Metanorma document revision format specification

113:2025, Version 1

113:2025, Version 1

Ribose

Contents

Warning for Drafts

This document is not a Ribose Standard. It is distributed for review and comment, and is subject to change without notice and may not be referred to as a Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Ribose Asia Limited

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

copyright@ribose.com
www.ribose.com

Introduction

This specification defines the Metanorma document revision format, a structured data format for tracking and documenting changes to technical documents across multiple revisions and editions.

Document revision history is a critical component of standards development and technical documentation, providing transparency and traceability for changes over time. A well-structured revision history enables:

- Clear documentation of when and why changes were made
- Attribution of changes to specific contributors or organizations
- Precise identification of affected document sections or components
- Classification of changes by type, severity, or other attributes
- Consistent presentation of revision information across different output formats

The Metanorma document revision format addresses these needs with:

1. standardized data models for revision history that work consistently across formats
2. support for both simple and complex revision tracking requirements
3. flexible amendment location referencing for precise change documentation
4. extensible classification system for categorizing types of changes
5. support for serialization in both YAML and XML formats

This format enables standards developers and document maintainers to capture comprehensive revision information that can be consistently represented across different document renderings while preserving both the structural integrity and semantic meaning of the revision history.

1. Scope

This document defines a revision history format called the “Metanorma document revision format” for documenting changes made to technical documents over time.

It specifies the data models and formats for representing revision history in Metanorma documents, supporting both YAML and XML serialization.

It is a flexible, format-independent revision model, allowing defined revision histories to be consistently presented across various output formats including PDF, HTML, and Word.

2. Terms and definitions

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

2.1. revision history PREFERRED

structured collection of revisions documenting changes made to a document over time

2.2. revision PREFERRED

set of related changes made to a document at a specific point in time, typically associated with a particular edition or version

2.3. edition PREFERRED

version number or identifier associated with a specific revision of a document

2.4. amendment PREFERRED

specific change or set of related changes made to the document content as part of a revision

EXAMPLE: “Updated clause 4.0 and 12.0. Populated clause 9.0.”

2.5. location PREFERRED

specific part of the document affected by an amendment

EXAMPLE: A location can be a clause (“clause 4.0”), a whole document (“whole”), or specific components like figures, tables, or annexes.

2.6. classification PREFERRED

categorization of an amendment according to specific attributes such as type or severity

EXAMPLE: Classifications might include severity (“major”, “minor”) or type (“editorial”, “technical”).

2.7. contributor PREFERRED

person or organization responsible for making amendments in a revision

2.8. date information PREFERRED

temporal information associated with a revision, including the specific type of date (e.g., published, updated)

