© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

UnitsML — UnitsDB specification

Committee Draft Standard

Warning for drafts

This document is not a CalConnect 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.

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.

The Calendaring and Scheduling Consortium, Inc. 2025

© 2025 The Calendaring and Scheduling Consortium, Inc.

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.

The Calendaring and Scheduling Consortium, Inc.

4390 Chaffin Lane
McKinleyville
California 95519
United States of America

copyright@calconnect.org
www.calconnect.org

Contents

Foreword v

Introduction vi

1. Scope 1

2. Normative references 1

3. Terms and definitions 1

4. Principles 3
4.1. General 3
4.2. Scientific accuracy and completeness 3
4.3. Interoperability 3
4.4. Declarative content 3
4.5. Ease of use and maintenance 3
4.6. Interoperability 3
4.7. Ease of use and maintenance 4
4.8. Extensibility 4

5. Architecture 4
5.1. General 4
5.2. File organization 4
5.3. Entity relationships 5
5.4. Cross-reference mechanism 5
5.5. Information models 5
5.6. Inheritance hierarchies 6
5.7. Extension points 6
5.8. Model mapping to YAML 6
5.9. Implementation considerations 7

6. Unit system 7
6.1. General 7
6.2. Unit system model 8
6.3. Unit system relationship model 9

7. Prefix 10
7.1. General 10
7.2. Prefix model 10
7.3. Prefix symbols model 11

8. Quantity 12
8.1. General 12
8.2. Quantity model 13
8.3. Quantity type model 15
8.4. Unit reference model 16

9. Dimension 17
9.1. General 17
9.2. Dimension model 17
9.3. Dimension component model 19
9.4. Base dimension enumeration 20
9.5. Dimension representation symbol model 22

10. Unit 23
10.1. Unit model 23
10.2. YAML structure 25
10.3. Unit symbols model 27
10.4. Dimension reference model 29
10.5. Unit system reference model 29
10.6. SI root unit derivation reference model 30
10.7. Root unit reference model 31
10.8. Quantity reference model 33

11. Data file normalization rules 34
11.1. General 34
11.2. Normalization principles model 34

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

11.3. File normalization model....34
11.4. Cross-file normalization model....35
11.5. Data integrity model....36
11.6. Change management model....36

12. Common models....37
12.1. General....37
12.2. Symbol representation model....37
12.3. Cross-reference model....38

Appendix A (normative) Test suite....39
A.1. Test cases....39
A.2. Cross-reference validation....43
A.3. Validation techniques....43

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

Foreword

The Calendaring and Scheduling Consortium (“CalConnect”) is a global non-profit organization with the aim to facilitate interoperability of collaborative technologies and tools through open standards.

CalConnect works closely with international and regional partners, of which the full list is available on our website (https://www.calconnect.org/about/liaisons-and-relationships).

The procedures used to develop this document and those intended for its further maintenance are described in the CalConnect Directives.

In particular the different approval criteria needed for the different types of CalConnect documents should be noted. This document was drafted in accordance with the editorial rules of the CalConnect Directives.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CalConnect shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be provided in the Introduction.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

This document was prepared by Technical Committee *UnitsML*.

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

Introduction

UnitsDB is a comprehensive database of units of measurement, quantities, dimensions, and prefixes that serves as the authoritative reference for the UnitsML framework.

UnitsDB provides a structured and standardized way to represent and reference measurement units and related concepts across various scientific, engineering, and technical domains.

UnitsDB is designed to support multiple applications, including:

— Providing an authoritative source of measurement units for scientific and technical documentation
— Supporting unit conversion systems
— Enabling semantic interoperability between different systems that work with measurement data

This specification defines the data models, structures, and normalization rules for UnitsDB. It is intended for developers implementing UnitsML, database administrators managing UnitsDB installations, and contributors who wish to extend or modify the database.

By following this specification, implementers and users of UnitsDB can ensure consistent and accurate representation of measurement units and related concepts across different applications and domains.

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

UnitsML — UnitsDB specification

1. Scope

This document specifies the data models, structures, and normalization rules for UnitsDB, the authoritative database of measurement units, quantities, dimensions, and prefixes for the UnitsML framework.
NOTE This document applies to UnitsDB version 1.0.0.

The specification defines:

— The overall architecture of UnitsDB
— The data models for unit systems, prefixes, quantities, dimensions, and units
— The format and structure of YAML files that comprise the database
— The normalization rules for data integrity and consistency
— The relationships between different entities in the database
— A test suite for validation of UnitsDB implementations

This document:

— Does not specify implementation details for using UnitsDB in applications
— Does not define algorithms for unit conversions or dimensional analysis (though the data provided supports these operations)
— Does not provide a comprehensive list of all units, quantities, dimensions, and prefixes (these are provided in the actual database)

This specification is intended for:

— Developers implementing UnitsML systems
— Developers managing UnitsDB database instances
— Contributors extending or modifying the UnitsDB database
— Standards developers referencing UnitsDB

The specification enables interoperable representation of measurement units and related concepts across different applications, domains, and systems.

2. Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

BIPM SI Brochure sur le SI 9e v3.01 (2019/2024, F) / BIPM SI Brochure 9e v3.01 (2019/2024, E), BUREAU INTERNATIONAL DES POIDS ET MESURES (BIPM) and BUREAU INTERNATIONAL DES POIDS ET MESURES. *Le Système international d'unités*. Rapport no. SI Brochure 9e v3.01 (2019/2024, E). 9th edition. Bureau international des poids et mesures.

3. Terms and definitions

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

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

3.1.
unit
reference quantity with which other quantities of the same kind are compared to express their value

Note 1 to entry: Units are identified by conventionally assigned names and symbols.

EXAMPLE The meter is a unit of length.

3.2.
quantity
property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed by a number and a reference

EXAMPLE Length, mass, and time are quantities.

3.3.
dimension
expression of the dependence of a quantity on the base quantities of a system of quantities as a product of powers of factors corresponding to the base quantities

Note 1 to entry: In UnitsDB, dimensions are represented as combinations of seven base dimensions: length (L), mass (M), time (T), electric current (I), thermodynamic temperature (Θ), amount of substance (N), and luminous intensity (J).

3.4.
prefix
name or symbol that, when attached to a unit, multiplies that unit by a specified factor

EXAMPLE The kilo prefix multiplies a unit by 1000.

3.5.
unit system
set of units used to express quantities in a consistent manner

3.6.
SI
International System of Units

Note 1 to entry: SI is the modern form of the metric system and is the most widely used system of measurement.

3.7.
base unit
unit of measurement of a base quantity in a given system of quantities

3.8.
derived unit
unit of measurement of a derived quantity in a given system of quantities

3.9.
normalization rule
constraint that ensures the consistency, integrity, and standardization of data

3.10.
data model
formal description of the structure, attributes, and relationships of entities

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

4. Principles

4.1. General

The design and architecture of the UnitsDB adheres to the following principles.

— Scientific accuracy and completeness
— Interoperability
— Declarative content
— Ease of use and maintenance
— Extensibility

4.2. Scientific accuracy and completeness

UnitsDB adheres to the International System of Units (SI) and aims to facilitate accurate measurement across various domains towards international standards.

The database aims to accurately represent:

— Base and derived units as defined by SI.
— Information models used in the representation and realization of a digital units system.
— Historical and non-SI units that are still in use.

4.3. Interoperability

UnitsDB supports interoperability through:

— Simple realization of the database in human-readable formats
— Globally unique identifiers
— Support for presentational needs of units
— Well-defined data models allowing straightforward serialization to various formats

4.4. Declarative content

UnitsDB maintains structural consistency through:

— Well-defined data models for each entity type
— Separation of concerns in information models
— Explicit definitions of relationships between information models

4.5. Ease of use and maintenance

UnitsDB applies data normalization principles to:

— Eliminate redundancy in the database
— Ensure data integrity across related entities
— Simplify maintenance and updates
— Support efficient querying and processing
— Enable validation and consistency checking

4.6. Interoperability

UnitsDB supports interoperability through:

— Use of industry-standard YAML for data representation
— Clear separation of content from presentation

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

— Consistent identifiers and cross-references
— Support for multiple symbol representations (ASCII, Unicode, MathML, LaTeX, HTML)
— Well-defined data models that can be mapped to various implementation formats

This interoperability enables UnitsDB to serve as a reference database for diverse applications and systems.

