Metanorma Markdown syntax specification

110:2024, Version 1

110:2024, Version 1

Ribose

Contents

Ribose Asia Limited

Suite 1, 8/F, New Henry House
10 Ice House Street
Central
Hong Kong

copyright@ribose.com
www.ribose.com

Foreword

This document is part of the Metanorma specifications series that defines requirements for document authoring.

Introduction

This document specifies the Markdown syntax for use with Metanorma document processing. It defines both the base CommonMark syntax support and Metanorma-specific extensions necessary for standards document authoring.

0.1. Purpose and scope

This specification:

- Defines the Markdown syntax subset based on CommonMark that Metanorma supports
- Specifies Metanorma-specific extensions to Markdown
- Provides migration guidance from Metanorma AsciiDoc to Metanorma Markdown
- Establishes compatibility requirements between Markdown and existing Metanorma features

0.2. Conformance

A conforming Metanorma Markdown processor SHALL:

- Implement all CommonMark syntax features as specified
- Support all Metanorma extension blocks
- Process document metadata as specified
- Generate identical output to equivalent AsciiDoc input
- Support all specified bibliography features
- Process all specified math expressions
- Handle all cross-reference mechanisms

1. Normative references

There are no normative references in this document.

2. Terms and definitions

For the purposes of this document, the following terms and definitions apply.

2.1. Markdown PREFERRED

lightweight markup language using plain text formatting syntax

2.2. CommonMark PREFERRED

strict specification of Markdown syntax with unambiguous parsing rules

2.3. Metanorma Markdown PREFERRED

extension of CommonMark syntax that supports Metanorma document authoring features

2.4. Frontmatter PREFERRED

metadata section at the beginning of a document using YAML syntax

3. Core document structure

3.1. Document metadata

3.1.1. Purpose

Document metadata provides essential information about the document including its identity, status, and administrative details.

3.1.2. Syntax

EXAMPLE 1 — Basic document metadata
The following example shows the minimal required metadata for a Metanorma document.

```
---
docnumber: "110"
```

```
title: "Metanorma Markdown Syntax"
edition: 1
revdate: 2024-10-27
-
```

Key

1	Triple hyphens delimit the YAML frontmatter block

EXAMPLE 2 — Extended document metadata

This example demonstrates additional metadata fields for document classification and management.

```
---
docnumber: "110"
title: "Metanorma Markdown Syntax"
edition: 1
revdate: 2024-10-27
language: en
doctype: standard
status: draft
committee: Technical
technical-committee: "Syntax and Document Formats"
keywords:
  - markdown
  - syntax
  - metanorma <1>
---
```

Key

<1>	Arrays in YAML are denoted with hyphens

EXAMPLE 3 — Complex metadata with nested structures

This example shows how to represent complex metadata relationships and contributor information.

```
---
docnumber: "110"
title: "Metanorma Markdown Syntax"
edition: 1
revdate: 2024-10-27
contributors:
  - role: author
    name: Jane Smith
    organization: Standards Organization <1>
  - role: editor
    name: John Doe
    organization: Technical Committee
relationships:
  obsoletes: MN-109 <2>
  related:
    - MN-108
    - MN-107
---
```

Key

<1>	Nested structures use indentation to indicate hierarchy
<2>	Single values and arrays can be mixed in the same document

3.1.3. Implementation notes

1. All metadata fields SHALL support UTF-8 encoding
2. Date fields SHALL follow ISO 8601 format
3. Multiple values SHALL be represented as YAML arrays
4. Nested structures SHALL be preserved in processing
5. Unknown metadata fields SHALL be ignored by the processor

3.2. Document title

3.2.1. Purpose

The document title identifies the document and appears as the main heading in the rendered output.

3.2.2. Syntax

EXAMPLE 1 — Title in frontmatter
The title can be specified in the document's frontmatter.

```
---
title: "Document Title"
-
```

Key

1	Title in frontmatter takes precedence over other title declarations

EXAMPLE 2 — Title as level-1 heading
Alternatively, the title can be specified as a level-1 heading in the document body.

```
# Document Title

## First Section
```

Key

1	The hash symbol followed by a space denotes a level-1 heading

EXAMPLE 3 — Title with subtitle

Documents can include both a title and subtitle.

```
---
title: "Document Title"
subtitle: "A comprehensive guide" <1>
---
```

Key

<1>	The subtitle field is optional and can contain formatting

3.2.3. Implementation notes

1. When both frontmatter title and level-1 heading exist, frontmatter SHALL take precedence
2. Title SHALL be encoded in UTF-8
3. Title SHALL be treated as plain text with formatting markers ignored

3.3. Basic text formatting

3.3.1. Purpose

Basic text formatting provides emphasis, strong emphasis, and combinations for inline text styling.

