetitle* attributes (positional attributes 2 and 3) specify the quote’s author and source. For example:

```
[quote, Sir Arthur Conan Doyle, The Adventures of Sherlock Holmes]
____________________________________________________________________
As he spoke there was the sharp sound of horses' hoofs and
grating wheels against the curb, followed by a sharp pull at the
bell. Holmes whistled.

"A pair, by the sound," said he. "Yes," he continued, glancing
out of the window. "A nice little brougham and a pair of
beauties. A hundred and fifty guineas apiece. There's money in
this case, Watson, if there is nothing else."
____________________________________________________________________
```

Which is rendered as:

> As he spoke there was the sharp sound of horses' hoofs and grating wheels against the curb, followed by a sharp pull at the bell. Holmes whistled.
>
> "A pair, by the sound," said he. "Yes," he continued, glancing out of the window. "A nice little brougham and a pair of beauties. A hundred and fifty guineas apiece. There’s money in this case, Watson, if there is nothing else."

— Sir Arthur Conan Doyle
The Adventures of Sherlock Holmes

ExampleBlocks encapsulate the DocBook Example element and are used for, well, examples. Example blocks can be titled by preceding them with a *BlockTitle*. DocBook toolchains will normally automatically number examples and generate a *List of Examples* backmatter section.

Example blocks are delimited by lines of equals characters and can contain any block elements apart from Titles, BlockTitles and Sidebars) inside an example block. For example:

```
.An example
=====================================================================
Qui in magna commodo, est labitur dolorum an. Est ne magna primis
adolescens.
=====================================================================
```

Renders:

Example 1. An example

Qui in magna commodo, est labitur dolorum an. Est ne magna primis adolescens.

A title prefix that can be inserted with the `caption` attribute (HTML backends). For example:

```
[caption="Example 1: "]
.An example with a custom caption
=====================================================================
Qui in magna commodo, est labitur dolorum an. Est ne magna primis
adolescens.
=====================================================================
```

The *ExampleBlock* definition includes a set of admonition styles (*NOTE*, *TIP*, *IMPORTANT*, *WARNING*, *CAUTION*) for generating admonition blocks (admonitions containing more than a single paragraph). Just precede the *ExampleBlock* with an attribute list specifying the admonition style name. For example:

```
[NOTE]
.A NOTE admonition block
=====================================================================
Qui in magna commodo, est labitur dolorum an. Est ne magna primis
adolescens.

. Fusce euismod commodo velit.
. Vivamus fringilla mi eu lacus.
  .. Fusce euismod commodo velit.
  .. Vivamus fringilla mi eu lacus.
. Donec eget arcu bibendum
  nunc consequat lobortis.
=====================================================================
```

Renders:

Note

A NOTE admonition block

Qui in magna commodo, est labitur dolorum an. Est ne magna primis adolescens.

1. Fusce euismod commodo velit.
2. Vivamus fringilla mi eu lacus.
    a. Fusce euismod commodo velit.
    b. Vivamus fringilla mi eu lacus.
3. Donec eget arcu bibendum nunc consequat lobortis.

See also Admonition Icons and Captions.

Open blocks are special:

- The open block delimiter is line containing two hyphen characters (instead of four or more repeated characters).
- They can be used to group block elements for List item continuation.
- Open blocks can be styled to behave like any other type of delimited block. The following built-in styles can be applied to open blocks: *literal*, *verse*, *quote*, *listing*, *TIP*, *NOTE*, *IMPORTANT*, *WARNING*, *CAUTION*, *abstract*, *partintro*, *comment*, *example*, *sidebar*, *source*, *music*, *latex*, *graphviz*. For example, the following open block and listing block are functionally identical:

  ```
  [listing]
  --
  Lorum ipsum ...
  --

  ---------------
  Lorum ipsum ...
  ```

```
---------------
```

- An unstyled open block groups section elements but otherwise does nothing.

Open blocks are used to generate document abstracts and book part introductions:

- Apply the *abstract* style to generate an abstract, for example:

  ```
  [abstract]
  --
  In this paper we will ...
  --
  ```

  1. Apply the *partintro* style to generate a book part introduction for a multi-part book, for example:

     ```
     [partintro]
     .Optional part introduction title
     --
     Optional part introduction goes here.
     --
     ```

List types

- Bulleted lists. Also known as itemized or unordered lists.
- Numbered lists. Also called ordered lists.
- Labeled lists. Sometimes called variable or definition lists.
- Callout lists (a list of callout annotations).

List behavior

- List item indentation is optional and does not determine nesting, indentation does however make the source more readable.
- Another list or a literal paragraph immediately following a list item will be implicitly included in the list item; use list item continuation to explicitly append other block elements to a list item.
- A comment block or a comment line block macro element will terminate a list — use inline comment lines to put comments inside lists.
- The `listindex` intrinsic attribute is the current list item index (1..). If this attribute is used outside a list then it's value is the number of items in the most recently closed list. Useful for displaying the number of items in a list.

Bulleted list items start with a single dash or one to five asterisks followed by some white space then some text. Bulleted list syntaxes are:

```
- List item.
* List item.
** List item.
*** List item.
**** List item.
***** List item.
```

List item numbers are explicit or implicit.

List items begin with a number followed by some white space then the item text. The numbers can be decimal (arabic), roman (upper or lower case) or alpha (upper or lower case). Decimal and alpha numbers are terminated with a period, roman numbers are terminated with a closing parenthesis. The different terminators are necessary to ensure *i*, *v* and *x* roman numbers are are distinguishable from *x*, *v* and *x* alpha numbers. Examples:

```
1.   Arabic (decimal) numbered list item.
a.   Lower case alpha (letter) numbered list item.
F.   Upper case alpha (letter) numbered list item.
iii) Lower case roman numbered list item.
IX)  Upper case roman numbered list item.
```

List items begin one to five period characters, followed by some white space then the item text. Examples:

```
. Arabic (decimal) numbered list item.
.. Lower case alpha (letter) numbered list item.
... Lower case roman numbered list item.
.... Upper case alpha (letter) numbered list item.
..... Upper case roman numbered list item.
```

You can use the *style* attribute (also the first positional attribute) to specify an alternative numbering style. The numbered list style can be one of the following values: *arabic*, *loweralpha*, *upperalpha*, *lowerroman*, *upperroman*.

Here are some examples of bulleted and numbered lists:

```
- Praesent eget purus quis magna eleifend eleifend.
  1. Fusce euismod commodo velit.
    a. Fusce euismod commodo velit.
    b. Vivamus fringilla mi eu lacus.
    c. Donec eget arcu bibendum nunc consequat lobortis.
  2. Vivamus fringilla mi eu lacus.
    i)  Fusce euismod commodo velit.
    ii) Vivamus fringilla mi eu lacus.
  3. Donec eget arcu bibendum nunc consequat lobortis.
  4. Nam fermentum mattis ante.
- Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
  * Fusce euismod commodo velit.
  ** Qui in magna commodo, est labitur dolorum an. Est ne magna primis
     adolescens. Sit munere ponderum dignissim et. Minim luptatum et
     vel.
  ** Vivamus fringilla mi eu lacus.
  * Donec eget arcu bibendum nunc consequat lobortis.
- Nulla porttitor vulputate libero.
  . Fusce euismod commodo velit.
  . Vivamus fringilla mi eu lacus.
[upperroman]
    .. Fusce euismod commodo velit.
    .. Vivamus fringilla mi eu lacus.
  . Donec eget arcu bibendum nunc consequat lobortis.
```

Which render as:

- Praesent eget purus quis magna eleifend eleifend.
    1. Fusce euismod commodo velit.
        a. Fusce euismod commodo velit.
        b. Vivamus fringilla mi eu lacus.
        c. Donec eget arcu bibendum nunc consequat lobortis.
    2. Vivamus fringilla mi eu lacus.
        i. Fusce euismod commodo velit.
        ii. Vivamus fringilla mi eu lacus.
    3. Donec eget arcu bibendum nunc consequat lobortis.
    4. Nam fermentum mattis ante.
- Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
    - Fusce euismod commodo velit.
        - Qui in magna commodo, est labitur dolorum an. Est ne magna primis adolescens. Sit munere ponderum dignissim et. Minim luptatum et vel.
        - Vivamus fringilla mi eu lacus.
    - Donec eget arcu bibendum nunc consequat lobortis.
- Nulla porttitor vulputate libero.
    1. Fusce euismod commodo velit.
    2. Vivamus fringilla mi eu lacus.
        I. Fusce euismod commodo velit.
        II. Vivamus fringilla mi eu lacus.
    3. Donec eget arcu bibendum nunc consequat lobortis.

A predefined *compact* option is available to bulleted and numbered lists — this translates to the DocBook *spacing="compact"* lists attribute which may or may not be processed by the DocBook toolchain. Example:

```
[options="compact"]
- Compact list item.
- Another compact list item.
```

To apply the *compact* option globally define a document-wide *compact-option* attribute, e.g. using the `-a compact-option` command-line

Tip option.

You can set the list start number using the *start* attribute (works for HTML outputs and DocBook outputs processed by DocBook XSL Stylesheets). Example:

```
[start=7]
. List item 7.
. List item 8.
```

Labeled list items consist of one or more text labels followed by the text of the list item.

An item label begins a line with an alphanumeric character hard against the left margin and ends with two, three or four colons or two semi-colons. A list item can have multiple labels, one per line.

The list item text consists of one or more lines of text starting after the last label (either on the same line or a new line) and can be followed by nested List or ListParagraph elements. Item text can be optionally indented.

Here are some examples:

```
In::
Lorem::
  Fusce euismod commodo velit.

  Fusce euismod commodo velit.

Ipsum:: Vivamus fringilla mi eu lacus.
  * Vivamus fringilla mi eu lacus.
  * Donec eget arcu bibendum nunc consequat lobortis.
Dolor::
  Donec eget arcu bibendum nunc consequat lobortis.
  Suspendisse;;
    A massa id sem aliquam auctor.
  Morbi;;
    Pretium nulla vel lorem.
  In;;
    Dictum mauris in urna.
    Vivamus::: Fringilla mi eu lacus.
    Donec:::   Eget arcu bibendum nunc consequat lobortis.
```

Which render as:

In
Lorem

Fusce euismod commodo velit.

```
Fusce euismod commodo velit.
```

Ipsum

Vivamus fringilla mi eu lacus.

- Vivamus fringilla mi eu lacus.
- Donec eget arcu bibendum nunc consequat lobortis.

Dolor

Donec eget arcu bibendum nunc consequat lobortis.

Suspendisse

A massa id sem aliquam auctor.

Morbi

Pretium nulla vel lorem.

In

Dictum mauris in urna.

Vivamus

Fringilla mi eu lacus.

Donec

Eget arcu bibendum nunc consequat lobortis.

The *horizontal* labeled list style (also the first positional attribute) places the list text side-by-side with the label instead of under the label. Here is an example:

```
[horizontal]
*Lorem*:: Fusce euismod commodo velit.  Qui in magna commodo, est
labitur dolorum an. Est ne magna primis adolescens.

  Fusce euismod commodo velit.

*Ipsum*:: Vivamus fringilla mi eu lacus.
- Vivamus fringilla mi eu lacus.
- Donec eget arcu bibendum nunc consequat lobortis.

*Dolor*::
  - Vivamus fringilla mi eu lacus.
  - Donec eget arcu bibendum nunc consequat lobortis.
```

Which render as:

Lorem	Fusce euismod commodo velit. Qui in magna commodo, est labitur dolorum an. Est ne magna primis adolescens. `Fusce euismod commodo velit.`
Ipsum	Vivamus fringilla mi eu lacus. • Vivamus fringilla mi eu lacus. • Donec eget arcu bibendum nunc consequat lobortis.
Dolor	• Vivamus fringilla mi eu lacus. • Donec eget arcu bibendum nunc consequat lobortis.

Note

- Current PDF toolchains do not make a good job of determining the relative column widths for horizontal labeled lists.
- Nested horizontal labeled lists will generate DocBook validation errors because the *DocBook XML V4.2* DTD does not permit nested informal tables (although DocBook XSL Stylesheets and dblatex process them correctly).
- The label width can be set as a percentage of the total width by setting the *width* attribute e.g. `width="10%"`

AsciiDoc comes pre-configured with a *qanda* style labeled list for generating DocBook question and answer (Q&A) lists. Example:

```
[qanda]
Question one::
        Answer one.
Question two::
        Answer two.
```

Renders:

1. *Question one*

   Answer one.

2. *Question two*

   Answer two.

AsciiDoc comes pre-configured with a *glossary* style labeled list for generating DocBook glossary lists. Example:

```
[glossary]
A glossary term::
    The corresponding definition.
A second glossary term::
    The corresponding definition.
```

For working examples see the `article.txt` and `book.txt` documents in the AsciiDoc `./doc` distribution directory.

Note To generate valid DocBook output glossary lists must be located in a section that uses the *glossary* section markup template.

AsciiDoc comes with a predefined *bibliography* bulleted list style generating DocBook bibliography entries. Example:

```
[bibliography]
.Optional list title
- [[[taoup]]] Eric Steven Raymond. 'The Art of UNIX
  Programming'. Addison-Wesley. ISBN 0-13-142901-9.
- [[[walsh-muellner]]] Norman Walsh & Leonard Muellner.
  'DocBook - The Definitive Guide'. O'Reilly & Associates. 1999.
  ISBN 1-56592-580-7.
```

The `[[[<reference>]]]` syntax is a bibliography entry anchor, it generates an anchor named `<reference>` and additionally displays `[<reference>]` at the anchor position. For example `[[[taoup]]]` generates an anchor named `taoup` that displays `[taoup]` at the anchor position. Cite the reference from elsewhere your document using `<<taoup>>`, this displays a hyperlink (`[taoup]`) to the corresponding bibliography entry anchor.

For working examples see the `article.txt` and `book.txt` documents in the AsciiDoc `./doc` distribution directory.

Note To generate valid DocBook output bibliography lists must be located in a bibliography section.

Another list or a literal paragraph immediately following a list item is implicitly appended to the list item; to append other block elements to a list item you need to explicitly join them to the list item with a *list continuation* (a separator line containing a single plus character). Multiple block elements can be appended to a list item using list continuations (provided they are legal list item children in the backend markup).

Here are some examples of list item continuations: list item one contains multiple continuations; list item two is continued with an OpenBlock containing multiple elements:

```
1. List item one.
+
List item one continued with a second paragraph followed by an
Indented block.
+
.................
$ ls *.sh
$ mv *.sh ~/tmp
.................
+
List item continued with a third paragraph.

2. List item two continued with an open block.
+
--
This paragraph is part of the preceding list item.

a. This list is nested and does not require explicit item continuation.
+
This paragraph is part of the preceding list item.

b. List item b.

This paragraph belongs to item two of the outer list.
--
```

Renders:

1. List item one.

   List item one continued with a second paragraph followed by an Indented block.

   ```
   $ ls *.sh
   $ mv *.sh ~/tmp
   ```

   List item continued with a third paragraph.

2. List item two continued with an open block.

   This paragraph is part of the preceding list item.

   a. This list is nested and does not require explicit item continuation.

