LutaML — XML Namespaces

RS 3006:2025, Version 1.0

Ronald Tse

Unrestricted
February 22, 2025
RS 3006:2025, Version 1.0
LutaML

Ribose

Contents

Introduction

This document specifies how lutaml-model implements XML namespace handling, including namespace inheritance, prefix resolution, and attribute namespace management. It serves as both a technical specification and a practical guide for developers using lutaml-model's XML namespace capabilities.

1. Scope

This document specifies:

- XML namespace handling in lutaml-model
- Namespace inheritance and resolution rules
- Prefix management and attribute namespace handling
- Best practices and common patterns for namespace usage

2. Normative references

There are no normative references in this document.

3. Terms and definitions

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

3.1. xml namespace PREFERRED

method to avoid element name conflicts by qualifying element and attribute names with namespace IRIs

3.2. default namespace PREFERRED

namespace that applies to unprefixed element names

3.3. prefixed namespace PREFERRED

namespace that applies to qualified names

3.4. namespace scope PREFERRED

range of elements where a namespace declaration is valid, starting from the declaring element and including all child elements

4. XML namespace fundamentals

The W3C XML Namespaces 1.0 specification defines the following key concepts for XML namespace handling:

- Namespace declarations using the `xmlns` attribute
- Default namespaces that apply to unprefixed elements
- Prefixed namespaces that apply to qualified names
- Scoping rules for namespace declarations

The W3C specification defines these core rules for namespace inheritance:

1. Element names inherit the namespace specified by an `xmlns` attribute on themselves or the nearest ancestor element

2. Unprefixed attributes do not inherit any namespace, even with a default namespace declaration
3. Prefixed attributes use the namespace associated with their prefix
4. Namespace declarations remain valid from the declaring element through all descendants

5. Namespace operations

5.1. Model-level

5.1.1. Namespace definition

When a namespace is set at the model level, it becomes the default namespace for that model and all its unnamespaced inner elements:

FIGURE 1

```
xml do
  namespace "{namespace-uri}" # Sets default namespace
end
```

Where,

`namespace-uri`	The URI of the namespace to be used.

5.1.2. Prefix declaration

The prefix is optional.

Prefixes can be declared at the model level to be used by inner elements.

FIGURE 2

```
xml do
  # Mandatory prefix declaration
  prefix "{prefix}"

  # Declares optional prefix when needed
  prefix "{optional-prefix}", optional: true
```

```
    end
```

Where,

`prefix`	The prefix to be used for the element.
`optional-prefix`	(optional) The prefix to be used for the element when needed.

5.2. Mapping-level

5.2.1. Element mapping

For XML elements that need to be mapped to model attributes, the `map_element` is used to specify how XML elements correspond to attributes in the model.

FIGURE 3

```
    xml do
      # ... model-level settings
      map_element "{xml-element-name}",
        to: {attribute-name},
        namespace: {namespace}, # Optional
        prefix: {prefix}, # Optional
    end
```

Where,

`xml-element-name`	The name of the element in the XML.
`attribute-name`	The name of the attribute in the model.
`namespace`	(optional) The namespace to be applied to the element.
`prefix`	(optional) The prefix to be used for the element.

The behavior is as follows:

1. If the target model attribute has no namespace, it is treated as an unprefixed element, meaning that adopts the current namespace.
2. If the target model attribute has a namespace:
   a. If the target model attribute namespace is the same as the parent element, then it is identical to the unprefixed element.
   b. If the target model attribute namespace is different from the parent element:
      i. If the target element is to be unprefixed and inherits the namespace from the parent element (which causes inner prefixing), then the mapping needs to specify `namespace: inherit` to alter the target model's namespace. The element will receive an `xmlns` declaration for the target model attribute namespace at the target element (not the parent element).

ii. If the target element is to be unprefixed with an empty namespace, then the mapping needs to specify `namespace: nil` to clear the namespace. The element will be an unprefixed element.

iii. If the target element is to be prefixed, the prefix must be either declared in the mapping (through mapping-level `prefix:`) or the target model (through model-level `prefix`). The namespace declaration with prefix will be applied to the parent element and the target element will be prefixed.

