Preliminaries

the basic structure of the model

In Literate Data Modeling, the main components of interest are typically Classes, Attributes, Models, and Subjects. However, to streamline the model and promote reusability, we introduce a supertype called Component. By defining common attributes and behaviors in the Component class, we can inherit them in the subclasses, ensuring consistency and reducing duplication throughout the model.

We present the Component class first because it is a best practice in modeling to introduce supertypes before their subtypes. This approach allows readers to understand the general concepts and shared properties before delving into the specifics of each specialized component.

Component

An element or building block of the literate data model

normalName — the name of the component, not in camel case

name — The name of the component *(String value)*

qualifiedName
abbreviatedName — a short form of *(QualifiedCamel value)* *(CamelName value)*
the component's name, used for cross references and improved readability.

default: Example *(CamelName value)*
"LDM" is the short form of "Literate Data Model".

oneLiner — A brief, one-line definition or description of the component, suitable for use in a descriptive table of contents. _

elaboration — A more detailed explanation or discussion of *(RichLine value)*
the component _
For Machinery *(RichText value)*
mechanical attributes

isEmbellishment — Indicates whether this component is an embellishment added during post-parsing processing _

default: Note *(Boolean value)*
This attribute is set to true for components that are automatically generated or added during the fleshing out, review, or rendering processes, such as implied attributes or suggested model elements. It helps distinguish embellishments from the core model elements defined in the original LDM source.

AnnotationType

a kind of note, or aside, used to call attention to additional information about some Component.

based_on Literate Data Model

Note

Each LDM declares a set of Annotation Types, with defined labels, emojis, and clearly documented purposes. These are *recognized* or *registered* Annotation Types.

emoji an emoji

emojiName an emoji *(Emoji value)*

emojiUnicode the Unicode for the emoji *(String value)*

label A short label to indicate the purpose of the annotation _ *(Unicode value)*

plural the plural form of the label *(CamelName value)*

default: *(UpperCamel value)*

Purpose the intended reason for the annotation.

ValueType Annotation A note or comment associated with a model element

based_on Component

annotationType **Note** *(Annotation Type value)*

An Annotation is considered to *recognized* if the label is associated with an Annotation Type. otherwise it is *ad hoc*.

label A short label to indicate the purpose of the annotation _

But any short label is valid. *(CamelName value)*

default:

Emoji The content or body of the annotation *(Emoji value)*

default: For Machinery

content isEmbellishment Indicates whether this annotation is an embellishment added during post-parsing processing _ *(RichText value)*

default: **Note** *(Boolean value)*

This attribute is set to true for annotations that are automatically generated or added during the fleshing out, review, or rendering processes, such as suggestions, issues, or diagnostic messages. It helps distinguish embellishment annotations from the annotations defined in the original LDM source.

The Model and its Subjects

LiterateDataModel

A representation of a domain's entities, attributes, and relationships, along with explanatory text and examples

abbreviation: LDM

plural: LiterateDataModels

subtype_of Component bySomething

name
allSubjects

list of all classes in the model, as ordered in the definition of the model. *(UpperCamel value)*

derivation: list of all classes in the model, as ordered in the definition of the model. *(Classes value)*

allClasses

derivation: Modeling Configuration *(Classes value)*

annotationTypes
Preferred Coding Language

the recommended lanquage for expressing derivation, defaults, and constraints *(AnnotationTypes value)*

(Coding Language value)

default: the recommended lanquage for expressing derivation, defaults, and constraints *(Coding Languages value)*

alternate Coding Languages
Preferred Template Language

(Template Language value)

A list of functions that require sophisticated AI-powered implementation *

default: **Subject A specific topic or theme within the model** *(Template Languages value)*

alternate Template Languages
aiFunctions

Subjects
Component bySomething
LiterateDataModel *(String value)*

derivation: Subjects are the chapters an sections of the model. + A subject need not contain any Classes if itâ€™s just expository.

plural:

subtype_of
dependent_of
name
parentSubject

The parent subject, if any, under which this subject is nested _ *(UpperCamel value)*

Classes

The major classes related to this subject, in the order in which they should be presented _ *(Subject value)*

Issue *(Classes value)*

define chapter, section, subsection as levels?