      This paragraph is part of the preceding list item.

   b. List item b.

   This paragraph belongs to item two of the outer list.

The shipped AsciiDoc configuration includes three footnote inline macros:

```
footnote:[<text>]
```

Generates a footnote with text `<text>`.

`footnoteref:[<id>,<text>]`

Generates a footnote with a reference ID `<id>` and text `<text>`.

`footnoteref:[<id>]`

Generates a reference to the footnote with ID `<id>`.

The footnote text can span multiple lines.

The *xhtml11* and *html5* backends render footnotes dynamically using JavaScript; *html4* outputs do not use JavaScript and leave the footnotes inline; *docbook* footnotes are processed by the downstream DocBook toolchain.

Example footnotes:

```
A footnote footnote:[An example footnote.];
a second footnote with a reference ID footnoteref:[note2,Second footnote.];
finally a reference to the second footnote footnoteref:[note2].
```

Renders:

A footnote [2]; a second footnote with a reference ID [3]; finally a reference to the second footnote [3].

The shipped AsciiDoc configuration includes the inline macros for generating DocBook index entries.

`indexterm:[<primary>,<secondary>,<tertiary>]`
`(((<primary>,<secondary>,<tertiary>)))`

This inline macro generates an index term (the `<secondary>` and `<tertiary>` positional attributes are optional). Example: `indexterm:[Tigers,Big cats]` (or, using the alternative syntax `(((Tigers,Big cats)))`. Index terms that have secondary and tertiary entries also generate separate index terms for the secondary and tertiary entries. The index terms appear in the index, not the primary text flow.

`indexterm2:[<primary>]`
`((<primary>))`

This inline macro generates an index term that appears in both the index and the primary text flow. The `<primary>` should not be padded to the left or right with white space characters.

For working examples see the `article.txt` and `book.txt` documents in the AsciiDoc `./doc` distribution directory.

Note Index entries only really make sense if you are generating DocBook markup — DocBook conversion programs automatically generate an index at the point an *Index* section appears in source document.

Callouts are a mechanism for annotating verbatim text (for example: source code, computer output and user input). Callout markers are placed inside the annotated text while the actual annotations are presented in a callout list after the annotated text. Here's an example:

```
 .MS-DOS directory listing
 -----------------------------------------------------
 10/17/97   9:04         <DIR>    bin
 10/16/97  14:11         <DIR>    DOS            <1>
 10/16/97  14:40         <DIR>    Program Files
 10/16/97  14:46         <DIR>    TEMP
 10/17/97   9:04         <DIR>    tmp
 10/16/97  14:37         <DIR>    WINNT
 10/16/97  14:25             119  AUTOEXEC.BAT   <2>
  2/13/94   6:21          54,619  COMMAND.COM    <2>
 10/16/97  14:25             115  CONFIG.SYS     <2>
 11/16/97  17:17      61,865,984  pagefile.sys
  2/13/94   6:21           9,349  WINA20.386     <3>
 -----------------------------------------------------

 \<1> This directory holds MS-DOS.
 \<2> System startup code for DOS.
 \<3> Some sort of Windows 3.1 hack.
```

Which renders:

MS-DOS directory listing

```
10/17/97   9:04         <DIR>    bin
10/16/97  14:11         <DIR>    DOS            1
10/16/97  14:40         <DIR>    Program Files
10/16/97  14:46         <DIR>    TEMP
10/17/97   9:04         <DIR>    tmp
10/16/97  14:37         <DIR>    WINNT
```

```
10/16/97  14:25              119  AUTOEXEC.BAT     2
 2/13/94   6:21           54,619  COMMAND.COM      2
10/16/97  14:25              115  CONFIG.SYS       2
11/16/97  17:17       61,865,984  pagefile.sys
 2/13/94   6:21            9,349  WINA20.386       3
```

1 This directory holds MS-DOS.
2 System startup code for DOS.
3 Some sort of Windows 3.1 hack.

Explanation

- The callout marks are whole numbers enclosed in angle brackets — they refer to the correspondingly numbered item in the following callout list.
- By default callout marks are confined to *LiteralParagraphs*, *LiteralBlocks* and *ListingBlocks* (although this is a configuration file option and can be changed).
- Callout list item numbering is fairly relaxed — list items can start with `<n>`, `n>` or `>` where `n` is the optional list item number (in the latter case list items starting with a single `>` character are implicitly numbered starting at one).
- Callout lists should not be nested.
- Callout lists cannot be used within tables.
- Callout lists start list items hard against the left margin.
- If you want to present a number inside angle brackets you'll need to escape it with a backslash to prevent it being interpreted as a callout mark.

Note Define the AsciiDoc *icons* attribute (for example using the `-a icons` command-line option) to display callout icons.

Callout marks are generated by the *callout* inline macro while callout lists are generated using the *callout* list definition. The *callout* macro and *callout* list are special in that they work together. The *callout* inline macro is not enabled by the normal *macros* substitutions option, instead it has its own *callouts* substitution option.

The following attributes are available during inline callout macro substitution:

`{index}`

The callout list item index inside the angle brackets.

`{coid}`

An identifier formatted like `CO<listnumber>-<index>` that uniquely identifies the callout mark. For example `CO2-4` identifies the fourth callout mark in the second set of callout marks.

The `{coids}` attribute can be used during callout list item substitution — it is a space delimited list of callout IDs that refer to the explanatory list item.

You can annotate working code examples with callouts — just remember to put the callouts inside source code comments. This example displays the `test.py` source file (containing a single callout) using the *source* (code highlighter) filter:

AsciiDoc source

```
 [source,python]
 -------------------------------------------
 \include::test.py[]
 -------------------------------------------

 \<1> Print statement.
```

Included `test.py` source

```
print 'Hello World!'   # <1>
```

Macros are a mechanism for substituting parametrized text into output documents.

Macros have a *name*, a single *target* argument and an *attribute list*. The usual syntax is `<name>:<target>[<attrlist>]` (for inline macros) and `<name>::<target>[<attrlist>]` (for block macros). Here are some examples:

```
http://www.docbook.org/[DocBook.org]
include::chapt1.txt[tabsize=2]
mailto:srackham@gmail.com[]
```

Macro behavior

- `<name>` is the macro name. It can only contain letters, digits or dash characters and cannot start with a dash.
- The optional `<target>` cannot contain white space characters.

- `<attrlist>` is a list of attributes enclosed in square brackets.
- `]` characters inside attribute lists must be escaped with a backslash.
- Expansion of macro references can normally be escaped by prefixing a backslash character (see the AsciiDoc *FAQ* for examples of exceptions to this rule).
- Attribute references in block macros are expanded.
- The substitutions performed prior to Inline macro macro expansion are determined by the inline context.
- Macros are processed in the order they appear in the configuration file(s).
- Calls to inline macros can be nested inside different inline macros (an inline macro call cannot contain a nested call to itself).
- In addition to `<name>`, `<target>` and `<attrlist>` the `<passtext>` and `<subslist>` named groups are available to passthrough macros. A macro is a passthrough macro if the definition includes a `<passtext>` named group.

Inline Macros occur in an inline element context. Predefined Inline macros include *URLs*, *image* and *link* macros.

http, *https*, *ftp*, *file*, *mailto* and *callto* URLs are rendered using predefined inline macros.

- If you don't need a custom link caption you can enter the *http*, *https*, *ftp*, *file* URLs and email addresses without any special macro syntax.
- If the `<attrlist>` is empty the URL is displayed.

Here are some examples:

```
http://www.docbook.org/[DocBook.org]
http://www.docbook.org/
mailto:joe.bloggs@foobar.com[email Joe Bloggs]
joe.bloggs@foobar.com
```

Which are rendered:

DocBook.org

http://www.docbook.org/

email Joe Bloggs

joe.bloggs@foobar.com

If the `<target>` necessitates space characters use `%20`, for example `large%20image.png`.

Two AsciiDoc inline macros are provided for creating hypertext links within an AsciiDoc document. You can use either the standard macro syntax or the (preferred) alternative.

Used to specify hypertext link targets:

```
[[<id>,<xreflabel>]]
anchor:<id>[<xreflabel>]
```

The `<id>` is a unique string that conforms to the output markup's anchor syntax. The optional `<xreflabel>` is the text to be displayed by captionless *xref* macros that refer to this anchor. The optional `<xreflabel>` is only really useful when generating DocBook output. Example anchor:

```
[[X1]]
```

You may have noticed that the syntax of this inline element is the same as that of the BlockId block element, this is no coincidence since they are functionally equivalent.

Creates a hypertext link to a document anchor.

```
<<<id>,<caption>>>
xref:<id>[<caption>]
```

The `<id>` refers to an anchor ID. The optional `<caption>` is the link's displayed text. Example:

```
<<X21,attribute lists>>
```

If `<caption>` is not specified then the displayed text is auto-generated:

- The AsciiDoc *xhtml11* and *html5* backends display the `<id>` enclosed in square brackets.
- If DocBook is produced the DocBook toolchain is responsible for the displayed text which will normally be the referenced figure, table or section title number followed by the element's title text.

Here is an example:

```
[[tiger_image]]
.Tyger tyger
image::tiger.png[]

This can be seen in <<tiger_image>>.
```

Hypertext links to files on the local file system are specified using the *link* inline macro.

```
link:<target>[<caption>]
```

The *link* macro generates relative URLs. The link macro `<target>` is the target file name (relative to the file system location of the referring document). The optional `<caption>` is the link's displayed text. If `<caption>` is not specified then `<target>` is displayed. Example:

```
link:downloads/foo.zip[download foo.zip]
```

You can use the `<filename>#<id>` syntax to refer to an anchor within a target document but this usually only makes sense when targeting HTML documents.

Inline images are inserted into the output document using the *image* macro. The inline syntax is:

```
image:<target>[<attributes>]
```

The contents of the image file `<target>` is displayed. To display the image its file format must be supported by the target backend application. HTML and DocBook applications normally support PNG or JPG files.

`<target>` file name paths are relative to the location of the referring document.

Image macro attributes

- The optional *alt* attribute is also the first positional attribute, it specifies alternative text which is displayed if the output application is unable to display the image file (see also Use of ALT texts in IMGs). For example:

  ```
  image:images/logo.png[Company Logo]
  ```

- The optional *title* attribute provides a title for the image. The block image macro renders the title alongside the image. The inline image macro displays the title as a popup "tooltip" in visual browsers (AsciiDoc HTML outputs only).

- The optional `width` and `height` attributes scale the image size and can be used in any combination. The units are pixels. The following example scales the previous example to a height of 32 pixels:

  ```
  image:images/logo.png["Company Logo",height=32]
  ```

- The optional `link` attribute is used to link the image to an external document. The following example links a screenshot thumbnail to a full size version:

  ```
  image:screen-thumbnail.png[height=32,link="screen.png"]
  ```

- The optional `scaledwidth` attribute is only used in DocBook block images (specifically for PDF documents). The following example scales the images to 75% of the available print width:

  ```
  image::images/logo.png[scaledwidth="75%",alt="Company Logo"]
  ```

- The image `scale` attribute sets the DocBook `imagedata` element `scale` attribute.

- The optional `align` attribute aligns block macro images horizontally. Allowed values are `center`, `left` and `right`. For example:

  ```
  image::images/tiger.png["Tiger image",align="left"]
  ```

- The optional `float` attribute floats the image `left` or `right` on the page (works with HTML outputs only, has no effect on DocBook outputs). `float` and `align` attributes are mutually exclusive. Use the `unfloat::[]` block macro to stop floating.

See comment block macro.

A Block macro reference must be contained in a single line separated either side by a blank line or a block delimiter.

Block macros behave just like Inline macros, with the following differences:

- They occur in a block context.

- The default syntax is `<name>::<target>[<attrlist>]` (two colons, not one).

- Markup template section names end in `-blockmacro` instead of `-inlinemacro`.

The Block Identifier macro sets the `id` attribute and has the same syntax as the anchor inline macro since it performs essentially the same function — block templates use the `id` attribute as a block element ID. For example:

```
[[X30]]
```

This is equivalent to the `[id="X30"]` AttributeList element).

The *image* block macro is used to display images in a block context. The syntax is:

```
image::<target>[<attributes>]
```

The block `image` macro has the same macro attributes as it's inline image macro counterpart.

Warning Unlike the inline `image` macro, the entire block `image` macro must be on a single line.

Block images can be titled by preceding the *image* macro with a *BlockTitle*. DocBook toolchains normally number titled block images and optionally list them in an automatically generated *List of Figures* backmatter section.

This example:

```
.Main circuit board
image::images/layout.png[J14P main circuit board]
```

is equivalent to:

```
image::images/layout.png["J14P main circuit board", title="Main circuit board"]
```

A title prefix that can be inserted with the `caption` attribute (HTML backends). For example:

```
.Main circuit board
[caption="Figure 2: "]
image::images/layout.png[J14P main circuit board]
```

Embedding images in XHTML documents

If you define the `data-uri` attribute then images will be embedded in XHTML outputs using the data URI scheme. You can use the *data-uri* attribute with the *xhtml11* and *html5* backends to produce single-file XHTML documents with embedded images and CSS, for example:

```
$ asciidoc -a data-uri mydocument.txt
```

Note

- All current popular browsers support data URIs, although versions of Internet Explorer prior to version 8 do not.
- Some browsers limit the size of data URIs.

Single lines starting with two forward slashes hard up against the left margin are treated as comments. Comment lines do not appear in the output unless the *showcomments* attribute is defined. Comment lines have been implemented as both block and inline macros so a comment line can appear as a stand-alone block or within block elements that support inline macro expansion. Example comment line:

```
// This is a comment.
```

If the *showcomments* attribute is defined comment lines are written to the output:

- In DocBook the comment lines are enclosed by the *remark* element (which may or may not be rendered by your toolchain).
- The *showcomments* attribute does not expose Comment Blocks. Comment Blocks are never passed to the output.

System macros are block macros that perform a predefined task and are hardwired into the asciidoc(1) program.

- You can escape system macros with a leading backslash character (as you can with other macros).
- The syntax and tasks performed by system macros is built into asciidoc(1) so they don't appear in configuration files. You can however customize the syntax by adding entries to a configuration file `[macros]` section.

The `include` and `include1` system macros to include the contents of a named file into the source document.

The `include` macro includes a file as if it were part of the parent document — tabs are expanded and system macros processed. The contents of `include1` files are not subject to tab expansion or system macro processing nor are attribute or lower priority substitutions performed. The `include1` macro's intended use is to include verbatim embedded CSS or scripts into configuration file headers. Example:

```
include::chapter1.txt[tabsize=4]
```

Include macro behavior

- If the included file name is specified with a relative path then the path is relative to the location of the referring document.