4.7. Ease of use and maintenance

UnitsDB aims to provide users an easy to use information database while keeping it easy and simple to maintain.

— Human-readable content facilitates inspection and understanding by humans and transparent review and validation of content
— Automated validation and consistency checking
— Allows incofmproation into software sytsemss
— Efficient querying and filtering
— Transformation to other formats and representations

4.8. Extensibility

UnitsDB is designed to be extensible to accommodate:

— New units as they are defined, come into use or encoded historically
— Additional quantities in emerging domains
— New prefixes defined in the future
— Domain-specific extensions and annotations

5. Architecture

5.1. General

The UnitsDB architecture follows a modular design that organizes measurement-related concepts into distinct but interconnected data models. These data models are implemented as YAML files, with each file focusing on a specific aspect of the measurement system.

This section describes the overall architecture, the relationships between different entities, and how the lutaml-model language is used to define the data models.

5.2. File organization

UnitsDB is organized into five primary YAML files, each responsible for a different component of the measurement system.

`dimensions.yaml`	Contains definitions of physical dimensions (length, mass, time, etc.)
`quantities.yaml`	Contains definitions of measurable quantities (length, velocity, force, etc.)
`units.yaml`	Contains definitions of measurement units (meter, second, newton, etc.)
`prefixes.yaml`	Contains definitions of unit prefixes (kilo, milli, mega, etc.)
`unit_systems.yaml`	Contains definitions of unit systems (SI, non-SI) and their categories

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

Each file contains entries that follow a consistent model structure as defined using the lutaml-model language.

5.3. Entity relationships

The UnitsDB architecture establishes explicit relationships between different entities to create a coherent measurement system:

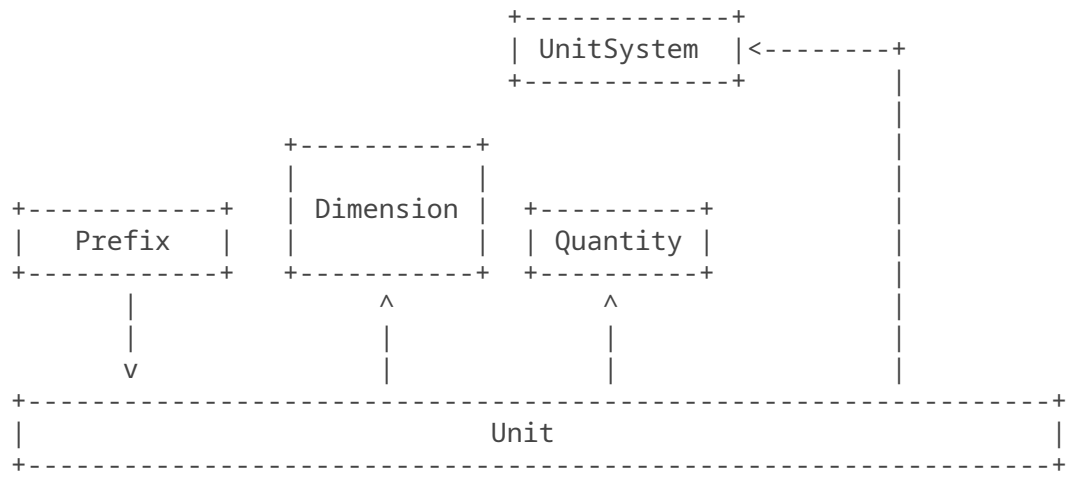


Figure 1

The key relationships are described below.

— Dimension ↔ Quantity: Each quantity has exactly one dimension, but multiple quantities can share the same dimension
— Quantity ↔ Unit: Each quantity can be measured by multiple units, and each unit can measure multiple quantities (many-to-many relationship)
— Unit ↔ Dimension: Each unit has exactly one dimension, which must match the dimension of the quantities it measures
— Prefix ↔ Unit: Prefixes can be applied to units to create new prefixed units
— UnitSystem ↔ Unit: Each unit belongs to a specific unit system

These relationships are implemented through cross-references between the different YAML files.

5.4. Cross-reference mechanism

UnitsDB uses a consistent cross-reference mechanism to link entities across different files:

— References to dimensions use the format `#NISTdX` (e.g., `#NISTd1` for length).
— References to quantities use the format `#NISTqX` (e.g., `#NISTq1` for length).
— References to units use the format `#NISTuX` (e.g., `#NISTu1` for meter).
— References to prefixes use the format `#NISTpX_Y` (e.g., `#NISTp10_3` for kilo).

These cross-references create a web of relationships that maintain the integrity and coherence of the measurement system.

5.5. Information models

UnitsDB defines five primary classes.

`Dimension`	Represents the mathematical concept of a physical dimension
`Quantity`	Represents a measurable property with a specific dimension

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

`Unit`	Represents a reference for measuring quantities
`Prefix`	Represents a multiplier for units
`UnitSystem`	Represents a system of units (e.g., SI, non-SI)

Each class definition specifies the attributes that should be present in the corresponding YAML files.

5.6. Inheritance hierarchies

UnitsDB does not use formal inheritance hierarchies, but it does establish conceptual hierarchies through relationships:

— Base dimensions vs. derived dimensions: Derived dimensions are composed of combinations of base dimensions
— Base quantities vs. derived quantities: Derived quantities reference dimensions that are combinations of base dimensions
— Base units vs. derived units: Derived units are composed of combinations of base units

These hierarchies are reflected in the data structure rather than through formal inheritance mechanisms.

5.7. Extension points

The UnitsDB architecture includes several extension points for future development:

— Addition of new dimensions, quantities, units, prefixes, or unit systems
— Enhancement of metadata for existing entities
— Addition of conversion factors between units with the same dimension
— Extension of symbol representations to support additional formats
— Domain-specific annotations for specialized fields

These extension points allow UnitsDB to evolve while maintaining backward compatibility.

5.8. Model mapping to YAML

The lutaml-model class definitions are mapped to YAML structures in the database files.

For example, the `Unit` class defined in lutaml-model:

```
class Unit {
  attribute id, String {
    definition "Unique identifier for the unit"
  }
  attribute dimension_url, String {
    definition "Reference to the dimension of the unit"
  }
  // Additional attributes...
}
```

Figure 2

Maps to YAML entries like:

```
NISTu1:
  dimension_url: "#NISTd1"
  short: meter
```


© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
# Additional properties...
```

Figure 3

This mapping ensures that the YAML files conform to the defined data models while maintaining human readability.

5.9. Implementation considerations

When implementing UnitsDB for specific applications, several architectural considerations are important:

— YAML parsing: Implementations should handle YAML 1.2 syntax correctly
— Cross-reference resolution: Applications need to resolve references between entities
— Validation: Implementations should validate entries against the defined models
— Memory management: Depending on the application, different caching strategies may be appropriate
— Query capabilities: Applications might need to support various ways to search and retrieve entities

These considerations influence how UnitsDB is integrated into larger systems and applications.

6. Unit system

6.1. General

A unit system is a defined set of units.

UnitsDB differentiates unit systems in accordance to BIPM guidelines.

SI units	Units defined by the International System of Units (SI) as defined by the CGPM and described by the BIPM. SI units are the international standard for measurement. These are the 7 base units
Units derived from SI units and special	Units that are not part of the SI system but are directly derived from SI units and therefore acceptable for use. Examples include traditionally used units like newton, pascal, joule, watt.
Units derived from SI units but not special	Units that are not part of the SI system, directly derived from SI units but are usually dimensionless or compound units and not accepted for international use. Examples include ppm, ppb, square meters.
Non-SI units, acceptable for use	Units that are not part of the SI system but are generally accepted for technical use. These units include imperial units like hour, minute, angstrom, watt hour.
Non-SI units, temporarily acceptable for use	Units that are not part of the SI system but are temporarily accepted for use in specific contexts. These units only include “are” and “hectare” for area.
Non-SI units, not acceptable for use	Units that are not part of the SI system but are generally not accepted for technical use. These units include imperial units like foot, inch, mile, metric ton, and pound.

NOTE Prefixes are modifiers that act on units, they do not affect the units’ categorization in unit systems.

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

6.2. Unit system model

6.2.1. Definition

The UnitSystem model is defined in the LutaML data definition language.

