Label Auto-assignment Definition Language (LADL) specification

112:2025, Version 1

112:2025, Version 1
© 2025 Ribose Asia Limited. All rights reserved

Ribose

Contents

© 2025 Ribose Asia Limited

© 2025 Ribose Asia Limited

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.

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from the address below.

Ribose Asia Limited

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

copyright@ribose.com
www.ribose.com

 © 2025 Ribose Asia Limited

Introduction

This specification defines the Label Auto-assignment Definition Language (LADL), a declarative modeling language for defining automatic label assignment requirements across technical documentation and publishing systems.

NOTE "LADL" is pronounced as "ladel".

Label auto-assignment is a critical component of structured documentation, enabling consistent referencing and organization of content elements throughout a document. The act is to automatically create correct Label objects that attach to the content element. A Labeller creates a string label from a Label object, depending on the usage context (e.g., the same item can be labelled differently in different parts of the content hierarchy).

LADL addresses these challenges with:

1. platform-independent labeling model that works consistently across rendering engines
2. built-in support for international labeling systems (including CJK)
3. hierarchical labeling with contextual awareness and scope binding
4. clear separation between semantic structure and presentation formatting
5. support for transformations to CSS, Word Styles, and XSLT labellers
6. string pattern interpolation for flexible label construction

LADL enables document architects and standards developers to define once and apply everywhere, ensuring that labeling schemes work predictably across all outputs while preserving both semantic structure and visual presentation requirements.

© 2025 Ribose Asia Limited

1. Scope

This document defines a label definition language called the “Label Auto-assignment Definition Language” (LADL) for defining auto-labeling schemes.

It also defines requirements for auto-labeling systems that can perform the necessary labeling activity based on the defined schemes.

It is a platform-independent, language-agnostic labeling model, allowing defined schemes to be deployed towards various rendering formats.

© 2025 Ribose Asia Limited

2. Terms and definitions

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

2.1. label PREFERRED

machine-readable object that attaches to a content element for consistent and unambiguous reference of information in an information container

EXAMPLE: A label like “1.1.2” indicates it is the “first clause in document, first subclause, second subsubclause”, while “Figure A.2” means it is the second image in Annex A.

2.2. labeller PREFERRED

numeric or alphanumeric variable that tracks sequence position within a document context

2.3. sequence PREFERRED

ordered progression of values following defined rules for generation and formatting

2.4. labeling format PREFERRED

rules for representing a labeller value in a particular alphanumeric system

EXAMPLE: Arabic numerals, Roman numerals, alphabetic characters.

2.5. label context PREFERRED

document boundary within which a labeller operates, defining when the labeller resets or continues

2.6. string interpolation PREFERRED

process of substituting variable placeholders in a pattern with actual values to construct a rendered label

© 2025 Ribose Asia Limited

2.7. label template PREFERRED

template combining labeller values and literal characters to form string labels

EXAMPLE: “Figure %n” is a template that generates labels like “Figure 1”, “Figure 2”, etc.

2.8. hierarchical labeling PREFERRED

labeling system where elements inherit context from parent elements

EXAMPLE: In a labeling system, “1.1.2” can represent the locality of “section 1, subsection 1, clause 2”.

2.9. labeller PREFERRED

component that assigns labels to content elements based on their position in the document structure and labeling context, and generates a human-readable label in string form from a label object

2.10. reference PREFERRED

general term for pointing to specific content elements using labels, enabling navigation, identification, and citation

2.11. local reference PREFERRED

reference to content within the same context or document section, using labels to provide precise location information

2.12. global reference PREFERRED

reference to content across different contexts or document sections, using labels to provide unambiguous identification regardless of location

2.13. citation PREFERRED

formal reference to content using labels, enabling precise identification in both internal and external referencing systems

© 2025 Ribose Asia Limited

3. Principles and requirements

3.1. Purpose

Labels serve as the foundation for consistent and unambiguous reference of information within documents.

Their primary purpose is to enable:

- Precise navigation through hierarchical document structures
- Unambiguous identification of content elements regardless of their position
- Citation systems that work both internally and externally to the document

Labels provide machine-readable identifiers that can be rendered in human-readable form, allowing both automated systems and human readers to locate and reference specific content with precision.

A consistent, predictable auto-labeling language enables consistent, predictable auto-labeling with the following benefits:

- Preserving semantic relationships between labeled elements
- Adapting to different display requirements by format
- Supporting internationalization requirements
- Allowing for hierarchical labeling schemes
- Enabling flexible formatting of labels
- Managing labeller scopes across document boundaries
- Ability to deploy to different rendering engines, such as CSS, Word Styles, and XSLT.

3.2. Principles

LADL follows these fundamental design principles:

Semantics and presentation separation	Numbers represent both position in a sequence and visual labeling. These concerns must be separable for proper processing.
Context awareness	Labeling systems must understand their position within content hierarchies.
Format-independence	The model must define labeling in a way that can be consistently applied across multiple forms of media.
Implementation-free	The defined labeling system must not depend on any specific rendering engine or language outside of LADL.
Internationalization support	Non-Latin labeling systems, including right-to-left languages, must be fully supported.

© 2025 Ribose Asia Limited

Extensible	Support for complex labeling schemes required by standards organizations.

3.3. Reference resolution from labels

The process of resolving references involves:

1. Identification: Determining the target element based on its label
2. Context mapping: Understanding the relationship between the reference context and the target context
3. Rendering: Presenting the reference in an appropriate format for the current context

The LADL model provides the foundation for this resolution process by ensuring that labels are:

- Unique: Each label uniquely identifies a specific content element
- Structured: Labels follow a consistent structure that reflects the document hierarchy
- Context-aware: Labels can be interpreted correctly regardless of where they appear

3.4. Label reference types

3.4.1. General

Labels support different types of references depending on their scope and usage:

- Global references point to content across different contexts
- Local references point to content within the same context
- Citations provide formal references to external content

3.4.2. Global references

A global reference label is a unique reference that refers to content scoped within an information container.

Global references require string labels that include all necessary context information to ensure unambiguous identification.

A label reference is considered fully-qualified as a global reference if:

- All its labeling contexts are shown in the label
- The label uniquely refers to a content element within the container

© 2025 Ribose Asia Limited

EXAMPLE: A reference from Annex A to "Figure 3.2" clearly identifies the second figure in Clause 3, regardless of where the reference appears.

3.4.3. Local references

A local reference label is a unique reference that refers to content scoped within a part of the information container.

EXAMPLE 1: Within Clause 3, a reference to a list item "a)" is a local reference instead of the global reference counterpart of "Clause 3, a)".

Local references can be preferred over global references when referring content within the same context is more concise for the reader, due to omission of the duplicated self-reference.

EXAMPLE 2: Within a list, a reference to "a)" is more concise than "List 1, a)".

3.4.4. Citations

Citations are external references to the information container that the labels are attached to.

Citations to labelled content in the current environment may be created by external document references or in bibliographic entries.

For external users to create consistent citations to the content, the labels must be predictable and consistent across different documents.

3.5. Runtime requirements

The LADL language runtime is responsible for interpreting and executing label auto-assignment definitions. The runtime:

- Processes LADL definitions to create a label assignment model
- Maintains labeller states and sequences throughout document processing
- Manages context hierarchies and inheritance relationships
- Executes label generation according to defined patterns
- Handles format-specific rendering requirements

© 2025 Ribose Asia Limited

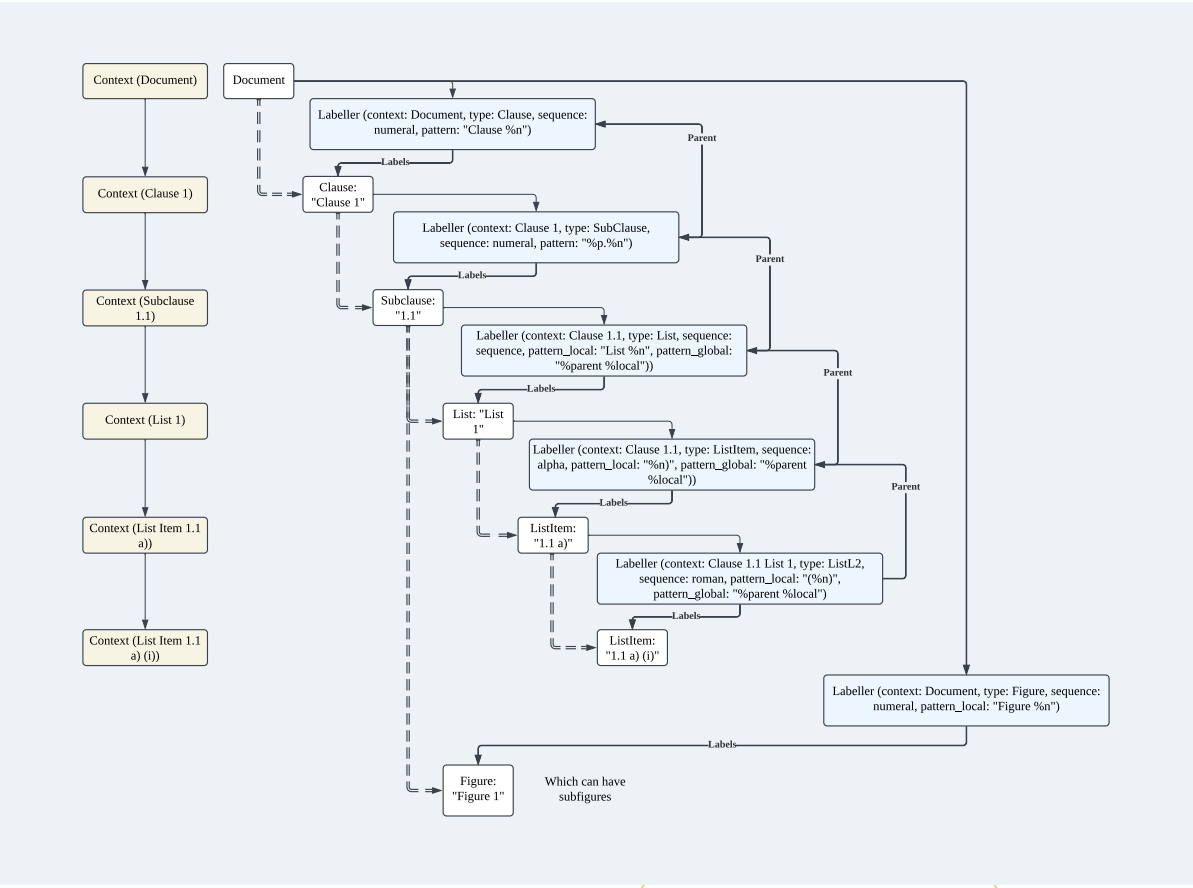
4. Architecture

4.1. General

The LADL architecture consists of:

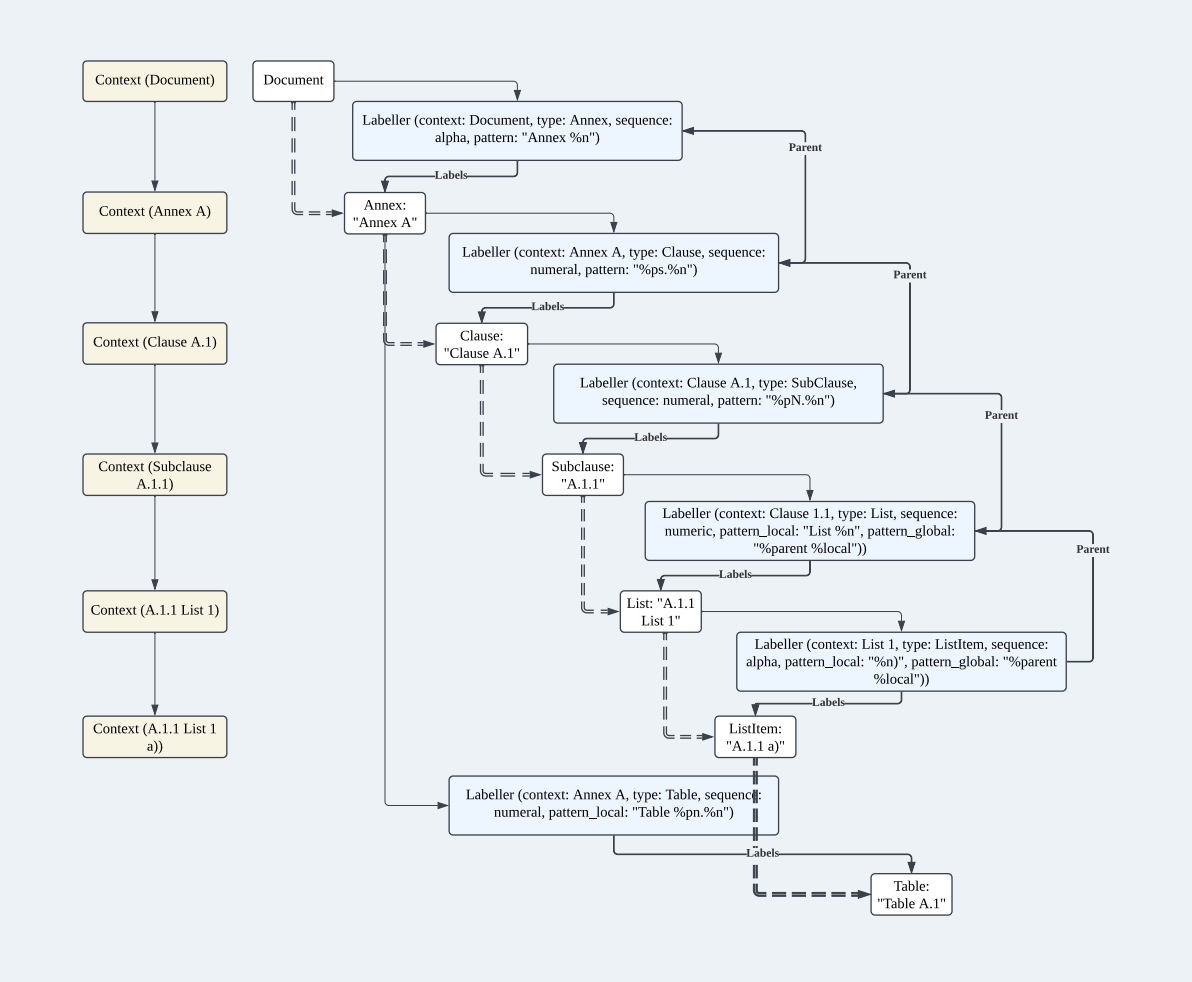
- Core models
- Hierarchical context management
- Context hierarchy framework
- Hierarchical pattern interpolation

FIGURE 1: Document clause hierarchy and labellers



© 2025 Ribose Asia Limited

FIGURE 2: Document annex hierarchy and labellers



This diagram demonstrates how:

- The LabelingContext is different from the actual content hierarchy
- Elements may be labeled according to a higher label context even if they belong to a lower document hierarchy
- Some elements are labeled according to label contexts belonging to the document hierarchy

The diagram illustrates the critical distinction between content hierarchy and label context in the LADL model:

- **Double dashed lines** represent content hierarchy (e.g., document containing clauses containing subclauses)
- **Unlabeled directional lines** represent ownership (e.g., document owning a ClauseLabeller and an AnnexLabeller)
- **Lines labeled "Parent"** show labeller inheritance relationships between parent and child labellers

© 2025 Ribose Asia Limited

- Lines labeled "Labels" show where a Labeller/Labeller assigns a Label to a content element

The diagram also illustrates the 3 key model trees in the LADL model:

- Content tree: Contains models that can be used to contain ContentElements that have Labels. These are boxes in white.
- Context tree: Made of LabelContext objects that provide label contexts. These are the boxes in brown.
- Labeller tree: Made of Labeller objects that allow hierarchical labeling. These are boxes in blue.

5. Core models

5.1. General

The core models in the LADL language framework are:

- `Label`: Represents a label assigned to a content element
- `Labeller`: Transforms a label template into a final label string
- `Sequence`: Defines the progression of values for a labeller
- `LabelContext`: Defines the scope and type of label context for labeling elements
- `LabelTemplate`: Defines how labeller values are combined with fixed text to create labels

In this document, we define the following models for illustration purposes of the LADL language:

- `ContentElement`: Represents a content element in an information container

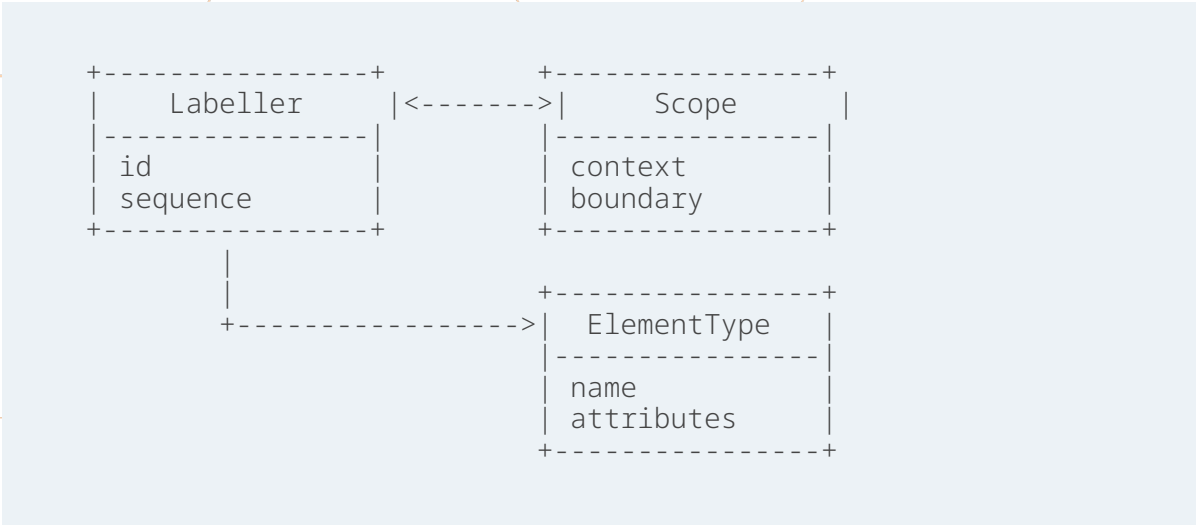
The following 3 model trees exist in the LADL model:

- Content tree: an object tree that contains models that can be used to contain ContentElements that have Labels.
- Context tree: an object tree made of LabelContext objects (linked via the `parent` attribute) that provides label contexts.
- Labeller tree: an object tree made of Labeller objects (linked via the `parent` attribute) that allows hierarchical labeling.

© 2025 Ribose Asia Limited

The following diagrams illustrate the key relationships in the LADL model.

FIGURE 3: Scope and Element Binding



5.2. ContentElement

The ContentElement class represents a content element in an information container that can potentially hold a LabelContext object.

FIGURE 4

```
class ContentElement {
  attribute id, String {
    definition "Unique identifier for the content element"
  }
  attribute parent, ref:(ContentElement) {
    definition "Parent element to which this element belongs"
  }
  attribute type, String {
    definition "Type of the content element"
  }
}
```

The `type` attribute is used to match whether a Labeller can label it.