Each Class should be followed first by the classes that are dependent on it, and then By its subtype classes.
childSubjects

Any child subjects nested under this subject, in the order in which they should be presented _
Subject
DSL: the Classes within a Subject are always displayed before the childSubjects.

SubjectArea A main topic or area of focus within the model, containing related subjects and classes *(Subjects value)*

SubjectAreas

inverse:

CLASS_NAME:

ATTRIBUTE_NAME:
parentSubject.

plural:

subtype_of Subject bySomething

where: parentSubject is absent

Classes

Class

A key entity or object type in the model, often corresponding to a real-world concept

plural: Classes

subtype_of Component bySomething

pluralForm

the normal English plural form of the name of the Class

Might be Books for the Book class or other regular plurals. + But also might be People for Person.

(*UpperName value*)

Note

When inputting a model, you will rarely need to specify the plural form. The input program will just look it up.

default:

basedOn

the Class or Classes on which this class is dependent

This is solely based on **Existence Dependency**. A true dependent entity cannot logically exist without the related parent entity. For instance, an Order Item cannot exist without an Order. If removing the parent entity logically implies removing the dependent entity, then it is a dependent entity.

(***element_type:*** *Classes value*)

Note

that basedOn and dependentOf are being used synonymousle in this metamodel.

supertypes

The parent class

subtypings

the criteria, or dimensions, by which the class can be divided into subtypes

(*es value*)

Example

in a library model, the `Book` class could have subtypings based on genre (e.g., Fiction, Non-fiction), format (e.g., Hardcover, Paperback), or subject (e.g., Science, History).

(*Subtypings value*)

subtypes

Any subtypes or specializations of this class based on itâ€™s subtypings. _

Example

For instance, using the `Book` example, the subtypes could include `FictionBook`, `Non-fictionBook`, `HardcoverBook`, `PaperbackBook`, `ScienceBook`, and `HistoryBook`.

(*Classes value*)

attributes

The attributes or properties of the class, in the order in which they should be presented _

attributeSections

additional attributes or properties of the class, grouped for clarity and elaboration. _

(*Attributes value*)

(*AttributeSections value*)

constraints

Any constraints, rules, or validations specific to this class _

Note

Constraints may be expressed on either the Class or the Attribute. Always?

(Constraints value)

methods

Any behaviors or operations associated with this class _

Implied Attributes

(Methods value)

dependents

the Classes which are basedOn this Class

inverse: Class

CLASS_NAME: UniqueKey

(element_type: Classes value)

ATTRIBUTE_NAME:

basedOn UniqueKeys

Subtyping

a way in which subtypes of a Class may be

inverse: classified

CLASS_NAME: Class

(element_type: UniqueKeys value)

ATTRIBUTE_NAME:

basedOn

dependent_of

name

DSL: Shown in the DSL as + > Subbtypes: byBrand - Brand1, Brand2,... (non exclusive, exhaustive) + on the super class. And as + > Subtype of: SuperClass byBrand + on the subclass.

(Upper Name value)

is exclusive

default:

(Boolean value)

is exhaustive

default:

Note

every class can have an unnamed subtyping.

(Boolean value)

classes

(Classes value)

ValueType

subtype_of

Class. bySomething

Reference Type

subtype_of

Class. bySomething

CodeType A data type or enumeration used in the model

subtype_of

ValueType. bySomething

Note

Often, a CodeType will be assigned to just one attribute in the model. In such cases, there's no need to declare a new Code Type and invent a name for it. Instead:

List the code values as a bulletted list inside the description of the attribute in the form â€˜code descriptionâ€™

A Code Type will be created with the name [class] [attribute]Code and the code values listed. That CodeType will be marked as isCaptive.
isCaptive
code
description

the code type was implied by use in an attribute and is only used for that attribute

Code Value
A possible value for an enumerated data class DependentOf CodeType

A short code or abbreviationi for the value _

an explanation of what the code means

(Boolean value)
(NameString value)
(RichText value)

Key
a list of attributes of a class

subtype_of Component bySomething
dependent_of Class
keyAttributes

the attributes of the base Class.