- Include macros can appear inside configuration files.
- Files included from within *DelimitedBlocks* are read to completion to avoid false end-of-block underline termination.
- Attribute references are expanded inside the include *target*; if an attribute is undefined then the included file is silently skipped.
- The *tabsize* macro attribute sets the number of space characters to be used for tab expansion in the included file (not applicable to `include1` macro).
- The *depth* macro attribute sets the maximum permitted number of subsequent nested includes (not applicable to `include1` macro which does not process nested includes). Setting *depth* to *1* disables nesting inside the included file. By default, nesting is limited to a depth of ten.
- If the he *warnings* attribute is set to *False* (or any other Python literal that evaluates to boolean false) then no warning message is printed if the included file does not exist. By default *warnings* are enabled.
- Internally the `include1` macro is translated to the `include1` system attribute which means it must be evaluated in a region where attribute substitution is enabled. To inhibit nested substitution in included files it is preferable to use the `include` macro and set the attribute `depth=1`.

Lines of text in the source document can be selectively included or excluded from processing based on the existence (or not) of a document attribute.

Document text between the `ifdef` and `endif` macros is included if a document attribute is defined:

```
ifdef::<attribute>[]
:
endif::<attribute>[]
```

Document text between the `ifndef` and `endif` macros is not included if a document attribute is defined:

```
ifndef::<attribute>[]
:
endif::<attribute>[]
```

`<attribute>` is an attribute name which is optional in the trailing `endif` macro.

If you only want to process a single line of text then the text can be put inside the square brackets and the `endif` macro omitted, for example:

```
ifdef::revnumber[Version number 42]
```

Is equivalent to:

```
ifdef::revnumber[]
Version number 42
endif::revnumber[]
```

ifdef and *ifndef* macros also accept multiple attribute names:

- Multiple *,* separated attribute names evaluate to defined if one or more of the attributes is defined, otherwise it's value is undefined.
- Multiple + separated attribute names evaluate to defined if all of the attributes is defined, otherwise it's value is undefined.

Document text between the `ifeval` and `endif` macros is included if the Python expression inside the square brackets is true. Example:

```
ifeval::[{rs458}==2]
:
endif::[]
```

- Document attribute references are expanded before the expression is evaluated.
- If an attribute reference is undefined then the expression is considered false.

Take a look at the `*.conf` configuration files in the AsciiDoc distribution for examples of conditional inclusion macro usage.

The *eval*, *sys* and *sys2* block macros exhibit the same behavior as their same named system attribute references. The difference is that system macros occur in a block macro context whereas system attributes are confined to inline contexts where attribute substitution is enabled.

The following example displays a long directory listing inside a literal block:

```
------------------
sys::[ls -l *.txt]
------------------
```

Note There are no block macro versions of the *eval3* and *sys3* system attributes.

The `template` block macro allows the inclusion of one configuration file template section within another. The following example includes the `[admonitionblock]` section in the `[admonitionparagraph]` section:

```
[admonitionparagraph]
template::[admonitionblock]
```

Template macro behavior

- The `template::[]` macro is useful for factoring configuration file markup.
- `template::[]` macros cannot be nested.
- `template::[]` macro expansion is applied after all configuration files have been read.

Passthrough macros are analogous to passthrough blocks and are used to pass text directly to the output. The substitution performed on the text is determined by the macro definition but can be overridden by the `<subslist>`. The usual syntax is `<name>:<subslist>[<passtext>]` (for inline macros) and `<name>::<subslist>[<passtext>]` (for block macros). Passthroughs, by definition, take precedence over all other text substitutions.

pass

Inline and block. Passes text unmodified (apart from explicitly specified substitutions). Examples:

```
pass:[<q>To be or not to be</q>]
pass:attributes,quotes[<u>the '{author}'</u>]
```

asciimath, latexmath

Inline and block. Passes text unmodified. Used for mathematical formulas.

+++

Inline and block. The triple-plus passthrough is functionally identical to the *pass* macro but you don't have to escape `]` characters and you can prefix with quoted attributes in the inline version. Example:

```
Red [red]+++`sum_(i=1)\^n i=(n(n+1))/2`$+++ AsciiMathML formula
```

$$

Inline and block. The double-dollar passthrough is functionally identical to the triple-plus passthrough with one exception: special characters are escaped. Example:

```
$$`[[a,b],[c,d]]((n),(k))`$$
```

`

Text quoted with single backtick characters constitutes an *inline literal* passthrough. The enclosed text is rendered in a monospaced font and is only subject to special character substitution. This makes sense since monospace text is usually intended to be rendered literally and often contains characters that would otherwise have to be escaped. If you need monospaced text containing inline substitutions use a plus character instead of a backtick.

Each entry in the configuration `[macros]` section is a macro definition which can take one of the following forms:

`<pattern>=<name>[<subslist]`

Inline macro definition.

`<pattern>=#<name>[<subslist]`

Block macro definition.

`<pattern>=+<name>[<subslist]`

System macro definition.

`<pattern>`

Delete the existing macro with this `<pattern>`.

`<pattern>` is a Python regular expression and `<name>` is the name of a markup template. If `<name>` is omitted then it is the value of the regular expression match group named *name*. The optional `[<subslist]` is a comma-separated list of substitution names enclosed in `[]` brackets, it sets the default substitutions for passthrough text, if omitted then no passthrough substitutions are performed.

The following named groups can be used in macro `<pattern>` regular expressions and are available as markup template attributes:

name

The macro name.

target

The macro target.

attrlist

The macro attribute list.

passtext

Contents of this group are passed unmodified to the output subject only to *subslist* substitutions.

subslist

Processed as a comma-separated list of substitution names for *passtext* substitution, overrides the the macro definition *subslist*.

Here's what happens during macro substitution

- Each contextually relevant macro *pattern* from the `[macros]` section is matched against the input source line.
- If a match is found the text to be substituted is loaded from a configuration markup template section named like `<name>-inlinemacro` or `<name>-blockmacro` (depending on the macro type).
- Global and macro attribute list attributes are substituted in the macro's markup template.
- The substituted template replaces the macro reference in the output document.

The *html5* backend *audio* and *video* block macros generate the HTML 5 *audio* and *video* elements respectively. They follow the usual AsciiDoc block macro syntax `<name>::<target>[<attrlist>]` where:

`<name>`	*audio* or *video*.
`<target>`	The URL or file name of the video or audio file.
`<attrlist>`	A list of named attributes (see below).

Table 4. Audio macro attributes

Name	Value
options	A comma separated list of one or more of the following items: *autoplay*, *loop* which correspond to the same-named HTML 5 *audio* element boolean attributes. By default the player *controls* are enabled, include the *nocontrols* option value to hide them.

Table 5. Video macro attributes

Name	Value
height	The height of the player in pixels.
width	The width of the player in pixels.
poster	The URL or file name of an image representing the video.
options	A comma separated list of one or more of the following items: *autoplay*, *loop* and *nocontrols*. The *autoplay* and *loop* options correspond to the same-named HTML 5 *video* element boolean attributes. By default the player *controls* are enabled, include the *nocontrols* option value to hide them.

Examples:

```
audio::images/example.ogg[]

video::gizmo.ogv[width=200,options="nocontrols,autoplay"]

.Example video
video::gizmo.ogv[]

video::http://www.808.dk/pics/video/gizmo.ogv[]
```

If your needs are more complex put raw HTML 5 in a markup block, for example (from http://www.808.dk/?code-html-5-video):

```
++++
<video poster="pics/video/gizmo.jpg" id="video" style="cursor: pointer;" >
  <source src="pics/video/gizmo.mp4" />
  <source src="pics/video/gizmo.webm" type="video/webm" />
  <source src="pics/video/gizmo.ogv" type="video/ogg" />
  Video not playing? <a href="pics/video/gizmo.mp4">Download file</a> instead.
</video>

<script type="text/javascript">
  var video = document.getElementById('video');
  video.addEventListener('click',function(){
    video.play();
  },false);
</script>
++++
```

The AsciiDoc table syntax looks and behaves like other delimited block types and supports standard block configuration entries. Formatting is easy to read and, just as importantly, easy to enter.

- Cells and columns can be formatted using built-in customizable styles.
- Horizontal and vertical cell alignment can be set on columns and cell.
- Horizontal and vertical cell spanning is supported.

Use tables sparingly

When technical users first start creating documents, tables (complete with column spanning and table nesting) are often considered very important. The reality is that tables are seldom used, even in technical documentation.

Try this exercise: thumb through your library of technical books, you’ll be surprised just how seldom tables are actually used, even less seldom are tables containing block elements (such as paragraphs or lists) or spanned cells. This is no accident, like figures, tables are outside the normal document flow — tables are for consulting not for reading.

Tables are designed for, and should normally only be used for, displaying column oriented tabular data.

Table 6. Simple table

1	2	A
3	4	B
5	6	C

AsciiDoc source

```
[width="15%"]
|=======
|1 |2 |A
|3 |4 |B
|5 |6 |C
|=======
```

Table 7. Columns formatted with strong, monospaced and emphasis styles

	Columns 2 and 3	
1	`Item 1`	*Item 1*
2	`Item 2`	*Item 2*
3	`Item 3`	*Item 3*
4	`Item 4`	*Item 4*
footer 1	`footer 2`	*footer 3*

AsciiDoc source

```
.An example table
[width="50%",cols=">s,^m,e",frame="topbot",options="header,footer"]
```

```
|==========================
|      2+|Columns 2 and 3
|1       |Item 1  |Item 1
|2       |Item 2  |Item 2
|3       |Item 3  |Item 3
|4       |Item 4  |Item 4
|footer 1|footer 2|footer 3
|==========================
```

Table 8. Horizontal and vertical source data

Date	Duration	Avg HR	Notes
22-Aug-08	10:24	157	Worked out MSHR (max sustainable heart rate) by going hard for this interval.
22-Aug-08	23:03	152	Back-to-back with previous interval.
24-Aug-08	40:00	145	Moderately hard interspersed with 3x 3min intervals (2min hard + 1min really hard taking the HR up to 160).

Short cells can be entered horizontally, longer cells vertically. The default behavior is to strip leading and trailing blank lines within a cell. These characteristics aid readability and data entry.

AsciiDoc source

```
.Windtrainer workouts
[width="80%",cols="3,^2,^2,10",options="header"]
|=========================================================
|Date |Duration |Avg HR |Notes

|22-Aug-08 |10:24 | 157 |
Worked out MSHR (max sustainable heart rate) by going hard
for this interval.

|22-Aug-08 |23:03 | 152 |
Back-to-back with previous interval.

|24-Aug-08 |40:00 | 145 |
Moderately hard interspersed with 3x 3min intervals (2min
hard + 1min really hard taking the HR up to 160).

|=========================================================
```

Table 9. A table with externally sourced CSV data

ID	Customer Name	Contact Name	Customer Address	Phone

AsciiDoc source

```
[format="csv",cols="^1,4*2",options="header"]
|===================================================
ID,Customer Name,Contact Name,Customer Address,Phone
include::customers.csv[]
|===================================================
```

Table 10. Cell spans, alignments and styles

1	**2**	3	**4**
5	`6`		
8			`7`
9	`10`		

AsciiDoc source

```
[cols="e,m,^,>s",width="25%"]
|============================
|1 >s|2 |3 |4
^|5 2.2+^.^|6 .3+<.>m|7
^|8
```

```
|9 2+>|10
|=============================
```

AsciiDoc table data can be *psv*, *dsv* or *csv* formatted. The default table format is *psv*.

AsciiDoc *psv* (*Prefix Separated Values*) and *dsv* (*Delimiter Separated Values*) formats are cell oriented — the table is treated as a sequence of cells — there are no explicit row separators.

- *psv* prefixes each cell with a separator whereas *dsv* delimits cells with a separator.
- *psv* and *dsv* separators are Python regular expressions.
- The default *psv* separator contains cell specifier related named regular expression groups.
- The default *dsv* separator is `:|\n` (a colon or a new line character).
- *psv* and *dsv* cell separators can be escaped by preceding them with a backslash character.

Here are four *psv* cells (the second item spans two columns; the last contains an escaped separator):

```
|One 2+|Two and three |A \| separator character
```

csv is the quasi-standard row oriented *Comma Separated Values (CSV)* format commonly used to import and export spreadsheet and database data.

Tables can be customized by the following attributes:

format

: *psv* (default), *dsv* or *csv* (See Table Data Formats).

separator

: The cell separator. A Python regular expression (*psv* and *dsv* formats) or a single character (*csv* format).

frame

: Defines the table border and can take the following values: *topbot* (top and bottom), *all* (all sides), *none* and *sides* (left and right sides). The default value is *all*.

grid

: Defines which ruler lines are drawn between table rows and columns. The *grid* attribute value can be any of the following values: *none*, *cols*, *rows* and *all*. The default value is *all*.

align

: Use the *align* attribute to horizontally align the table on the page (works with HTML outputs only, has no effect on DocBook outputs). The following values are valid: *left*, *right*, and *center*.

float

: Use the *float* attribute to float the table *left* or *right* on the page (works with HTML outputs only, has no effect on DocBook outputs). Floating only makes sense in conjunction with a table *width* attribute value of less than 100% (otherwise the table will take up all the available space). *float* and *align* attributes are mutually exclusive. Use the `unfloat::[]` block macro to stop floating.

halign

: Use the *halign* attribute to horizontally align all cells in a table. The following values are valid: *left*, *right*, and *center* (defaults to *left*). Overridden by Column specifiers and Cell specifiers.

valign

: Use the *valign* attribute to vertically align all cells in a table. The following values are valid: *top*, *bottom*, and *middle* (defaults to *top*). Overridden by Column specifiers and Cell specifiers.

options

: The *options* attribute can contain comma separated values, for example: *header*, *footer*. By default header and footer rows are omitted. See attribute options for a complete list of available table options.

cols

: The *cols* attribute is a comma separated list of column specifiers. For example `cols="2<p,2*,4p,>"`.

  - If *cols* is present it must specify all columns.

- If the *cols* attribute is not specified the number of columns is calculated as the number of data items in the **first line** of the table.
- The degenerate form for the *cols* attribute is an integer specifying the number of columns e.g. `cols=4`.

width

The *width* attribute is expressed as a percentage value (*"1%"…"99%"*). The width specifies the table width relative to the available width. HTML backends use this value to set the table width attribute. It's a bit more complicated with DocBook, see the DocBook table widths sidebar.

filter

The *filter* attribute defines an external shell command that is invoked for each cell. The built-in *asciidoc* table style is implemented using a filter.

DocBook table widths

The AsciiDoc docbook backend generates CALS tables. CALS tables do not support a table width attribute — table width can only be controlled by specifying absolute column widths.

Specifying absolute column widths is not media independent because different presentation media have different physical dimensions. To get round this limitation both DocBook XSL Stylesheets and dblatex have implemented table width processing instructions for setting the table width as a percentage of the available width. AsciiDoc emits these processing instructions if the *width* attribute is set along with proportional column widths (the AsciiDoc docbook backend *pageunits* attribute defaults to *).

To generate DocBook tables with absolute column widths set the *pageunits* attribute to a CALS absolute unit such as *pt* and set the *pagewidth* attribute to match the width of the presentation media.

