LutaML — Model Register

RS 3004:2025, Version 1.0

Ronald Tse

Unrestricted
February 20, 2025
RS 3004:2025, Version 1.0
LutaML

Ribose

Contents

Ribose Inc.

167-169 Great Portland Street
5th Floor
London
W1W 5PF
United Kingdom

copyright@ribose.com
www.ribose.com

1. Scope

This document specifies the model register capabilities in LutaML Model, which enable:

- Dynamic model attribute modification
- Model hierarchy manipulation
- Cross-model attribute type substitution

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. model register PREFERRED

registry that maintains and manages dynamic model configurations

3.2. model path PREFERRED

path expression that identifies a specific model or attribute within a model hierarchy

3.3. model hierarchy PREFERRED

tree structure that represents the relationships between different models and their attributes

3.4. attribute substitution PREFERRED

process of replacing one attribute type with another within a model

3.5. model tree PREFERRED

collection of interconnected models that form a hierarchical structure

4. Principles of model registers

4.1. General

A LutaML model register provides a way to dynamically modify and reconfigure model hierarchies without changing the original model definitions.

Common use cases include:

- Replacing attribute types in models

- Swapping model subtrees with alternative implementations
- Adding or removing attributes from models
- Converting between different model namespaces

4.2. Architecture

A model register acts as a dynamic configuration layer that sits above the static model definitions:

FIGURE 1

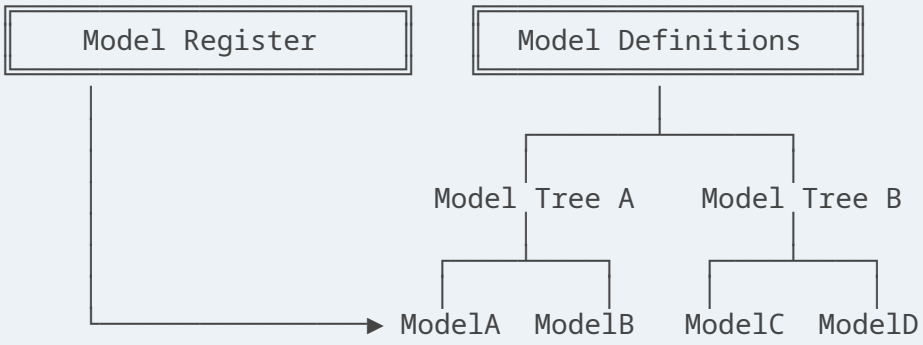

FIGURE 2: Subtree Swap

```
Before:                          After:
  ModelA                          ModelA
  ├─ModelB                        ├─ModelB
  │ └─ModelC                      │ └─NewSubtree
  └─ModelD                        └─ModelD
```

FIGURE 3: Single Model Swap

```
Before:                          After:
  ModelA                          ModelA
  ├─ModelB                        ├─NewModel
```

```
    └─ModelC                        └─ModelC
```

FIGURE 4: Global Type Substitution

```
Before:                          After:
  Document                         Document
  ├─NameString                     ├─StructuredName
  └─Author                         └─Author
    └─NameString                     └─StructuredName
```

4.3. Basic usage

A model register is created and populated with model definitions:

FIGURE 5

```ruby
class StorageRecordCollection < Lutaml::ModelCollection
  instances :items, StorageRecord
end

class StorageRecord
  attribute :ceramic_info, GeneralCeramicInfo
end

class GeneralCeramicInfo < Lutaml::Model::Serializable
  attribute :material, :string
  attribute :production_date, :date
end

register = Lutaml::ModelRegister.new
register.register_model_tree(StorageRecordCollection)
```

4.4. Model paths

Model paths identify specific locations within a model hierarchy using the LutaML Path syntax.

Common path patterns:

- Single model: `ModelName`
- Nested model: `ParentModel > ChildModel`
- Model attribute: `Model.attribute`

- Nested attribute: `ParentModel > ChildModel.attribute`

EXAMPLE: Given this model hierarchy:

```
class Publication
  attribute :metadata, Metadata
end

class Book < Publication
  attribute :chapters, Chapter, collection: true
end

class Metadata
  attribute :title, :string
  attribute :date, :date
end
```

Valid model paths include:

- `Publication` — References the Publication model
- `Publication > Book` — References Book as a child of Publication
- `Publication.metadata` — References the metadata attribute
- `Publication > Book.chapters` — References the chapters collection

5. Model registration

5.1. General

Models must be registered before they can be dynamically modified.