3.3.2. Syntax

EXAMPLE 1 — Basic emphasis and strong emphasis

This example shows the fundamental text formatting syntax.

```
This is *emphasized* and this is **strong**. <1>
This is _also emphasized_ and this is __also strong__. <2>
```

Key

<1>	Asterisks can be used for emphasis and strong emphasis
<2>	Underscores can be used as an alternative to asterisks

EXAMPLE 2 — Combined emphasis

Text can be both emphasized and strong simultaneously.

```
This is ***strong and emphasized***. <1>
This is ___also strong and emphasized___. <2>
```

Key

<1> Triple asterisks combine strong and emphasis

<2> Triple underscores provide an alternative syntax

EXAMPLE 3 — Complex formatting
Formatting can be nested and combined within text.

```
This **strong text contains *emphasis* within it**. <1>
This paragraph has both *emphasized* and **strong** elements. <2>
```

Key

<1> Nested emphasis is processed inside-out

<2> Different formatting styles can be mixed in the same paragraph

3.3.3. Implementation notes

1. Emphasis markers SHALL NOT span multiple paragraphs
2. Nested emphasis SHALL be processed inside-out
3. Emphasis markers within words SHALL be treated as literal characters

3.4. Code spans

3.4.1. Purpose

Code spans mark text as computer code, technical terms, or other literal content.

3.4.2. Syntax

EXAMPLE 1 — Basic code spans
Code spans are used for inline code references.

```
Use the `print()` function. <1>
```

Key

<1> Single backticks denote inline code

EXAMPLE 2 — Code spans with backticks
When the code itself contains backticks, double backticks can be used as delimiters.

```
``Use `backticks` within code`` <1>
```

Key

<1>	Double backticks allow inclusion of single backticks

EXAMPLE 3 — Code spans with attributes

Code spans can include language and other attributes.

```
`{language=ruby} puts "Hello"` <1>
```

Key

<1>	Attributes in curly braces affect processing and display

3.4.3. Implementation notes

1. Code spans SHALL preserve whitespace
2. Markdown syntax within code spans SHALL be treated as literal text
3. Language attributes SHALL be preserved for syntax highlighting

3.5. Links and cross-references

3.5.1. Purpose

Links and cross-references connect document sections and external resources.

3.5.2. Syntax

EXAMPLE 1 — External links

External links connect to resources outside the document.

```
Visit [Metanorma](https://www.metanorma.org "Metanorma Homepage"). <1>
See <https://www.metanorma.org> for more information. <2>
```

Key

<1>	Links can include optional titles in quotes
<2>	URLs can be automatically linked using angle brackets

EXAMPLE 2 — Internal cross-references

Internal cross-references link to sections within the document.

```
See [Section 3.2](#section-3-2) for details. <1>
Refer to [Terms and definitions](#terms-and-definitions). <2>
```

Key

<1>	Section references use the section's ID

<2> IDs are automatically generated from heading text

EXAMPLE 3 — Reference-style links

Reference-style links separate the link text from the URL definition.

```
This specification uses [CommonMark][cm] syntax.
See the [Metanorma documentation][mn] for more.

[cm]: https://commonmark.org "CommonMark Spec" <1>
[mn]: https://www.metanorma.org/docs/ "Documentation" <2>
```

Key

<1> Link references can be defined anywhere in the document

<2> References support optional titles

3.5.3. Implementation notes

1. Internal cross-references SHALL be validated during processing
2. External links SHALL be checked for well-formed URLs
3. Reference-style link definitions SHALL be collected and processed globally

3.6. Mathematics

3.6.1. Purpose

Mathematical expressions in both inline and display modes.