Column specifiers define how columns are rendered and appear in the table cols attribute. A column specifier consists of an optional column multiplier followed by optional alignment, width and style values and is formatted like:

```
[<multiplier>*][<align>][<width>][<style>]
```

- All components are optional. The multiplier must be first and the style last. The order of `<align>` or `<width>` is not important.
- Column `<width>` can be either an integer proportional value (1…) or a percentage (1%…100%). The default value is 1. To ensure portability across different backends, there is no provision for absolute column widths (not to be confused with output column width markup attributes which are available in both percentage and absolute units).
- The *<align>* column alignment specifier is formatted like:

  ```
  [<horizontal>][.<vertical>]
  ```

  Where `<horizontal>` and `<vertical>` are one of the following characters: `<`, `^` or `>` which represent *left*, *center* and *right* horizontal alignment or *top*, *middle* and *bottom* vertical alignment respectively.
- A `<multiplier>` can be used to specify repeated columns e.g. `cols="4*<"` specifies four left-justified columns. The default multiplier value is 1.
- The `<style>` name specifies a table style to used to markup column cells (you can use the full style names if you wish but the first letter is normally sufficient).
- Column specific styles are not applied to header rows.

Cell specifiers allow individual cells in *psv* formatted tables to be spanned, multiplied, aligned and styled. Cell specifiers prefix *psv* `|` delimiters and are formatted like:

```
[<span>*|+][<align>][<style>]
```

- *<span>* specifies horizontal and vertical cell spans (+ operator) or the number of times the cell is replicated (* operator). *<span>* is formatted like:

  ```
  [<colspan>][.<rowspan>]
  ```

  Where `<colspan>` and `<rowspan>` are integers specifying the number of columns and rows to span.
- `<align>` specifies horizontal and vertical cell alignment an is the same as in column specifiers.
- A `<style>` value is the first letter of table style name.

For example, the following *psv* formatted cell will span two columns and the text will be centered and emphasized:

```
`2+^e| Cell text`
```

Table styles can be applied to the entire table (by setting the *style* attribute in the table's attribute list) or on a per column basis (by specifying the style in the table's cols attribute). Table data can be formatted using the following predefined styles:

default

The default style: AsciiDoc inline text formatting; blank lines are treated as paragraph breaks.

emphasis

Like default but all text is emphasised.

monospaced

Like default but all text is in a monospaced font.

strong

Like default but all text is bold.

header

Apply the same style as the table header. Normally used to create a vertical header in the first column.

asciidoc

With this style table cells can contain any of the AsciiDoc elements that are allowed inside document sections. This style runs asciidoc(1) as a filter to process cell contents. See also Docbook table limitations.

literal

No text formatting; monospaced font; all line breaks are retained (the same as the AsciiDoc LiteralBlock element).

verse

All line breaks are retained (just like the AsciiDoc verse paragraph style).

AsciiDoc makes a number of attributes available to table markup templates and tags. Column specific attributes are available when substituting the *colspec* cell data tags.

pageunits

DocBook backend only. Specifies table column absolute width units. Defaults to *.

pagewidth

DocBook backend only. The nominal output page width in *pageunit* units. Used to calculate CALS tables absolute column and table widths. Defaults to *425*.

tableabswidth

Integer value calculated from *width* and *pagewidth* attributes. In *pageunit* units.

tablepcwidth

Table width expressed as a percentage of the available width. Integer value (0..100).

colabswidth

Integer value calculated from *cols* column width, *width* and *pagewidth* attributes. In *pageunit* units.

colpcwidth

Column width expressed as a percentage of the table width. Integer value (0..100).

colcount

Total number of table columns.

rowcount

Total number of table rows.

halign

Horizontal cell content alignment: *left*, *right* or *center*.

valign

Vertical cell content alignment: *top*, *bottom* or *middle*.

colnumber, colstart

: The number of the leftmost column occupied by the cell (1…).

colend

: The number of the rightmost column occupied by the cell (1…).

colspan

: Number of columns the cell should span.

rowspan

: Number of rows the cell should span (1…).

morerows

: Number of additional rows the cell should span (0…).

An alternative *psv* separator character *!* can be used (instead of |) in nested tables. This allows a single level of table nesting. Columns containing nested tables must use the *asciidoc* style. An example can be found in `./examples/website/newtables.txt`.

Fully implementing tables is not trivial, some DocBook toolchains do better than others. AsciiDoc HTML table outputs are rendered correctly in all the popular browsers — if your DocBook generated tables don't look right compare them with the output generated by the AsciiDoc *xhtml11* backend or try a different DocBook toolchain. Here is a list of things to be aware of:

- Although nested tables are not legal in DocBook 4 the FOP and dblatex toolchains will process them correctly. If you use `a2x(1)` you will need to include the `--no-xmllint` option to suppress DocBook validation errors.

  Note In theory you can nest DocBook 4 tables one level using the *entrytbl* element, but not all toolchains process *entrytbl*.

- DocBook only allows a subset of block elements inside table cells so not all AsciiDoc elements produce valid DocBook inside table cells. If you get validation errors running `a2x(1)` try the `--no-xmllint` option, toolchains will often process nested block elements such as sidebar blocks and floating titles correctly even though, strictly speaking, they are not legal.

- Text formatting in cells using the *monospaced* table style will raise validation errors because the DocBook *literal* element was not designed to support formatted text (using the *literal* element is a kludge on the part of AsciiDoc as there is no easy way to set the font style in DocBook.

- Cell alignments are ignored for *verse*, *literal* or *asciidoc* table styles.

Sooner or later, if you program in a UNIX environment, you're going to have to write a man page.

By observing a couple of additional conventions (detailed below) you can write AsciiDoc files that will generate HTML and PDF man pages plus the native manpage roff format. The easiest way to generate roff manpages from AsciiDoc source is to use the a2x(1) command. The following example generates a roff formatted manpage file called `asciidoc.1` (a2x(1) uses asciidoc(1) to convert `asciidoc.1.txt` to DocBook which it then converts to roff using DocBook XSL Stylesheets):

```
a2x --doctype manpage --format manpage asciidoc.1.txt
```

Viewing and printing manpage files

Use the `man(1)` command to view the manpage file:

```
$ man -l asciidoc.1
```

To print a high quality man page to a postscript printer:

```
$ man -l -Tps asciidoc.1 | lpr
```

You could also create a PDF version of the man page by converting PostScript to PDF using `ps2pdf(1)`:

```
$ man -l -Tps asciidoc.1 | ps2pdf - asciidoc.1.pdf
```

The `ps2pdf(1)` command is included in the Ghostscript distribution.

To find out more about man pages view the `man(7)` manpage (`man 7 man` and `man man-pages` commands).

A manpage document Header is mandatory. The title line contains the man page name followed immediately by the manual section number in brackets, for example *ASCIIDOC(1)*. The title name should not contain white space and the manual section number is a single digit optionally followed by a single character.

The first manpage section is mandatory, must be titled *NAME* and must contain a single paragraph (usually a single line) consisting of a list of one or more comma separated command name(s) separated from the command purpose by a dash character. The dash must have at least one white space character on either side. For example:

```
printf, fprintf, sprintf - print formatted output
```

The second manpage section is mandatory and must be titled *SYNOPSIS*.

In addition to the automatically created man page intrinsic attributes you can assign DocBook refmiscinfo element *source*, *version* and *manual* values using AsciiDoc `{mansource}`, `{manversion}` and `{manmanual}` attributes respectively. This example is from the AsciiDoc header of a man page source file:

```
:man source:   AsciiDoc
:man version:  {revnumber}
:man manual:   AsciiDoc Manual
```

The *asciimath* and *latexmath* passthrough blocks along with the *asciimath* and *latexmath* passthrough macros provide a (backend dependent) mechanism for rendering mathematical formulas. You can use the following math markups:

LaTeX math can be included in documents that are processed by dblatex(1). Example inline formula:

```
latexmath:[$C = \alpha + \beta Y^{\gamma} + \epsilon$]
```

For more examples see the AsciiDoc website or the distributed `doc/latexmath.txt` file.

MathJax allows LaTeX Math style formulas to be included in XHTML documents generated via the AsciiDoc *xhtml11* and *html5* backends. This route overcomes several restrictions of the MathML-based approaches, notably, restricted support of MathML by many mainstream browsers. To enable *MathJax* support you must define the *mathjax* attribute, for example using the `-a mathjax` command-line option. Equations are specified as explained above using the *latexmath* passthrough blocks. By default, rendering of equations with *MathJax* requires a working internet connection and will thus not work if you are offline (but it can be configured differently).

LaTeXMathML allows LaTeX Math style formulas to be included in XHTML documents generated using the AsciiDoc *xhtml11* and *html5* backends. AsciiDoc uses the original LaTeXMathML by Douglas Woodall. *LaTeXMathML* is derived from ASCIIMathML and is for users who are more familiar with or prefer using LaTeX math formulas (it recognizes a subset of LaTeX Math, the differences are documented on the *LaTeXMathML* web page). To enable LaTeXMathML support you must define the *latexmath* attribute, for example using the `-a latexmath` command-line option. Example inline formula:

```
latexmath:[$\sum_{n=1}^\infty \frac{1}{2^n}$]
```

For more examples see the AsciiDoc website or the distributed `doc/latexmathml.txt` file.

Note The *latexmath* macro used to include *LaTeX Math* in DocBook outputs is not the same as the *latexmath* macro used to include *LaTeX MathML* in XHTML outputs. *LaTeX Math* applies to DocBook outputs that are processed by dblatex and is normally used to generate PDF files. *LaTeXMathML* is very much a subset of *LaTeX Math* and applies to XHTML documents. This remark does not apply to *MathJax* which does not use any of the *latexmath* macros (but only requires the *latexmath* passthrough blocks for identification of the equations).

ASCIIMathML formulas can be included in XHTML documents generated using the *xhtml11* and *html5* backends. To enable ASCIIMathML support you must define the *asciimath* attribute, for example using the `-a asciimath` command-line option. Example inline formula:

```
asciimath:[`x/x={(1,if x!=0),(text{undefined},if x=0):}`]
```

For more examples see the AsciiDoc website or the distributed `doc/asciimathml.txt` file.

MathML is a low level XML markup for mathematics. AsciiDoc has no macros for MathML but users familiar with this markup could use passthrough macros and passthrough blocks to include MathML in output documents.

AsciiDoc source file syntax and output file markup is largely controlled by a set of cascading, text based, configuration files. At runtime The AsciiDoc default configuration files are combined with optional user and document specific configuration files.

Configuration files contain named sections. Each section begins with a section name in square brackets []. The section body consists of the lines of text between adjacent section headings.

- Section names consist of one or more alphanumeric, underscore or dash characters and cannot begin or end with a dash.
- Lines starting with a # character are treated as comments and ignored.
- If the section name is prefixed with a + character then the section contents is appended to the contents of an already existing same-named section.
- Otherwise same-named sections and section entries override previously loaded sections and section entries (this is sometimes referred to as *cascading*). Consequently, downstream configuration files need only contain those sections and section entries that need to be overridden.

Tip When creating custom configuration files you only need to include the sections and entries that differ from the default configuration.

Tip The best way to learn about configuration files is to read the default configuration files in the AsciiDoc distribution in conjunction with asciidoc(1) output files. You can view configuration file load sequence by turning on the asciidoc(1) `-v` (`--verbose`) command-line option.

AsciiDoc reserves the following section names for specific purposes:

miscellaneous

Configuration options that don't belong anywhere else.

attributes

Attribute name/value entries.

specialcharacters

Special characters reserved by the backend markup.

tags

Backend markup tags.

quotes

Definitions for quoted inline character formatting.

specialwords

Lists of words and phrases singled out for special markup.

replacements, replacements2, replacements3

Find and replace substitution definitions.

specialsections

Used to single out special section names for specific markup.

macros

Macro syntax definitions.

titles

Heading, section and block title definitions.

paradef-*

Paragraph element definitions.

blockdef-*

DelimitedBlock element definitions.

listdef-*

List element definitions.

listtags-*

List element tag definitions.

tabledef-*

Table element definitions.

tabletags-*

Table element tag definitions.

Each line of text in these sections is a *section entry*. Section entries share the following syntax:

name=value

The entry value is set to value.

name=

The entry value is set to a zero length string.

name!

The entry is undefined (deleted from the configuration). This syntax only applies to *attributes* and *miscellaneous* sections.

Section entry behavior

- All equals characters inside the `name` must be escaped with a backslash character.
- `name` and `value` are stripped of leading and trailing white space.
- Attribute names, tag entry names and markup template section names consist of one or more alphanumeric, underscore or dash characters. Names should not begin or end with a dash.
- A blank configuration file section (one without any entries) deletes any preceding section with the same name (applies to non-markup template sections).

The optional `[miscellaneous]` section specifies the following `name=value` options:

newline

> Output file line termination characters. Can include any valid Python string escape sequences. The default value is `\r\n` (carriage return, line feed). Should not be quoted or contain explicit spaces (use `\x20` instead). For example:
>
> ```
> $ asciidoc -a 'newline=\n' -b docbook mydoc.txt
> ```

outfilesuffix

> The default extension for the output file, for example `outfilesuffix=.html`. Defaults to backend name.

tabsize

> The number of spaces to expand tab characters, for example `tabsize=4`. Defaults to 8. A *tabsize* of zero suppresses tab expansion (useful when piping included files through block filters). Included files can override this option using the *tabsize* attribute.

pagewidth, pageunits

> These global table related options are documented in the Table Configuration File Definitions sub-section.

Note `[miscellaneous]` configuration file entries can be set using the asciidoc(1) `-a` (`--attribute`) command-line option.

sectiontitle

> Two line section title pattern. The entry value is a Python regular expression containing the named group *title*.

underlines

> A comma separated list of document and section title underline character pairs starting with the section level 0 and ending with section level 4 underline. The default setting is:
>
> ```
> underlines="==","--","~~","^^","++"
> ```

sect0…sect4

> One line section title patterns. The entry value is a Python regular expression containing the named group *title*.

blocktitle

> BlockTitle element pattern. The entry value is a Python regular expression containing the named group *title*.

subs

> A comma separated list of substitutions that are performed on the document header and section titles. Defaults to *normal* substitution.

The `[tags]` section contains backend tag definitions (one per line). Tags are used to translate AsciiDoc elements to backend markup.

An AsciiDoc tag definition is formatted like `<tagname>=<starttag>|<endtag>`. For example:

```
emphasis=<em>|</em>
```

In this example asciidoc(1) replaces the | character with the emphasized text from the AsciiDoc input file and writes the result to the output file.

Use the `{brvbar}` attribute reference if you need to include a | pipe character inside tag text.

The optional `[attributes]` section contains predefined attributes.

If the attribute value requires leading or trailing spaces then the text text should be enclosed in quotation mark (") characters.