```
class UnitSystem {
  attribute id, String {
    definition {
      Unique identifier for the unit system.
    }
  }
  attribute name, String {
    definition {
      Name of the unit system (e.g., SI, non-SI).
    }
  }
  attribute acceptable, Boolean {
    definition {
      Whether the unit system is generally acceptable for use in scientific and
      technical documentation.
    }
  }
}
```

Figure 4

6.2.2. Attributes

6.2.3. id

The `id` field is a unique identifier for the unit system type. It indicates both the name of the system and the category of unit within that system. For example, `SI_base` refers to base units in the SI system.

6.2.4. name

The `name` field specifies the overarching unit system, either “SI” or “non-SI”.

6.2.5. acceptable

The `acceptable` field is a boolean indicating whether units in this system are generally acceptable for use in scientific and technical documentation according to international standards.

6.2.6. YAML structure

The unit system data is stored in the `unit_systems.yaml` file with the following structure.

EXAMPLE

```yaml
- id: SI_base
  name: SI
  acceptable: true
- id: SI_derived_special
  name: SI
  acceptable: true
- id: SI_derived_non-special
  name: SI
```

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
    acceptable: true
  - id: non-SI_acceptable
    name: non-SI
    acceptable: true
  - id: non-SI_not_acceptable
    name: non-SI
    acceptable: false
```

6.3. Unit system relationship model

6.3.1. General

Unit systems in UnitsDB maintain relationships with other entities in the database, primarily units.

6.3.2. Definition

```
class UnitSystemRelationship {
  attribute unit_id, String {
    definition "Reference to the unit"
  }
  attribute unit_system_id, String {
    definition "Reference to the unit system"
  }
}
```

Figure 5

In the UnitsDB structure, each unit is associated with a unit system through the `unit_system` field in the Unit model. This link establishes the context for the unit and provides guidance on its appropriate usage.

For example, a unit like the meter would be associated with the `SI_base` unit system, indicating it is a base unit in the SI system and generally acceptable for use in scientific and technical documentation.

6.3.3. Attributes

6.3.3.1. unit_id

The `unit_id` field is a reference to the unit that is associated with the unit system. This field links the unit to its corresponding unit system, allowing for easy identification and categorization.

6.3.3.2. unit_system_id

The `unit_system_id` field is a reference to the unit system that the unit belongs to. This field establishes the relationship between the unit and its unit system, providing context for its usage and interpretation.

6.3.4. YAML structure

The unit system relationship data is used in the `units.yaml` file with the following structure.

EXAMPLE

```
  - unit_id: NISTu1
    unit_system_id: SI_base
  - unit_id: NISTu2
    unit_system_id: SI_base
  - unit_id: NISTu3
```

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
        unit_system_id: non-SI_acceptable
```

7. Prefix

7.1. General

A prefix is a modifier that, when attached to a unit, multiplies that unit by a specified factor.

In UnitsDB, prefixes are used to create multiples and submultiples of units, allowing for the expression of quantities across a wide range of magnitudes.

7.2. Prefix model

7.2.1. Definition

The Prefix model is defined as follows.

```
class Prefix {
  attribute id, String {
    definition "Unique identifier for the prefix"
  }
  attribute name, String {
    definition "Name of the prefix (e.g., kilo, milli)"
  }
  attribute symbol, PrefixSymbols {
    definition "Symbol representations in different formats"
  }
  attribute base, Integer {
    definition "The base of the exponentiation (10 for SI prefixes, 2 for binary
prefixes)"
  }
  attribute power, Integer {
    definition "The power to which the base is raised"
  }
}
```

Figure 6

7.2.2. YAML structure

The prefix data is stored in the `prefixes.yaml` file with the following structure:

```
NISTp10_3:
  name: kilo
  symbol:
    ascii: k
    html: k
    latex: k
    unicode: k
  base: 10
  power: 3

NISTp10_-6:
  name: micro
  symbol:
    ascii: u
    html: "&micro;"
    latex: "$mu$"
    unicode: µ
```


© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
  base: 10
  power: -6

NISTp2_10:
  name: kibi
  symbol:
    ascii: Ki
    html: Ki
    latex: Ki
    unicode: Ki
  base: 2
  power: 10
```

Figure 7

7.2.3. Attributes

7.2.3.1. id

The `id` field is a unique identifier for the prefix. The format follows a pattern of “NISTp” followed by the base and power of the prefix, separated by an underscore. For SI prefixes, the format is “NISTp10_X” where X is the power, which can be positive or negative. For binary prefixes, the format is “NISTp2_X”.

7.2.3.2. name

The `name` field contains the name of the prefix, such as “kilo”, “milli”, or “giga”.

7.2.3.3. symbol

The `symbol` field contains representations of the prefix symbol in different formats using the `PrefixSymbols` model.

7.2.3.4. base

The `base` field indicates the base of the exponentiation.

— For SI prefixes, this is always 10.
— For binary prefixes, it is 2.

7.2.3.5. power

The `power` field indicates the power to which the base is raised to obtain the multiplier for the prefix.

For example, a power of 3 with a base of 10 gives a multiplier of $10^3 = 1000$.

7.3. Prefix symbols model

7.3.1. Definition

```
class PrefixSymbols {
  attribute ascii, String {
    definition "ASCII text representation"
  }
  attribute html, String {
    definition "HTML representation (may include HTML entities)"
  }
```

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
  attribute mathml, String {
    definition "MathML representation in XML format."
  }
  attribute latex, String {
    definition "LaTeX representation in LaTeX typeset format."
  }
  attribute unicode, String {
    definition "Unicode representation in UTF-8 encoding."
  }
}
```

Figure 8

7.3.2. Attributes

7.3.2.1. ascii

The `ascii` field contains the ASCII text representation of the prefix symbol. This is the most basic representation and is suitable for plain text environments.

7.3.2.2. html

The `html` field contains the HTML representation of the prefix symbol. This may include HTML entities for special characters.

7.3.2.3. mathml

The `mathml` field contains the MathML representation of the prefix symbol and is a MathML string. This is for use within MathML-supported environments.

7.3.2.4. latex

The `latex` field contains the LaTeX representation of the prefix symbol. This is for use within LaTeX documents or LaTeX-supported environments.

7.3.2.5. unicode

The `unicode` field contains the Unicode representation of the prefix symbol. This is mainly for use within UnicodeMath or other Unicode-enabled environments.
NOTE This allows for the representation of characters that may not be available in ASCII or HTML.

8. Quantity

8.1. General

A quantity is a property of a phenomenon, body, or substance that can be quantified by measurement.

In UnitsDB, quantities provide the bridge between theoretical dimensions and practical units of measurement.

They represent physical properties like length, mass, time, speed, or energy, which can be measured using specific units.

There are two represented types of quantities in UnitsDB.

Base quantities — The fundamental quantities that cannot be derived from other quantities. The SI system defines seven base quantities: length,

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

mass, time, electric current, thermodynamic temperature, amount of substance, and luminous intensity.

Dimensionless quantities

The dimension-independent quantities that are not associated with any physical dimension. These quantities are often used in mathematical or statistical contexts, such as probability, ratios, or pure numbers. While dimensionless quantities do not have a specific dimension, they can still be associated with a unit of measurement for semantic interoperability.

EXAMPLE 1

The ratio of two lengths is dimensionless, but it can be expressed as a quantity as a “ratio for lengths”.

EXAMPLE 2

The unit ‘parts-per-million’ (ppm) is a unit that measures a quantity that represents a ratio of one part of a substance to one million parts of the total mixture.

Derived quantities

Quantities that are defined through algebraic relationships with other quantities, typically expressed as products of powers of base quantities. They can differ in units depending on their dependencies and can be used to express complex measures in a simplified form.

8.2. Quantity model

8.2.1. Definition

The Quantity model is defined as follows.

```
class Quantity {
  attribute id, String {
    definition {
      "Unique identifier for the quantity.
    }
  }
  attribute dimension_url, String {
    definition {
      Reference to the dimension of the quantity.
    }
  }
  attribute quantity_type, QuantityType {
    definition {
      Type of quantity.
    }
  }
  attribute quantity_name, String {
    collection true
    definition {
      Names of the quantity.
    }
  }
  attribute unit_reference, UnitReference {
    collection true
    definition {
      References to units that can measure this quantity
    }
```

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
  }
}
```

Figure 9

8.2.2. YAML structure

The quantity data is stored in the `quantities.yaml` file with the following structure:

```yaml
NISTq1:
  dimension_url: "#NISTd1"
  quantity_type: base
  quantity_name:
  - length
  unit_reference:
  - name: meter
    url: "#NISTu1"
  - name: foot
    url: "#NISTu78"
  - name: inch
    url: "#NISTu8"
  - name: mile
    url: "#NISTu83"
  - name: yard
    url: "#NISTu84"

NISTq12:
  dimension_url: "#NISTd11"
  quantity_type: derived
  quantity_name:
  - velocity
  - speed
  unit_reference:
  - name: meter per second
    url: "#NISTu1.u3e-1/1"
  - name: kilometer per hour
    url: "#NISTu160"
  - name: mile per hour
    url: "#NISTu110"
```

Figure 10

8.2.3. Attributes

8.2.3.1. id

The `id` field is a unique identifier for the quantity, typically in the format `NISTqX` where X is a number.