A ContentElement must be associated with a LabelContext object if it has children to be labeled.

Every ContentElement that can either:

- Serve as a label context
- Be labeled

© 2025 Ribose Asia Limited

5.3. Sequence

5.3.1. General

A sequence defines the progression of values used for labeling.

There are two types of Sequence models that inherit from Sequence:

- CharacterSequence: Defines a fixed sequence of characters
- NumericSequence: Defines a monotonic sequence of numerical values

FIGURE 5

```
class Sequence {
  attribute char_direction, String {
    definition "Whether the sequence contains characters in
a left-to-right or right-to-left direction"

    default "ltr"
    values {
      value "ltr" {
        definition "Left-to-right"
      }
      value "rtl" {
        definition "Right-to-left"
      }
    }
  }
  method value_for_position(position: Integer) {
    definition "Provide the value for a given order in the
sequence"
  }

  method set_initial_value(position: Integer) {
    definition "Set the sequence to start at a specific
value"
  }
}
```

5.3.2. NumericSequence

FIGURE 6

```
class NumericSequence < Sequence {
  attribute initial_value, Integer {
    definition "Starting value for the sequence"
    default: 1
  }
  attribute increment, Integer {
```


© 2025 Ribose Asia Limited

```
        definition "Value to increment by for each step"
        default: 1
      }
      attribute cycle_length, Integer {
        definition "Length after which sequence notation expands"
      }
      attribute base_sequence, Sequence {
        definition "Base sequence that this sequence builds upon"
      }
    }
```

A LADL-compliant runtime should provide the following default labeling sequences:

Left-to-right sequences	**DecimalNumericSequence**	Standard decimal numbers (1, 2, 3...)
	RomanUpperNumericSequence	Uppercase Roman numerals (I, II, III...)
	RomanLowerNumericSequence	Lowercase Roman numerals (i, ii, iii...)
	AlphaUpperNumericSequence	Uppercase letters (A, B, C...)
	AlphaLowerNumericSequence	Lowercase letters (a, b, c...)
	CyrillicUpperCharacterSequence	Uppercase Cyrillic alphabet characters (А, Б, В, Г, Д, Е, Ж, З...)
	CyrillicLowerCharacterSequence	Lowercase Cyrillic alphabet characters (а, б, в, г, д, е, ж, з...)
	GreekUpperCharacterSequence	Uppercase Greek alphabet characters (Α, Β, Γ, Δ, Ε, Ζ, Η, Θ...)
	GreekLowerCharacterSequence	Lowercase Greek alphabet characters (α, β, γ, δ, ε, ζ, η, θ...)
	ChineseNumericSequence	Chinese numerals (一, 二, 三...)
	JapaneseNumericSequence	Japanese numerals (一, 二, 三...)
	KoreanNumericSequence	Korean numerals (일, 이, 삼...)
Right-to-left sequences	**ArabicNumericSequence**	Arabic numerals in Arabic script (,# # ,#...). This is a sequence of RTL numbers.
	HebrewNumericSequence	Hebrew numerals (א, ב, ג...). This is a sequence of RTL numbers.

EXAMPLE 1 — An Arabic numeric sequence that increments by 3: The following definition describes a custom numeral sequence based on ArabicNumericSequence that jumps every 3 positions.

© 2025 Ribose Asia Limited

```
instance ThreeIncrementArabicNumericSequence < NumericSequence {
  base_sequence = ArabicNumericSequence
  initial_value = 1
  increment = 3
}
```

Produces a sequence of "1, 4, 7, 10, ...".

EXAMPLE 2 — A Chinese numeric sequence that only gives even numbers: The following definition describes a custom numeric sequence based on Chinese numeric sequence only giving even numbers.

```
instance EvenChineseNumericSequence {
  base_sequence = ChineseNumericSequence
  initial_value = 2
  increment = 2
}
```

Produces a sequence of "二, 四, 六, 八, ...".

5.3.3. CharacterSequence

FIGURE 7

```
class CharacterSequence < Sequence {
  attribute characters, String {
    definition "Strings that define a label for an ordered
element in the sequence"
    cardinality 1..n
  }
  attribute expansion_rule, String {
    definition "Rule for expanding sequence after completion"
    values {
      value "repeat_label" {
        definition "Repeat the label character"
      }
      value "prepend_prefix" {
        definition "Add a new prefix character"
      }
      value "double_character" {
        definition "Double the character for expansion"
      }
      value "none" {
        definition "No expansion applied"
      }
      value "append_suffix" {
        definition "Append a suffix character after
expansion"
      }
    }
  }
  attribute direction, String {
    definition "Direction of the sequence"
    default "ltr"
    values {
```


© 2025 Ribose Asia Limited

```
        value "ltr" {
          definition "Left-to-right"
        }
        value "rtl" {
          definition "Right-to-left"
        }
      }
    }
  }
```

EXAMPLE 1: Certain ancient texts apply a labeling system of [”元“, “亨“, “利“, “貞“] as book labels if there are 4 books in a series. The following definition applies the expansion rule “prepend_prefix” to the sequence.

```
instance YiJingCharacterSequence {
  characters = ["元", "亨", "利", "貞"]
  expansion_rule = "prepend_prefix"
  expansion_prefix = ["乾", "坤", "巽", "震", "坎", "艮", "離", "兌"]
}
```

Produces a sequence of “元, 亨, 利, 貞, 乾元, 乾亨, 乾利, 乾貞, 坤元, 坤亨, …”.

EXAMPLE 2: Greek literature uses the Greek alphabet and numerals for labeling, which includes both letters and specific numeral characters for enumeration. The following definition applies the expansion rule “double_character” to the sequence.

```
instance GreekCharacterSequence {
  characters = ["α", "β", "γ", "δ", "ε", "ζ", "η", "θ"]
  expansion_rule = "double_character"
}
```

Produces a sequence of “α, β, γ, δ, ε, ζ, η, θ, αα, ββ, γγ, δδ, εε, …”.

5.4. Label template

The LabelTemplate class defines how string labels are constructed from labeller values and fixed text.

FIGURE 8

```
  class LabelTemplate {
    attribute pattern, String {
      definition "Template string with placeholders for
  labeller values"
    }
    attribute direction, String {
      definition "Direction of the pattern string"
      default "ltr"
      values {
        value "ltr" {
          definition "Left-to-right"
        }
```


© 2025 Ribose Asia Limited

```
          value "rtl" {
            definition "Right-to-left"
          }
        }
      }
      attribute condition, LabelContextCondition {
        definition {
          Condition of the LabelContext for selecting the
    pattern, select conditions
          that return "true" first, then select the first
    pattern.
        }
        cardinality 0..1
      }
    }
```

The `pattern` attribute is a string that contains placeholders for labeller values and any fixed text necessary.

The following placeholders are supported:

%n Labeller value

EXAMPLE 1: The pattern `"Figure %n"` may generate labels like "Figure 1", "Figure 2", "Figure 3", etc.

%pn Parent labeller value

EXAMPLE 2: The pattern `"Figure %pn.%n"`, where the parent is a labeller for an Annex context, may generate the labels "Figure A.1", "Figure B.2", "Figure C.3".

%parent Rendered global label value of parent.

EXAMPLE 3: The pattern "%parent, List %n", where the parent is a labeller for a Clause context, may generate the labels "Clause 1, List 1", "Clause 1, List 2", "Clause 2, List 1", etc.

%self Rendered local label value of self.

EXAMPLE 4: The pattern "%self" in a Clause labeller context, where the local reference template is "Clause %n", may generate the label "Clause 1", "Clause 2", "Clause 3".

The `direction` attribute defines the direction of the pattern string, which can be either "ltr" (left-to-right) or "rtl" (right-to-left).

EXAMPLE 5 — An unordered list label template: The following definition describes a label template for unordered lists with no string interpolation.

```
Level 1: •
Level 2: ◦
Level 3: ▪
```


© 2025 Ribose Asia Limited

```
instance UnorderedListLabelLevel1Template {
  pattern = "・"
}

instance UnorderedListLabelLevel2Template {
  pattern = "◦"
}

instance UnorderedListLabelLevel3Template {
  pattern = "▪"
}
```

EXAMPLE 6 — A simple figure labeling pattern: The following definition describes a label template for figures that uses the labeller value.
`Figure %n`

```
instance FigureLabelTemplate < LabelTemplate {
  pattern = "Figure %n"
}
```

Output: “Figure 1”, “Figure 2”, “Figure 3”, etc.

EXAMPLE 7 — A Japanese clause hierarchical labeling pattern: The following definition describes a label template for clauses that uses the labeller value.
`%parentの%n`

```
instance JapaneseClauseLabelTemplate < LabelTemplate {
  pattern = "%parentの%n"
}
```

If parent clause is labelled as “二“, the sequence is the JapaneseNumericSequence, the output would be “二の一“, “二の二“, “二の三“, etc.

If parent clause is labelled as “二の三“, the sequence is the JapaneseNumericSequence, the output would be “二の三の一“, “二の三の二“, “二の三の三“, etc.

A label template can be in a RTL direction. The following rules apply:

- Numbers flow correctly from right-to-left
- Hierarchical labels maintain proper RTL rendering
- Global references preserve the RTL format when used in different contexts

EXAMPLE 8 — RTL clause labels: Arabic documents use right-to-left (RTL) text direction.

```
instance ArabicClauseLabelTemplate < LabelTemplate {
  pattern = "%n ###"
  direction = "rtl"
}

instance ArabicClauseLabelGlobalTemplate < LabelTemplate {
  pattern = "%n #### %parent"
  direction = "rtl"
}
```


© 2025 Ribose Asia Limited

```
instance ArabicClauseLabeller {
  id = "arabic_clause_labeller"
  sequence = ArabicNumericSequence // This is a sequence of RTL numbers
  context = DocumentContext
  type = "clause"
  template = ArabicClauseLabelTemplate
  global_template = ArabicClauseLabelGlobalTemplate
}
```

The values would be:

```
> ArabicClauseLabeller.value_for_position(1) => "### #"
> ArabicClauseLabeller.value_for_position(2) => "### #"
> ArabicClauseLabeller.value_for_position(3) => "### #"
```

The `condition` attribute is a LabelContextCondition object that defines the condition for selecting the pattern.