To delete a attribute insert a `name!` entry in a downstream configuration file or use the asciidoc(1) `--attribute name!` command-line option (an attribute name suffixed with a `!` character deletes the attribute)

The `[specialcharacters]` section specifies how to escape characters reserved by the backend markup. Each translation is specified on a single line formatted like:

```
<special_character>=<translated_characters>
```

Special characters are normally confined to those that resolve markup ambiguity (in the case of HTML and XML markups the ampersand, less than and greater than characters). The following example causes all occurrences of the < character to be replaced by `&lt;`.

```
<=&lt;
```

Quoting is used primarily for text formatting. The `[quotes]` section defines AsciiDoc quoting characters and their corresponding backend markup tags. Each section entry value is the name of a of a `[tags]` section entry. The entry name is the character (or characters) that quote the text. The following examples are taken from AsciiDoc configuration files:

```
[quotes]
_=emphasis
```

```
[tags]
emphasis=<em>|</em>
```

You can specify the left and right quote strings separately by separating them with a | character, for example:

```
``|''=quoted
```

Omitting the tag will disable quoting, for example, if you don't want superscripts or subscripts put the following in a custom configuration file or edit the global `asciidoc.conf` configuration file:

```
[quotes]
^=
~=
```

Unconstrained quotes are differentiated from constrained quotes by prefixing the tag name with a hash character, for example:

```
__=#emphasis
```

Quoted text behavior

- Quote characters must be non-alphanumeric.
- To minimize quoting ambiguity try not to use the same quote characters in different quote types.

The `[specialwords]` section is used to single out words and phrases that you want to consistently format in some way throughout your document without having to repeatedly specify the markup. The name of each entry corresponds to a markup template section and the entry value consists of a list of words and phrases to be marked up. For example:

```
[specialwords]
strongwords=NOTE IMPORTANT
```

```
[strongwords]
<strong>{words}</strong>
```

The examples specifies that any occurrence of `NOTE` or `IMPORTANT` should appear in a bold font.

Words and word phrases are treated as Python regular expressions: for example, the word `^NOTE` would only match `NOTE` if appeared at the start of a line.

AsciiDoc comes with three built-in Special Word types: *emphasizedwords*, *monospacedwords* and *strongwords*, each has a corresponding (backend specific) markup template section. Edit the configuration files to customize existing Special Words and to add new ones.

Special word behavior

- Word list entries must be separated by space characters.
- Word list entries with embedded spaces should be enclosed in quotation (") characters.
- A `[specialwords]` section entry of the form `name=word1 [word2…]` adds words to existing `name` entries.
- A `[specialwords]` section entry of the form `name` undefines (deletes) all existing `name` words.
- Since word list entries are processed as Python regular expressions you need to be careful to escape regular expression special characters.
- By default Special Words are substituted before Inline Macros, this may lead to undesirable consequences. For example the special word `foobar` would be expanded inside the macro call `http://www.foobar.com[]`. A possible solution is to emphasize whole words only by defining the word using regular expression characters, for example `\bfoobar\b`.
- If the first matched character of a special word is a backslash then the remaining characters are output without markup i.e. the backslash can be used to escape special word markup. For example the special word `\\?\b[Tt]en\b` will mark up the words `Ten` and `ten` only if they are not

preceded by a backslash.

`[replacements]`, `[replacements2]` and `[replacements3]` configuration file entries specify find and replace text and are formatted like:

```
<find_pattern>=<replacement_text>
```

The find text can be a Python regular expression; the replace text can contain Python regular expression group references.

Use Replacement shortcuts for often used macro references, for example (the second replacement allows us to backslash escape the macro name):

```
NEW!=image:./images/smallnew.png[New!]
\\NEW!=NEW!
```

The only difference between the three replacement types is how they are applied. By default *replacements* and *replacements2* are applied in normal substitution contexts whereas *replacements3* needs to be configured explicitly and should only be used in backend configuration files.

Replacement behavior

- The built-in replacements can be escaped with a backslash.
- If the find or replace text has leading or trailing spaces then the text should be enclosed in quotation (") characters.
- Since the find text is processed as a regular expression you need to be careful to escape regular expression special characters.
- Replacements are performed in the same order they appear in the configuration file replacements section.

Markup template sections supply backend markup for translating AsciiDoc elements. Since the text is normally backend dependent you'll find these sections in the backend specific configuration files. Template sections differ from other sections in that they contain a single block of text instead of per line *name=value* entries. A markup template section body can contain:

- Attribute references
- System macro calls.
- A document content placeholder

The document content placeholder is a single | character and is replaced by text from the source element. Use the `{brvbar}` attribute reference if you need a literal | character in the template.

Configuration files have a `.conf` file name extension; they are loaded from the following locations:

1. The directory containing the asciidoc executable.
2. If there is no `asciidoc.conf` file in the directory containing the asciidoc executable then load from the global configuration directory (normally `/etc/asciidoc` or `/usr/local/etc/asciidoc`) i.e. the global configuration files directory is skipped if AsciiDoc configuration files are installed in the same directory as the asciidoc executable. This allows both a system wide copy and multiple local copies of AsciiDoc to coexist on the same host PC.
3. The user's `$HOME/.asciidoc` directory (if it exists).
4. The directory containing the AsciiDoc source file.
5. Explicit configuration files specified using:
    - The `conf-files` attribute (one or more file names separated by a `|` character). These files are loaded in the order they are specified and prior to files specified using the `--conf-file` command-line option.
    - The asciidoc(1) `--conf-file`) command-line option. The `--conf-file` option can be specified multiple times, in which case configuration files will be processed in the same order they appear on the command-line.
6. Backend plugin configuration files are loaded from subdirectories named like `backends/<backend>` in locations 1, 2 and 3.
7. Filter configuration files are loaded from subdirectories named like `filters/<filter>` in locations 1, 2 and 3.

Configuration files from the above locations are loaded in the following order:

- The `[attributes]` section only from:
    - `asciidoc.conf` in location 3
    - Files from location 5.

      This first pass makes locally set attributes available in the global `asciidoc.conf` file.
- `asciidoc.conf` from locations 1, 2, 3.

- *attributes*, *titles* and *specialcharacters* sections from the `asciidoc.conf` in location 4.
- The document header is parsed at this point and we can assume the *backend* and *doctype* have now been defined.
- Backend plugin `<backend>.conf` and `<backend>-<doctype>.conf` files from locations 6. If a backend plugin is not found then try locations 1, 2 and 3 for `<backend>.conf` and `<backend>-<doctype>.conf` backend configuration files.
- Filter conf files from locations 7.
- `lang-<lang>.conf` from locations 1, 2, 3.
- `asciidoc.conf` from location 4.
- `<backend>.conf` and `<backend>-<doctype>.conf` from location 4.
- Filter conf files from location 4.
- `<docfile>.conf` and `<docfile>-<backend>.conf` from location 4.
- Configuration files from location 5.

Where:

- `<backend>` and `<doctype>` are values specified by the asciidoc(1) `-b` (`--backend`) and `-d` (`--doctype`) command-line options.
- `<infile>` is the path name of the AsciiDoc input file without the file name extension.
- `<lang>` is a two letter country code set by the the AsciiDoc *lang* attribute.

Note The backend and language global configuration files are loaded **after** the header has been parsed. This means that you can set most attributes in the document header. Here's an example header:

```
Life's Mysteries
================
:author: Hu Nose
:doctype: book
:toc:
:icons:
:data-uri:
:lang: en
:encoding: iso-8859-1
```

Attributes set in the document header take precedence over configuration file attributes.

Tip Use the asciidoc(1) `-v` (`--verbose`) command-line option to see which configuration files are loaded and the order in which they are loaded.

A document attribute is comprised of a *name* and a textual *value* and is used for textual substitution in AsciiDoc documents and configuration files. An attribute reference (an attribute name enclosed in braces) is replaced by the corresponding attribute value. Attribute names are case insensitive and can only contain alphanumeric, dash and underscore characters.

There are four sources of document attributes (from highest to lowest precedence):

- Command-line attributes.
- AttributeEntry, AttributeList, Macro and BlockId elements.
- Configuration file `[attributes]` sections.
- Intrinsic attributes.

Within each of these divisions the last processed entry takes precedence.

Note If an attribute is not defined then the line containing the attribute reference is dropped. This property is used extensively in AsciiDoc configuration files to facilitate conditional markup generation.

The `AttributeEntry` block element allows document attributes to be assigned within an AsciiDoc document. Attribute entries are added to the global document attributes dictionary. The attribute name/value syntax is a single line like:

```
:<name>: <value>
```

For example:

```
:Author Initials: JB
```

This will set an attribute reference `{authorinitials}` to the value *JB* in the current document.

To delete (undefine) an attribute use the following syntax:

```
:<name>!:
```

AttributeEntry behavior

- The attribute entry line begins with colon — no white space allowed in left margin.
- AsciiDoc converts the `<name>` to a legal attribute name (lower case, alphanumeric, dash and underscore characters only — all other characters deleted). This allows more human friendly text to be used.
- Leading and trailing white space is stripped from the `<value>`.
- Lines ending in a space followed by a plus character are continued to the next line, for example:

  ```
  :description: AsciiDoc is a text document format for writing notes, +
                documentation, articles, books, slideshows, web pages +
                and man pages.
  ```

- If the `<value>` is blank then the corresponding attribute value is set to an empty string.
- Attribute references contained in the entry `<value>` will be expanded.
- By default AttributeEntry values are substituted for `specialcharacters` and `attributes` (see above), if you want to change or disable AttributeEntry substitution use the inline macro syntax.
- Attribute entries in the document Header are available for header markup template substitution.
- Attribute elements override configuration file and intrinsic attributes but do not override command-line attributes.

Here are some more attribute entry examples:

```
AsciiDoc User Manual
====================
:author:    Stuart Rackham
:email:     srackham@gmail.com
:revdate:   April 23, 2004
:revnumber: 5.1.1
```

Which creates these attributes:

```
{author}, {firstname}, {lastname}, {authorinitials}, {email},
{revdate}, {revnumber}
```

The previous example is equivalent to this document header:

```
AsciiDoc User Manual
====================
Stuart Rackham <srackham@gmail.com>
5.1.1, April 23, 2004
```

A variant of the Attribute Entry syntax allows configuration file section entries and markup template sections to be set from within an AsciiDoc document:

```
:<section_name>.[<entry_name>]: <entry_value>
```

Where `<section_name>` is the configuration section name, `<entry_name>` is the name of the entry and `<entry_value>` is the optional entry value. This example sets the default labeled list style to *horizontal*:

```
:listdef-labeled.style: horizontal
```

It is exactly equivalent to a configuration file containing:

```
[listdef-labeled]
style=horizontal
```

- If the `<entry_name>` is omitted then the entire section is substituted with the `<entry_value>`. This feature should only be used to set markup template sections. The following example sets the *xref2* inline macro markup template:

  ```
  :xref2-inlinemacro.: <a href="#{1}">{2?{2}}</a>
  ```

- No substitution is performed on configuration file attribute entries and they cannot be undefined.
- This feature can only be used in attribute entries — configuration attributes cannot be set using the asciidoc(1) command `--attribute` option.

Attribute entries promote clarity and eliminate repetition

URLs and file names in AsciiDoc macros are often quite long — they break paragraph flow and readability suffers. The problem is compounded by redundancy if the same name is used repeatedly. Attribute entries can be used to make your documents easier to read and write, here are some examples:

```
:1:         http://freshmeat.net/projects/asciidoc/
:homepage:  http://asciidoc.org[AsciiDoc home page]
:new:       image:./images/smallnew.png[]
:footnote1: footnote:[A meaningless latin term]
```

```
Using previously defined attributes: See the {1}[Freshmeat summary]
or the {homepage} for something new {new}. Lorem ispum {footnote1}.
```

Note

- The attribute entry definition must precede it's usage.
- You are not limited to URLs or file names, entire macro calls or arbitrary lines of text can be abbreviated.
- Shared attributes entries could be grouped into a separate file and included in multiple documents.

- An attribute list is a comma separated list of attribute values.
- The entire list is enclosed in square brackets.
- Attribute lists are used to pass parameters to macros, blocks (using the AttributeList element) and inline quotes.

The list consists of zero or more positional attribute values followed by zero or more named attribute values. Here are three examples: a single unquoted positional attribute; three unquoted positional attribute values; one positional attribute followed by two named attributes; the unquoted attribute value in the final example contains comma (`&#44;`) and double-quote (`&#34;`) character entities:

```
[Hello]
[quote, Bertrand Russell, The World of Mathematics (1956)]
["22 times", backcolor="#0e0e0e", options="noborders,wide"]
[A footnote&#44; &#34;with an image&#34; image:smallnew.png[]]
```

Attribute list behavior

- If one or more attribute values contains a comma the all string values must be quoted (enclosed in double quotation mark characters).
- If the list contains any named or quoted attributes then all string attribute values must be quoted.
- To include a double quotation mark (") character in a quoted attribute value the the quotation mark must be escaped with a backslash.
- List attributes take precedence over existing attributes.
- List attributes can only be referenced in configuration file markup templates and tags, they are not available elsewhere in the document.
- Setting a named attribute to `None` undefines the attribute.
- Positional attributes are referred to as `{1}`,`{2}`,`{3}`,…
- Attribute `{0}` refers to the entire list (excluding the enclosing square brackets).
- Named attribute names cannot contain dash characters.

If the attribute list contains an attribute named `options` it is processed as a comma separated list of option names:

- Each name generates an attribute named like `<option>-option` (where `<option>` is the option name) with an empty string value. For example `[options="opt1,opt2,opt3"]` is equivalent to setting the following three attributes `[opt1-option="",opt2-option="",opt2-option=""]`.
- If you define a an option attribute globally (for example with an attribute entry) then it will apply to all elements in the document.
- AsciiDoc implements a number of predefined options which are listed in the Attribute Options appendix.

Macros calls are suffixed with an attribute list. The list may be empty but it cannot be omitted. List entries are used to pass attribute values to macro markup templates.

An attribute reference is an attribute name (possibly followed by an additional parameters) enclosed in curly braces. When an attribute reference is encountered it is evaluated and replaced by its corresponding text value. If the attribute is undefined the line containing the attribute is dropped.

There are three types of attribute reference: *Simple*, *Conditional* and *System*.

Attribute reference evaluation

- You can suppress attribute reference expansion by placing a backslash character immediately in front of the opening brace character.

- By default attribute references are not expanded in *LiteralParagraphs*, *ListingBlocks* or *LiteralBlocks*.
- Attribute substitution proceeds line by line in reverse line order.
- Attribute reference evaluation is performed in the following order: *Simple* then *Conditional* and finally *System*.

Simple attribute references take the form `{<name>}`. If the attribute name is defined its text value is substituted otherwise the line containing the reference is dropped from the output.

Additional parameters are used in conjunction with attribute names to calculate a substitution value. Conditional attribute references take the following forms:

`{<names>=<value>}`

> `<value>` is substituted if the attribute `<names>` is undefined otherwise its value is substituted. `<value>` can contain simple attribute references.

`{<names>?<value>}`

> `<value>` is substituted if the attribute `<names>` is defined otherwise an empty string is substituted. `<value>` can contain simple attribute references.

`{<names>!<value>}`

> `<value>` is substituted if the attribute `<names>` is undefined otherwise an empty string is substituted. `<value>` can contain simple attribute references.

`{<names>#<value>}`

> `<value>` is substituted if the attribute `<names>` is defined otherwise the undefined attribute entry causes the containing line to be dropped. `<value>` can contain simple attribute references.

`{<names>%<value>}`

> `<value>` is substituted if the attribute `<names>` is not defined otherwise the containing line is dropped. `<value>` can contain simple attribute references.