5.2.2. Attribute mapping

For XML attributes that need to be mapped to model attributes, the `map_attribute` is used to specify how XML attributes correspond to attributes in the model.

FIGURE 4

```
xml do
  # ... model-level settings
  map_attribute "{xml-attribute-name}",
    to: {attribute-name},
    namespace: {namespace}, # Optional
    prefix: {prefix}, # Optional
end
```

Where,

`xml-attribute-name`	The name of the attribute in the XML.
`attribute-name`	The name of the attribute in the model.
`namespace`	(optional) The namespace to be applied to the attribute.
`prefix`	(optional) The prefix to be used for the attribute.

The behavior is as follows:

1. If the target attribute model has no namespace:
   a. If the target XML attribute is to be unprefixed, nothing needs to be done.
   b. If the target XML attribute is to be a prefixed XML attribute, the namespace and the prefix must be provided either by the target attribute model (through model-level `namespace` and `prefix`) or the mapping (through mapping-level`namespace` and `prefix`). The namespace declaration with prefix will be applied to the XML element that contains the attribute, and the XML attribute will be prefixed.
2. If the target attribute has a namespace:
   a. If it is identical to the parent element's namespace:
      i. If it is to be an unprefixed XML attribute:

            A. The namespace has to be cleared (see [attribute-namespace] for details). Then `namespace: nil` must be specified in the mapping.

        ii. If it is to be a prefixed XML attribute:

            A. Either the mapping or the target attribute must provide the prefix. Then the namespace declaration with prefix will be applied to the XML attribute. This prefixed namespace will be declared at the parent XML element.

    b. If it is different from the parent element's namespace:

        i. If it is to be an unprefixed XML attribute:

            A. The namespace has to be cleared (see [attribute-namespace] for details). Then `namespace: nil` must be specified in the mapping.

        ii. If it is to be a prefixed XML attribute:

            A. The namespace declaration should be set to `namespace: :inherit` to inherit the parent model's namespace. The prefixed namespace will be declared at the parent XML element. Either the mapping or the target attribute must provide the prefix (or the optional `prefix`).

6. Scenarios

6.1. Element namespace handling

6.1.1. Namespace inheritance

When the attribute's model shares the same namespace, it inherits the namespace from the parent without needing prefixes.

FIGURE 5

```
module Ceramic
  class Ceramic < Lutaml::Model::Serializable
    attribute :category, Category # Same namespace

    xml do
      namespace "http://example.com/ceramic"
      map_element "category", to: :category # No prefix
needed
    end
  end
```

```
    class Category < Lutaml::Model::Serializable
      xml do
        namespace "http://example.com/ceramic"
      end
    end
  end
```

Results in:

FIGURE 6

```
<ceramic xmlns="http://example.com/ceramic">
  <category>Ornamental</category> <!-- Same namespace, no
prefix -->
</ceramic>
```

6.1.2. Different namespaces

When the attribute’s model does not share the same namespace, the attribute will be under a different namespace.

FIGURE 7

```
module Ceramic
  class Ceramic < Lutaml::Model::Serializable
    attribute :potter, Potter # Different namespace

    xml do
      namespace "http://example.com/ceramic"
      map_element "potter", to: :potter
    end
  end

  class Potter < Lutaml::Model::Serializable
    attribute :name, :string

    xml do
      namespace "http://example.com/potter"
      map_element "name", to: :name
    end
  end
```

```
end
```

Results in:

FIGURE 8

```
<ceramic xmlns="http://example.com/ceramic">
  <potter xmlns="http://example.com/potter">
    <name>Alice Perrin</name>
  </potter>
</ceramic>
```

6.1.3. Different namespaces with prefix

When the attribute's model does not share the same namespace, it can be given a prefix at the parent level to reference the different namespace.