3.6.2. Syntax

EXAMPLE 1 — Inline mathematics

Inline math expressions are embedded within text.

```
The equation $E = mc^2$ shows the relationship. <1>
```

Key

<1> Single dollar signs denote inline math mode

EXAMPLE 2 — Display mathematics

Display math appears as separate blocks.

```
$$
\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} <1>
$$
```

Key

<1> Double dollar signs create display math blocks

EXAMPLE 3 — Numbered equations
Equations can be numbered and referenced.

```
$$
\begin{equation}
  \label{eq:1} <1>
  F = ma
\end{equation}
$$
```

Key

<1> Labels enable equation referencing

3.6.3. Implementation notes

1. LaTeX math syntax SHALL be supported
2. Equation numbers SHALL be automatically generated if not specified
3. Cross-references SHALL be supported

3.7. Block elements

3.7.1. Purpose

Block elements structure document content into distinct sections.

3.7.2. Syntax

EXAMPLE 1 — Basic blocks
Paragraphs are separated by blank lines.

```
This is the first paragraph.

This is the second paragraph. <1>
```

Key

<1> Blank lines separate paragraphs

EXAMPLE 2 — Blocks with attributes
Blocks can have attributes that affect their processing.

```
{.note} <1>
This is a note block.
```

```
{#custom-id .warning} <2>
This is a warning block.
```

Key

<1>	Class attributes affect block styling
<2>	IDs enable block referencing

EXAMPLE 3 — Complex blocks

Blocks can contain other blocks and inline elements.

```
{.requirement #req-1}
This requirement block contains:

1. Ordered list
2. With *emphasized* text
3. And `code spans` <1>
```

Key

<1>	Blocks can contain mixed content types

3.7.3. Implementation notes

1. Block attributes SHALL be parsed before block content
2. Nested blocks SHALL maintain proper hierarchy
3. Block types SHALL determine valid attribute sets

3.8. Lists

3.8.1. Purpose

Lists organize content in ordered, unordered, and definition formats.

3.8.2. Syntax

EXAMPLE 1 — Basic list types

Lists can be ordered or unordered with nested structures.

```
* Unordered item 1
* Unordered item 2 <1>
  * Nested item 2.1 <2>
  * Nested item 2.2
* Unordered item 3

1. Ordered item 1 <3>
2. Ordered item 2
   1. Nested item 2.1
   2. Nested item 2.2
3. Ordered item 3
```

Key

<1>	Asterisks denote unordered list items
<2>	Two spaces indent creates nested lists
<3>	Numbers with periods create ordered lists

EXAMPLE 2 — Definition lists

Definition lists associate terms with their definitions.

```
Term 1
: Definition 1 <1>
: Another definition 1 <2>

Term 2
: Definition 2
```

Key

<1>	Colon indicates a definition
<2>	Terms can have multiple definitions

EXAMPLE 3 — Lists with attributes

Lists can have attributes affecting their appearance and behavior.

```
{.checklist} <1>
* [ ] Task 1 <2>
* [x] Task 2
  1. Subtask 2.1
  2. Subtask 2.2
* [ ] Task 3
```

Key

<1>	Class attributes modify list behavior
<2>	Checkbox syntax for task lists

3.8.3. Implementation notes

1. List markers SHALL be consistent within the same level
2. Indentation SHALL be preserved for nested lists
3. Definition lists SHALL support multiple definitions per term

3.9. Code blocks

3.9.1. Purpose

Code blocks present source code, technical content, or other preformatted text.

3.9.2. Syntax

EXAMPLE 1 — Fenced code blocks

Fenced code blocks use triple backticks with optional language specification.

````
```ruby <1>
def hello_world
  puts "Hello, world!"
end
``` <2>
````

Key

<1>	Language identifier enables syntax highlighting
<2>	Triple backticks delimit the code block

EXAMPLE 2 — Code blocks with attributes

Code blocks can have additional attributes for processing and display.

````
```{.ruby #example-1 .numbered} <1>
def hello_world
  puts "Hello, world!"
end
```
````

Key

<1>	Attributes in curly braces affect block processing

EXAMPLE 3 — Indented code blocks

Code blocks can also be created by indentation.

```
    def hello_world <1>
      puts "Hello, world!"
    end
```

Key

<1>	Four spaces indent creates a code block

3.9.3. Implementation notes

1. Both fenced and indented code blocks SHALL be supported
2. Language identifiers SHALL be preserved for syntax highlighting
3. Line numbers SHALL be added when specified
4. Custom attributes SHALL be preserved

3.10. Tables

3.10.1. Purpose

Tables organize data in rows and columns with optional formatting.