- If the condition is met, the pattern is selected.
- If multiple patterns have conditions that are met, the first pattern is selected.

  It is crucial to note that it is possible to create multiple matching templates that have overlapping conditions.

  EXAMPLE 9: A template with a `condition` of "element_count_gt" and a threshold of 1, and another template with a condition of "element_count_eq" and a threshold of 2, overlap when the count is equal to 2.

- If no conditions are met, the first pattern is selected.

5.5. Label context condition

The LabelContextCondition class defines a context condition for a labeller to determine which pattern to use.

In some labeling systems, the labeller may need to switch between different conditions based on the defined thresholds.

EXAMPLE: In the ISO/IEC labelling system, a "NOTE" label takes on these forms depending on the number of notes in a clause:

- "NOTE" if there is only one note
- "NOTE 1", "NOTE 2" if there are more than one notes

© 2025 Ribose Asia Limited

NOTE This is only used for conditional label templates.

FIGURE 9

```
class LabelContextCondition {
  attribute type, String {
    definition "Type of label context condition"

    values {
      value "element_count_gt" {
        definition "Element count greater than"
      }
      value "element_count_eq" {
        definition "Element count equal to"
      }
      value "element_count_lt" {
        definition "Element count less than"
      }
    }
  }
  attribute threshold, Integer {
    definition "Threshold value for the label context
condition"
  }
}
```

5.6. Label context

The LabelContext class defines the scope of labeling (and a label hierarchy) within an information container.

FIGURE 10

```
class LabelContext {
  attribute name, String {
    definition "Name of the label context"
  }
  attribute parent, ref:(LabelContext) {
    definition "Parent label context to which this label
context relates"
  }
  attribute element, ContentElement {
    definition "Content element associated with the label
context"
  }
  attribute labellers, ref:(Labeller) {
    definition "Labellers (of different types) that are
bound to this label context"
    collection true
```


© 2025 Ribose Asia Limited

```
    }
    // attribute targets, Label {
    //   definition "Labels that are assigned to content
  elements within this label context"
    //   collection true
    // }
  }
```

The `name` attribute is a string that identifies the label context.

The `parent` attribute is a reference to the parent label context to which this label context relates.

The `element` attribute is a reference to the content element associated with the label context.

The `labellers` attribute is a collection of Labeller objects that are bound to this label context.

EXAMPLE: The following definition describes a label context for a document clause that contains a labeller for figures.

```
instance ClauseContext {
  name = "clause"
  parent = DocumentContext
  content_element = SomeClauseElement
  labellers = ["id:figure_labeller", "id:notes_labeller"]
}
```

5.7. Labeller

The Labeller model defines the fundamental mechanism for tracking sequence position within a label context.

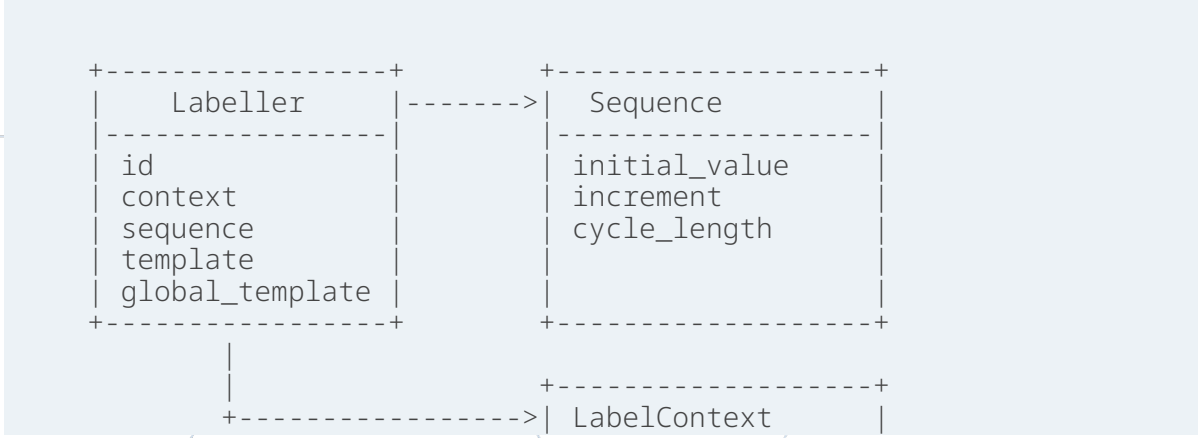
FIGURE 11

```
  class Labeller {
    attribute sequence, Sequence {
      definition "Sequence that defines the label's possible
  values"
    }

    attribute context, LabelContext {
      definition "LabelContext to which this labeller is bound"
    }

    attribute type, String {
      definition "Type of the ContentElement the labeller can
  label"
    }

    attribute parent, ref:(Labeller) {
```

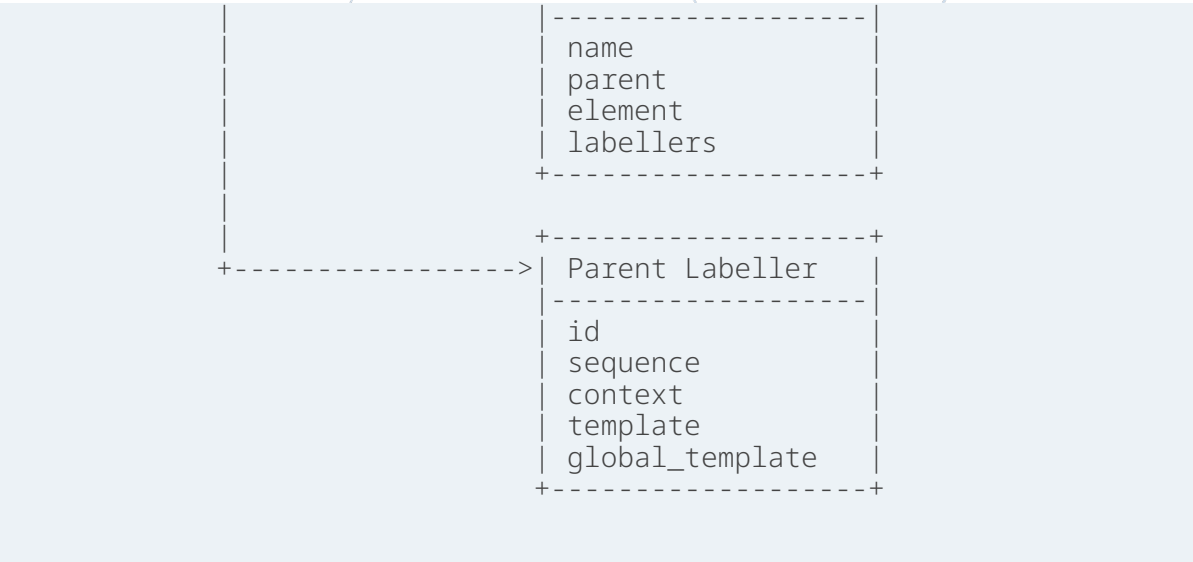

© 2025 Ribose Asia Limited

```
    definition "Parent labeller to which this labeller
relies upon to generate labels, if any"
  }

  attribute template, LabelTemplate {
    definition {
      Template for generating the label. The template
direction must be
      respected when generating the label. If there are more
than one templates,
      the labeller should select the appropriate template
based on the condition.
    }
    collection true
  }

  attribute global_template, LabelTemplate {
    definition {
      Template for generating the fully qualified, global
label. The template
      direction must be respected when generating the label.
If there are multiple
      global templates, the labeller should choose the
appropriate one based
      on the context and condition.
    }
    collection true
  }

  method value_for_position(position: Integer) {
    definition "Provide the rendered string label for local
reference."
  }

  method global_value_for_position(position: Integer) {
    definition "Provide the rendered string label for global
reference."
  }
}
```