FIGURE 9

```
class Ceramic < Lutaml::Model::Serializable
  attribute :potter, Potter # Different namespace

  xml do
    namespace "http://example.com/ceramic"
    map_element "potter", to: :potter, prefix: "p"
  end
end

class Potter < Lutaml::Model::Serializable
  attribute :name, :string

  xml do
    namespace "http://example.com/potter"
    map_element "name", to: :name
  end
end
```

Results in:

FIGURE 10

```
<ceramic xmlns="http://example.com/ceramic"
         xmlns:p="http://example.com/potter">
  <p:potter>
    <p:name>Alice Perrin</p:name>
  </p:potter>
</ceramic>
```

6.2. Attribute namespace handling

6.2.1. General

Attributes differ from Elements when namespaces are involved.

> A default namespace declaration applies to all unprefixed element names within its scope. Default namespace declarations do not apply directly to attribute names; the interpretation of unprefixed attributes is determined by the element on which they appear.
>
> If there is a default namespace declaration in scope, the expanded name corresponding to an unprefixed element name has the IRI of the default namespace as its namespace name. If there is no default namespace declaration in scope, the namespace name has no value. The namespace name for an unprefixed attribute name always has no value. In all cases, the local name is local part (which is of course the same as the unprefixed name itself).
>
> — >

A namespaced attribute in XML requires a prefix as per [w3c-xml11].

It is also important to understand that the interpretation of unprefixed attributes is determined by the element on which they appear.

An attribute when assigned a type that is a model can have multiple scenarios:

1. Attribute has no namespace
   a. No namespace, no prefix
   b. Adopt the model's namespace if the parent element is under a prefixed namespace
   c. Add a namespace, with or without prefix
2. Same namespace as the model that includes it
   a. Use as an unprefixed attribute (clear namespace)
   b. Retain that namespace, with prefix
   c. Change that namespace, with prefix
3. Different namespace from the model that includes it
   a. Retain that namespace, with prefix
   b. Change that namespace, to another or the current namespace, with prefix
   c. Clear that namespace and use it as an unprefixed attribute

6.2.2. Scenario 1: Attribute has no namespace

6.2.3. Scenario 1.1: No namespace, no prefix

When the attribute has no namespace, and the parent mapping does not have specify a namespace, the attribute is an unprefixed attribute.

EXAMPLE 1

```
class Ceramic < Lutaml::Model::Serializable
  attribute :code, String

  xml do
    namespace "http://example.com/ceramic"
    map_attribute "code", to: :code # No namespace, no prefix
  end
end

class CommonCode < Lutaml::Model::Serializable
  attribute :code, String

  xml do
    map_attribute "code", to: :code
  end
end
```

Corresponding XML:

```
<ceramic xmlns="http://example.com/ceramic" code="Value">
```

==== Scenario 1.2: Adopt the model's namespace if the parent element is under a prefixed namespace

When the attribute has no namespace, it adopts the model's namespace if the parent element is under a prefixed namespace.

Syntax:

```
xml do
  namespace {namespace}
  prefix {prefix}

  map_attribute {xml-attribute-name}, to: {attribute-name}
  # Add more attributes as necessary
end
```

Where,

`namespace`	The namespace to be applied to the element.
`prefix`	The prefix to be used for the element.
`xml-attribute-name`	The name of the attribute in the XML.

`attribute-name`	The name of the attribute in the model.

FIGURE 11

```
class Ceramic < Lutaml::Model::Serializable
  attribute :code, String

  xml do
    root "ceramic"
    namespace "http://example.com/ceramic"
    prefix "c"
    map_attribute "code", to: :code
  end
end

class CommonCode < Lutaml::Model::Serializable
  attribute :code, String

  xml do
    root "common_code"
    namespace "http://example.com/common"
    map_attribute "code", to: :code
  end
end
```

Corresponding XML:

FIGURE 12

```
<c:ceramic xmlns:c="http://example.com/ceramic" c:code=
"Value">
```

EXAMPLE 2
==== Scenario 1.3: Add a namespace, with or without prefix

When the attribute has no namespace, it can be given a namespace and a prefix.