3.10.2. Syntax

EXAMPLE 1 — Basic tables

Simple tables use pipe characters to separate columns.

```
| Header 1 | Header 2 | <1>
|----------|----------| <2>
| Cell 1   | Cell 2   |
| Cell 3   | Cell 4   |
```

Key

<1>	Header row is separated by pipes
<2>	Delimiter row indicates column alignment

EXAMPLE 2 — Aligned columns

Column alignment is specified in the delimiter row.

```
| Left | Center | Right | <1>
|:-----|:------:|------:| <2>
| 1    | 2      | 3     |
| 4    | 5      | 6     |
```

Key

<1>	Headers establish column count
<2>	Colons in delimiter row specify alignment

EXAMPLE 3 — Complex tables with attributes

Tables can have attributes and contain formatted content.

```
{.bordered #table-1 title="Sample Table"} <1>
| Function | Description | Example |
|----------|-------------|---------|
| `sum()`  | Adds numbers | `sum(1, 2)` |
| `avg()`  | Calculates average | `avg([1, 2])` |
```

Key

<1>	Table attributes affect presentation and processing

3.10.3. Implementation notes

1. Table alignments SHALL be preserved
2. Header rows SHALL be distinguished in output

3. Tables SHALL support block-level elements in cells
4. Table captions and attributes SHALL be preserved

4. Metanorma extensions

4.1. Requirement blocks

4.1.1. Purpose

Requirement blocks specify normative requirements, recommendations, and permissions.

4.1.2. Syntax

EXAMPLE 1 — Basic requirement
Simple requirement blocks state single requirements.

```
::: requirement <1>
This system SHALL support UTF-8 encoding. <2>
:::
```

Key

<1>	Triple colons denote extension blocks
<2>	Requirements use normative language

EXAMPLE 2 — Requirement with attributes
Requirements can have identifiers and specific obligation levels.

```
::: requirement{#req-1 level="shall"} <1>
The processor SHALL implement all CommonMark features. <2>
:::
```

Key

<1>	Attributes specify requirement properties
<2>	Content states the normative requirement

EXAMPLE 3 — Complex requirement
Requirements can include structured content and metadata.

```
::: requirement{#req-2}
level:: shall <1>
inherit:: req-1 <2>
classification:: technical <3>
```

```
---
The system SHALL:

1. Parse all valid inputs
2. Report errors clearly
3. Maintain backward compatibility
:::
```

Key

<1>	Requirement level specified as metadata
<2>	Inheritance indicates requirement dependencies
<3>	Classification aids in requirement management

4.1.3. Implementation notes

1. Requirements SHALL support nested content
2. Requirement IDs SHALL be unique
3. Inheritance SHALL be validated
4. Classification SHALL affect rendering

4.2. Admonitions

4.2.1. Purpose

Admonitions highlight important information with specific semantic meaning.

4.2.2. Syntax

EXAMPLE 1 — Basic admonitions

Simple admonitions use the triple exclamation mark syntax.

```
!!! note <1>
    This is a note. <2>

!!! warning
    This is a warning.
```

Key

<1>	Admonition type follows the markers
<2>	Content is indented by 4 spaces

EXAMPLE 2 — Admonitions with titles

Admonitions can have custom titles.

```
!!! important "Critical Information" <1>
```

```
    This must be considered.
```

Key

<1>	Custom title in quotes after type

EXAMPLE 3 — Complex admonitions
Admonitions can contain structured content and attributes.

```
!!! note{#note-1 .special} <1>
    This note contains:

    * Important points
    * Critical information
    * Key considerations
```

Key

<1>	Attributes customize admonition behavior

4.2.3. Implementation notes

1. Standard admonition types SHALL be supported
2. Custom admonition types MAY be defined
3. Nested content SHALL be supported
4. Attributes SHALL be preserved

4.3. Bibliography

4.3.1. Purpose

Bibliography entries provide structured reference information.