`{<names>@<regexp>:<value1>[:<value2>]}`

> `<value1>` is substituted if the value of attribute `<names>` matches the regular expression `<regexp>` otherwise `<value2>` is substituted. If attribute `<names>` is not defined the containing line is dropped. If `<value2>` is omitted an empty string is assumed. The values and the regular expression can contain simple attribute references. To embed colons in the values or the regular expression escape them with backslashes.

`{<names>$<regexp>:<value1>[:<value2>]}`

> Same behavior as the previous ternary attribute except for the following cases:
>
> `{<names>$<regexp>:<value>}`
>
> > Substitutes `<value>` if `<names>` matches `<regexp>` otherwise the result is undefined and the containing line is dropped.
>
> `{<names>$<regexp>::<value>}`
>
> > Substitutes `<value>` if `<names>` does not match `<regexp>` otherwise the result is undefined and the containing line is dropped.

The attribute `<names>` parameter normally consists of a single attribute name but it can be any one of the following:

- A single attribute name which evaluates to the attributes value.
- Multiple `,` separated attribute names which evaluates to an empty string if one or more of the attributes is defined, otherwise it's value is undefined.
- Multiple `+` separated attribute names which evaluates to an empty string if all of the attributes are defined, otherwise it's value is undefined.

Conditional attributes with single attribute names are evaluated first so they can be used inside the multi-attribute conditional `<value>`.

Conditional attributes are mainly used in AsciiDoc configuration files — see the distribution `.conf` files for examples.

Attribute equality test

> If `{backend}` is *docbook45* or *xhtml11* the example evaluates to "DocBook 4.5 or XHTML 1.1 backend" otherwise it evaluates to "some other backend":
>
> ```
> {backend@docbook45|xhtml11:DocBook 4.5 or XHTML 1.1 backend:some other backend}
> ```

Attribute value map

This example maps the `frame` attribute values [`topbot`, `all`, `none`, `sides`] to [`hsides`, `border`, `void`, `vsides`]:

```
{frame@topbot:hsides}{frame@all:border}{frame@none:void}{frame@sides:vsides}
```

System attribute references generate the attribute text value by executing a predefined action that is parametrized by one or more arguments. The syntax is `{<action>:<arguments>}`.

`{counter:<attrname>[:<seed>]}`

Increments the document attribute (if the attribute is undefined it is set to `1`). Returns the new attribute value.

- Counters generate global (document wide) attributes.
- The optional `<seed>` specifies the counter's initial value; it can be a number or a single letter; defaults to *1*.
- `<seed>` can contain simple and conditional attribute references.
- The *counter* system attribute will not be executed if the containing line is dropped by the prior evaluation of an undefined attribute.

`{counter2:<attrname>[:<seed>]}`

Same as `counter` except the it always returns a blank string.

`{eval:<expression>}`

Substitutes the result of the Python `<expression>`.

- If `<expression>` evaluates to `None` or `False` the reference is deemed undefined and the line containing the reference is dropped from the output.
- If the expression evaluates to `True` the attribute evaluates to an empty string.
- `<expression>` can contain simple and conditional attribute references.
- The *eval* system attribute can be nested inside other system attributes.

`{eval3:<command>}`

Passthrough version of `{eval:<expression>}` — the generated output is written directly to the output without any further substitutions.

`{include:<filename>}`

Substitutes contents of the file named `<filename>`.

- The included file is read at the time of attribute substitution.
- If the file does not exist a warning is emitted and the line containing the reference is dropped from the output file.
- Tabs are expanded based on the current *tabsize* attribute value.

`{set:<attrname>[!][:<value>]}`

Sets or unsets document attribute. Normally only used in configuration file markup templates (use AttributeEntries in AsciiDoc documents).

- If the attribute name is followed by an exclamation mark the attribute becomes undefined.
- If `<value>` is omitted the attribute is set to a blank string.
- `<value>` can contain simple and conditional attribute references.
- Returns a blank string unless the attribute is undefined in which case the return value is undefined and the enclosing line will be dropped.

`{set2:<attrname>[!][:<value>]}`

Same as `set` except that the attribute scope is local to the template.

`{sys:<command>}`

Substitutes the stdout generated by the execution of the shell `<command>`.

`{sys2:<command>}`

Substitutes the stdout and stderr generated by the execution of the shell `<command>`.

`{sys3:<command>}`

Passthrough version of `{sys:<command>}` — the generated output is written directly to the output without any further substitutions.

`{template:<template>}`

Substitutes the contents of the configuration file section named `<template>`. Attribute references contained in the template are substituted.

System reference behavior

- System attribute arguments can contain non-system attribute references.
- Closing brace characters inside system attribute arguments must be escaped with a backslash.

Intrinsic attributes are simple attributes that are created automatically from: AsciiDoc document header parameters; asciidoc(1) command-line arguments; attributes defined in the default configuration files; the execution context. Here's the list of predefined intrinsic attributes:

```
{amp}                 ampersand (&) character entity
{asciidoc-args}       used to pass inherited arguments to asciidoc filters
{asciidoc-confdir}    the asciidoc(1) global configuration directory
{asciidoc-dir}        the asciidoc(1) application directory
{asciidoc-file}       the full path name of the asciidoc(1) script
{asciidoc-version}    the version of asciidoc(1)
{author}              author's full name
{authored}            empty string '' if {author} or {email} defined,
{authorinitials}      author initials (from document header)
{backend-<backend>}   empty string ''
{<backend>-<doctype>} empty string ''
{backend}             document backend specified by `-b` option
{backend-confdir}     the directory containing the <backend>.conf file
{backslash}           backslash character
{basebackend-<base>}  empty string ''
{basebackend}         html or docbook
{blockname}           current block name (note 8).
{brvbar}              broken vertical bar (|) character
{docdate}             document last modified date
{docdir}              document input directory name  (note 5)
{docfile}             document file name  (note 5)
{docname}             document file name without extension (note 6)
{doctime}             document last modified time
{doctitle}            document title (from document header)
{doctype-<doctype>}   empty string ''
{doctype}             document type specified by `-d` option
{email}               author's email address (from document header)
{empty}               empty string ''
{encoding}            specifies input and output encoding
{filetype-<fileext>}  empty string ''
{filetype}            output file name file extension
{firstname}           author first name (from document header)
{gt}                  greater than (>) character entity
{id}                  running block id generated by BlockId elements
{indir}               input file directory name (note 2,5)
{infile}              input file name (note 2,5)
{lastname}            author last name (from document header)
{ldquo}               Left double quote character (note 7)
{level}               title level 1..4 (in section titles)
{listindex}           the list index (1..) of the most recent list item
{localdate}           the current date
{localtime}           the current time
{lsquo}               Left single quote character (note 7)
{lt}                  less than (<) character entity
{manname}             manpage name (defined in NAME section)
{manpurpose}          manpage (defined in NAME section)
{mantitle}            document title minus the manpage volume number
{manvolnum}           manpage volume number (1..8) (from document header)
{middlename}          author middle name (from document header)
{nbsp}                non-breaking space character entity
{notitle}             do not display the document title
{outdir}              document output directory name (note 2)
{outfile}             output file name (note 2)
{plus}                plus character
{python}              the full path name of the Python interpreter executable
{rdquo}               right double quote character (note 7)
{reftext}             running block xreflabel generated by BlockId elements
{revdate}             document revision date (from document header)
{revnumber}           document revision number (from document header)
{rsquo}               right single quote character (note 7)
{sectnum}             formatted section number (in section titles)
{sp}                  space character
{showcomments}        send comment lines to the output
{title}               section title (in titled elements)
{two-colons}          two colon characters
{two-semicolons}      two semicolon characters
{user-dir}            the ~/.asciidoc directory (if it exists)
{verbose}             defined as '' if --verbose command option specified
```

`{wj}`	word-joiner
`{zwsp}`	zero-width space character entity

Note

1. Intrinsic attributes are global so avoid defining custom attributes with the same names.
2. `{outfile}`, `{outdir}`, `{infile}`, `{indir}` attributes are effectively read-only (you can set them but it won't affect the input or output file paths).
3. See also the Backend Attributes section for attributes that relate to AsciiDoc XHTML file generation.
4. The entries that translate to blank strings are designed to be used for conditional text inclusion. You can also use the `ifdef`, `ifndef` and `endif` System macros for conditional inclusion. [4]
5. `{docfile}` and `{docdir}` refer to root document specified on the asciidoc(1) command-line; `{infile}` and `{indir}` refer to the current input file which may be the root document or an included file. When the input is being read from the standard input (`stdin`) these attributes are undefined.
6. If the input file is the standard input and the output file is not the standard output then `{docname}` is the output file name sans file extension.
7. See non-English usage of quotation marks.
8. The `{blockname}` attribute identifies the style of the current block. It applies to delimited blocks, lists and tables. Here is a list of `{blockname}` values (does not include filters or custom block and style names):

   delimited blocks

      comment, sidebar, open, pass, literal, verse, listing, quote, example, note, tip, important, caution, warning, abstract, partintro

   lists

      arabic, loweralpha, upperalpha, lowerroman, upperroman, labeled, labeled3, labeled4, qanda, horizontal, bibliography, glossary

   tables

      table

The syntax and behavior of Paragraph, DelimitedBlock, List and Table block elements is determined by block definitions contained in AsciiDoc configuration file sections.

Each definition consists of a section title followed by one or more section entries. Each entry defines a block parameter controlling some aspect of the block's behavior. Here's an example:

```
[blockdef-listing]
delimiter=^-{4,}$
template=listingblock
presubs=specialcharacters,callouts
```

Configuration file block definition sections are processed incrementally after each configuration file is loaded. Block definition section entries are merged into the block definition, this allows block parameters to be overridden and extended by later loading configuration files.

AsciiDoc Paragraph, DelimitedBlock, List and Table block elements share a common subset of configuration file parameters:

delimiter

   A Python regular expression that matches the first line of a block element — in the case of DelimitedBlocks and Tables it also matches the last line.

template

   The name of the configuration file markup template section that will envelope the block contents. The pipe (|) character is substituted for the block contents. List elements use a set of (list specific) tag parameters instead of a single template. The template name can contain attribute references allowing dynamic template selection a the time of template substitution.

options

   A comma delimited list of element specific option names. In addition to being used internally, options are available during markup tag and template substitution as attributes with an empty string value named like `<option>-option` (where `<option>` is the option name). See attribute options for a complete list of available options.

subs, presubs, postsubs

- *presubs* and *postsubs* are lists of comma separated substitutions that are performed on the block contents. *presubs* is applied first, *postsubs* (if specified) second.
- *subs* is an alias for *presubs*.
- If a *filter* is allowed (Paragraphs, DelimitedBlocks and Tables) and has been specified then *presubs* and *postsubs* substitutions are performed before and after the filter is run respectively.
- Allowed values: *specialcharacters*, *quotes*, *specialwords*, *replacements*, *macros*, *attributes*, *callouts*.
- The following composite values are also allowed:

  none

  No substitutions.

  normal

  The following substitutions in the following order: *specialcharacters*, *quotes*, *attributes*, *specialwords*, *replacements*, *macros*, *replacements2*.

  verbatim

  The following substitutions in the following order: *specialcharacters* and *callouts*.
- *normal* and *verbatim* substitutions can be redefined by with `subsnormal` and `subsverbatim` entries in a configuration file `[miscellaneous]` section.
- The substitutions are processed in the order in which they are listed and can appear more than once.

filter

This optional entry specifies an executable shell command for processing block content (Paragraphs, DelimitedBlocks and Tables). The filter command can contain attribute references.

posattrs

Optional comma separated list of positional attribute names. This list maps positional attributes (in the block's attribute list) to named block attributes. The following example, from the QuoteBlock definition, maps the first and section positional attributes:

```
posattrs=attribution,citetitle
```

style

This optional parameter specifies the default style name.

<stylename>-style

Optional style definition (see Styles below).

The following block parameters behave like document attributes and can be set in block attribute lists and style definitions: *template*, *options*, *subs*, *presubs*, *postsubs*, *filter*.

A style is a set of block parameter bundled as a single named parameter. The following example defines a style named *verbatim*:

```
verbatim-style=template="literalblock",subs="verbatim"
```

If a block's attribute list contains a *style* attribute then the corresponding style parameters are be merged into the default block definition parameters.

- All style parameter names must be suffixed with `-style` and the style parameter value is in the form of a list of named attributes.
- The *template* style parameter is mandatory, other parameters can be omitted in which case they inherit their values from the default block definition parameters.
- Multi-item style parameters (*subs*,*presubs*,*postsubs*,*posattrs*) must be specified using Python tuple syntax (rather than a simple list of values as they in separate entries) e.g. `postsubs=("callouts",)` not `postsubs="callouts"`.

Paragraph translation is controlled by `[paradef-*]` configuration file section entries. Users can define new types of paragraphs and modify the behavior of existing types by editing AsciiDoc configuration files.

Here is the shipped Default paragraph definition:

```
[paradef-default]
delimiter=(?P<text>\S.*)
template=paragraph
```

The normal paragraph definition has a couple of special properties:

1. It must exist and be defined in a configuration file section named `[paradef-default]`.
2. Irrespective of its position in the configuration files default paragraph document matches are attempted only after trying all other paragraph types.

Paragraph specific block parameter notes:

delimiter

> This regular expression must contain the named group *text* which matches the text on the first line. Paragraphs are terminated by a blank line, the end of file, or the start of a DelimitedBlock.

options

> The *listelement* option specifies that paragraphs of this type will automatically be considered part of immediately preceding list items. The *skip* option causes the paragraph to be treated as a comment (see CommentBlocks).

Paragraph processing proceeds as follows:

1. The paragraph text is aligned to the left margin.
2. Optional *presubs* inline substitutions are performed on the paragraph text.
3. If a filter command is specified it is executed and the paragraph text piped to its standard input; the filter output replaces the paragraph text.
4. Optional *postsubs* inline substitutions are performed on the paragraph text.
5. The paragraph text is enveloped by the paragraph's markup template and written to the output file.

DelimitedBlock *options* values are:

sectionbody

> The block contents are processed as a SectionBody.

skip

> The block is treated as a comment (see CommentBlocks). Preceding attribute lists and block titles are not consumed.

presubs, *postsubs* and *filter* entries are ignored when *sectionbody* or *skip* options are set.

DelimitedBlock processing proceeds as follows:

1. Optional *presubs* substitutions are performed on the block contents.
2. If a filter is specified it is executed and the block's contents piped to its standard input. The filter output replaces the block contents.
3. Optional *postsubs* substitutions are performed on the block contents.
4. The block contents is enveloped by the block's markup template and written to the output file.

Tip Attribute expansion is performed on the block filter command before it is executed, this is useful for passing arguments to the filter.

List behavior and syntax is determined by `[listdef-*]` configuration file sections. The user can change existing list behavior and add new list types by editing configuration files.

List specific block definition notes:

type

> This is either *bulleted*,*numbered*,*labeled* or *callout*.

delimiter

> A Python regular expression that matches the first line of a list element entry. This expression can contain the named groups *text* (bulleted groups), *index* and *text* (numbered lists), *label* and *text* (labeled lists).

tags

> The `<name>` of the `[listtags-<name>]` configuration file section containing list markup tag definitions. The tag entries (*list*, *entry*, *label*, *term*, *text*) map the AsciiDoc list structure to backend markup; see the *listtags* sections in the AsciiDoc distributed backend `.conf` configuration files for examples.