Syntax:

```
xml do
  # ...
  map_attribute "xml-attribute-name",
    to: {attribute-name},
    prefix: {prefix}, # Optional
    namespace: {namespace}
```

```
end
```

Where,

`xml-attribute-name`	The name of the attribute in the XML.
`attribute-name`	The name of the attribute in the model.
`prefix`	(optional) The prefix to be used for the attribute.
`namespace`	The namespace to be applied to the attribute.

FIGURE 13

```
class Ceramic < Lutaml::Model::Serializable
  attribute :code, String

  xml do
    root "ceramic"
    namespace "http://example.com/ceramic"
    map_attribute "code", to: :code, prefix: "c", namespace:
"http://example.com/common"
  end
end

class CommonCode < Lutaml::Model::Serializable
  attribute :code, String

  xml do
    root "common_code"
    namespace "http://example.com/common"
    map_attribute "code", to: :code
  end
end
```

Corresponding XML:

FIGURE 14

```
<ceramic xmlns="http://example.com/common"
          xmlns:c="http://example.com/common" c:code="Value">
```

6.2.4. Scenario 2: Same namespace as the model that includes it

When the attribute is a model that has a namespace, and the parent element has the same namespace, the attribute inherits that namespace.

6.2.5. Scenario 2.1: Use as an unprefixed attribute (clear namespace)

When the attribute has no namespace, it is used as an unprefixed attribute.

EXAMPLE

```
class Ceramic < Lutaml::Model::Serializable
  attribute :code, CommonCode

  xml do
    root "ceramic"
    namespace "http://example.com/ceramic"
    map_attribute "code", to: :code
  end
end

class CommonCode < Lutaml::Model::Serializable
  attribute :code, String

  xml do
    root "common_code"
    namespace "http://example.com/ceramic" # Same namespace as parent
    map_attribute "code", to: :code # Notice that the XML is an
unprefixed attribute.
  end
end
```

Notice that the XML is an unprefixed attribute.

Corresponding XML:

```
<ceramic xmlns="http://example.com/ceramic" code="Value">
```

6.2.6. Scenario 2.2: Retain that namespace, with prefix

When the attribute model has a namespace and you want to retain that namespace, the namespace must be specified.

Syntax:

FIGURE 15

```
xml do
  # ...
  prefix "c", optional: true
  map_attribute "code", to: :code, namespace: :inherit
end
```

EXAMPLE

```
class Ceramic < Lutaml::Model::Serializable
  attribute :code, CommonCode

  xml do
```

```
    root "ceramic"
    namespace "http://example.com/ceramic"
    prefix "c", optional: true
    map_attribute "code", to: :code, namespace: :inherit
  end
end

class CommonCode < Lutaml::Model::Serializable
  attribute :code, String

  xml do
    root "common_code"
    namespace "http://example.com/ceramic" # Same namespace as parent
    map_attribute "code", to: :code
  end
end
```

Corresponding XML:

```
<c:ceramic xmlns="http://example.com/ceramic" c:code="Value">
```

6.2.7. Scenario 2.3: Change that namespace, with prefix

When the attribute model has a namespace and you want to change that namespace, the new namespace and its prefix must be specified.

Syntax:

FIGURE 16

```
xml do
  # ...
  prefix "d", optional: true
  map_attribute "code", to: :code, namespace: {namespace}
end
```

EXAMPLE

```
class Ceramic < Lutaml::Model::Serializable
  attribute :code, CommonCode

  xml do
    root "ceramic"
    namespace "http://example.com/ceramic"
    # Change to different namespace
    map_attribute "code", to: :code, prefix: "d", namespace: "http://
example.com/common"
  end
end

class CommonCode < Lutaml::Model::Serializable
  attribute :code, String

  xml do
```

```
    root "common_code"
    namespace "http://example.com/ceramic" # Same namespace as parent
    map_attribute "code", to: :code
  end
end
```

Corresponding XML:

```
<ceramic xmlns="http://example.com/ceramic"
  xmlns:d="http://example.com/common" d:code="Value">
```

6.2.8. Scenario 3: Different namespace from the model that includes it

This is the case where the attribute is a model that has a namespace different from the parent element, the attribute must be given a prefix.