4.3.2. Syntax

EXAMPLE 1 — Basic citation
Citations reference bibliography entries.

```
[@reference-key] <1>
```

Key

<1>	Citation key in square brackets with @ prefix

EXAMPLE 2 — Citation with prefix and locator
Citations can include additional context.

```
[see @smith2024, p. 23-45] <1>
```

Key

<1>	Prefixes and page numbers add citation context

EXAMPLE 3 — Bibliography entry
Bibliography data is defined in YAML format.

```yaml
---
references:
- id: smith2024 <1>
  type: book
  author:
    - family: Smith
      given: John
  title: Example Book
  publisher: Publisher Name
  year: 2024
---
```

Key

<1>	Entry ID matches citation keys

4.3.3. Implementation notes

1. Multiple citation styles SHALL be supported
2. Bibliography data SHALL be validated
3. Citations SHALL be linked to entries
4. Custom CSL styles SHALL be supported

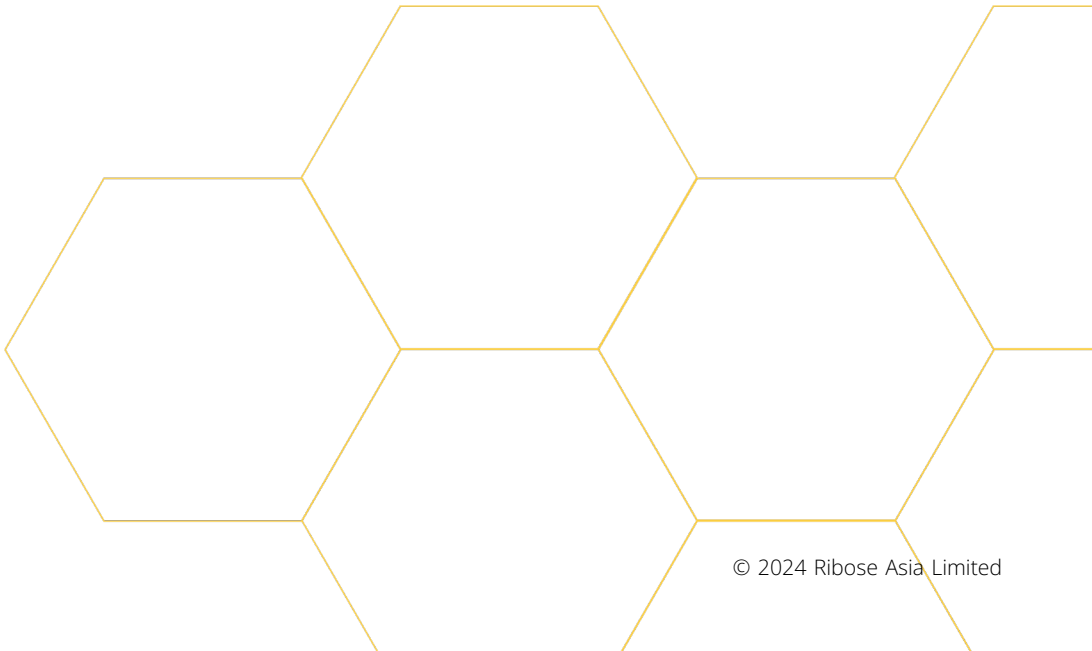
Annex A
(normative)

Complete example document

EXAMPLE — Full document example: This example demonstrates a complete Metanorma Markdown document.

```
---
title: "Sample Metanorma Document"
docnumber: "MN-SAMPLE-1"
date: 2024-10-28
type: standard
status: draft
---

# Sample Metanorma document

## Introduction

This document demonstrates the Metanorma Markdown syntax.

### Purpose

The purpose is to show syntax examples.

## Technical requirements

::: requirement{#req-1}
level:: shall
The system SHALL support all specified features.
:::

### Mathematical expressions

The formula $E = mc^2$ shows mass-energy equivalence.

$$
F = ma
$$

## References

See [@smith2024] for background.

---
references:
```

```
- id: smith2024
  type: article
  author:
    - family: Smith
      given: J.
  title: Sample Article
  journal: Journal Name
  year: 2024
---
```



Bibliography

[1] CommonMark Specification 0.30

[2] Pandoc User's Guide

[3] GitHub Flavored Markdown Specification