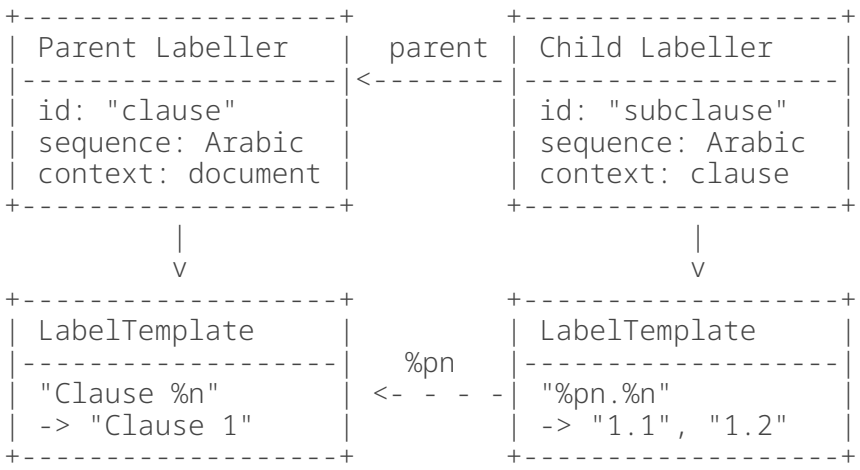
FIGURE 12: Labeller model



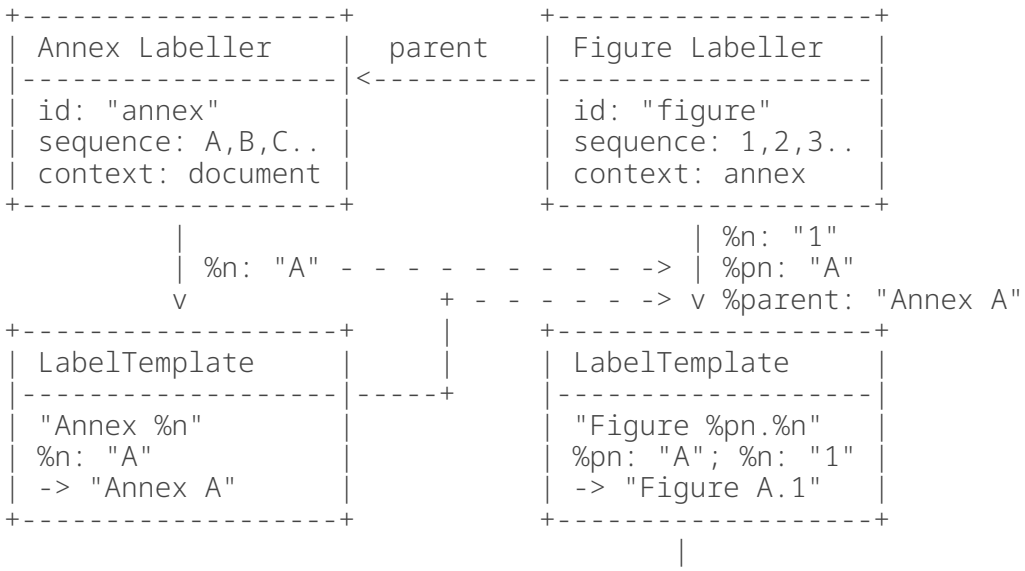
© 2025 Ribose Asia Limited



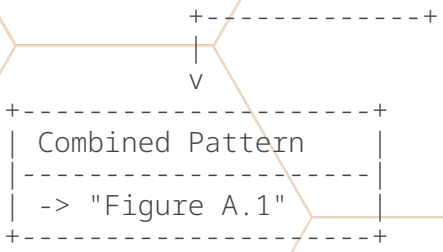
EXAMPLE 1 — Illustration of labelling a subclause as “1.1” with a parent labeller for “Clause 1”



EXAMPLE 2 — Illustration of labelling a figure as “Figure A.1”, where A is the parent label (an annex)



© 2025 Ribose Asia Limited



EXAMPLE 3 — An unordered list label template: The following definition describes a labeller for unordered lists with no string interpolation.

```
instance UnorderedListLabelTemplate {
  pattern = "List %n"
}

instance UnorderedListLabelLevel1Template {
  pattern = "•"
}

instance UnorderedListLabelLevel2Template {
  pattern = "◦"
}

instance UnorderedListLabelLevel3Template {
  pattern = "▪"
}

instance UnorderedListLabeller {
  id = "unordered_list_labeller"
  sequence = CharacterSequence
  context = ClauseContext
  type = "list"
  template = UnorderedListLabelTemplate
}
instance UnorderedListLevel1Labeller {
  id = "unordered_list_labeller_1"
  sequence = CharacterSequence
  context = ClauseContext
  type = "list_item"
  template = UnorderedListLabelLevel1Template
}
instance UnorderedListLevel2Labeller {
  id = "unordered_list_labeller_1_1"
  sequence = CharacterSequence
  context = ClauseContext
  type = "list_item"
  template = UnorderedListLabelLevel2Template
  parent = UnorderedListLevel1Labeller
}
instance UnorderedListLevel3Labeller {
  id = "unordered_list_labeller_1_1_3"
  sequence = CharacterSequence
  context = ClauseContext
  type = "list_item"
  template = UnorderedListLabelLevel3Template
  parent = UnorderedListLevel2Labeller
}
```


© 2025 Ribose Asia Limited

```
// Content elements
instance ListContentElement < ContentElement {
  // Content element definition
  labellers = ["id:unordered_list_labeller"]
}
instance ListItem_1_ContentElement < ContentElement {
  // Content element definition
  labellers = ["id:unordered_list_item_labeller_1"]
}
instance ListItem_1_1_ContentElement < ContentElement {
  // Content element definition
  labellers = ["id:unordered_list_item_labeller_1_1"]
}
instance ListItem_1_1_3_ContentElement < ContentElement {
  // Content element definition
  labellers = ["id:unordered_list_item_labeller_1_1_3"]
}
```

The values would be:

```
> UnorderedListLabeller.value_for_position(1) => "List 1"
> UnorderedListLabeller.value_for_position(3) => "List 3"

// Level 1, same per position
> UnorderedListLevel1Labeller.value_for_position(1) => "•"
> UnorderedListLevel1Labeller.value_for_position(8) => "•"

// Level 2, same per position
> UnorderedListLevel2Labeller.value_for_position(1) => "◦"
> UnorderedListLevel2Labeller.value_for_position(3) => "◦"

// Level 3, same per position
> UnorderedListLevel3Labeller.value_for_position(1) => "▪"
> UnorderedListLevel3Labeller.value_for_position(4) => "▪"
```

For hierarchical labeling, a Labeller may have a parent labeller that provides label context for the current labeller's position in the parent labeller's label context'.

EXAMPLE 4: Simple labeller for tracking Notes in a Clause.

```
instance ClauseContext {
  labellers = ["id:notes_labeller"]
}

instance NotesLabeller {
  id = "notes_labeller"
  sequence = ArabicNumericSequence
  context = ClauseContext
  parent = null // Does not rely on a parent Labeller
  template = NoteLabelTemplate
  global_template = NoteGlobalLabelTemplate
}
instance NoteLabelTemplate < LabelTemplate {
  pattern = "NOTE %n"
}

instance NoteGlobalLabelTemplate < LabelTemplate {
  pattern = "%parent, %self"
```


© 2025 Ribose Asia Limited

```
}
```

The values would be:

```
> NotesLabeller.value_for_position(1) => "NOTE 1"
> NotesLabeller.value_for_position(2) => "NOTE 2"
> NotesLabeller.value_for_position(3) => "NOTE 3"
> NotesLabeller.global_value_for_position(1) => "Clause 1, NOTE 1"
> NotesLabeller.global_value_for_position(2) => "Clause 1, NOTE 2"
> NotesLabeller.global_value_for_position(3) => "Clause 1, NOTE 3"
```

EXAMPLE 5: Simple labeller for tracking Figures:

```
instance ClauseContext {
  type = "section"
  labellers = ["id:figure_labeller"]
}

instance FigureLabeller {
  id = "figure_labeller"
  sequence = RomanUpperNumericSequence
    context = ClauseContext
  parent = null // Does not rely on a parent Labeller
  template = FigureLabelTemplate
  global_template = FigureGlobalLabelTemplate
}
instance FigureLabelTemplate < LabelTemplate {
  pattern = "Figure %n"
}

instance FigureGlobalLabelTemplate < LabelTemplate {
  pattern = "%parent, %self"
}
```

The values would be:

```
> FigureLabelTemplate.value_for_position(1) => "Figure I"
> FigureLabelTemplate.value_for_position(2) => "Figure II"
> FigureLabelTemplate.value_for_position(3) => "Figure III"
> FigureGlobalLabelTemplate.global_value_for_position(1) => "Section
1, Figure I"
> FigureGlobalLabelTemplate.global_value_for_position(2) => "Section
1, Figure II"
> FigureGlobalLabelTemplate.global_value_for_position(3) => "Section
1, Figure III"
```

EXAMPLE 6 — Conditional NOTE label: The following definition describes a labeller for tracking Notes in a Clause with a conditional label template.

```
instance ClauseContext < LabelContext {
  labellers = ["id:notes_labeller"]
}

instance NotesLabeller < Labeller {
  id = "notes_labeller"
  sequence = ArabicNumericSequence
  context = ClauseContext
  parent = null // Does not rely on a parent Labeller
```


© 2025 Ribose Asia Limited

```
  template = [NoteSingleLabelTemplate, NoteMultipleLabelTemplate]
  global_template = [NoteSingleGlobalLabelTemplate, NoteMultipleGlobalL
abelTemplate]
}
instance NoteSingleLabelTemplate < LabelTemplate {
  pattern = "NOTE"
}

instance NoteMultipleLabelTemplate < LabelTemplate {
  pattern = "NOTE %n"
  condition = {
    type = "element_count_gt"
    threshold = 1
  }
}

instance NoteSingleGlobalLabelTemplate < LabelTemplate {
  pattern = "%parent, NOTE"
}

instance NoteMultipleGlobalLabelTemplate < LabelTemplate {
  pattern = "%parent, NOTE %n"
  condition = {
    type = "element_count_gt"
    threshold = 1
  }
}
```

The values would be:

```
// If there is only one note
> NotesLabeller.value_for_position(1) => "NOTE"

// If there are more than one notes
> NotesLabeller.value_for_position(1) => "NOTE 1"
> NotesLabeller.value_for_position(2) => "NOTE 2"
```

5.8. Label

The Label model represents a label assigned to a content element.

FIGURE 13

```
    class Label {
      attribute value, String {
        definition "Value of the label"
      }
      attribute content_element, ContentElement {
        definition "Content element to which this label is
    assigned"
      }
```


© 2025 Ribose Asia Limited

```
    }
```

EXAMPLE: The following definition describes a label for a figure with the value “Figure 1”.

```
instance FigureLabel {
  value = "Figure 1"
  content_element = SomeFigureElement
}
```

5.9. Label style

The LabelStyle class defines a full set of label models sufficient to implement a coherent labeling system for a publisher.

