Table of Contents

- ChapterÂ 21.Â Macros
- 21.1.Â Inline Macros
    - 21.1.1.Â URLs
    - 21.1.2.Â Internal Cross References
    - 21.1.3.Â Linking to Local Documents
    - 21.1.4.Â Images
    - 21.1.5.Â Comment Lines
- 21.2.Â Block Macros
    - 21.2.1.Â Block Identifier
    - 21.2.2.Â Images
    - 21.2.3.Â Comment Lines
- 21.3.Â System Macros
    - 21.3.1.Â Include Macros
    - 21.3.2.Â Conditional Inclusion Macros
    - 21.3.3.Â Executable system macros
    - 21.3.4.Â Template System Macro
- 21.4.Â Passthrough macros
- 21.5.Â Macro Definitions

ChapterÂ 21.Â Macros

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:

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.

21.1.Â Inline Macros

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

21.1.1.Â URLs

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

- If you donâ€™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:

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` .

21.1.2.Â Internal Cross References

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.

anchor

Used to specify hypertext link targets:

The `<id>` is a unique string that conforms to the output markupâ€™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:

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.

xref

Creates a hypertext link to a document anchor.

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

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 elementâ€™s title text.

Here is an example:

21.1.3.Â Linking to Local Documents

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

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 linkâ€™s displayed text. If `<caption>` is not specified then `<target>` is displayed. Example:

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.

21.1.4.Â Images

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

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:

-
- 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 â€œtooltipâ€ 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:
-
- 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:
- screen thumbnail
- 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.

21.1.5.Â Comment Lines

See comment block macro .

21.2.Â Block Macros

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`* .

21.2.1.Â Block Identifier

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

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

21.2.2.Â Images

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

The block `image` macro has the same macro attributes as itâ€™sinline image macro counterpart.

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:

is equivalent to:

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

Embedding images in XHTML documents

If you define the `data-uri` attribute then images will be embedded in XHTML outputs using thedata 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:

21.2.3.Â Comment Lines

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:

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.

21.3.Â System Macros

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 donâ€™t appear in configuration files. You can however customize the syntax by adding entries to a configuration file `[macros]`* section.

21.3.1.Â Include Macros

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 documentâ€‰â€”â€‰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` macroâ€™s intended use is to include verbatim embedded CSS or scripts into configuration file headers. Example:

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`* .

21.3.2.Â Conditional Inclusion Macros

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:

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

`<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:

Is equivalent to:

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 itâ€™s value is undefined.
- Multiple
- +
- separated attribute names evaluate to defined if all of the attributes is defined, otherwise itâ€™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:

- 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.

21.3.3.Â Executable system macros

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:

21.3.4.Â Template System Macro

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:

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.

21.4.Â Passthrough macros

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.

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

mathematical formulas

Inline and block. The triple-plus passthrough is functionally identical to the *pass* macro but you donâ€™t have to escape `]` characters and you can prefix with quoted attributes in the inline version. Example:

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

plus character instead of a backtick

21.5.Â Macro Definitions

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

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

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

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

`<pattern>`

`<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.

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

Hereâ€™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 macroâ€™s markup template.
- The substituted template replaces the macro reference in the output document.