Each quantity has a unique ID that is used for cross-referencing throughout the UnitsDB system.

8.2.3.2. dimension_url

The `dimension_url` field contains a reference to the dimension associated with the quantity. This reference establishes the dimensional characteristics of the quantity and links it to the mathematical representation in the dimension model.

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

8.2.3.3. quantity_type

The `quantity_type` field indicates whether the quantity is a base quantity or a derived quantity, using the `QuantityType` enumeration.

8.2.3.4. quantity_name

The `quantity_name` field is a collection of strings that each represents a name for the quantity.

The first name in the array is the primary name, and any additional names are alternative names or synonyms.

EXAMPLE

"velocity" might have the synonym "speed".

8.2.3.5. unit_reference

The `unit_reference` field is a collection of UnitReference objects that can be used to measure the quantity.

8.3. Quantity type model

8.3.1. General

UnitsDB classifies quantities into different types based on their role in the system of physical quantities.

8.3.2. Definition

```
enum QuantityType {
  base {
    definition "Base quantity"
  }
  derived {
    definition "Derived quantity"
  }
}
```

Figure 11

8.3.3. Enumerated values

8.3.3.1. base

Base quantities are the fundamental physical quantities that cannot be derived from other quantities. They are defined independently and serve as the foundation for all other quantities.

The seven base quantities in the SI system form the foundation of all physical measurements, and are currently defined in the UnitsDB.
NOTE These quantity entries are protected entries in the UnitsDB.

(NISTq1) Length, (NISTq48) Distance	The measurement of distance or dimension in one direction.

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

(NISTq2) Mass	The measurement of the amount of matter in an object.
(NISTq3) Time/Duration	The measurement of duration or sequential events.
(NISTq4) Electric current	The flow of electric charge.
(NISTq5) Thermodynamic temperature	The measurement of thermal energy.
(NISTq6) Amount of substance	The number of entities (atoms, molecules, ions, etc.) in a system.
(NISTq7) Luminous intensity	The measure of the power of light in a given direction.

8.3.3.2. `derived`

A derived quantity is a physical quantity that is defined in terms of base quantities. Derived quantities are expressed as products of powers of base quantities, and they can be represented in terms of their dimensional relationships.

Derived quantities can be expressed in terms of base quantities using dimensional analysis.

For example, velocity is a derived quantity that can be expressed as length divided by time (L/T). Other examples of derived quantities include:

— acceleration (NISTq13), which is expressed as length divided by time squared (L/T^2)
— force (NISTq14), which is expressed as mass times acceleration ($M \cdot L/T^2$)
— pressure (NISTq15), which is expressed as force divided by area ($M \cdot L/T^2 \cdot L^2$)

8.4. Unit reference model

8.4.1. General

A quantity can be measured by multiple units, and this relationship is captured in the `unit_reference` field.

This many-to-many relationship allows:

— Selection of appropriate units for a given quantity
— Identification of all quantities that can be measured with a particular unit
— Conversion between different units for the same quantity

For example, the quantity “length” (NISTq1) can be measured using units such as meter, foot, inch, mile, and yard.

8.4.2. Definition

```
class UnitReference {
  attribute name, String {
    definition "Name of the unit"
  }

  attribute url, String {
    definition "Reference to the unit"
  }
```

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
}
```

Figure 12

8.4.3. Attributes

8.4.3.1. name

The `name` field is a string that represents the name of the unit that can measure the quantity.

This field is optional

8.4.3.2. url

The `url` field is a string that contains a reference to the unit in the UnitsDB system.

9. Dimension

9.1. General

A dimension is a physical quantity that can be measured.

According to the SI Brochure, a dimension is defined as “a physical quantity that can be expressed in terms of base quantities.” Dimensions are fundamental to the system of units and provide a framework for understanding and comparing different physical properties.

The SI standard defines the following dimensions and corresponding dimension symbols:

— time (T)
— length (L)
— mass (M)
— electric current (I)
— absolute temperature (Θ)
— amount of substance (N)
— luminous intensity (J).

In UnitsDB, dimensions are represented as mathematical combinations of base dimensions, using the same approach as dimensional analysis in physics.

9.2. Dimension model

9.2.1. Definition

The Dimension model is defined as follows.

```
class Dimension {
  attribute id, String {
    definition {
      Unique identifier for the dimension
    }
  }

  attribute dimensionless, Boolean {
    definition {
      Whether the dimension is dimensionless. A dimensionless dimension
      does not have any associated dimension components in `components`.
    }
  }
```

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
  attribute components, DimensionComponent {
    collection true
    definition {
      Components of the dimension, including base quantities and their powers.
      Not applicable for dimensionless quantities.
    }
  }
}
```

Figure 13

9.2.2. YAML structure

The dimension data is stored in the `dimensions.yaml` file with the following structure:

```
NISTd1:
  length:
    powerNumerator: 1
    symbol: L
    dim_symbols:
    - id: dim_L
      ascii: L
      html: "&#x1D5AB;"
      mathml: "<mi mathvariant='sans-serif'>L</mi>"
      latex: "\\ensuremath{\\mathsf{L}}"
      unicode: "\U0001D5AB"
NISTd12:
  length:
    powerNumerator: 1
    symbol: L
  mass:
    powerNumerator: 1
    symbol: M
  time:
    powerNumerator: -2
    symbol: T
NISTd67:
  dimensionless: true
```

Figure 14

9.2.3. Attributes

9.2.3.1. id

The `id` field is a unique identifier for the dimension, typically in the format `NISTdX` where X is a number. Each dimension has a unique ID that is used for cross-referencing throughout the UnitsDB system.

9.2.3.2. dimensionless

The `dimensionless` field is a boolean that indicates whether the dimension is dimensionless (true) or has a physical dimension (false or not present). Dimensionless quantities are pure numbers without units, such as ratios, angles, and counts.

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

9.2.3.3. dimension components

Each dimension is composed of one or more base dimension components represented as a DimensionComponent object.

Each dimension can only have a maximum of one base dimension component of each type of the 7 base dimensions.

9.3. Dimension component model

9.3.1. General

A dimension component represents a base quantity and its exponent (power) in the dimension.

The seven base quantities in SI are:

— (NISTq1) length
— (NISTq2) mass
— (NISTq3) time
— (NISTq4) electric current
— (NISTq5) thermodynamic temperature
— (NISTq6) amount of substance
— (NISTq7) luminous intensity

9.3.2. Definition

```
class DimensionComponent {
  attribute base_dimension, BaseDimensionEnum {
    definition {
      The base dimension represented by this component.
    }
  }

  attribute power_numerator, Integer {
    definition "The exponent (power) of the base quantity in the dimension"
  }

  attribute symbol, String {
    definition "The symbol used to represent the base quantity in dimensional
formulas"
  }

  attribute dim_symbols, DimensionSymbol {
    collection true
    definition "Symbol representations in different formats"
  }
}
```

Figure 15

9.3.3. Attributes

9.3.3.1. base_dimension

The `base_dimension` field is an enumeration that specifies the base dimension represented by this component, with possible values listed in the `BaseDimensionEnum` model.

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

9.3.3.2. power_numerator

The `power_numerator` field is an integer that represents the exponent (power) of the base quantity in the dimension. This indicates how many times the base dimension is multiplied in the dimension formula.

EXAMPLE 1

A power of 2 indicates that the base dimension is squared.

EXAMPLE 2

For a dimension with a power of -1, it indicates the reciprocal of the base dimension.

9.3.3.3. symbol

The `symbol` field represents a general notation used for the dimension in various contexts.

EXAMPLE

For a dimension expressed with a power of 0, it indicates that the dimension is dimensionless.