FIGURE 14

```
class LabelStyle {
  attribute name, String {
    definition "Name of the label style"
  }
  attribute labellers, Labeller {
    definition "Labellers that are bound to this label style"
    collection true
  }
  attribute sequences, Sequence {
    definition "Sequences that are bound to this label style"
    collection true
  }
  attribute templates, LabelTemplate {
    definition "Templates that are bound to this label style"
    collection true
  }
  attribute contexts, LabelContext {
    definition "Label contexts that are bound to this label
style"
    collection true
  }
}
```


© 2025 Ribose Asia Limited

6. Usage

6.1. Defining a label style

A label style (the LabelStyle class) represents a coherent collection of LADL models that define a coherent labeling system for a document.

Steps to creating a label style are detailed below.

1. Understand the ContentElement tree that will be labeled.
   a. Consider the types of ContentElements to label.

      EXAMPLE 1: In a content tree there may be clauses, annexes and notes.

   b. Consider the label contexts of those ContentElements.

      EXAMPLE 2: Consider whether a ContentElement type will have its label scoped by the document, or a subclause.

   c. Consider the templates that will be used to format the labels.

      EXAMPLE 3: A clause may be labeled as “Clause 1”, an annex as “Annex A”, and a note as “Note 1”.

   d. Consider whether the label for a ContentElement will differ between global and local references.

      EXAMPLE 4: The first ordered list item may be labelled locally as “a)”, but globally as “List N, a)”.

2. Define the LADL models from the bottom up, per type of ContentElement.
   a. Specify the LabelTemplate for generating the label for the type of ContentElement.

      EXAMPLE 5: A clause labeled as “Clause 1” is specified as a LabelTemplate with the pattern “Clause %n”, where the sequence used is ArabicNumericSequence.

   b. Specify the Sequence for generating numbering formats if a custom implementation is required.

      EXAMPLE 6: An Annex labeled as “Annex A” requires the sequence RomanUpperNumericSequence, which is available by default.

   c. Specify the LabelContext for the type of ContentElement.

      EXAMPLE 7: A clause numbered according to a document-level scope requires a LabelContext that is linked to the ContentElement(Document).

   d. Construct the Labeller for the type of ContentElement using the defined models in previous steps.

© 2025 Ribose Asia Limited

EXAMPLE 8: A labeller requires a sequence, context, template (local and maybe global), potentially a parent labeller depending on whether the template incorporates the ContentElement's parent's label.

3. Construct the LabelStyle object with the defined models.

EXAMPLE 9 — Full label style definition of SimpleLabelStyle

The following definition describes a label style for a document with a clause context and a figure labeller.

The desired outputs are:

For `ContentElement(Clause)`	"Clause 1" for the clause labeller, with label context of the document, identical global and local label.
For `ContentElement(SubClause)`	"1.1" for the sub-clause labeller, with label context of the clause, identical global and local label.
For `ContentElement(Figure)`	"Figure 1" for the figure labeller, with label context of the document, identical global and local label.

Assume that we have this ContentElement tree:

```
Document
  Clause 1
    Sub-clause 1
    Sub-clause 2
      Figure 1
  Clause 2
    Figure 2
    Sub-clause 1
    Sub-clause 2
      Sub-clause 1
      Sub-clause 2
```

The label contexts are:

- The DocumentContext scopes the labels of clauses and figures
- The ClauseContext scopes the labels of the first sub-clause level
- The SubClauseContext scopes the labels of all other sub-clause levels
- The Document scopes the labels of figures

```
Document
  Clause 1          [label scoped by Document: DocumentContext with
ClauseLabeller]
    Sub-clause 1    [label scoped by Clause 1: ClauseContext with
SubClauseLabeller]
    Sub-clause 2    [label scoped by Clause 1: ClauseContext with
SubClauseLabeller]
      Figure 1      [label scoped by Document: DocumentContext with
FigureLabeller]
  Clause 2          [label scoped by Document: DocumentContext with
ClauseLabeller]
```


© 2025 Ribose Asia Limited

```
    Figure 2       [label scoped by Document: DocumentContext with
FigureLabeller]
    Sub-clause 1   [label scoped by Clause 2: ClauseContext with
SubClauseLabeller]
    Sub-clause 2   [label scoped by Clause 2: ClauseContext with
SubClauseLabeller]
      Sub-clause 1 [label scoped by 2.2: SubClauseContext with
SubClauseLabeller]
      Sub-clause 2 [label scoped by 2.2: SubClauseContext with
SubClauseLabeller]
        Figure 3   [label scoped by Document: DocumentContext with
FigureLabeller]

instance SimpleLabelStyle {
  name = "Simple Label Style"
  labellers = ["id:figure_labeller", "id:clause_labeller", "id:
subclause_labeller"]
  sequences = [ArabicNumericSequence]
  templates = [FigureLabelTemplate, ClauseLabelTemplate,
SubClauseLabelTemplate]
  contexts = [DocumentContext, ClauseContext, SubClauseContext]
}

instance DocumentContext < LabelContext {
  name = "document"
}

instance ClauseContext < LabelContext {
  // One per clause, link to ContentElement(Clause)
  name = "clause"
  parent = DocumentContext
}

instance SubClauseContext < LabelContext {
  // One per subclause, link to ContentElement(Clause)
  name = "subclause"
  // Implementation set parent to ClauseContext
}

instance ClauseLabeller < Labeller {
  id = "clause_labeller"
  sequence = ArabicNumericSequence
  context = DocumentContext
  parent = null // Does not rely on a parent Labeller
  template = ClauseLabelTemplate
}
instance ClauseLabelTemplate < LabelTemplate {
  pattern = "Clause %n"
}

instance SubClauseLabeller < Labeller {
  id = "subclause_labeller"
  sequence = ArabicNumericSequence
  context = ClauseContext
  parent = ClauseLabeller // Rely on ClauseLabeller for parent label
  template = SubClauseLabelTemplate
}
instance SubClauseLabelTemplate < LabelTemplate {
```


© 2025 Ribose Asia Limited

```
  pattern = "%parent.%n"
}

instance FigureLabeller < Labeller {
  id = "figure_labeller"
  sequence = ArabicNumericSequence
  context = DocumentContext
  parent = null // Does not rely on a parent Labeller
  template = FigureLabelTemplate
  global_template = FigureGlobalLabelTemplate
}
instance FigureLabelTemplate < LabelTemplate {
  pattern = "Figure %n"
}

instance FigureGlobalLabelTemplate < LabelTemplate {
  pattern = "%parent, %self"
}
```

6.2. Algorithm to resolve the label for a content element

6.2.1. General

The following algorithm describe how to resolve the label for a ContentElement object in a LADL-compliant runtime.

6.2.2. Prerequisites

Before resolving a label, the following prerequisites must be met:

1. A valid LabelStyle must be defined with appropriate Labeller objects.
2. ContentElement objects must be correctly structured in a content tree.
3. LabelContext objects must be properly associated with ContentElement objects.
4. Each Labeller must have at least one valid LabelTemplate.

The label resolution algorithm also requires:

- Access to the current element count of similar elements in the same context (for conditional templates)
- Ability to track labeller state across the document
- Proper parent-child relationships between Labeller objects

6.2.3. Algorithm

1. From the ContentElement, find its applicable LabelContext object. The applicable LabelContext object is one that has a Labeller that can label the type of the ContentElement.
   a. If the LabelContext object is on it, use it.

© 2025 Ribose Asia Limited

b. If not, navigate through its `parent` attribute subsequently until the closest LabelContext object is obtained.

2. From the LabelContext object, find the Labeller object that is applicable to the ContentElement.

3. Ask the Labeller object to generate a label for the ContentElement.

a. The Labeller object will use its LabelTemplate object to generate the label.

i. If the Labeller has multiple LabelTemplate objects with conditions, evaluate each condition:

A. For each LabelTemplate, check if its LabelContextCondition evaluates to true for the current context.

B. Select the first LabelTemplate whose condition evaluates to true.

C. If no conditions evaluate to true, select the first LabelTemplate without a condition, or the first one if all have conditions

ii. If the LabelTemplate requires a sequence element, it will ask its Sequence object to generate a value for the position of the ContentElement in the LabelContext.

iii. If the LabelTemplate requires a parent labeller, it will ask its parent Labeller object to generate a label for the ContentElement.

4. Done.

© 2025 Ribose Asia Limited

Annex A
(normative)

Elements often auto-labeled

A.1. General

Many types of content elements in technical documents are auto-labeled to provide a consistent structure and reference system.

This annex provides examples of common auto-labeled elements for the reader's reference.

A.2. Lists

Lists are among the most commonly labeled elements in documents, with both ordered and unordered variants.

A.2.1. Ordered Lists

Ordered lists use sequential labellers to number items in a list, increasing in value with each new item. They may also include nested hierarchies for sub-items.

EXAMPLE:

1. Item 1
2. Item 2
    a. Sub-item 2.1
    b. Sub-item 2.2
3. Item 3

A.2.2. Unordered Lists

Unordered lists use non-sequential markers for visual distinction.

EXAMPLE:

- Item A

© 2025 Ribose Asia Limited

- Item B
  - Sub-item B.1
  - Sub-item B.2

A.3. Clause labeling

Clauses form the primary structural elements of technical documents and use hierarchical labeling.

EXAMPLE: "Clause 1: Introduction"
"Clause 2: Background"

"Sub-clause 2.1: Context"

"Sub-clause 2.2: Importance"

A.4. Annex and appendix labeling

Annexes and appendices typically use different labeling schemes than the main document.