3. Core models

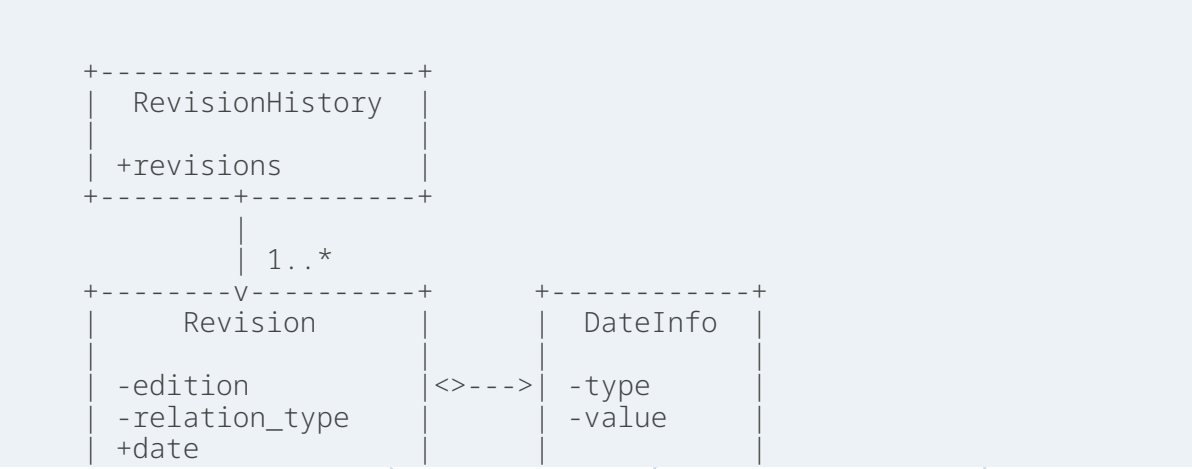
3.1. General

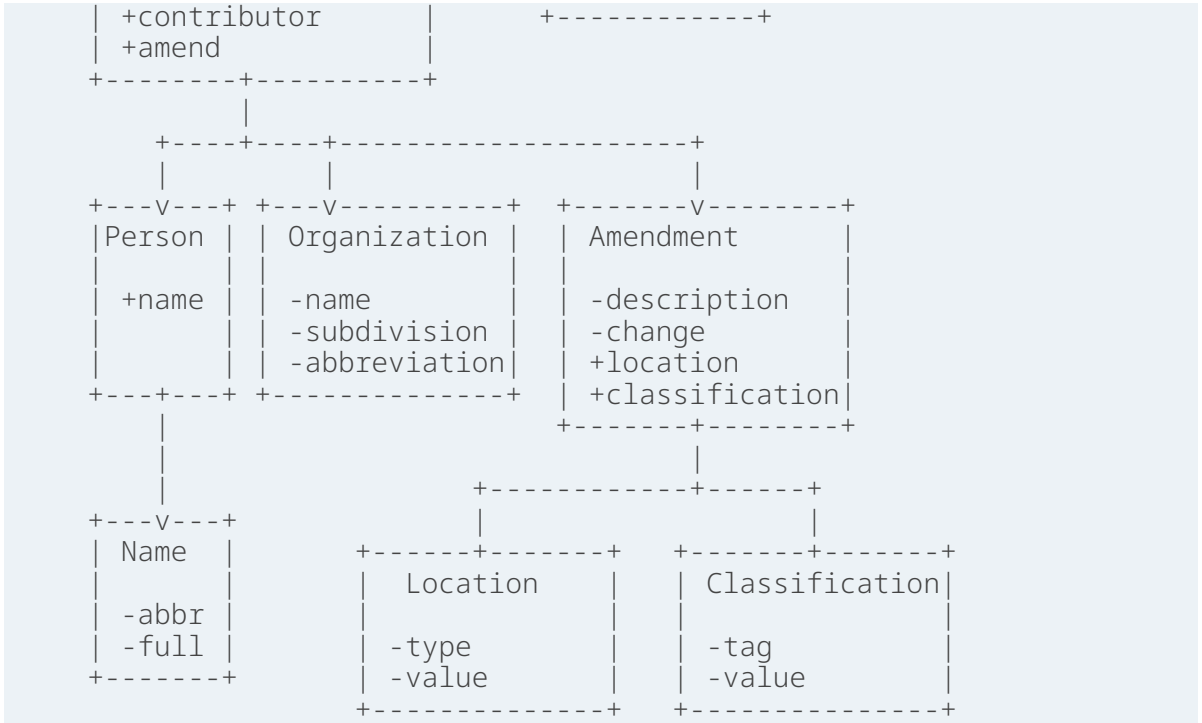
The Metanorma document revision format consists of the following core models:

- `RevisionHistory`: The main container for all revisions
- `Revision`: Represents a single revision entry
- `Amendment`: Represents a specific change or set of changes
- `Location`: Specifies the part of the document affected
- `Classification`: Categorizes an amendment
- `Contributor`: Identifies who made the changes
- `DateInfo`: Provides date information for a revision

The revision format follows this hierarchical structure:

FIGURE 1





3.2. RevisionHistory

The RevisionHistory class is the main container for all revisions.

FIGURE 2

```
class RevisionHistory
  attribute :revisions, Revision, collection: true
end
```

3.3. Revision

The Revision class represents a single revision entry, which may include multiple amendments.

FIGURE 3

```
class Revision
  attribute :date, DateInfo, collection: true
  attribute :edition, :string
  attribute :contributor, Contributor, collection: true
  attribute :amend, Amendment, collection: true
```

```
    attribute :relation_type, Amendment, collection: true
  end
```

The `date` attribute is a collection of DateInfo objects that provide information about when the revision occurred.

