BSR/ASHRAE Standard 232P

Public Review Draft

Common Content and Specifications for Building Data Schemas

First Public Review (March 2024)
(Complete Draft for Full Review)

This draft has been recommended for public review by the responsible project committee. To submit a comment on this proposed standard, go to the ASHRAE website at www.ashrae.org/standards-research--technology/public-review-drafts and access the online comment database. The draft is subject to modification until it is approved for publication by the Board of Directors and ANSI. Until this time, the current edition of the standard (as modified by any published addenda on the ASHRAE website) remains in effect. The current edition of any standard may be purchased from the ASHRAE Online Store at www.ashrae.org/bookstore or by calling 404-636-8400 or 1-800-727-4723 (for orders in the U.S. or Canada).

The appearance of any technical data or editorial material in this public review document does not constitute endorsement, warranty, or guaranty by ASHRAE of any product, service, process, procedure, or design, and ASHARE expressly disclaims such.

ASHRAE, 180 Technology Parkway, Peachtree Corners GA 30092

Table of Contents

(This foreword is not part of this standard. It is merely informative and does not contain requirements necessary for conformance to the standard. It has not been processed according to the ANSI requirements for a standard and may contain material that has not been subject to public review or a consensus process. Unresolved objectors on informative material are not offered the right to appeal at ASHRAE or ANSI.)

FOREWORD

When a standard needs to include a data model, the first thing that the authors need to decide is how to format and document the data model. This effort takes time away from developing the content of the standard. Standard 232 aims to provide a standard method for designing and documenting data models that are included in other standards. This standardization has the benefit of freeing up authors of other standards to concentrate on just the content of the standard and for adopters of multiple standards with data models from having to take time to determine how the data model is formatted and specified.

This standard defines the structures, conventions, and formats (i.e., meta-schema) for data models based on schemas, such as, XML or JSON. It is not intended for semantic schemas. As much as possible, the standard is schema format agnostic and describes the content in a generic fashion. In this way, the standard can continue to be relevant even as new schema formats are developed. The standard does not include any data models of its own.

The standard has multiple parts. The first part of the standard describes how to structure and format the data model. The next part describes how to document the data model in a human-readable format. Then data groups that are common to all Standard 232 compliant data models and data groups that are generic and universal for use in other data models are described. Finally, informative naming guidelines and examples are included.

The developers of Standard 232 recognize that there are many ways to format and document a data model and some alternatives may work better than those chosen here for a specific data model. But the benefits of overall standardization across multiple data models outweigh the limits of standardization. Since it is impossible to cover everything that may need to be included in a data model, the intent is for the information included in the standard to be extensible when needed, as long as the rules established in the standard are not violated. The hope that any extensions needed by other standards will be suggested back to this standard for possible inclusion in future versions.

The content of Standard 232 was first developed in the writing of ASHRAE Standard 205 and further refined in the IBPSA-USA Building Data Exchange committee. It was moved into a stand-alone standard to improve referencing by other standards such as ASHRAE Standard 229.

1. PURPOSE

1.1 Purpose. Define metaschemas (such as data types, data elements, naming conventions, and formats) to specify and validate other standard schemas for data exchange among building performance and HVAC&R software.

2. SCOPE

2.1 Scope. This standard applies to data models and schemas specified in other standards for the design, operation, and performance of buildings.

3. DEFINITIONS AND SYMBOLS

3.1 Definitions

alternative: a set of data types allowed for a data element where one, and only one corresponding value is provided.

array: an ordered collection of values of a single data type.

attribute: provides extra information about a data element. Attributes have name and type properties and are defined within the data model specification. An attribute can appear 0 or 1 times within a given element. Attributes are either optional or required (by default they are optional).

data element: a named data item with an explicit data type.

data group: multiple data elements grouped together

data model: a collection of data groups and enumerations.

data model specification: a document of a data model and its relevant context.

data type: an attribute that specifies how to interpret the value of the data (see Section 5.3 for data type definitions).

enumeration: a data type consisting of a finite set of enumerators. Enumerations define a "controlled vocabulary" for the value for an attribute.

enumerator: a member of an enumeration set whose name is a unique identifier that behaves as a constant in a computer language.

regular expression: (sometimes called a rational expression) a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching.

serialization: the process of converting a data object—a combination of code and data represented within a region of data storage—into a series of bytes that saves the state of the object in an easily transmittable form.

***serialization format*:** the file format used to transmit a serialization

unit system: units of measurement used within a data model (e.g., I-P vs SI).

4. COMPLIANCE WITH THE STANDARD

To comply with this standard, a data model shall include a schema which validates versus the specifications in the standard and documentation which includes the sections specified in the standard.

4.1 Extensibility

A data model that extends the specification included in the standard or includes additional documentation is still compliant if it does not change or contradict the requirements included in the standard.

A data model specification shall clearly state whether the data included in the data model is extensible or not.

4.2 Schema Format

The data model shall state which schema formats are allowed (JSON schema, XSD, etc.), to define the data model.

4.3 Serialization Format

The data model shall state which serialization formats (XML, JSON, etc.), are allowed to transmit data complying to the specification.

5. DATA MODEL CONTENT

A data model organizes data elements in data groups and standardizes how the data elements relate to one another.

All named data model components shall be case-sensitive and unique within their scope.

Informative note: Conventions used within this document generally follow those defined in the Python Style Guide (Rossum 2013).

5.1 Data Group Composition.

A data group is a collection of data elements and embedded data groups.