EXAMPLE: "Annex A: Glossary"
"Annex B: References"

"Annex A, Appendix 1: Sample Data"

"Annex A, Appendix 2: Test Cases"

A.5. Figure labeling

Figures typically use a combination of sequential and hierarchical labeling.

EXAMPLE: "Figure 1: System Architecture"
"Figure A.2: Data Flow Diagram"

A.6. Table labeling

Tables use labeling schemes similar to figures but often with different presentation.

EXAMPLE: "Table 1: Data Summary"
"Table A.2: Test Results"

© 2025 Ribose Asia Limited

A.7. Note labeling

Notes may appear throughout a document and are typically labeled within their label context.

EXAMPLE: "NOTE 1: Important Information"
"Clause 2, NOTE 1: Additional Details"

A.8. Example labeling

Examples often follow similar label templates to notes.

EXAMPLE: "EXAMPLE 1: Basic Usage"
"Clause 3, EXAMPLE 1: Advanced Features"

A.9. Requirement labeling

Requirements often need special labeling for traceability.

EXAMPLE: "REQ-1.1: System shall support multiple users"
"REQ-1.2: The system shall ensure data security"

© 2025 Ribose Asia Limited

Annex B
(normative)

Instance model examples

B.1. Illustration diagram

The following instance model definitions illustrate the relationships shown in the diagram.

EDITORIAL NOTE

Add instance model definitions

© 2025 Ribose Asia Limited

Annex C
(normative)

ISO/IEC label style definition

C.1. Introduction

This document defines the ISO/IEC label style using the Label Auto-assignment Definition Language (LADL) as specified in MN 112. The ISO/IEC label style originates from and implements the requirements of:

- the ISO/IEC Directives, Part 2
- the ISO House Style

This style defines the labeling scheme used by ISO/IEC deliverables.

C.2. Specification

The element labeling scheme for ISO/IEC is specified as follows.

TABLE C.1: Labeling specification

Element	Labeling specification	LabelContext
Clause	Local and global: “Clause 1”, “Clause 2”...	Document
Subclause	Local and global: “1.1”, “1.2”, “2.1.1”... Uses parent’s sequence number	Parent Clause
Annex	Local and global: “Annex A”, “Annex B”, “Annex AA”...	Document
Annex clause	Local and global: “A.1”, “A.2”...	Annex
Figure	Local and global: “Figure 1”, “Figure A.1” Subfigure: “Figure A.1 a)”	Document (for main document) Annex (for annex figures)
Table	Local and global: “Table 1”, “Table A.1”	Document (for main document) Annex (for annex tables)
NOTE	Local: “NOTE 1”, “NOTE 2”... Global: “Clause X, NOTE 1”	Clause or Subclause

© 2025 Ribose Asia Limited

Element	Labeling specification	LabelContext
	Special case: If single note, simply "NOTE" without number	
EXAMPLE	Local: "EXAMPLE 1", "EXAMPLE 2"... Global: "Clause X, EXAMPLE 1", "1.2.3, EXAMPLE 2"	Clause or Subclause
Formula	Same pattern as Figure/Table	Clause
List	Local and global: "List 1", "List 2", "List A.1"...	Clause
List item (unordered)	All levels: "—"	List (first level), Parent ListItem (subsequent levels)
List item (ordered)	First level: a), b), c)... Second level: 1), 2), 3)... Third level: i), ii), iii)... Fourth level: a), b), c)...	List (first level), Parent ListItem (subsequent levels)

C.3. Clause labeling

The clause labeling scheme for ISO/IEC documents follows a hierarchical structure with Arabic numerals.

FIGURE C.1

```
// Clause templates
instance ClauseLabelTemplate < LabelTemplate {
  pattern = "Clause %n"
  definition "Template for local clause references"
}

instance ClauseGlobalLabelTemplate < LabelTemplate {
  pattern = "Clause %n"
  definition "Template for global clause references"
}

instance ClauseLabeller < Labeller {
  id = "clause_labeller"
  sequence = ArabicNumericSequence
  context = DocumentContext
  type = "clause"
  template = ClauseLabelTemplate
  global_template = ClauseGlobalLabelTemplate
  definition "Labeller for clauses in the main document"
}
```


© 2025 Ribose Asia Limited

C.4. Subclause labeling

Subclauses are labeled hierarchically, inheriting the parent clause number.

FIGURE C.2

```
// Subclause templates
instance SubclauseLabelTemplate < LabelTemplate {
  pattern = "%pn.%n"
  definition "Template for local subclause references"
}

instance SubclauseGlobalLabelTemplate < LabelTemplate {
  pattern = "%pn.%n"
  definition "Template for global subclause references"
}

instance SubclauseLabeller < Labeller {
  id = "subclause_labeller"
  sequence = ArabicNumericSequence
  context = ClauseContext
  parent = ClauseLabeller
  type = "subclause"
  template = SubclauseLabelTemplate
  global_template = SubclauseGlobalLabelTemplate
  definition "Labeller for subclauses within clauses"
}
```

C.5. Annex labeling

Annexes use uppercase alphabetic labeling.

FIGURE C.3

```
instance AnnexLabelTemplate < LabelTemplate {
  pattern = "Annex %n"
  definition "Template for local annex references"
}

instance AnnexGlobalLabelTemplate < LabelTemplate {
  pattern = "Annex %n"
  definition "Template for global annex references"
}

instance AnnexLabeller < Labeller {
  id = "annex_labeller"
  sequence = AlphaUpperNumericSequence
  context = DocumentContext
  type = "annex"
  template = AnnexLabelTemplate
```


© 2025 Ribose Asia Limited

```
  global_template = AnnexGlobalLabelTemplate
  definition "Labeller for annexes in the document"
}
```

C.6. Annex clause labeling

Clauses within annexes combine the annex letter with an Arabic numeral.

FIGURE C.4

```
instance AnnexClauseLabelTemplate < LabelTemplate {
  pattern = "%pn.%n"
  definition "Template for local annex clause references"
}

instance AnnexClauseGlobalLabelTemplate < LabelTemplate {
  pattern = "%pn.%n"
  definition "Template for global annex clause references"
}

instance AnnexClauseLabeller < Labeller {
  id = "annex_clause_labeller"
  sequence = ArabicNumericSequence
  context = AnnexContext
  parent = AnnexLabeller
  type = "clause"
  template = AnnexClauseLabelTemplate
  global_template = AnnexClauseGlobalLabelTemplate
  definition "Labeller for clauses within annexes"
}
```

C.7. Figure labeling

Figures are labeled with “Figure” followed by a sequential number.

FIGURE C.5

```
instance FigureLabelTemplate < LabelTemplate {
  pattern = "Figure %n"
  definition "Template for local figure references in the
main document"
}

instance FigureGlobalLabelTemplate < LabelTemplate {
  pattern = "Figure %n"
  definition "Template for global figure references in the
main document"
```


© 2025 Ribose Asia Limited

```
}

instance FigureLabeller < Labeller {
  id = "figure_labeller"
  sequence = ArabicNumericSequence
  context = DocumentContext
  type = "figure"
  template = FigureLabelTemplate
  global_template = FigureGlobalLabelTemplate
  definition "Labeller for figures in the main document"
}
```

C.8. Annex figure labeling

Figures in annexes combine the annex letter with a figure number.

FIGURE C.6

```
instance AnnexFigureLabelTemplate < LabelTemplate {
  pattern = "Figure %pn.%n"
  definition "Template for local figure references in
annexes"
}

instance AnnexFigureGlobalLabelTemplate < LabelTemplate {
  pattern = "Figure %pn.%n"
  definition "Template for global figure references in
annexes"
}

instance AnnexFigureLabeller < Labeller {
  id = "annex_figure_labeller"
  sequence = ArabicNumericSequence
  context = AnnexContext
  parent = AnnexLabeller
  type = "figure"
  template = AnnexFigureLabelTemplate
  global_template = AnnexFigureGlobalLabelTemplate
  definition "Labeller for figures in annexes"
}
```


© 2025 Ribose Asia Limited

C.9. Subfigure labeling

Subfigures use lowercase alphabetic labels.

FIGURE C.7

```
instance SubfigureLabelTemplate < LabelTemplate {
  pattern = "Figure %pn %n)"
  definition "Template for local subfigure references"
}

instance SubfigureGlobalLabelTemplate < LabelTemplate {
  pattern = "Figure %pn %n)"
  definition "Template for global subfigure references"
}

instance SubfigureLabeller < Labeller {
  id = "subfigure_labeller"
  sequence = AlphaLowerNumericSequence
  context = DocumentContext
  parent = FigureLabeller
  type = "subfigure"
  template = SubfigureLabelTemplate
  global_template = SubfigureGlobalLabelTemplate
  definition "Labeller for subfigures in the document"
}
```

C.10. Table labeling

Tables are labeled with “Table” followed by a sequential number.

FIGURE C.8

```
instance TableLabelTemplate < LabelTemplate {
  pattern = "Table %n"
  definition "Template for local table references in the
main document"
}

instance TableGlobalLabelTemplate < LabelTemplate {
  pattern = "Table %n"
  definition "Template for global table references in the
main document"
}

instance TableLabeller < Labeller {
  id = "table_labeller"
  sequence = ArabicNumericSequence
  context = DocumentContext
  type = "table"
```


© 2025 Ribose Asia Limited

```
  template = TableLabelTemplate
  global_template = TableGlobalLabelTemplate
  definition "Labeller for tables in the main document"
}
```

C.11. Annex table labeling

Tables in annexes combine the annex letter with a table number.