The `edition` attribute is a string that represents the version number or identifier for the revision.

The `contributor` attribute is a collection of Contributor objects representing the people or organizations responsible for the revision.

The `amend` attribute is a collection of Amendment objects representing the changes made in the revision.

The `relation_type` attribute is a collection of Amendment objects representing the relationship between this revision and others, if applicable.

3.4. DateInfo

The DateInfo class represents date information associated with a revision.

FIGURE 4

```
class DateInfo
  attribute :type, :string
  attribute :value, :string
end
```

The `type` attribute is a string indicating the type of date, such as “published” or “updated”.

The `value` attribute is a string containing the date value.

3.5. Contributor

The Contributor class represents a person or organization responsible for a revision.

FIGURE 5

```
class Contributor
  attribute :person, Person
  attribute :organization, Organization
end
```

3.6. Person

The Person class represents an individual contributor.

FIGURE 6

```
class Person
  attribute :name, Name
end
```

3.7. Organization

The Organization class represents an organizational contributor.

FIGURE 7

```
class Organization
  attribute :name, :string
  attribute :subdivision, :string
  attribute :abbreviation, :string
end
```

The `name` attribute is a string containing the organization's name.

The `subdivision` attribute is an optional string specifying a department or division within the organization.

The `abbreviation` attribute is an optional string providing an abbreviated form of the organization's name.

3.8. Name

The Name class represents a person's name.

FIGURE 8

```
class Name
  attribute :abbreviation, :string
  attribute :completename, :string
end
```

The `abbreviation` attribute is a string containing the person's initials or abbreviation.

The `completename` attribute is a string containing the person's full name.

3.9. Amendment

The Amendment class represents a specific change or set of changes made to the document.

FIGURE 9

```
class Amendment
  attribute :description, :string
  attribute :location, Location, collection: true
  attribute :classification, Classification, collection: true
  attribute :change, :string, default: -> { "modify" }
end
```

The `description` attribute is a string describing the changes made.

The `location` attribute is a collection of Location objects specifying the parts of the document affected by the amendment.

The `classification` attribute is a collection of Classification objects categorizing the amendment.

The change attribute is a string indicating the type of change, defaulting to "modify".

3.10. Location

The Location class specifies the part of the document affected by an amendment.

FIGURE 10

```
class Location
  VALID_TYPES = %w[
    section clause part paragraph chapter page line table
annex figure
    example note formula list time anchor whole
  ].freeze

  attribute :value, :string
  attribute :type, :string, values: VALID_TYPES
end
```

The `value` attribute is a string specifying the specific value of the location, such as "4.0" for clause 4.0, or "B" for Annex B.

The `type` attribute is a string indicating the type of location, such as “clause”, “annex”, or “whole”.

3.11. Classification

The Classification class categorizes an amendment according to specific attributes.

FIGURE 11

```
class Classification
  attribute :tag, :string
  attribute :value, :string
end
```

The `tag` attribute is a string indicating the classification category, such as “severity” or “type”.

The `value` attribute is a string specifying the classification value, such as “major” or “editorial”.

4. Document format serialization

4.1. YAML format

The Metanorma document revision format can be serialized as YAML, which is often used in configuration files and is human-readable.

EXAMPLE

```
revisions:
  - date:
    - type: published
      value:  2012-04
    edition: 1.0.0
    contributor:
    - person:
        name:
          abbreviation: JMS
          completename: J. Michael Straczynski
    amend:
      - description: Approved edition of S-102
  - date:
    - type: updated
      value:  2017-03
```

```
    edition: 2.0.0
    contributor:
    - organization:
        name: S-102PT
    amend:
      - description: |
          Updated clause 4.0 and 12.0.

          Populated clause 9.0.
        location:
          - type: clause
            value: 4.0
          - type: clause
            value: 12.0
          - type: clause
            value: 9.0
          - type: whole
        classification:
          - tag: severity
            value: major
          - tag: type
            value: editorial
      - description: Deleted contents of Annex B in preparation for
updated S-100 Part 10C guidance.
        location:
          - type: annex
            value: B
```

4.2. XML format

The same revision history can also be serialized as XML, which is commonly used for structured data interchange.

EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
<revision-history>
  <revision>
    <date>
      <type>published</type>
      <value>2012-04</value>
    </date>
    <edition>1.0.0</edition>
    <contributor>
      <person>
        <name>
          <abbreviation>JMS</abbreviation>
          <completename>J. Michael Straczynski</completename>
        </name>
      </person>
    </contributor>
    <amend>
      <description>Approved edition of S-102</description>
    </amend>
  </revision>
  <revision>
    <date>
```

```
      <type>updated</type>
      <value>2017-03</value>
    </date>
    <edition>2.0.0</edition>
    <contributor>
      <organization>
        <name>S-102PT</name>
      </organization>
    </contributor>
    <amend>
      <description>Updated clause 4.0 and 12.0.

Populated clause 9.0.</description>
      <location type="clause">4.0</location>
      <location type="clause">12.0</location>
      <location type="clause">9.0</location>
      <location type="whole"></location>
      <classification>
        <tag>severity</tag>
        <value>major</value>
      </classification>
      <classification>
        <tag>type</tag>
        <value>editorial</value>
      </classification>
    </amend>
    <amend>
      <description>Deleted contents of Annex B in preparation for
updated S-100 Part 10C guidance.</description>
      <location type="annex">B</location>
    </amend>
  </revision>
</revision-history>
```

5. Usage

5.1. Parsing revision history

5.1.1. From YAML

The following Ruby code demonstrates how to parse a revision history from a YAML string:

FIGURE 12

```ruby
require 'revix'

# Parse from YAML string
```

```ruby
yaml_content = File.read('revision_history.yaml')
history = Revix::RevisionHistory.from_yaml(yaml_content)

# Access revision data
history.revisions.each do |revision|
  puts "Edition: #{revision.edition}"

  revision.date.each do |date|
    puts "Date: #{date.value} (#{date.type})"
  end

  revision.contributor.each do |contributor|
    if contributor.person
      puts "Contributor:
#{contributor.person.name.completename}
(#{contributor.person.name.abbreviation})"
    elsif contributor.organization
      puts "Contributor: #{contributor.organization.name}"
    end
  end

  revision.amend.each do |amendment|
    puts "Amendment: #{amendment.description}"

    amendment.location&.each do |location|
      if location.value
        puts "  Location: #{location.type}=#{location.value}"
      else
        puts "  Location: #{location.type}"
      end
    end

    amendment.classification&.each do |classification|
      puts "  Classification: #{classification.tag} =
#{classification.value}"
    end
  end
end
```

5.1.2. From XML

The following Ruby code demonstrates how to parse a revision history from an XML string:

FIGURE 13

```ruby
require 'revix'

# Parse from XML string
xml_content = File.read('revision_history.xml')
history = Revix::RevisionHistory.from_xml(xml_content)

# Access revision data (same as with YAML)
```

```
# ...
```

5.2. Creating revision history

The following Ruby code demonstrates how to create a revision history programmatically:

FIGURE 14

```
require 'revix'

# Create a revision history object
history = Revix::RevisionHistory.new(revisions: [
  Revix::Revision.new(
    date: [Revix::DateInfo.new(type: "published", value:
"2012-04")],
    edition: "1.0.0",
    contributor: [
      Revix::Contributor.new(
        person: Revix::Person.new(
          name: Revix::Name.new(
            abbreviation: "JMS",
            completename: "J. Michael Straczynski"
          )
        )
      )
    ],
    amend: [
      Revix::Amendment.new(
        description: "Approved edition of S-102",
        location: [
          Revix::Location.new(type: "clause", value: "4.0"),
          Revix::Location.new(type: "whole")
        ]
      )
    ]
  )
])
```

5.3. Serializing revision history

5.3.1. To YAML

The following Ruby code demonstrates how to serialize a revision history to YAML:

FIGURE 15

```ruby
# Serialize to YAML
yaml_content = history.to_yaml
File.write('revision_history.yaml', yaml_content)
```

5.3.2. To XML

The following Ruby code demonstrates how to serialize a revision history to XML:

FIGURE 16