Issue *(Attributes value)*
need ascending descending to support index keys or ordering keys.

UniqueKey
a list of attributes on which instances of the base class may be keyed.

subtype_of Key bySomething

Note
order unimportant for Unique Keys.
Attributes

Attribute Section
a group of attributes for a class that merit a shared explanation.

subtype_of Component. bySomething
based_on Class

isOptional

whether the attributes in this section, taken together, are optional. If the Attribute Section is required,then each Attribute within the sectional is optional ot required, depending on how it is marked. + Â + But if the Arrribute Section is optional each attribute in the section is only required if any attribute in the section is ptresent. *(Boolean value)*

Attribute A property or characteristic of a class
plural: Attributes

subtype_of Component bySomething
based_on AttributeSection

name CamelName *(Lower Camel value)*

overrides:

CLASS_NAME:

ATTRIBUTE_NAME: The kind of object to which the attribute refers. _

dataType

But, *(DataType value)*

List of Editions
Set of Edition
... and more complicated cases.

See
the section below on Data Type Specifiers.
Cardinalities.

isOptional

Indicates whether the attribute must have a value for every instance of the class _

default: *(Boolean value)*

cardinality

The cardinality of the relationship represented by the attribute _

default: ***DSL***: the cardinality of *(CardinalityCode value)*
an attribute, if stated explicitly, should be placed just before the class name in the parenthetical data type specification after the one-liner.

For example

author *(Invented Name)*
books *(Invented Name)*

Note
how this works with optionality
Inverse Attributes

isInvertible *(Boolean value)*

derivation: the class which contains, or would contain the inverse attribute

inverseClass *(Class value)*

derivation:

inverseAttribute *(Attribute value)*
inverseIsOptional *(Attribute value)*
default

Formulas

The rule or formula for calculating the value, if no value is supplied Now running to a second line with the parenthentical on yet a third line

Note *(Derivation value)*
even when an Attribute has a default derivation, thereâ€™s no guarantee that every instance will have an assigned value.
Example needed.

derivation

For derived attributes, the rule or formula for calculating the value _

Issue *(Derivation value)*
on insert vs on access?

constraints

Any validation rules specific to this attribute _

Note *(Constraints value)*
from Class.constraints
Override Tracking

Overrides

ValueType Derivation A rule or formula for deriving the value of an attribute

plural: Derivations

statement An English language statement of the derivation rule _

expression The formal expression of the derivation in a programming language _ *(RichText value)*

(CodeExpression value)

ValueType Constraint A rule, condition, or validation that must be satisfied by the model

plural: Constraints

subtype_of Component bySomething

statement An English language statement of the constraint _

expression The formal expression of the constraint in a programming language *(RichText value)*

severity **Warning** - nothing fatal; just a caution *(Code value)* *(Invented Name)*

Error - serious. Fix now

Message *(Template value)*

Class Constraint

subtype_of Constraint bySomething

based_on Class.

Attribute Constraint

subtype_of Constraint bySomething

based_on Attribute

CodeExpression

Language the programming language

OCL: Object Constraint Language *(Code value)*

Java: Java

Expression Methods *(String value)*

Method A behavior or operation associated with a class

plural: Methods

subtype_of Component bySomething

parameters The input parameters of the method _

returnType The data type of the value returned by the method _ *(Parameters value)*

(DataType value)

Parameter An input to a method

plural: Parameters

subtype_of Component bySomething

type The data type of the parameter _

cardinality The cardinality of the parameter *(DataType value)*

Data Types *(Invented Name)*

ValueType:**Data Type**

Simple Data Type SubtpeOf DataType

coreClass *(Class value)*

Complex Data Type

aggregation *(Aggregating Operator value)*
aggregatedTypes **Aggregating Operator** *(DataTypes value)*

Name *(Code value)*

SetOf

ListOf

Mapping

arity *(Integer value)*
spelling Low level Data Types *(Template value)*
insert Camel Case.md

ValueType CamelName
A short string without punctuation or spaces, suitable for names, labels, or identifiers and presented in camel case.

subtype_of String bySomething

value the string *(String value)*

Example
"firstName", "orderDate", "customerID"

ModelingNote
* *CamelName* is presented here, just after its first usage by another class (Component), to provide context and understanding before it is used further in the model.