9.3.3.4. dim_symbols

The `dim_symbols` field is a collection of `DimensionSymbol` objects that represent the symbol representations of the dimension in different formats, in the `DimensionSymbol` model.

9.4. Base dimension enumeration

9.4.1. General

The `BaseDimensionEnum` model defines the enumeration of base dimensions according to the SI system. Each base dimension is represented by a unique identifier and has a corresponding definition.

9.4.2. Definition

```
class BaseDimensionEnum {
  definition {
    Enumeration of base dimensions.
  }

  value length {
    definition {
      The base dimension of length
    }
  }
  value mass {
    definition {
      The base dimension of mass
    }
  }
  value time {
    definition {
      The base dimension of time
    }
  }
```

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
  value electric_current {
    definition {
      The base dimension of electric current
    }
  }
  value thermodynamic_temperature {
    definition {
      The base dimension of thermodynamic temperature
    }
  }
  value amount_of_substance {
    definition {
      The base dimension of amount of substance
    }
  }
  value luminous_intensity {
    definition {
      The base dimension of luminous intensity
    }
  }

  value plane_angle {
    definition {
      The base dimension of the dimensionless plane angle. This is a special
      case that is not a base dimension in the SI system but is often treated as
      such in dimensional analysis.
    }
  }
}
```

Figure 16

9.4.3. Enumerated values

9.4.3.1. length

The `length` base dimension represents the physical quantity of length.

9.4.3.2. mass

The `mass` base dimension represents the physical quantity of mass.

9.4.3.3. time

The `time` base dimension represents the physical quantity of time.

9.4.3.4. electric_current

The `electric_current` base dimension represents the physical quantity of electric current.

9.4.3.5. thermodynamic_temperature

The `thermodynamic_temperature` base dimension represents the physical quantity of thermodynamic temperature.

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

9.4.3.6. amount_of_substance

The `amount_of_substance` base dimension represents the physical quantity of the amount of substance.

9.4.3.7. luminous_intensity

The `luminous_intensity` base dimension represents the physical quantity of luminous intensity.

9.4.3.8. plane_angle

The `plane_angle` base dimension represents the physical quantity of a plane angle. This is a special case that is not a base dimension in the SI system but is often treated as such in dimensional analysis.

9.5. Dimension representation symbol model

9.5.1. General

The dimension representation symbols model defines the various symbol representations for a dimension. These symbols are used in different contexts, such as mathematical expressions, HTML rendering, and LaTeX typesetting.

This model is only used for base dimensions defined in the `BaseDimensionEnum` model.
NOTE Only base dimensions have officially assigned dimension symbols by the SI System.

9.5.2. Definition

```
class DimensionSymbol {
  attribute id, String {
    definition "Unique identifier for the symbol representation"
  }

  attribute ascii, String {
    definition "ASCII text representation"
  }

  attribute html, String {
    definition "HTML representation (may include HTML entities)"
  }

  attribute mathml, String {
    definition "MathML representation in XML format."
  }

  attribute latex, String {
    definition "LaTeX representation in LaTeX typeset format."
  }

  attribute unicode, String {
    definition "Unicode representation in UTF-8 encoding."
  }
}
```

Figure 17

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

9.5.3. Attributes

9.5.3.1. id

The `id` field is a unique identifier for the symbol representation.

9.5.3.2. ascii

The `ascii` field contains the ASCII text representation of the symbol. This is the most basic representation and is suitable for plain text environments.

9.5.3.3. html

The `html` field contains the HTML representation of the symbol. This may include HTML entities for special characters.

9.5.3.4. mathml

The `mathml` field contains the MathML representation of the symbol. This is an XML-based markup language for representing mathematical notations.

9.5.3.5. latex

The `latex` field contains the LaTeX representation of the symbol.

9.5.3.6. unicode

The `unicode` field contains the Unicode representation of the symbol.

9.5.4. YAML structure

The dimension representation symbols data is stored in the `dimensions.yaml` file with the following structure:

EXAMPLE

```yaml
id: dim_L
ascii: L
html: "&#x1D5AB;"
mathml: "<mi mathvariant='sans-serif'>L</mi>"
latex: "\\ensuremath{\\mathsf{L}}"
unicode: "\U0001D5AB"
```

10. Unit

10.1. Unit model

10.1.1. General

A unit is a reference quantity adopted by convention with which other quantities of the same kind can be compared to express their value.

In UnitsDB, units are organized according to their system (SI, non-SI), dimensionality, and relationships to other units.

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

10.1.2. Definition

The Unit model is defined as follows.

```
class Unit {
  attribute id, String {
    definition {
      Unique identifier for the unit.

      NOTE: Currently this is in the pattern `NISTuX` where X is a number.
    }
  }
  attribute short_name, String {
    definition {
      Short name for the unit. This is a unique identifier for the unit.
    }
  }
  attribute root, Boolean {
    definition {
      Whether the unit can be used as a root unit to create derived units.
    }
  }
  attribute prefixed, Boolean {
    definition {
      Whether the unit is prefixed itself.
    }
  }
  attribute unit_name, String {
    collection true
    definition {
      One or more names of the unit.
    }
  }
  attribute unit_symbol, UnitSymbol {
    collection true
    definition {
      One or more symbol representations in different formats.
    }
  }
  attribute unit_system, UnitSystemReference {
    definition {
      Reference to the unit system to which the unit belongs.
    }
  }
  attribute dimension_ref, DimensionReference {
    definition {
      Reference to the dimension of the unit.
    }
  }
  attribute quantity_ref, QuantityReference {
    collection true
    definition {
      Reference to the quantity measured by the unit.
    }
  }
  attribute root_units, RootUnitReference {
    collection true
    definition {
      For derived units, the component root units with exponents.
```

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
    }
  }
  attribute si_derived_bases, SiBaseUnitReference {
    collection true
    definition {
      For SI-derived units, the SI base units with exponents.
    }
  }
}
```

Figure 18

10.2. YAML structure

The unit data is stored in the `units.yaml` file with the following structure.

EXAMPLE 1 — Example of an SI unit

```
NISTu1:
  dimension_url: "#NISTd1"
  short: meter
  root: true
  unit_system:
    type: SI_base
    name: SI
  unit_name:
  - meter
  unit_symbols:
  - id: m
    ascii: m
    html: m
    mathml: "<mi mathvariant='normal'>m</mi>"
    latex: "\\ensuremath{\\mathrm{m}}"
    unicode: m
  root_units:
  - unit: meter
    power_numerator: 1
  quantity_reference:
  - name: length
    url: "#NISTq1"
  - name: diameter
    url: "#NISTq100"
  - name: length of path
    url: "#NISTq101"
  - name: cartesian coordinates
    url: "#NISTq102"
  - name: position vector
    url: "#NISTq103"
  - name: displacement
    url: "#NISTq104"
  - name: radius of curvature
    url: "#NISTq105"
  - name: wavelength
    url: "#NISTq114"
  - name: distance
    url: "#NISTq48"
  - name: breadth
    url: "#NISTq95"
  - name: height
    url: "#NISTq96"
```

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
    - name: thickness
      url: "#NISTq97"
    - name: radius
      url: "#NISTq98"
    - name: radial distance
      url: "#NISTq99"
```

EXAMPLE 2 — Example of a non-SI unit

```
  NISTu8:
    dimension_url: "#NISTd1"
    short: inch
    root: true
    unit_system:
      type: non-SI_not_acceptable
      name: non-SI
    unit_name:
    - inch
    unit_symbols:
    - id: in
      ascii: in
      html: in
      mathml: "<mi mathvariant='normal'>in</mi>"
      latex: "\\ensuremath{\\mathrm{in}}"
      unicode: in
    - id: "\"_in"
      ascii: "\""
      html: "&#8243;"
      mathml: "<mi mathvariant='normal'>&#8243;</mi>"
      latex: "\\ensuremath{\\mathrm{''}}"
      unicode: "″"
    - id: dprime_in
      ascii: "\""
      html: "&#8243;"
      mathml: "<mi mathvariant='normal'>&#8243;</mi>"
      latex: "\\ensuremath{\\mathrm{''}}"
      unicode: "″"
    root_units:
    - unit: inch
      power_numerator: 1
    quantity_reference:
    - name: length
      url: "#NISTq1"
    - name: distance
      url: "#NISTq48"
```

EXAMPLE 3 — Example of an SI-derived unit

```
  NISTu18:
    dimension_url: "#NISTd20"
    short: ohm
    root: true
    unit_system:
      type: SI_derived_special
      name: SI
    unit_name:
    - ohm
    unit_symbols:
    - id: Ohm
      ascii: Ohm
      html: "&#8486;"
```


© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
      mathml: "<mi mathvariant='normal'>&#8486;</mi>"
      latex: "\\ensuremath{\\mathrm{\\Omega}}"
      unicode: Ω
    root_units:
    - unit: ohm
      power_numerator: 1
    quantity_reference:
    - name: electric resistance
      url: "#NISTq28"
    si_derived_bases:
    - id: NISTu1
      prefix:
      power: 2
    - id: NISTu27
      prefix: p10_3
      power: 1
    - id: NISTu3
      prefix:
      power: 4
    - id: NISTu4
      prefix:
      power: 2
```

10.2.1. Attributes

10.3. Unit symbols model

10.3.1. General

The unit symbols model describes the various representations of a unit in different formats, such as ASCII, HTML, MathML, LaTeX, and Unicode. Each representation is associated with a unique identifier.