6.2.9. Scenario 3.1: Retain that namespace, with prefix

When the attribute has a different namespace from the model that includes it, it must be given a prefix.

Syntax:

FIGURE 17

```
map_attribute "xml-attribute-name", to: {attribute-name},
prefix: {prefix}
```

Where,

`xml-attribute-name`	The name of the attribute in the XML.
`attribute-name`	The name of the attribute in the model.
`prefix`	The prefix to be used for the attribute.

EXAMPLE

```
class Ceramic < Lutaml::Model::Serializable
  attribute :id, Identifier

  xml do
    namespace "http://example.com/ceramic"
    map_attribute "id", to: :id, prefix: "c"
  end
end
```

Results in:

```
<ceramic xmlns="http://example.com/ceramic"
         xmlns:c="http://example.com/identifier"
```

```
        c:id="1234">
```

6.2.10. Scenario 3.2: Change that namespace, with prefix

When the attribute has a different namespace from the model that includes it, it can be given a different namespace and a prefix.

Syntax:

FIGURE 18

```
map_attribute "xml-attribute-name", to: {attribute-name},
prefix: {prefix}, namespace: {namespace}
```

Where,`xml-attribute-name`:: The name of the attribute in the XML.`attribute-name`:: The name of the attribute in the model.`prefix`:: The prefix to be used for the attribute.`namespace`:: The namespace to be applied to the attribute.

EXAMPLE

```
class Ceramic < Lutaml::Model::Serializable
  attribute :code, CommonCode

  xml do
    namespace "http://example.com/ceramic"
    # Explicit namespace and prefix
    map_attribute "code", to: :code, prefix: "d", namespace: "http://
example.com/common"
  end
end
```

Results in:

```
<ceramic xmlns="http://example.com/ceramic"
         xmlns:d="http://example.com/common"
         d:code="ABC">
```

6.2.11. Scenario 3.3: Clear that namespace and use it as an unprefixed attribute

When the attribute has a different namespace from the model that includes it, it can be cleared to use as an unprefixed attribute.

Syntax:

FIGURE 19

```
xml do
  # ...
  map_attribute "xml-attribute-name", to: {attribute-name},
namespace: nil
```

```
    end
```

Where,

`xml-attribute-name`	The name of the attribute in the XML.
`attribute-name`	The name of the attribute in the model.

EXAMPLE

```
class Ceramic < Lutaml::Model::Serializable
  attribute :code, CommonCode

  xml do
    root "ceramic"
    namespace "http://example.com/ceramic"
    map_attribute "code", to: :code, namespace: nil
  end
end

class CommonCode < Lutaml::Model::Serializable
  attribute :code, String

  xml do
    root "common_code"
    namespace "http://example.com/identifier"
    map_attribute "code", to: :code
  end
end
```

Corresponding XML:

```
<ceramic xmlns="http://example.com/ceramic" code="Value">
```

7. Namespace scenarios

7.1. Default namespace inheritance

When elements share the same namespace, they inherit the namespace from their parent without needing prefixes.

FIGURE 20

```
    class Ceramic < Lutaml::Model::Serializable
      attribute :category, Category
    xml do
      namespace "http://example.com/ceramic"
      map_element "category", to: :category # No prefix needed
    end
  end

  class Category < Lutaml::Model::Serializable
    xml do
      namespace "http://example.com/ceramic" # Same namespace
  as parent
    end
  end
```

Results in:

FIGURE 21

```
<ceramic xmlns="http://example.com/ceramic">
  <category>Ornamental</category>
</ceramic>
```

7.2. Explicit prefix declaration

When elements use different namespaces, prefixes must be explicitly declared.