UpperCamel
a CamelName that begins with a capital letter

subtype_of CamelName bySomething

where: content begins with an upper case letter.

Example
_ "Customer", "ProductCategory", "PaymentMethod"

LowerCamel
a CamelName that begins with a lower case letter

subtype_of CamelName bySomething

where: content begins with a lower case letter.

Example
"firstName", "orderTotal", "shippingAddress"

Qualified Camel
an expression consisting of Camel Names separated by periods

subtype_of String bySomething

RichText. A string with markup for block level formatting.

subtype_of String bySomething

value	the string content	
format	the rich text coding language used	*(string value)*
HTML MarkDown		*(Code value)*

RichLine

String with markup for line level formatting.

subtype_of	RichText bySomething
value	the string content *(string value)*

PrimitiveType

subtype_of	ValueType A basic bySomething

built-in data type bySomething

Values

Appendices Insert More Sidebars.md Insert Overrides.md insert LDM Intro.md Insert OCL.md Insert Camel Case.md

Annotation Types Used

These are the recognized Annotation Types for the LDM model. And this is how you register the AnnotationTyped for a model. By including this sort of array in the DSL document for the model.

```typescript
interface AnnotationType {
label: string;
emoji: string;
emojiName: string;
emojiUnicode: string;
purpose: string;
}
// LINK: LiterateDataModel.annotationTypes
const annotationTypes: AnnotationType[] = [
{
label: "Error",
emoji: "",
emojiName: "cross_mark",
emojiUnicode: "U+274C",
purpose: "Indicates a critical error or
failure in the model."
},
{
label: "Warning",
emoji: "",
emojiName: "warning",
emojiUnicode: "U+26A0",
purpose: "Indicates a potential issue or
warning in the model."
```

```
},
{
label: "Note",
emoji: "",
emojiName: "blue_book",
emojiUnicode: "U+1F4D8",
purpose: "Provides additional context,
explanations, or clarifications for the
annotated element."
},
{
label: "Issue",
emoji: "",
emojiName: "warning",
emojiUnicode: "U+26A0",
purpose: "Highlights a potential issue or
error that needs to be addressed or resolved."
},
{
label: "Question",
emoji: "",
emojiName: "question",
emojiUnicode: "U+2753",
purpose: "Raises a question or seeks further
clarification about the annotated element."
},
{
label: "Suggestion",
emoji: "",
emojiName: "bulb",
emojiUnicode: "U+1F4A1",
purpose: "Provides a suggestion or
recommendation for improving the model or the
annotated element."
},
{
label: "Info",
emoji: "",
emojiName: "information_source",
emojiUnicode: "U+2139",
purpose: "Offers relevant information, facts,
or details about the annotated element."
},
{
label: "Todo",
```

```
emoji: "",
emojiName: "pushpin",
emojiUnicode: "U+1F4CC",
purpose: "Indicates a pending task, action
item, or future work related to the annotated
element."
},
{
label: "Reference",
emoji: "",
emojiName: "globe_with_meridians",
emojiUnicode: "U+1F310",
purpose: "Provides a reference or link to an
external resource or documentation."
},
{
label: "See",
emoji: "",
emojiName: "mag",
emojiUnicode: "U+1F50D",
purpose: "Indicates a cross-reference to
another relevant element within the model."
}
];
```

===

Appendices Insert More Sidebars.md Insert Overrides.md insert LDM Intro.md Insert OCL.md Insert Camel Case.md

== content to add