FIGURE C.9

```
instance AnnexTableLabelTemplate < LabelTemplate {
  pattern = "Table %pn.%n"
  definition "Template for local table references in annexes"
}

instance AnnexTableGlobalLabelTemplate < LabelTemplate {
  pattern = "Table %pn.%n"
  definition "Template for global table references in
annexes"
}

instance AnnexTableLabeller < Labeller {
  id = "annex_table_labeller"
  sequence = ArabicNumericSequence
  context = AnnexContext
  parent = AnnexLabeller
  type = "table"
  template = AnnexTableLabelTemplate
  global_template = AnnexTableGlobalLabelTemplate
  definition "Labeller for tables in annexes"
}
```

C.12. Note labeling

Notes use conditional labeling based on the number of notes in a context. A single note is labeled simply as “NOTE” without a number, while multiple notes in the same context are labeled as “NOTE 1”, “NOTE 2”, etc.

FIGURE C.10

```
instance NoteSingleLabelTemplate < LabelTemplate {
  pattern = "NOTE"
  definition "Template for single note references"
}
```


© 2025 Ribose Asia Limited

```
instance NoteMultipleLabelTemplate < LabelTemplate {
  pattern = "NOTE %n"
  definition "Template for multiple note references"
  condition = {
    type = "element_count_gt"
    threshold = 1
  }
}

instance NoteSingleGlobalLabelTemplate < LabelTemplate {
  pattern = "%parent, NOTE"
  definition "Template for global single note references"
}

instance NoteMultipleGlobalLabelTemplate < LabelTemplate {
  pattern = "%parent, NOTE %n"
  definition "Template for global multiple note references"
  condition = {
    type = "element_count_gt"
    threshold = 1
  }
}

instance NoteLabeller < Labeller {
  id = "note_labeller"
  sequence = ArabicNumericSequence
  context = ClauseContext
  type = "note"
  template = [NoteSingleLabelTemplate,
NoteMultipleLabelTemplate]
  global_template = [NoteSingleGlobalLabelTemplate, NoteMulti
pleGlobalLabelTemplate]
  definition "Labeller for notes in clauses with conditional
templates based on note count"
}
```

The `condition` property uses a LabelContextCondition object to determine which template to use based on element count. The condition `element_count_gt` with threshold 1 means "use this template when there are more than 1 notes in the context."

C.13. Example labeling

Examples use conditional labeling similar to notes. A single example is labeled simply as "EXAMPLE" without a number, while multiple examples in the same context are labeled as "EXAMPLE 1", "EXAMPLE 2", etc.

FIGURE C.11

```
instance ExampleSingleLabelTemplate < LabelTemplate {
  pattern = "EXAMPLE"
  definition "Template for single example references"
```


© 2025 Ribose Asia Limited

```
}

instance ExampleMultipleLabelTemplate < LabelTemplate {
  pattern = "EXAMPLE %n"
  definition "Template for multiple example references"
  condition = {
    type = "element_count_gt"
    threshold = 1
  }
}

instance ExampleSingleGlobalLabelTemplate < LabelTemplate {
  pattern = "%parent, EXAMPLE"
  definition "Template for global single example references"
}

instance ExampleMultipleGlobalLabelTemplate < LabelTemplate {
  pattern = "%parent, EXAMPLE %n"
  definition "Template for global multiple example
references"
  condition = {
    type = "element_count_gt"
    threshold = 1
  }
}

instance ExampleLabeller < Labeller {
  id = "example_labeller"
  sequence = ArabicNumericSequence
  context = ClauseContext
  type = "example"
  template = [ExampleSingleLabelTemplate, ExampleMultipleLabe
lTemplate]
  global_template = [ExampleSingleGlobalLabelTemplate, Exampl
eMultipleGlobalLabelTemplate]
  definition "Labeller for examples in clauses with
conditional templates based on example count"
}
```

Just like with notes, the LabelContextCondition determines which template to use based on the count of examples within the context.

C.14. Formula labeling

Formulas are labeled with parenthesized numbers.

FIGURE C.12

```
instance FormulaLabelTemplate < LabelTemplate {
  pattern = "(%n)"
  definition "Template for local formula references in the
main document"
```


© 2025 Ribose Asia Limited

```
}

instance FormulaGlobalLabelTemplate < LabelTemplate {
  pattern = "Formula (%n)"
  definition "Template for global formula references in the
main document"
}

instance FormulaLabeller < Labeller {
  id = "formula_labeller"
  sequence = ArabicNumericSequence
  context = DocumentContext
  type = "formula"
  template = FormulaLabelTemplate
  global_template = FormulaGlobalLabelTemplate
  definition "Labeller for formulas in the main document"
}
```

C.15. Annex formula labeling

Formulas in annexes combine the annex letter with a formula number.

FIGURE C.13

```
instance AnnexFormulaLabelTemplate < LabelTemplate {
  pattern = "(%pn.%n)"
  definition "Template for local formula references in
annexes"
}

instance AnnexFormulaGlobalLabelTemplate < LabelTemplate {
  pattern = "Formula (%pn.%n)"
  definition "Template for global formula references in
annexes"
}

instance AnnexFormulaLabeller < Labeller {
  id = "annex_formula_labeller"
  sequence = ArabicNumericSequence
  context = AnnexContext
  parent = AnnexLabeller
  type = "formula"
  template = AnnexFormulaLabelTemplate
  global_template = AnnexFormulaGlobalLabelTemplate
  definition "Labeller for formulas in annexes"
}
```


© 2025 Ribose Asia Limited

C.16. List labeling

Lists are labeled with “List” followed by a sequential number.

FIGURE C.14

```
instance ListLabelTemplate < LabelTemplate {
  pattern = "List %n"
  definition "Template for local list references in the main
document"
}

instance ListGlobalLabelTemplate < LabelTemplate {
  pattern = "List %n"
  definition "Template for global list references in the
main document"
}

instance ListLabeller < Labeller {
  id = "list_labeller"
  sequence = ArabicNumericSequence
  context = ClauseContext
  type = "list"
  template = ListLabelTemplate
  global_template = ListGlobalLabelTemplate
  definition "Labeller for lists in the main document"
}
```

C.17. Annex list labeling

Lists in annexes combine the annex letter with a list number.

FIGURE C.15

```
instance AnnexListLabelTemplate < LabelTemplate {
  pattern = "List %pn.%n"
  definition "Template for local list references in annexes"
}

instance AnnexListGlobalLabelTemplate < LabelTemplate {
  pattern = "List %pn.%n"
  definition "Template for global list references in annexes"
}

instance AnnexListLabeller < Labeller {
  id = "annex_list_labeller"
  sequence = ArabicNumericSequence
  context = AnnexContext
  parent = AnnexLabeller
  type = "list"
```


© 2025 Ribose Asia Limited

```
  template = AnnexListLabelTemplate
  global_template = AnnexListGlobalLabelTemplate
  definition "Labeller for lists in annexes"
}
```

C.18. Unordered list item labeling

Unordered list items use a dash character at all levels.

FIGURE C.16

```
instance UnorderedListItemLabelTemplate < LabelTemplate {
  pattern = "—"
  definition "Template for unordered list items at all
levels"
}

instance UnorderedListItemLabeller < Labeller {
  id = "unordered_list_item_labeller"
  sequence = ArabicNumericSequence  // Not used but required
  context = ListContext
  type = "list_item_unordered"
  template = UnorderedListItemLabelTemplate
  definition "Labeller for unordered list items"
}
```

C.19. Ordered list item labeling

Ordered list items use different sequences at different levels.

FIGURE C.17

```
// Level 1: a), b), c)...
instance OrderedListItemLevel1LabelTemplate < LabelTemplate {
  pattern = "%n)"
  definition "Template for first level ordered list items"
}

instance OrderedListItemLevel1Labeller < Labeller {
  id = "ordered_list_item_level1_labeller"
  sequence = AlphaLowerNumericSequence
  context = ListContext
  type = "list_item_ordered_level1"
  template = OrderedListItemLevel1LabelTemplate
  definition "Labeller for first level ordered list items"
}
```


© 2025 Ribose Asia Limited

```
// Level 2: 1), 2), 3)...
instance OrderedListItemLevel2LabelTemplate < LabelTemplate {
  pattern = "%n)"
  definition "Template for second level ordered list items"
}

instance OrderedListItemLevel2Labeller < Labeller {
  id = "ordered_list_item_level2_labeller"
  sequence = ArabicNumericSequence
  context = ListContext
  parent = OrderedListItemLevel1Labeller
  type = "list_item_ordered_level2"
  template = OrderedListItemLevel2LabelTemplate
  definition "Labeller for second level ordered list items"
}

// Level 3: i), ii), iii)...
instance OrderedListItemLevel3LabelTemplate < LabelTemplate {
  pattern = "%n)"
  definition "Template for third level ordered list items"
}

instance OrderedListItemLevel3Labeller < Labeller {
  id = "ordered_list_item_level3_labeller"
  sequence = RomanLowerNumericSequence
  context = ListContext
  parent = OrderedListItemLevel2Labeller
  type = "list_item_ordered_level3"
  template = OrderedListItemLevel3LabelTemplate
  definition "Labeller for third level ordered list items"
}

// Level 4: a), b), c)... (repeats first level)
instance OrderedListItemLevel4LabelTemplate < LabelTemplate {
  pattern = "%n)"
  definition "Template for fourth level ordered list items"
}

instance OrderedListItemLevel4Labeller < Labeller {
  id = "ordered_list_item_level4_labeller"
  sequence = AlphaLowerNumericSequence
  context = ListContext
  parent = OrderedListItemLevel3Labeller
  type = "list_item_ordered_level4"
  template = OrderedListItemLevel4LabelTemplate
  definition "Labeller for fourth level ordered list items"
}
```


© 2025 Ribose Asia Limited