Table behavior and syntax is determined by `[tabledef-*]` and `[tabletags-*]` configuration file sections. The user can change existing table behavior and add new table types by editing configuration files. The following `[tabledef-*]` section entries generate table output markup elements:

colspec

The table *colspec* tag definition.

headrow, footrow, bodyrow

Table header, footer and body row tag definitions. *headrow* and *footrow* table definition entries default to *bodyrow* if they are undefined.

headdata, footdata, bodydata

Table header, footer and body data tag definitions. *headdata* and *footdata* table definition entries default to *bodydata* if they are undefined.

paragraph

If the *paragraph* tag is specified then blank lines in the cell data are treated as paragraph delimiters and marked up using this tag.

Table behavior is also influenced by the following `[miscellaneous]` configuration file entries:

pagewidth

This integer value is the printable width of the output media. See table attributes.

pageunits

The units of width in output markup width attribute values.

Table definition behavior

- The output markup generation is specifically designed to work with the HTML and CALS (DocBook) table models, but should be adaptable to most XML table schema.
- Table definitions can be “mixed in” from multiple cascading configuration files.
- New table definitions inherit the default table and table tags definitions (`[tabledef-default]` and `[tabletags-default]`) so you only need to override those conf file entries that require modification.

AsciiDoc filters allow external commands to process AsciiDoc *Paragraphs*, *DelimitedBlocks* and *Table* content. Filters are primarily an extension mechanism for generating specialized outputs. Filters are implemented using external commands which are specified in configuration file definitions.

There’s nothing special about the filters, they’re just standard UNIX filters: they read text from the standard input, process it, and write to the standard output.

The asciidoc(1) command `--filter` option can be used to install and remove filters. The same option is used to unconditionally load a filter.

Attribute substitution is performed on the filter command prior to execution — attributes can be used to pass parameters from the AsciiDoc source document to the filter.

Warning Filters sometimes included executable code. Before installing a filter you should verify that it is from a trusted source.

If the filter command does not specify a directory path then asciidoc(1) recursively searches for the executable filter command:

- First it looks in the user’s `$HOME/.asciidoc/filters` directory.
- Next the global filters directory (usually `/etc/asciidoc/filters` or `/usr/local/etc/asciidoc`) directory is searched.
- Then it looks in the asciidoc(1) `./filters` directory.
- Finally it relies on the executing shell to search the environment search path (`$PATH`).

Standard practice is to install each filter in it’s own sub-directory with the same name as the filter’s style definition. For example the music filter’s style name is *music* so it’s configuration and filter files are stored in the `filters/music` directory.

Filters are normally accompanied by a configuration file containing a Paragraph or DelimitedBlock definition along with corresponding markup templates.

While it is possible to create new *Paragraph* or *DelimitedBlock* definitions the preferred way to implement a filter is to add a style to the existing Paragraph and ListingBlock definitions (all filters shipped with AsciiDoc use this technique). The filter is applied to the paragraph or delimited block by preceding it with an attribute list: the first positional attribute is the style name, remaining attributes are normally filter specific parameters.

asciidoc(1) auto-loads all `.conf` files found in the filter search paths unless the container directory also contains a file named `__noautoload__` (see previous section). The `__noautoload__` feature is used for filters that will be loaded manually using the `--filter` option.

AsciiDoc comes with a toy filter for highlighting source code keywords and comments. See also the `./filters/code/code-filter-readme.txt` file.

Note The purpose of this toy filter is to demonstrate how to write a filter — it's much to simplistic to be passed off as a code syntax highlighter. If you want a full featured multi-language highlighter use the source code highlighter filter.

The AsciiDoc distribution includes *source*, *music*, *latex* and *graphviz* filters, details are on the AsciiDoc website.

Table 11. Built-in filters list

Filter name	Description
music	A music filter is included in the distribution `./filters/` directory. It translates music in LilyPond or ABC notation to standard classical notation.
source	A source code highlight filter is included in the distribution `./filters/` directory.
latex	The AsciiDoc LaTeX filter translates LaTeX source to an image that is automatically inserted into the AsciiDoc output documents.
graphviz	Gouichi Iisaka has written a Graphviz filter for AsciiDoc. Graphviz generates diagrams from a textual specification. Gouichi Iisaka's Graphviz filter is included in the AsciiDoc distribution. Here are some AsciiDoc Graphviz examples.

Filter plugins are a mechanism for distributing AsciiDoc filters. A filter plugin is a Zip file containing the files that constitute a filter. The asciidoc(1) `--filter` option is used to load and manage filer plugins.

- Filter plugins take precedence over built-in filters with the same name.
- By default filter plugins are installed in `$HOME/.asciidoc/filters/<filter>` where `<filter>` is the filter name.

The AsciiDoc plugin architecture is an extension mechanism that allows additional backends, filters and themes to be added to AsciiDoc.

- A plugin is a Zip file containing an AsciiDoc backend, filter or theme (configuration files, stylesheets, scripts, images).
- The asciidoc(1) `--backend`, `--filter` and `--theme` command-line options are used to load and manage plugins. Each of these options responds to the plugin management *install*, *list*, *remove* and *build* commands.
- The plugin management command names are reserved and cannot be used for filter, backend or theme names.
- The plugin Zip file name always begins with the backend, filter or theme name.

Plugin commands and conventions are documented in the asciidoc(1) man page. You can find lists of plugins on the AsciiDoc website.

The asciidoc(1) command has a `--help` option which prints help topics to stdout. The default topic summarizes asciidoc(1) usage:

```
$ asciidoc --help
```

To print a help topic specify the topic name as a command argument. Help topic names can be shortened so long as they are not ambiguous. Examples:

```
$ asciidoc --help manpage
$ asciidoc -h m              # Short version of previous example.
$ asciidoc --help syntax
$ asciidoc -h s              # Short version of previous example.
```

To change, delete or add your own help topics edit a help configuration file. The help file name `help-<lang>.conf` is based on the setting of the `lang` attribute, it defaults to `help.conf` (English). The help file location will depend on whether you want the topics to apply to all users or just the current user.

The help topic files have the same named section format as other configuration files. The `help.conf` files are stored in the same locations and loaded in the same order as other configuration files.

When the `--help` command-line option is specified AsciiDoc loads the appropriate help files and then prints the contents of the section whose name matches the help topic name. If a topic name is not specified `default` is used. You don't need to specify the whole help topic name on the command-line, just enough letters to ensure it's not ambiguous. If a matching help file section is not found a list of available topics is printed.

Writing AsciiDoc documents will be a whole lot more pleasant if you know your favorite text editor. Learn how to indent and reformat text blocks, paragraphs, lists and sentences. Tips for *vim* users follow.

Use the vim `:gq` command to reformat paragraphs. Setting the *textwidth* sets the right text wrap margin; for example:

```
:set textwidth=70
```

To reformat a paragraph:

1. Position the cursor at the start of the paragraph.
2. Type `gq}`.

Execute `:help gq` command to read about the vim gq command.

Tip

- Assign the `gq}` command to the Q key with the `nnoremap Q gq}` command or put it in your `~/.vimrc` file to so it's always available (see the Example `~/.vimrc` file).
- Put `set` commands in your `~/.vimrc` file so you don't have to enter them manually.
- The Vim website (http://www.vim.org) has a wealth of resources, including scripts for automated spell checking and ASCII Art drawing.

The `gq` command can also be used to format bulleted, numbered and callout lists. First you need to set the `comments`, `formatoptions` and `formatlistpat` (see the Example `~/.vimrc` file).

Now you can format simple lists that use dash, asterisk, period and plus bullets along with numbered ordered lists:

1. Position the cursor at the start of the list.
2. Type `gq}`.

Indent whole paragraphs by indenting the fist line with the desired indent and then executing the `gq}` command.

```
" Use bold bright fonts.
set background=dark

" Show tabs and trailing characters.
"set listchars=tab:»·,trail:·,eol:¬
set listchars=tab:»·,trail:·
set list

" Reformat paragraphs and list.
nnoremap <Leader>r gq}

" Delete trailing white space and Dos-returns and to expand tabs to spaces.
nnoremap <Leader>t :set et<CR>:retab!<CR>:%s/[\r \t]\+$//<CR>

autocmd BufRead,BufNewFile *.txt,*.asciidoc,README,TODO,CHANGELOG,NOTES,ABOUT
        \ setlocal autoindent expandtab tabstop=8 softtabstop=2 shiftwidth=2 filetype=asciidoc
        \ textwidth=70 wrap formatoptions=tcqn
        \ formatlistpat=^\\s*\\d\\+\\.\\s\\+\\\\\|^\\s*<\\d\\+>\\s\\+\\\\\|^\\s*[a-zA-Z.]\\.\\s\\+\\\\\|^\\s*[ivxIVX]\\+\\.\\s\\+
        \ comments=s1:/*,ex:*/,://,b:#,:%,:XCOMM,fb:-,fb:*,fb:+,fb:.,fb:>
```

AsciiDoc diagnostic features are detailed in the Diagnostics appendix.

Incorrect character encoding

If you get an error message like `'UTF-8' codec can't decode ...` then you source file contains invalid UTF-8 characters — set the AsciiDoc encoding attribute for the correct character set (typically ISO-8859-1 (Latin-1) for European languages).

Invalid output

AsciiDoc attempts to validate the input AsciiDoc source but makes no attempt to validate the output markup, it leaves that to external tools such as `xmllint(1)` (integrated into `a2x(1)`). Backend validation cannot be hardcoded into AsciiDoc because backends are dynamically configured. The following example generates valid HTML but invalid DocBook (the DocBook `literal` element cannot contain an `emphasis` element):

```
+monospaced text with an _emphasized_ word+
```

Misinterpreted text formatting

You can suppress markup expansion by placing a backslash character immediately in front of the element. The following example suppresses inline monospaced formatting:

```
\+1 for C++.
```

Overlapping text formatting

Overlapping text formatting will generate illegal overlapping markup tags which will result in downstream XML parsing errors. Here's an example:

```
Some *strong markup _that overlaps* emphasized markup_.
```

Ambiguous underlines

A DelimitedBlock can immediately follow a paragraph without an intervening blank line, but be careful, a single line paragraph underline may be misinterpreted as a section title underline resulting in a "closing block delimiter expected" error.

Ambiguous ordered list items

Lines beginning with numbers at the end of sentences will be interpreted as ordered list items. The following example (incorrectly) begins a new list with item number 1999:

```
He was last sighted in
1999. Since then things have moved on.
```

The *list item out of sequence* warning makes it unlikely that this problem will go unnoticed.

Special characters in attribute values

Special character substitution precedes attribute substitution so if attribute values contain special characters you may, depending on the substitution context, need to escape the special characters yourself. For example:

```
$ asciidoc -a 'orgname=Bill &amp; Ben Inc.' mydoc.txt
```

Attribute lists

If any named attribute entries are present then all string attribute values must be quoted. For example:

```
["Desktop screenshot",width=32]
```

You have a number of stand-alone AsciiDoc documents that you want to process as a single document. Simply processing them with a series of `include` macros won't work because the documents contain (level 0) document titles. The solution is to create a top level wrapper document and use the `leveloffset` attribute to push them all down one level. For example:

```
Combined Document Title
=======================

// Push titles down one level.
:leveloffset: 1

include::document1.txt[]

// Return to normal title levels.
:leveloffset: 0

A Top Level Section
-------------------
Lorum ipsum.

// Push titles down one level.
:leveloffset: 1

include::document2.txt[]

include::document3.txt[]
```

The document titles in the included documents will now be processed as level 1 section titles, level 1 sections as level 2 sections and so on.

- Put a blank line between the `include` macro lines to ensure the title of the included document is not seen as part of the last paragraph of the previous document.
- You won't want non-title document header lines (for example, Author and Revision lines) in the included files — conditionally exclude them if they are necessary for stand-alone processing.

You have divided your AsciiDoc document into separate files (one per top level section) which are combined and processed with the following top level document:

```
Combined Document Title
=======================
Joe Bloggs
v1.0, 12-Aug-03

include::section1.txt[]

include::section2.txt[]

include::section3.txt[]
```

You also want to process the section files as separate documents. This is easy because asciidoc(1) will quite happily process `section1.txt`, `section2.txt` and `section3.txt` separately — the resulting output documents contain the section but have no document title.

Use the `-s` (`--no-header-footer`) command-line option to suppress header and footer output, this is useful if the processed output is to be included in another file. For example:

```
$ asciidoc -sb docbook section1.txt
```

asciidoc(1) can be used as a filter, so you can pipe chunks of text through it. For example:

```
$ echo 'Hello *World!*' | asciidoc -s -
<div class="paragraph"><p>Hello <strong>World!</strong></p></div>
```

See the `[footer]` section in the AsciiDoc distribution `xhtml11.conf` configuration file.

If the indentation and layout of the asciidoc(1) output is not to your liking you can:

1. Change the indentation and layout of configuration file markup template sections. The `{empty}` attribute is useful for outputting trailing blank lines in markup templates.
2. Use Dave Raggett's HTML Tidy program to tidy asciidoc(1) output. Example:

   ```
   $ asciidoc -b docbook -o - mydoc.txt | tidy -indent -xml >mydoc.xml
   ```

3. Use the `xmllint(1)` format option. Example:

   ```
   $ xmllint --format mydoc.xml
   ```

The conditional inclusion of DocBook SGML markup at the end of the distribution `docbook45.conf` file illustrates how to support minor DTD variations. The included sections override corresponding entries from preceding sections.

If you've ever tried to send someone an HTML document that includes stylesheets and images you'll know that it's not as straight-forward as exchanging a single file. AsciiDoc has options to create stand-alone documents containing embedded images, stylesheets and scripts. The following AsciiDoc command creates a single file containing embedded images, CSS stylesheets, and JavaScript (for table of contents and footnotes):

```
$ asciidoc -a data-uri -a icons -a toc -a max-width=55em article.txt
```

You can view the HTML file here: http://asciidoc.org/article-standalone.html

Reproducing presentation documents from someone else's source has one major problem: unless your configuration files are the same as the creator's you won't get the same output.

The solution is to create a single backend specific configuration file using the asciidoc(1) `-c` (`--dump-conf`) command-line option. You then ship this file along with the AsciiDoc source document plus the `asciidoc.py` script. The only end user requirement is that they have Python installed (and that they consider you a trusted source). This example creates a composite HTML configuration file for `mydoc.txt`:

```
$ asciidoc -cb xhtml11 mydoc.txt > mydoc-xhtml11.conf
```

Ship `mydoc.txt`, `mydoc-html.conf`, and `asciidoc.py`. With these three files (and a Python interpreter) the recipient can regenerate the HMTL output:

```
$ ./asciidoc.py -eb xhtml11 mydoc.txt
```

The `-e` (`--no-conf`) option excludes the use of implicit configuration files, ensuring that only entries from the `mydoc-html.conf` configuration are used.

Adjust your style sheets to add the correct separation between block elements. Inserting blank paragraphs containing a single non-breaking space character `{nbsp}` works but is an ad hoc solution compared to using style sheets.