5.2. Registration methods

5.2.1. Register individual model

Registers a single model class.

Syntax:

FIGURE 6

```
register.register_model(ModelClass)
```

EXAMPLE

```
register.register_model(StorageRecord)
```

5.2.2. Register model tree

Registers a model and all its dependent models.

Syntax:

FIGURE 7

```
register.register_model_tree(RootModelClass)
```

EXAMPLE

```
register.register_model_tree(StorageRecordCollection)
# Automatically registers:
# - StorageRecordCollection
# - StorageRecord
# - GeneralCeramicInfo
```

6. Dynamic modifications

6.1. Attribute type substitution

6.1.1. General

Replace an attribute's type with another model type.

6.1.2. Single attribute substitution

Replaces a specific attribute instance's type.

Syntax:

FIGURE 8

```
    register.register_dynamic_attribute(
      model_path: "Model > SubModel.attribute",
      attribute: :attribute_name,
      type: NewAttributeType
    )
```

EXAMPLE

```
class VaseCeramicInfo < GeneralCeramicInfo
  attribute :height, :float
  attribute :diameter, :float
end

register.register_dynamic_attribute(
  model_path: "StorageRecordCollection > StorageRecord",
  attribute: :ceramic_info,
  type: VaseCeramicInfo
)
```

6.1.3. Global type substitution

Replaces all instances of a type throughout the model hierarchy.

Syntax:

FIGURE 9

```
    register.register_global_type_substitution(
      from_type: OldType,
      to_type: NewType
    )
```

EXAMPLE

```
# Replace all Mml::Mi instances with Plurimath equivalents
register.register_global_type_substitution(
  from_type: Mml::Mi,
  to_type: Plurimath::Math::Symbols::Symbol
)
```

6.2. Model tree operations

6.2.1. Subtree replacement

Replaces an entire subtree in the model hierarchy.

Syntax:

FIGURE 10

```
    register.replace_subtree(
      model_path: "Path > To > Subtree",
      new_subtree: NewRootModel
    )
```

EXAMPLE

```
class NewMetadataTree < Lutaml::Model::Serializable
  attribute :title, :string
  attribute :description, :string
end

register.replace_subtree(
  model_path: "Document > Metadata",
  new_subtree: NewMetadataTree
)
```

6.2.2. Attribute modification

Add or remove attributes from a model.

Syntax:

FIGURE 11

```
    # Add attribute
    register.add_attribute(
      model_path: "Model",
      attribute: :new_attribute,
      type: AttributeType
    )

    # Remove attribute
    register.remove_attribute(
      model_path: "Model",
      attribute: :old_attribute
    )
```

EXAMPLE

```
register.add_attribute(
  model_path: "StorageRecord",
  attribute: :last_modified,
  type: :datetime
)

register.remove_attribute(
```

```
  model_path: "StorageRecord",
  attribute: :deprecated_field
)
```

6.3. Model resolution

6.3.1. General

After configuration, models are retrieved from the register using the resolve method.

6.3.2. Basic resolution

Retrieves a configured model class by name.

Syntax:

FIGURE 12

```
ModelClass = register.resolve("ModelName")
```

EXAMPLE

```
StorageRecordClass = register.resolve("StorageRecord")
record = StorageRecordClass.new(
  ceramic_info: VaseCeramicInfo.new(
    material: "clay",
    height: 10.0
  )
)
```

6.3.3. Path-based resolution

Retrieves a model class using a model path.

Syntax:

FIGURE 13

```
ModelClass = register.resolve_path("Path > To > Model")
```

EXAMPLE

```
VaseRecord = register.resolve_path(
  "StorageRecordCollection > StorageRecord"
```

```
)
```

7. Example scenarios

7.1. Namespace conversion

This example demonstrates converting models between different namespaces.

FIGURE 14

```ruby
# Original MathML models
module Mml
  class Expression < Lutaml::Model::Serializable
    attribute :operator, Mi
  end

  class Mi < Lutaml::Model::Serializable
    attribute :value, :string
  end
end

# Target Plurimath models
module Plurimath
  module Math
    module Symbols
      class Symbol < Lutaml::Model::Serializable
        attribute :value, :string
      end
    end
  end
end

# Register and configure conversion
register = Lutaml::ModelRegister.new
register.register_model_tree(Mml::Expression)

register.register_global_type_substitution(
  from_type: Mml::Mi,
  to_type: Plurimath::Math::Symbols::Symbol
)

# Use converted models
ExpressionClass = register.resolve("Mml::Expression")
expression = ExpressionClass.new(
  operator: Plurimath::Math::Symbols::Symbol.new(
    value: "+"
  )
```

```
)
```

7.2. Dynamic model extension

This example shows extending models with new attributes.