FIGURE 22

```
class Ceramic < Lutaml::Model::Serializable
  attribute :potter, Potter

  xml do
    namespace "http://example.com/ceramic"
    map_element "potter", to: :potter, prefix: "p" #
Different namespace needs prefix
  end
end

class Potter < Lutaml::Model::Serializable
  xml do
    namespace "http://example.com/potter"
  end
```

```
end
```

Results in:

FIGURE 23

```
<ceramic xmlns="http://example.com/ceramic"
         xmlns:p="http://example.com/potter">
  <p:potter>
    <p:name>Alice Perrin</p:name>
  </p:potter>
</ceramic>
```

7.3. Attribute namespace handling

Attributes have special namespace handling rules: 1. Unprefixed attributes have no namespace 2. Prefixed attributes must declare their namespace

FIGURE 24

```
class Ceramic < Lutaml::Model::Serializable
  attribute :type, :string
  attribute :id, Identifier

  xml do
    namespace "http://example.com/ceramic"
    map_attribute "type", to: :type  # No namespace
    map_attribute "id", to: :id, prefix: "c"  # Different
namespace
  end
end
```

Results in:

FIGURE 25

```
<ceramic xmlns="http://example.com/ceramic"
         xmlns:c="http://example.com/identifier"
         type="Fine Porcelain"
```

```
              c:id="1234">
```

7.4. Nested namespace changes

Child elements can override their parent's namespace and establish a new default namespace for their subtree.

FIGURE 26

```
class Ceramic < Lutaml::Model::Serializable
  attribute :production_site, ProductionSite

  xml do
    namespace "http://example.com/ceramic"
    map_element "production_site", to: :production_site
  end
end

class ProductionSite < Lutaml::Model::Serializable
  attribute :name, :string
  attribute :website, SiteUrl

  xml do
    namespace "http://example.com/production" # Different
from parent
    map_element "name", to: :name
    map_element "website", to: :website, prefix: "s"
  end
end
```

Results in:

FIGURE 27

```
<ceramic xmlns="http://example.com/ceramic">
  <production_site xmlns="http://example.com/production"
                   xmlns:s="http://example.com/url">
    <name>Bernardaud Factory</name>
    <s:website>http://www.bernardaud.com</s:website>
  </production_site>
```

```
</ceramic>
```

7.5. Multiple namespace declarations

Complex elements may need to declare multiple namespaces when they reference elements from different namespaces.

FIGURE 28

```
class Ceramic < Lutaml::Model::Serializable
  attribute :id, Identifier
  attribute :potter, Potter
  attribute :category, Category

  xml do
    namespace "http://example.com/ceramic"
    map_attribute "id", to: :id, prefix: "c"
    map_element "potter", to: :potter, prefix: "p"
    map_element "category", to: :category # Same namespace
  end
end
```

Results in:

FIGURE 29

```
<ceramic xmlns="http://example.com/ceramic"
         xmlns:c="http://example.com/identifier"
         xmlns:p="http://example.com/potter"
         c:id="1234">
  <p:potter>
    <p:name>Alice Perrin</p:name>
  </p:potter>
  <category>Ornamental</category>
```

```
</ceramic>
```

7.6. Collection element namespaces

Collections maintain proper namespace handling for each element.

FIGURE 30

```
class ProductionSite < Lutaml::Model::Serializable
  attribute :glazes_produced, :string, collection: true

  xml do
    namespace "http://example.com/production"
    map_element "glazes_produced", to: :glazes_produced
  end
end
```

Results in:

FIGURE 31

```
<production_site xmlns="http://example.com/production">
  <glazes_produced>Celadon</glazes_produced>
  <glazes_produced>Crystalline</glazes_produced>
```

```
</production_site>
```

8. Advanced namespace features

8.1. Automatic namespace merging

When multiple prefixes map to the same namespace URI, lutaml-model automatically merges them to use a single prefix declaration. This optimization reduces XML verbosity and improves namespace management.