You can close off section tags up to level `N` by calling the `eval::[Section.setlevel(N)]` system macro. This is useful if you want to include a section composed of raw markup. The following example includes a DocBook glossary division at the top section level (level 0):

```
ifdef::basebackend-docbook[]

\eval::[Section.setlevel(0)]

+++++++++++++++++++++++++++++++
<glossary>
  <title>Glossary</title>
  <glossdiv>
  ...
  </glossdiv>
</glossary>
+++++++++++++++++++++++++++++++
endif::basebackend-docbook[]
```

Use `xmllint(1)` to check the AsciiDoc generated markup is both well formed and valid. Here are some examples:

```
$ xmllint --nonet --noout --valid docbook-file.xml
$ xmllint --nonet --noout --valid xhtml11-file.html
$ xmllint --nonet --noout --valid --html html4-file.html
```

The `--valid` option checks the file is valid against the document type's DTD, if the DTD is not installed in your system's catalog then it will be fetched from its Internet location. If you omit the `--valid` option the document will only be checked that it is well formed.

The online W3C Markup Validation Service is the defacto standard when it comes to validating HTML (it validates all HTML standards including HTML5).

Block element

An AsciiDoc block element is a document entity composed of one or more whole lines of text.

Inline element

AsciiDoc inline elements occur within block element textual content, they perform formatting and substitution tasks.

Formal element

An AsciiDoc block element that has a BlockTitle. Formal elements are normally listed in front or back matter, for example lists of tables, examples and figures.

Verbatim element

The word verbatim indicates that white space and line breaks in the source document are to be preserved in the output document.

- A new set of quotes has been introduced which may match inline text in existing documents — if they do you'll need to escape the matched text with backslashes.
- The index entry inline macro syntax has changed — if your documents include indexes you may need to edit them.
- Replaced a2x(1) `--no-icons` and `--no-copy` options with their negated equivalents: `--icons` and `--copy` respectively. The default behavior has also changed — the use of icons and copying of icon and CSS files must be specified explicitly with the `--icons` and `--copy` options.

The rationale for the changes can be found in the AsciiDoc `CHANGELOG`.

Note If you want to disable unconstrained quotes, the new alternative constrained quotes syntax and the new index entry syntax then you can define the attribute `asciidoc7compatible` (for example by using the `-a asciidoc7compatible` command-line option).

Read the `README` and `INSTALL` files (in the distribution root directory) for install prerequisites and procedures. The distribution `Makefile.in` (used by `configure` to generate the `Makefile`) is the canonical installation procedure.

AsciiDoc *safe mode* skips potentially dangerous scripted sections in AsciiDoc source files by inhibiting the execution of arbitrary code or the inclusion of arbitrary files.

The safe mode is disabled by default, it can be enabled with the asciidoc(1) `--safe` command-line option.

Safe mode constraints

- `eval`, `sys` and `sys2` executable attributes and block macros are not executed.
- `include::<filename>[]` and `include1::<filename>[]` block macro files must reside inside the parent file's directory.
- `{include:<filename>}` executable attribute files must reside inside the source document directory.
- Passthrough Blocks are dropped.

Warning The safe mode is not designed to protect against unsafe AsciiDoc configuration files. Be especially careful when:

1. Implementing filters.
2. Implementing elements that don't escape special characters.
3. Accepting configuration files from untrusted sources.

AsciiDoc can process UTF-8 character sets but there are some things you need to be aware of:

- If you are generating output documents using a DocBook toolchain then you should set the AsciiDoc `lang` attribute to the appropriate language (it defaults to `en` (English)). This will ensure things like table of contents, figure and table captions and admonition captions are output in the specified language. For example:

  ```
  $ a2x -a lang=es doc/article.txt
  ```

- If you are outputting HTML directly from asciidoc(1) you'll need to set the various `*_caption` attributes to match your target language (see the list of captions and titles in the `[attributes]` section of the distribution `lang-*.conf` files). The easiest way is to create a language `.conf` file (see the AsciiDoc's `lang-en.conf` file).

  You still use the *NOTE*, *CAUTION*, *TIP*, *WARNING*, *IMPORTANT* captions in the AsciiDoc source, they get translated in the HTML

Note output file.

- asciidoc(1) automatically loads configuration files named like `lang-<lang>.conf` where `<lang>` is a two letter language code that matches the current AsciiDoc `lang` attribute. See also Configuration File Names and Locations.

Syntax highlighting is incredibly useful, in addition to making reading AsciiDoc documents much easier syntax highlighting also helps you catch AsciiDoc syntax errors as you write your documents.

The AsciiDoc distribution directory contains a Vim syntax highlighter for AsciiDoc (`./vim/syntax/asciidoc.vim`), you can find the latest version in the online AsciiDoc repository.

Install the highlighter by copying `asciidoc.vim` to your `$HOME/.vim/syntax` directory (create it if it doesn't already exist).

To enable syntax highlighing:

- Put a Vim *autocmd* in your Vim configuration file (see the example vimrc file).
- or execute the Vim command `:set syntax=asciidoc`.
- or add the following line to the end of you AsciiDoc source files:

```
// vim: set syntax=asciidoc:
```

Here is the list of predefined attribute list options:

Option	Backends	AsciiDoc Elements	Description
autowidth	xhtml11, html5, html4	table	The column widths are determined by the browser, not the AsciiDoc *cols* attribute. If there is no *width* attribute the table width is also left up to the browser.
unbreakable	xhtml11, html5	block elements	*unbreakable* attempts to keep the block element together on a single printed page c.f. the *breakable* and *unbreakable* docbook (XSL/FO) options below.
breakable, unbreakable	docbook (XSL/FO)	table, example, block image	The *breakable* options allows block elements to break across page boundaries; *unbreakable* attempts to keep the block element together on a single page. If neither option is specified the default XSL stylesheet behavior prevails.
compact	docbook, xhtml11, html5	bulleted list, numbered list	Minimizes vertical space in the list
footer	docbook, xhtml11, html5, html4	table	The last row of the table is rendered as a footer.
header	docbook, xhtml11, html5, html4	table	The first row of the table is rendered as a header.
pgwide	docbook (XSL/FO)	table, block image, horizontal labeled list	Specifies that the element should be rendered across the full text width of the page irrespective of the current indentation.
strong	xhtml11, html5, html4	labeled lists	Emboldens label text.

The `asciidoc(1) --verbose` command-line option prints additional information to stderr: files processed, filters processed, warnings, system attribute evaluation.

A special attribute named *trace* enables the output of element-by-element diagnostic messages detailing output markup generation to stderr. The *trace* attribute can be set on the command-line or from within the document using Attribute Entries (the latter allows tracing to be confined to specific portions of the document).

- Trace messages print the source file name and line number and the trace name followed by related markup.
- *trace names* are normally the names of AsciiDoc elements (see the list below).

- The trace message is only printed if the *trace* attribute value matches the start of a *trace name*. The *trace* attribute value can be any Python regular expression. If a trace value is not specified all trace messages will be printed (this can result in large amounts of output if applied to the whole document).
- In the case of inline substitutions:
    - The text before and after the substitution is printed; the before text is preceded by a line containing <<< and the after text by a line containing >>>.
    - The *subs* trace value is an alias for all inline substitutions.

Trace names

```
<blockname> block close
<blockname> block open
<subs>
dropped line (a line containing an undefined attribute reference).
floating title
footer
header
list close
list entry close
list entry open
list item close
list item open
list label close
list label open
list open
macro block (a block macro)
name (man page NAME section)
paragraph
preamble close
preamble open
push blockname
pop blockname
section close
section open: level <level>
subs (all inline substitutions)
table
```

Where:

- `<level>` is section level number *0…4*.
- `<blockname>` is a delimited block name: *comment*, *sidebar*, *open*, *pass*, *listing*, *literal*, *quote*, *example*.
- `<subs>` is an inline substitution type: *specialcharacters*,*quotes*,*specialwords*, *replacements*, *attributes*,*macros*,*callouts*, *replacements2*, *replacements3*.

Command-line examples:

1. Trace the entire document.

   ```
   $ asciidoc -a trace mydoc.txt
   ```

2. Trace messages whose names start with `quotes` or `macros`:

   ```
   $ asciidoc -a 'trace=quotes|macros'  mydoc.txt
   ```

3. Print the first line of each trace message:

   ```
   $ asciidoc -a trace mydoc.txt 2>&1 | grep ^TRACE:
   ```

Attribute Entry examples:

1. Begin printing all trace messages:

   ```
   :trace:
   ```

2. Print only matched trace messages:

   ```
   :trace: quotes|macros
   ```

3. Turn trace messages off:

   ```
   :trace!:
   ```

This table contains a list of optional attributes that influence the generated outputs.

Name	Backends	Description
badges	xhtml11, html5	Link badges (*XHTML 1.1* and *CSS*) in document footers. By default badges are omitted (*badges* is undefined). Note The path names of images, icons and scripts are relative path names to the output document not the source document.
data-uri	xhtml11, html5	Embed images using the data: uri scheme.
css-signature	html5, xhtml11	Set a *CSS signature* for the document (sets the *id* attribute of the HTML *body* element). CSS signatures provide a mechanism that allows users to personalize the document appearance. The term *CSS signature* was coined by Eric Meyer.
disable-javascript	xhtml11, html5	If the `disable-javascript` attribute is defined the `asciidoc.js` JavaScript is not embedded or linked to the output document. By default AsciiDoc automatically embeds or links the `asciidoc.js` JavaScript to the output document. The script dynamically generates table of contents and footnotes.
docinfo, docinfo1, docinfo2	All backends	These three attributes control which document information files will be included in the the header of the output file: docinfo — Include `<filename>-docinfo.<ext>` docinfo1 — Include `docinfo.<ext>` docinfo2 — Include `docinfo.<ext>` and `<filename>-docinfo.<ext>` Where `<filename>` is the file name (sans extension) of the AsciiDoc input file and `<ext>` is `.html` for HTML outputs or `.xml` for DocBook outputs. If the input file is the standard input then the output file name is used. The following example will include the `mydoc-docinfo.xml` docinfo file in the DocBook `mydoc.xml` output file: `$ asciidoc -a docinfo -b docbook mydoc.txt` This next example will include `docinfo.html` and `mydoc-docinfo.html` docinfo files in the HTML output file: `$ asciidoc -a docinfo2 -b html4 mydoc.txt`
encoding	html4, html5, xhtml11, docbook	Set the input and output document character set encoding. For example the `--attribute encoding=ISO-8859-1` command-line option will set the character set encoding to `ISO-8859-1`. • The default encoding is UTF-8. • This attribute specifies the character set in the output document. • The encoding name must correspond to a Python codec name or alias. • The *encoding* attribute can be set using an AttributeEntry inside the document header. For example: `:encoding: ISO-8859-1`
hr	html4	Defines the *html4* inter-section horizontal ruler element. By default *html4* top level sections are separated by a horizontal ruler element, undefine this attribute or set it to an empty string if you do not want them. The default *html4* backend value for the *hr* attribute is `<hr>`.
icons	xhtml11, html5	Link admonition paragraph and admonition block icon images and badge images. By default *icons* is undefined and text is used in place of icon images.

Name	Backends	Description
iconsdir	html4, html5, xhtml11, docbook	The name of the directory containing linked admonition icons, navigation icons and the `callouts` sub-directory (the `callouts` sub-directory contains callout number images). *iconsdir* defaults to `./images/icons`. If admonition icons are embedded using the data-uri scheme then the *iconsdir* attribute defaults to the location of the icons installed in the AsciiDoc configuration directory.
imagesdir	html4, html5, xhtml11, docbook	If this attribute is defined it is prepended to the target image file name paths in inline and block image macros.
keywords, description, title	html4, html5, xhtml11	The *keywords* and *description* attributes set the correspondingly named HTML meta tag contents; the *title* attribute sets the HTML title tag contents. Their principle use is for SEO (Search Engine Optimisation). All three are optional, but if they are used they must appear in the document header (or on the command-line). If *title* is not specified the AsciiDoc document title is used.
linkcss	html5, xhtml11	Link CSS stylesheets and JavaScripts. By default *linkcss* is undefined in which case stylesheets and scripts are automatically embedded in the output document.
max-width	html5, xhtml11	Set the document maximum display width (sets the *body* element CSS *max-width* property).
numbered	html4, html5, xhtml11, docbook (XSL Stylesheets)	Adds section numbers to section titles. The *docbook* backend ignores *numbered* attribute entries after the document header.
plaintext	All backends	If this global attribute is defined all inline substitutions are suppressed and block indents are retained. This option is useful when dealing with large amounts of imported plain text.
quirks	xhtml11	Include the `xhtml11-quirks.conf` configuration file and `xhtml11-quirks.css` stylesheet to work around IE6 browser incompatibilities. This feature is deprecated and its use is discouraged — documents are still viewable in IE6 without it.
revremark	docbook	A short summary of changes in this document revision. Must be defined prior to the first document section. The document also needs to be dated to output this attribute.
scriptsdir	html5, xhtml11	The name of the directory containing linked JavaScripts. See HTML stylesheets and JavaScript locations.
sgml	docbook45	The `--backend=docbook45` command-line option produces DocBook 4.5 XML. You can produce the older DocBook SGML format using the `--attribute sgml` command-line option.
stylesdir	html5, xhtml11	The name of the directory containing linked or embedded stylesheets. See HTML stylesheets and JavaScript locations.
stylesheet	html5, xhtml11	The file name of an optional additional CSS stylesheet.
theme	html5, xhtml11	Use alternative stylesheet (see Stylesheets).

Name	Backends	Description
toc	html5, xhtml11, docbook (XSL Stylesheets)	Adds a table of contents to the start of an article or book document. The `toc` attribute can be specified using the `--attribute toc` command-line option or a `:toc:` attribute entry in the document header. The *toc* attribute is defined by default when the *docbook* backend is used. To disable table of contents generation undefine the *toc* attribute by putting a `:toc!:` attribute entry in the document header or from the command-line with an `--attribute toc!` option. **xhtml11 and html5 backends** • JavaScript needs to be enabled in your browser. • The following example generates a numbered table of contents using a JavaScript embedded in the `mydoc.html` output document: `$ asciidoc -a toc -a numbered mydoc.txt`
toc2	html5, xhtml11	Adds a scrollable table of contents in the left hand margin of an article or book document. Use the *max-width* attribute to change the content width. In all other respects behaves the same as the *toc* attribute.
toc-placement	html5, xhtml11	When set to *auto* (the default value) asciidoc(1) will place the table of contents in the document header. When *toc-placement* is set to *manual* the TOC can be positioned anywhere in the document by placing the `toc::[]` block macro at the point you want the TOC to appear. Note If you use *toc-placement* then you also have to define the toc attribute.
toc-title	html5, xhtml11	Sets the table of contents title (defaults to *Table of Contents*).
toclevels	html5, xhtml11	Sets the number of title levels (1..4) reported in the table of contents (see the *toc* attribute above). Defaults to 2 and must be used with the *toc* attribute. Example usage: `$ asciidoc -a toc -a toclevels=3 doc/asciidoc.txt`