FIGURE 15

```ruby
class BaseDocument < Lutaml::Model::Serializable
  attribute :title, :string
end

class Chapter < Lutaml::Model::Serializable
  attribute :content, :string
end

register = Lutaml::ModelRegister.new
register.register_model_tree(BaseDocument)

# Add versioning attributes
register.add_attribute(
  model_path: "BaseDocument",
  attribute: :version,
  type: :string
)

register.add_attribute(
  model_path: "Chapter",
  attribute: :last_modified,
  type: :datetime
)

# Use extended models
DocumentClass = register.resolve("BaseDocument")
doc = DocumentClass.new(
  title: "Example",
  version: "1.0"
)
```

Annex A
(normative)

Tutorial: Building an adaptive document model

This tutorial demonstrates using model registers to create an adaptive document model system that can be customized for different use cases.

A.1. Step 1: Base document model

- Create initial model hierarchy
- Register models
- Understand basic model relationships

FIGURE A.1

```
# Define base models
class Document < Lutaml::Model::Serializable
  attribute :metadata, Metadata
  attribute :content, Content
end

class Metadata < Lutaml::Model::Serializable
  attribute :title, :string
  attribute :author, :string
  attribute :date, :date
end

class Content < Lutaml::Model::Serializable
  attribute :sections, Section, collection: true
end

class Section < Lutaml::Model::Serializable
  attribute :title, :string
  attribute :body, :string
end
```

```
# Create and populate register
register = Lutaml::ModelRegister.new
register.register_model_tree(Document)
```

A.2. Step 2: Technical documentation extension

- Extend models with new attributes
- Replace attribute types
- Use path-based modifications

FIGURE A.2

```
# Define technical documentation models
class TechnicalMetadata < Metadata
  attribute :version, :string
  attribute :status, :string
end

class CodeSection < Section
  attribute :language, :string
  attribute :code, :string
end

# Configure register
register.register_dynamic_attribute(
  model_path: "Document",
  attribute: :metadata,
  type: TechnicalMetadata
)

register.add_attribute(
  model_path: "Document > Content > Section",
  attribute: :type,
  type: :string
)

# Allow code sections
register.register_dynamic_attribute(
  model_path: "Document > Content",
  attribute: :sections,
  type: CodeSection
)
```

A.3. Step 3: Academic publication extension

- Replace model subtrees

- Add nested attributes
- Handle collections

FIGURE A.3

```
# Define academic models
class AcademicMetadata < Metadata
  attribute :abstract, :string
  attribute :keywords, :string, collection: true
  attribute :references, Reference, collection: true
end

class Reference < Lutaml::Model::Serializable
  attribute :authors, :string, collection: true
  attribute :title, :string
  attribute :journal, :string
  attribute :year, :integer
end

# Configure register
register.replace_subtree(
  model_path: "Document > Metadata",
  new_subtree: AcademicMetadata
)

# Add citation support
register.add_attribute(
  model_path: "Document > Content > Section",
  attribute: :citations,
  type: Reference,
  collection: true
)
```

A.4. Step 4: Global modifications

- Apply global type substitutions
- Manage cross-cutting concerns
- Handle model relationships

FIGURE A.4

```
# Define enhanced types
class EnhancedString < Lutaml::Model::Serializable
  attribute :value, :string
  attribute :language, :string
```

```
    attribute :format, :string
  end

  # Replace all string attributes with enhanced strings
  register.register_global_type_substitution(
    from_type: :string,
    to_type: EnhancedString
  )

  # Add tracking to all models
  register.add_attribute(
    model_path: "*",
    attribute: :created_at,
    type: :datetime
  )

  register.add_attribute(
    model_path: "*",
    attribute: :updated_at,
    type: :datetime
  )
```

A.5. Summary

This tutorial demonstrated:

- Basic model registration and configuration
- Dynamic attribute type substitution
- Model subtree replacement
- Global type modifications
- Cross-cutting attribute addition

The progression shows how model registers enable flexible and maintainable model configurations that can adapt to different requirements while maintaining model consistency.