FIGURE 32

```
class Ceramic < Lutaml::Model::Serializable
  attribute :id, Identifier
  attribute :code, CommonCode

  xml do
    namespace "http://example.com/ceramic"
    map_attribute "id", to: :id, prefix: "c", namespace:
"http://example.com/common"
    map_attribute "code", to: :code, prefix: "d", namespace:
"http://example.com/common"
  end
end
```

Results in a single prefix for the common namespace:

FIGURE 33

```
<ceramic xmlns="http://example.com/ceramic"
         xmlns:c="http://example.com/common"
         c:id="1234"
```

```
              c:code="ABC">
```

8.2. Polymorphic collection mapping

Polymorphic collections are automatically handled by preserving each element’s namespace while maintaining collection semantics.

FIGURE 34

```
class CeramicCollection < Lutaml::Model::Serializable
  attribute :items, Ceramic, collection: true # Can contain
any subclass of Ceramic

  xml do
    namespace "http://example.com/collection"
    map_element "items", to: :items
  end
end

# Different ceramic types with their own namespaces
class Vase < Ceramic
  xml do
    namespace "http://example.com/vase"
  end
end

class Bowl < Ceramic
  xml do
    namespace "http://example.com/bowl"
  end
end
```

Collection with mixed types preserves individual namespaces:

FIGURE 35

```
<collection xmlns="http://example.com/collection">
  <items>
    <vase xmlns="http://example.com/vase">...</vase>
    <bowl xmlns="http://example.com/bowl">...</bowl>
  </items>
```

```
</collection>
```

8.3. Namespace prefix conflict resolution

Prefix conflicts are automatically resolved by generating unique prefixes when the same prefix is requested for different namespaces:

FIGURE 36

```
class Ceramic < Lutaml::Model::Serializable
  attribute :id, Identifier
  attribute :metadata, MetaData

  xml do
    namespace "http://example.com/ceramic"
    map_attribute "id", to: :id, prefix: "meta", namespace:
"http://example.com/identifier"
    map_element "metadata", to: :metadata, prefix: "meta",
namespace: "http://example.com/metadata"
  end
end
```

Automatically resolves to unique prefixes:

FIGURE 37

```
<ceramic xmlns="http://example.com/ceramic"
         xmlns:meta1="http://example.com/identifier"
         xmlns:meta2="http://example.com/metadata"
         meta1:id="1234">
  <meta2:metadata>...</meta2:metadata>
</ceramic>
```

Annex A
(normative)