```ruby
# Serialize to XML
xml_content = history.to_xml
File.write('revision_history.xml', xml_content)
```

6. Best practices

6.1. Amendment descriptions

Amendment descriptions should be:

- Clear and concise, focusing on what changed
- Specific about the nature of the change
- Written in past tense (e.g., “Updated clause 4.0”)
- Grouped logically when multiple related changes occurred

6.2. Location specificity

Locations should be as specific as possible:

- Use the most precise location type available
- Include specific values where applicable
- Use “whole” only when the entire document is affected
- List multiple locations for changes that span different sections

6.3. Change classification

Classifications help users understand the impact and nature of changes:

- Use consistent classification tags across revisions
- Include severity classifications (e.g., major, minor) for significant changes
- Include type classifications (e.g., editorial, technical) to indicate the nature of changes
- Consider adding purpose classifications (e.g., clarification, correction) when relevant

6.4. Date information

Date information should:

- Include a clear type indicator (published, updated, etc.)
- Follow a consistent date format across all revisions
- Provide the appropriate level of precision (year, year-month, or full date)

6.5. Multiple contributors

When multiple contributors are involved:

- List all significant contributors to the revision
- Use organization contributors for changes made by groups
- Use person contributors for individual attributions
- Consider using both when an individual contributed on behalf of an organization

Annex A
(normative)

Common revision patterns

A.1. Initial publication

The initial publication of a document typically has a simple revision entry:

FIGURE A.1

```
revisions:
  - date:
    - type: published
      value: 2023-01-15
    edition: 1.0.0
    contributor:
    - organization:
        name: Technical Committee
    amend:
      - description: Initial release
        location:
          - type: whole
```

A.2. Editorial corrections

Minor editorial corrections are typically documented as follows:

FIGURE A.2

```
revisions:
  - date:
    - type: corrected
      value: 2023-02-20
    edition: 1.0.1
    contributor:
    - organization:
```

```
          name: Editorial Team
      amend:
        - description: Corrected typographical errors
          location:
            - type: clause
              value: 3.2
            - type: clause
              value: 5.1
          classification:
            - tag: severity
              value: minor
            - tag: type
              value: editorial
```

A.3. Technical amendments

Significant technical changes are typically documented with detailed classifications:

FIGURE A.3

```
revisions:
  - date:
    - type: updated
      value: 2023-06-10
    edition: 1.1.0
    contributor:
    - organization:
        name: Technical Committee
    amend:
      - description: Updated calculation method for
performance metrics
        location:
          - type: clause
            value: 8.3
          - type: formula
            value: 12
        classification:
          - tag: severity
            value: major
          - tag: type
            value: technical
          - tag: purpose
            value: improvement
```

A.4. Multiple amendments in one revision

A single revision often includes multiple amendments:

FIGURE A.4

```
revisions:
  - date:
    - type: updated
      value: 2023-09-05
    edition: 2.0.0
    contributor:
    - organization:
        name: Joint Working Group
    amend:
      - description: Restructured document organization
        location:
          - type: whole
        classification:
          - tag: severity
            value: major
          - tag: type
            value: structural
      - description: Added new section on security
considerations
        location:
          - type: clause
            value: 10
        classification:
          - tag: severity
            value: major
          - tag: type
            value: addition
      - description: Updated all diagrams for clarity
        location:
          - type: figure
        classification:
          - tag: severity
            value: minor
          - tag: type
            value: editorial
```

A.5. Multiple contributors

Revisions with multiple contributors can be documented as follows:

FIGURE A.5

```
revisions:
  - date:
```

```yaml
    - type: updated
      value: 2023-11-12
  edition: 2.1.0
  contributor:
  - person:
      name:
        abbreviation: JD
        completename: Jane Doe
  - person:
      name:
        abbreviation: JS
        completename: John Smith
  - organization:
      name: Expert Panel
      subdivision: Security Group
  amend:
    - description: Enhanced security protocols
      location:
        - type: clause
          value: 12.3
        - type: annex
          value: A
      classification:
        - tag: severity
          value: major
        - tag: type
          value: technical
```