10.3.2. Definition

```
class UnitSymbol {
  attribute id, String {
    definition "Unique identifier for the symbol representation"
  }
  attribute ascii, String {
    definition "ASCII text representation"
  }
  attribute html, String {
    definition "HTML representation (may include HTML entities)"
  }
  attribute mathml, String {
    definition "MathML representation in XML format."
  }
  attribute latex, String {
    definition "LaTeX representation in LaTeX typeset format."
  }
  attribute unicode, String {
    definition "Unicode representation in UTF-8 encoding."
  }
}
```

Figure 19

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

10.3.3. Attributes

10.3.3.1. id

The `id` field is a unique identifier for the symbol representation. This field is used to reference the symbol in various contexts, such as documentation and software applications.

10.3.3.2. ascii

The `ascii` field contains the ASCII text representation of the symbol. This field is used in plain text environments where special characters may not be available.

10.3.3.3. html

The `html` field contains the HTML representation of the symbol. This may include HTML entities for special characters, allowing for proper rendering in web browsers and other HTML-compatible environments.

10.3.3.4. mathml

The `mathml` field contains the MathML representation of the symbol. This field is used in mathematical contexts where precise formatting is required. MathML is an XML-based markup language for describing mathematical notation.

10.3.3.5. latex

The `latex` field contains the LaTeX representation of the symbol. This field is used in typesetting environments where LaTeX is applied.

10.3.3.6. unicode

The `unicode` field contains the Unicode representation of the symbol. This field is used in environments that support Unicode encoding, such as within UnicodeMath.

10.3.4. YAML structure

EXAMPLE — Example unit symbols from `NISTu8`

```yaml
unit_symbols:
- id: in
  ascii: in
  html: in
  mathml: "<mi mathvariant='normal'>in</mi>"
  latex: "\\ensuremath{\\mathrm{in}}"
  unicode: in
- id: "\"_in"
  ascii: "\""
  html: "&#8243;"
  mathml: "<mi mathvariant='normal'>&#8243;</mi>"
  latex: "\\ensuremath{\\mathrm{''}}"
  unicode: "″"
- id: dprime_in
  ascii: "\""
  html: "&#8243;"
  mathml: "<mi mathvariant='normal'>&#8243;</mi>"
  latex: "\\ensuremath{\\mathrm{''}}"
```

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
        unicode: "″"
```

10.4. Dimension reference model

10.4.1. General

The dimension reference model describes the relationship between the unit and a dimension. Each unit is associated with a dimension, which defines the physical quantity that the unit measures.

10.4.2. Definition

```
class DimensionReference {
  attribute dimension_url, String {
    definition "URL associated with the dimension"
  }
}
```

Figure 20

10.4.3. Attributes

10.4.3.1. dimension_url

The `dimension_url` field is a reference to the dimension that the unit is associated with. This field links the unit to its corresponding dimension, allowing for easy identification and categorization.

10.4.4. YAML structure

EXAMPLE

```
      dimension_url: "#NISTd1"
```

10.5. Unit system reference model

10.5.1. General

The unit system reference model describes the relationship between the unit and a unit system. Each unit is associated with a unit system, which defines the context in which the unit is used.

10.5.2. Definition

```
class UnitSystemReference {
  attribute type, String {
    definition "Type of the unit system"
  }
  attribute name, String {
    definition "Name of the unit system"
  }
}
```

Figure 21

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

10.5.3. Attributes

10.5.3.1. type

The `type` field is a string that represents the type of the unit system. This field indicates the classification of the unit system, such as SI base, SI derived, or non-SI.

10.5.3.2. name

The `name` field is a string that represents the name of the unit system. This field provides a human-readable label for the unit system, allowing for easy identification and categorization.

10.5.4. YAML structure

EXAMPLE 1

```yaml
unit_system:
  type: SI_base
  name: SI
```

EXAMPLE 2

```yaml
unit_system:
  type: non-SI_not_acceptable
  name: non-SI
```

10.6. SI root unit derivation reference model

10.6.1. General

The SI root unit derivation reference model describes the relationship between the unit and the SI root units. Each unit may be associated with one or more SI root units, which define the base units from which the unit is derived.

This only applies for the units that are `SI_derived_special`.

10.6.2. Definition

```
class SiBaseUnitReference {
  attribute id, String {
    definition {
      Unique identifier for the SI base unit. e.g. `NISTu1`.
    }
  }
  attribute prefix, String {
    definition {
      Prefix for the SI base unit, if applicable. Defaults to empty string.
    }
  }
  attribute power, Integer {
    definition {
      Power of the SI base unit in the derived unit. Defaults to 1.
    }
  }
}
```

Figure 22

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

10.6.3. Attributes

10.6.3.1. id

The `id` field is a string that represents a unique identifier for the SI base unit.

10.6.3.2. prefix

The `prefix` field is a string that represents the prefix for the SI base unit, if applicable.

10.6.3.3. power

The `power` field is an integer that indicates the power of the SI base unit in the derived unit. Defaults to 1.

10.6.4. YAML structure

EXAMPLE 1 — Example SI root unit derivation reference from `NISTu14` "watt", which is an SI-derived unit

```yaml
si_derived_bases:
  - id: NISTu1
    prefix:
    power: 2
  - id: NISTu27
    prefix: p10_3
    power: 1
  - id: NISTu3
    prefix:
    power: -3
```

EXAMPLE 2 — Example SI root unit derivation reference from `NISTu18` "ohm", which is an SI-derived unit

```yaml
si_derived_bases:
- id: NISTu1
  prefix:
  power: 2
- id: NISTu27
  prefix: p10_3
  power: 1
- id: NISTu3
  prefix:
  power: 4
- id: NISTu4
  prefix:
  power: 2
```

10.7. Root unit reference model

10.7.1. General

The root unit reference model describes the relationship between the unit and the root units. Each unit may be associated with one or more root units, which define the base units from which the unit is derived.

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

If the unit is a root unit, where the `root` field is set to true, then the root unit reference model points to the unit itself.

10.7.2. Definition

```
class RootUnitReference {
  attribute unit, String {
    definition "Unit `short` name of the root unit which is a Unit object."
  }
  attribute power_numerator, Integer {
    definition "Power of the numerator."
  }
  attribute power_denominator, Integer {
    definition "Power of the denominator."
  }
  attribute prefix, String {
    definition "Prefix for the unit, if applicable."
  }
}
```

Figure 23

10.7.3. Attributes

10.7.3.1. unit

The `unit` field is a string that represents the short name of the root unit.

10.7.3.2. power_numerator

The `power_numerator` field is an integer that represents the power of the numerator. This field indicates the exponent of the root unit in the numerator of the derived unit.

10.7.3.3. prefix

The `prefix` field is a string that represents the prefix for the unit, if applicable. This field indicates any prefix that modifies the root unit, such as “k” for kilogram or “m” for milli.

10.7.4. YAML structure

EXAMPLE 1

```yaml
root_units:
  enumerated_root_units:
  - unit: minute
    power_numerator: 1
```

Example root unit reference from `NISTu36` "minute", which is a root unit

EXAMPLE 2

```yaml
root_units:
  enumerated_root_units:
  - unit: meter
    power_numerator: 1
    prefix: u
```

Example root unit reference from `NISTu122` "micron", which is not a root unit

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

This indicates that the micron is derived from the meter with a prefix of “u” (micro). Hence the micron, a unit of length, is represented as the unit “meter” with a prefix “u” (effectively, “micrometer”) in the SI system.

EXAMPLE 3