Full example

FIGURE A.1

```
module Common
  # Simple content, unique namespace
  class Identifier < Lutaml::Model::Serializable
    attribute :name, :string

    xml do
      root "identifier"
      map_content to: :name
      namespace "http://example.com/identifier"
    end
  end

  # Simple content, unique namespace
  class SiteUrl < Lutaml::Model::Serializable
    attribute :url, :string

    xml do
      root "website"
      namespace "http://example.com/url"
      map_content to: :url
    end
  end
end

module Pottery
  # Simple content, unique namespace
  class Potter < Lutaml::Model::Serializable
    attribute :name, :string

    xml do
      root "potter"
      namespace "http://example.com/potter"
    end
  end
end

module Production
  # Nested content, unique namespace
```

```ruby
    class ProductionSite < Lutaml::Model::Serializable
      attribute :name, :string
      attribute :glazes_produced, :string, collection: true
      attribute :location, Location # Same namespace
      attribute :website, SiteUrl

      xml do
        root "production_site"
        namespace "http://example.com/production"
        map_element "name", to: :name
        map_element "glazes_produced", to: :glazes_produced
        # Same namespace, no need prefix
        map_element "location", to: :location
        # Different namespace
        map_element "established_at", to: :established_at
        # Different namespace
        map_element "website", to: :website, prefix: "s"
      end
    end

    # Simple content, sharing namespace with ProductionSite
    class Location < Lutaml::Model::Serializable
      attribute :address, :string
      attribute :city, :string
      attribute :country, :string

      xml do
        root "location"
        namespace "http://example.com/production"
      end
    end
  end

  module Ceramic
    # Complex content model
    class Ceramic < Lutaml::Model::Serializable
      attribute :type, :string
      attribute :composition, Composition
      attribute :id, Identifier
      attribute :glaze, :string
      attribute :category, Category
      attribute :production_site, ProductionSite
      attribute :potter, Potter

      xml do
        root "ceramic"
        namespace "http://example.com/ceramic"

        # Attribute with no namespace
        map_attribute "type", to: :type
        # Attribute with same namespace, no need prefix
        map_attribute "composition", to: :composition
        # Attribute with different namespace, need prefix
        map_attribute "id", to: :id, prefix: "c"
        # Element with no namespace
        map_element "glaze", to: :glaze
        # Element with same namespace, no need prefix
        map_element "category", to: :category
        # Element with different namespace with no prefix
        map_element "production_site", to: :production_site
        # Element with different namespace with prefix
        map_element "potter", to: :potter, prefix: "p"
      end
    end

    # Simple content, sharing namespace with Ceramic
    class Composition < Lutaml::Model::Serializable
      attribute :name, :string

      xml do
        root "composition"
        map_content to: :name
        namespace "http://example.com/ceramic"
      end
    end

    # Simple content, sharing namespace with Ceramic
    class Category < Lutaml::Model::Serializable
      attribute :name, :string

      xml do
        root "category"
        map_content to: :name
        namespace "http://example.com/ceramic"
      end
    end
  end

  location = Production::Location.new(address: "15 Rue du
  Temple", city: "Limoges", country: "France")
  production_site = Production::ProductionSite.new(name:
  "Bernardaud Factory", glazes_produced: ["Celadon",
  "Crystalline"], location: location)
  potter = Pottery::Potter.new(name: "Alice Perrin")
  composition = Ceramic::Composition.new(name: "Porcelain")
  identifier = Common::Identifier.new(name: "1234")
  category = Ceramic::Category.new(name: "Ornamental")
  ceramic = Ceramic::Ceramic.new(
    type: "Fine Porcelain",
    glaze: "Celadon",
    production_site: production_site,
    potter: potter,
    composition: composition,
    id: identifier,
    category: category,
  )

  puts ceramic.to_xml
  # =>
  <<~XML
    <?xml version="1.0" encoding="UTF-8"?>
    <ceramic xmlns="http://example.com/ceramic"
              xmlns:c="http://example.com/identifier"
              xmlns:p="http://example.com/potter"
              type="Fine Porcelain"
              composition="Porcelain"
```

```
              c:id="1234"
              category="Ornamental">
    <!-- Default namespace -->
    <glaze>Celadon</glaze>
    <!-- Same namespace, no need prefix -->
    <category>Ornamental</category>
    <!-- Different namespace, change default namespace
inside -->
    <production_site xmlns="http://example.com/production"
                      xmlns:s="http://example.com/url">
      <name>Bernardaud Factory</name>
      <glazes_produced>Celadon</glazes_produced>
      <glazes_produced>Crystalline</glazes_produced>
      <!-- Same namespace, no need prefix -->
      <location>
        <address>15 Rue du Temple</address>
        <city>Limoges</city>
        <country>France</country>
      </location>
      <!-- Different namespace -->
      <established_at xmlns="http://example.com/url">2010</
established_at>
      <!-- Different namespace, parent has explicitly set
prefix -->
      <s:website>http://www.bernardaud.com</s:website>
    </production_site>
    <!-- Different namespace, parent has explicitly set
prefix -->
    <p:potter>
      <p:name>Alice Perrin</p:name>
    </p:potter>
  </ceramic>
XML
```