```
root_units:
  enumerated_root_units:
  - unit: gram_force
    power_numerator: 1
    prefix: k
  - unit: second
    power_numerator: 2
  - unit: meter
    power_numerator: -1
```

Example root unit reference from `NISTu125` "kilogram-force second squared per meter", which is not a root unit

This indicates that the kilogram-force second squared per meter is derived from these root units:

— kilogram-force (with a prefix of “k”, effectively “gram_force” with a prefix “k”)
— second (with a power of 2)
— meter (with a power of -1)

The unicode representation would therefore be `kgf·s²m⁻¹`.

10.8. Quantity reference model

10.8.1. General

The quantity reference model describes the relationship between the unit and the quantity. Each unit may be associated with one or more quantities, which define the physical quantities that the unit measures.

10.8.2. Definition

```
class QuantityReference {
  attribute name, String {
    definition "Name of the quantity"
  }
  attribute url, String {
    definition "URL reference to the quantity"
  }
}
```

Figure 24

10.8.3. YAML structure

EXAMPLE — Example quantity reference from `NISTu15` “coulomb”, which is a root unit

```
quantity_reference:
- name: electric charge
  url: "#NISTq22"
- name: amount of electricity
```

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
        url: "#NISTq23"
```

11. Data file normalization rules

11.1. General

Data file normalization in UnitsDB refers to the process of organizing and structuring the YAML files that make up the database to ensure consistency, non-redundancy, and integrity. These rules help maintain the quality and reliability of the data while facilitating its use in various applications.

11.2. Normalization principles model

11.2.1. Unique identifiers

All model instances within UnitsDB (dimensions, quantities, units, prefixes, etc.) must have unique identifiers. These identifiers follow a consistent pattern:

— Dimensions: `NISTdX` (e.g., `NISTd1`)
— Quantities: `NISTqX` (e.g., `NISTq1`)
— Units: `NISTuX` (e.g., `NISTu1`)
— Prefixes: `NISTpX_Y` (e.g., `NISTp10_3`)

Where `X` and `Y` are numeric values.

11.2.2. References

References between entities must use consistent paths:

— References to dimensions use the format `#NISTdX`
— References to quantities use the format `#NISTqX`
— References to units use the format `#NISTuX`

These references create a web of relationships that maintain the integrity of the database.

11.2.3. Hierarchical organization

The data is organized in a hierarchical manner that reflects the logical relationship between entities:

— Dimensions provide the mathematical foundation for quantities
— Quantities are physical properties that can be measured
— Units are references for measuring quantities
— Prefixes modify units to create new scales

11.3. File normalization model

11.3.1. dimensions.yaml

— Each dimension entry must have a unique identifier.
— Dimensionless entries must be explicitly marked with `dimensionless: true`.
— Base quantity components must specify their `powerNumerator` and `symbol`.
— When `dim_symbols` are provided, they must include representations in all standard formats (ascii, html, mathml, latex, unicode).

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

11.3.2. quantities.yaml

— Each quantity entry must have a unique identifier.
— Each quantity must reference a valid dimension via `dimension_url`.
— Quantity types must be either `base` or `derived`.
— Quantity names must be provided as an array, with the primary name first.
— Unit references must include both a name and a URL to the unit.

11.3.3. units.yaml

— Each unit entry must have a unique identifier.
— Each unit must reference a valid dimension via `dimension_url`.
— The `root` field must be explicitly set to either `true` or `false`.
— If `prefixed` is true, there must be a prefix specified in the `root_units` section.
— Unit names must be provided as an array, with the primary name first.
— Unit symbols must include representations in all standard formats.
— The `unit_system` field must specify both a valid `type` and `name`.
— For derived units, the `root_units` section must properly enumerate all component units with their powers.
— Quantity references must include both a name and a URL to the quantity.

11.3.4. prefixes.yaml

— Each prefix entry must have a unique identifier.
— The `base` field must be either 10 (for SI prefixes) or 2 (for binary prefixes).
— The `power` field must be an integer.
— Prefix symbols must include representations in all standard formats.

11.3.5. unit_systems.yaml

— Each unit system entry must have a unique identifier.
— The `name` field must be either “SI” or “non-SI”.
— The `acceptable` field must be explicitly set to either `true` or `false`.

11.4. Cross-file normalization model

11.4.1. Dimension-quantity relationships

— Each quantity must reference a valid dimension.
— Multiple quantities can reference the same dimension.

11.4.2. Quantity-unit relationships

— Each unit must be associated with at least one quantity.
— Multiple units can measure the same quantity.
— The dimension of a unit must match the dimension of the quantities it measures.

11.4.3. Unit-prefix relationships

— Prefixed units must reference valid prefixes.
— The same prefix can be applied to multiple units.
— For the base unit of mass (kilogram), prefixes are applied to “gram” rather than “kilogram”.

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

11.5. Data integrity model

11.5.1. General

The data integrity model defines checks and validations that should be performed to ensure the overall consistency and correctness of the UnitsDB database.

```
class DataIntegrityCheck {
  attribute check_type, String {
    definition "Type of integrity check"
  }
  attribute description, String {
    definition "Description of the check"
  }
  attribute severity, String {
    definition "Severity level if check fails (error, warning, info)"
  }
}
```

Figure 25

11.5.2. Data integrity checks

To ensure the integrity of the UnitsDB data, the following checks should be performed:

— All reference links must point to valid entities.
— Circular references must be avoided.
— No duplicate identifiers should exist within each entity type.
— Derived dimensions must have consistent exponents in their base quantity components.
— Units must have dimensions that match their referenced quantities.
— All required fields must be present for each entity type.

11.6. Change management model

11.6.1. General

The change management model defines rules and procedures for updating the UnitsDB data files while maintaining backward compatibility and data integrity.

```
class ChangeManagement {
  attribute change_type, String {
    definition "Type of change"
  }
  attribute rules, Array {
    definition "Rules governing this type of change"
  }
  attribute impact, String {
    definition "Potential impact of this type of change"
  }
}
```

Figure 26

11.6.2. Versioning and change management

When updating the UnitsDB data files, the following rules should be applied:

— Existing identifier values should not be changed to avoid breaking existing references.

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

— New entities should be assigned new, unique identifiers.
— When an entity needs to be deprecated, it should be marked as such rather than removed.
— Major structural changes to the data format should be accompanied by a version increment and appropriate documentation.

12. Common models

12.1. General

This section defines models that are shared across multiple components of the UnitsDB specification. These common models provide consistent structures and behaviors that are reused throughout the database.

12.2. Symbol representation model

12.2.1. General

The Symbol representation model is used across various entities in UnitsDB to provide consistent representation of symbols in different formats.

Symbol representations are used in:

— Unit symbols
— Dimension symbols
— Prefix symbols

Each representation includes multiple formats to support different output contexts:

— Plain text (ASCII)
— Web display (HTML)
— Mathematical typesetting (MathML, LaTeX)
— Unicode-enabled environments

12.2.2. Definition

```
class SymbolRepresentation {
  attribute id, String {
    definition "Unique identifier for the symbol representation"
  }

  attribute ascii, String {
    definition "ASCII text representation"
  }

  attribute html, String {
    definition "HTML representation (may include HTML entities)"
  }

  attribute mathml, String {
    definition "MathML representation in XML format."
  }

  attribute latex, String {
    definition "LaTeX representation in LaTeX typeset format."
  }

  attribute unicode, String {
    definition "Unicode representation in UTF-8 encoding."
```

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
  }
}
```

Figure 27

12.2.3. YAML structure

EXAMPLE

Symbol representation for the meter unit:

```
- id: m
  ascii: m
  html: m
  mathml: "<mi mathvariant='normal'>m</mi>"
  latex: "\\ensuremath{\\mathrm{m}}"
  unicode: m
```

12.3. Cross-reference model

12.3.1. General

The Cross-reference model establishes a consistent mechanism for linking entities across different files in UnitsDB.

12.3.2. Definition

```
class CrossReference {
  attribute identifier, String {
    definition "Identifier of the referenced entity"
  }
}
```

Figure 28

12.3.3. YAML structure

Cross-references follow a consistent pattern:

— References to dimensions use the format `#NISTdX` (e.g., `#NISTd1` for length)
— References to quantities use the format `#NISTqX` (e.g., `#NISTq1` for length)
— References to units use the format `#NISTuX` (e.g., `#NISTu1` for meter)
— References to prefixes use the format `#NISTpX_Y` (e.g., `#NISTp10_3` for kilo)

EXAMPLE

Cross-reference to the length dimension from a quantity:

```
dimension_url: "#NISTd1"
```


© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

Appendix A
(normative)
Test suite

This appendix provides a test suite for UnitsDB, comprising examples from the YAML files that demonstrate the correct implementation of the data models and normalization rules.

A.1. Test cases

A.1.1. Dimension test cases

These test cases verify the correct implementation of the Dimension model:

```yaml
# Test case 1: Base dimension (length)
NISTd1:
  length:
    powerNumerator: 1
    symbol: L
    dim_symbols:
    - id: dim_L
      ascii: L
      html: "&#x1D5AB;"
      mathml: "<mi mathvariant='sans-serif'>L</mi>"
      latex: "\\ensuremath{\\mathsf{L}}"
      unicode: "\U0001D5AB"

# Test case 2: Derived dimension (force)
NISTd12:
  length:
    powerNumerator: 1
    symbol: L
  mass:
    powerNumerator: 1
    symbol: M
  time:
    powerNumerator: -2
    symbol: T

# Test case 3: Dimensionless quantity
NISTd67:
  dimensionless: true
```

Figure A.1

A.1.2. Quantity test cases

These test cases verify the correct implementation of the Quantity model:

```yaml
# Test case 1: Base quantity (length)
NISTq1:
  dimension_url: "#NISTd1"
  quantity_type: base
  quantity_name:
  - length
  unit_reference:
  - name: meter
    url: "#NISTu1"
  - name: foot
    url: "#NISTu78"
```

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
# Test case 2: Derived quantity (velocity)
NISTq12:
  dimension_url: "#NISTd11"
  quantity_type: derived
  quantity_name:
  - velocity
  - speed
  unit_reference:
  - name: meter per second
    url: "#NISTu1.u3e-1/1"
  - name: kilometer per hour
    url: "#NISTu160"

# Test case 3: Quantity with multiple names
NISTq3:
  dimension_url: "#NISTd3"
  quantity_type: base
  quantity_name:
  - time
  - duration
  unit_reference:
  - name: second
    url: "#NISTu3"
```

Figure A.2

A.1.3. Unit test cases

These test cases verify the correct implementation of the Unit model:

```
# Test case 1: Base unit (meter)
NISTu1:
  dimension_url: "#NISTd1"
  short: meter
  root: true
  unit_system:
    type: SI_base
    name: SI
  unit_name:
  - meter
  unit_symbols:
  - id: m
    ascii: m
    html: m
    mathml: "<mi mathvariant='normal'>m</mi>"
    latex: "\\ensuremath{\\mathrm{m}}"
    unicode: m
  root_units:
    enumerated_root_units:
    - unit: meter
      power_denominator: 1
      power_numerator: 1
  quantity_reference:
  - name: length
    url: "#NISTq1"

# Test case 2: Derived unit (meter per second)
NISTu1.u3e-1/1:
  dimension_url: "#NISTd11"
```

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
  short: meter per second
  root: false
  unit_system:
    type: SI_derived_non-special
    name: SI
  unit_name:
  - meter per second
  unit_symbols:
  - id: mps
    ascii: m/s
    html: m/s
    mathml: "<mi mathvariant='normal'>m</mi><mo>/</mo><mi mathvariant='normal'>s
</mi>"
    latex: "\\ensuremath{\\mathrm{m/s}}"
    unicode: m/s
  root_units:
    enumerated_root_units:
    - unit: meter
      power_denominator: 1
      power_numerator: 1
    - unit: second
      power_denominator: 1
      power_numerator: -1
  quantity_reference:
  - name: velocity
    url: "#NISTq12"

# Test case 3: Prefixed unit (kilometer)
NISTu1p10'3:
  dimension_url: "#NISTd1"
  short: kilometer
  prefixed: true
  root: false
  unit_system:
    type: SI_derived_non-special
    name: SI
  unit_name:
  - kilometer
  unit_symbols:
  - id: km
    ascii: km
    html: km
    mathml: "<mi mathvariant='normal'>km</mi>"
    latex: "\\ensuremath{\\mathrm{km}}"
    unicode: km
  root_units:
    enumerated_root_units:
    - unit: meter
      power_denominator: 1
      power_numerator: 1
      prefix: k
  quantity_reference:
  - name: length
    url: "#NISTq1"
```

Figure A.3

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

A.1.4. Prefix test cases

These test cases verify the correct implementation of the Prefix model:

```yaml
# Test case 1: Standard SI prefix (kilo)
NISTp10_3:
  name: kilo
  symbol:
    ascii: k
    html: k
    latex: k
    unicode: k
  base: 10
  power: 3

# Test case 2: Small SI prefix (micro)
NISTp10_-6:
  name: micro
  symbol:
    ascii: u
    html: "&micro;"
    latex: "$mu$"
    unicode: µ
  base: 10
  power: -6

# Test case 3: Binary prefix (kibi)
NISTp2_10:
  name: kibi
  symbol:
    ascii: Ki
    html: Ki
    latex: Ki
    unicode: Ki
  base: 2
  power: 10
```

Figure A.4

A.1.5. Unit system test cases

These test cases verify the correct implementation of the UnitSystem model:

```yaml
# Test case 1: SI base unit system
- id: SI_base
  name: SI
  acceptable: true

# Test case 2: SI derived unit system (with special names)
- id: SI_derived_special
  name: SI
  acceptable: true

# Test case 3: Non-SI but acceptable unit system
- id: non-SI_acceptable
  name: non-SI
  acceptable: true

# Test case 4: Non-SI and not acceptable unit system
- id: non-SI_not_acceptable
```

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

```
  name: non-SI
  acceptable: false
```

Figure A.5

A.2. Cross-reference validation

The following tests verify the integrity of cross-references across the UnitsDB files:

A.2.1. Dimension to quantity references

This test verifies that each dimension is referenced by at least one quantity:

```
# For each dimension NISTdX in dimensions.yaml:
# - Verify there exists at least one quantity in quantities.yaml where dimension_
url: "#NISTdX"

NISTd1 (length) → Referenced by NISTq1 (length), NISTq48 (distance), etc.
NISTd2 (mass) → Referenced by NISTq2 (mass)
NISTd3 (time) → Referenced by NISTq3 (time)
```

Figure A.6

A.2.2. Quantity to unit references

This test verifies that each quantity references valid units:

```
# For each quantity NISTqX in quantities.yaml:
# - For each unit_reference, verify that the referenced unit exists in units.yaml

NISTq1 (length) → References NISTu1 (meter), NISTu78 (foot), etc.
NISTq12 (velocity) → References NISTu1.u3e-1/1 (meter per second), NISTu160
(kilometer per hour)
```

Figure A.7

A.2.3. Unit to quantity references

This test verifies that each unit references valid quantities:

```
# For each unit NISTuX in units.yaml:
# - For each quantity_reference, verify that the referenced quantity exists in
quantities.yaml

NISTu1 (meter) → References NISTq1 (length)
NISTu1.u3e-1/1 (meter per second) → References NISTq12 (velocity)
```

Figure A.8

A.3. Validation techniques

The following techniques can be used to validate UnitsDB data files:

A.3.1. Schema validation

YAML schema validation can be performed to ensure that each file adheres to the expected structure. This includes checking:

— Required fields are present
— Field types are correct

© The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved

— Enumerated values are valid

A.3.2. Reference integrity validation

Reference integrity validation ensures that all cross-references between entities are valid:

— Verify dimension references from quantities
— Verify unit references from quantities
— Verify quantity references from units
— Verify dimension references from units

A.3.3. Dimensional consistency validation

Dimensional consistency validation ensures that the dimensional relationships are mathematically consistent:

— Verify that units have the same dimension as their referenced quantities
— Verify that derived dimensions have correct combinations of base dimensions
— Verify that derived units have correct combinations of root units

A.3.4. Normalization rule validation

Validation of normalization rules ensures that the data adheres to the specific formatting and structural requirements:

— Verify naming conventions for identifiers
— Verify the structure of root_units for derived units
— Verify the correct use of prefixes
— Verify unit system categorization

These test cases and validation techniques provide a comprehensive framework for ensuring the integrity and correctness of the UnitsDB database.

 © The Calendaring and Scheduling Consortium, Inc. 2025 – All rights reserved