Data group names shall be "CapitalizedWords". The names of data groups that conform with specific, reusable data structures (e.g., TimeSeries and LookupTable) shall be prefixed with their type name. Data groups that do not fall within this category shall not be prefixed with established type names.

Informative note: Regular expression pattern: ([A-Z]([A-Z]|[a-z]|[0-9])*$)

5.2 Data Element Attributes.

Data elements shall be characterized in data groups by the attributes shown below.

Attribute	**Description**	**Notes**
Name	Public name of data element	
Description	Text description that defines the meaning of the data element	
Type	Type of data element	See Section 5.3
Units	Units of data element	See Section 5.4
Constraints	A list of constraints on the data element value that can be verified against the schema	See Section 5.5
Required	Indicates whether data element is mandatory when containing data group is present	See Section 5.6

ID	Indicates whether data element can be used to reference an instance of the data group	See Section 5.7
Notes	Any supplementary information	

Data element names shall be “lower_case_with_underscores”.

Informative note: Regular expression pattern: (^([a-z]+)(_([a-z]|[0-9])+)*$)

5.3 Data Type Definitions.

Each data element shall have one of the data type attributes described below.

5.3.1 Fundamental Data Type Definitions

Data Type	Description	Examples
Integer	A positive or negative whole number (i.e., a number that can be written without a fractional part).	3, 19, -4
Numeric	A number that may include a fractional part with optional leading sign and optional exponent (engineering notation).	3.43, 0, -4, 1.03e4
Boolean	True or false.	true, false
String	A sequence of characters of any length using any (specified) character set.	Indirect evaporative cooler

5.3.2 Common String Data Type Definitions

The following data types are a predefined subset of the fundamental “string” data type conforming to the regular expression patterns provided.

Informative Note: Data models may append additional string data types that are useful for their purposes.

Data Type	Description	Regular Expression Pattern	Examples
UUID	An effectively unique character string conforming to ITU-T Recommendation X.667 (ITU-T 2012).	[0-9,a-f,A-F]{8}-[0-9,a-f,A-F]{4}-[0-9,a-f,A-F]{4}-[0-9,a-f,A-F]{4}-[0-9,a-f,A-F]{12}	“123e4567-e89b-12d3-a456-426655440000”
Date	A calendar date formatted per ISO 8601 (ISO 2019) Informative Note: for formats that include a fundamental data type	[0-9]{4}-[0-9]{2}-[0-9]{2}	“2015-04-29”

	for Date that can be used instead of a string		
Timestamp	Date with UTC time formatted per the extended format in ISO 8601 (ISO 2019) Informative Note: for formats that include a fundamental data type for DateTime that can be used instead of a string	[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}([\+-]?[0-9]{2}:[0-9]{2})?	“2016-06-29T14:35”
GenericTimestamp	Timestamp without denoting a specific year. In the form (G/F)(+/-)Y(YYY)-MM-DDTHH:MM:SS.f. G: Gregorian Calendar (leap years every 4 years except years divisible by 100) F: Fixed Calendar (no leap years) Informative Note: Negative years indicate years before year zero and may be used to indicate timestamps before timestamps of interest (e.g., initialization periods for simulated data).	[GF][\+-]?[0-9]+-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}([\+-]?[0-9]{2}:[0-9]{2})?	“F0-01-01T00:00:00”
TimeDuration	Duration of time formatted per ISO 8601 (ISO 2019) Informative Note: for formats that include a fundamental data type for DateTime that can be used instead of a string	^(-?)P(?=\d|T\d)(?:(\d+)Y)?(?:(\d+)M)?(?:(\d+)([DW]))?(?:T(?:(\d+)H)?(?:(\d+)M)?(?:(\d+(?:\.\d+)?)S)?)?$	“P1Y2DT0H3M”
Version	Version identifier in the form major.minor.patch as defined by SemVer	(0|[1-9][0-9]*)\.(0|[1-9]][0-9]*)\.(0|[1-9]][0-9]*)(?:-((?:0|[1-9]][0-9]*|][0-9]*[a-zA-Z-][0-9a-zA-Z-]*)(?:\.(?:0|[1-9]][0-9]*|][0-9]*[a-zA-Z-][0-9a-zA-Z-]*))*))?(?:\+([0-9a-	“1.1.3”, “1.2.0-beta-92”

	(SemVer 2016).	zA-Z-]+(?:\.[0-9a-zA-Z-]+)*))?	

Informative note: Examples of GenericTimestamps are provided in Informative Appendix B: Generic Timestamp Examples.

5.3.3 Derived and Composite Data Type Definitions

5.3.3.1 Data Group

Nested data groups shall be denoted by defining a data element whose data type is the name of a defined data group wrapped in round brackets and predicated by "Group". Example: Group(`PerformanceMapCooling`).The nested data group can be constrained to have a data element with a specific value (See Section 5.5.6).

5.3.3.2 Enumeration

An enumeration is a data type that takes one of a predefined set of named enumerator values. Each enumeration shall be given a unique data type name and shall define the set of string enumerators. Data elements representing enumerations shall be denoted in a data group by wrapping the name of the enumeration in round brackets predicated by "Enumeration" as the data type. Example: Enumeration(`CompressorType`).

Informative note: In computer science, an enumeration is a set of values that is converted to a list of constants (e.g., 0, 1, 2, 3, etc.).

Attribute	**Description**	**Notes**
Enumerator	Public name of enumerator	
Description	Text description that defines the meaning of the enumerator	
Display Text	Text used by an application for enumerator	
Notes	Any supplementary information	

Enumeration names shall be "CapitalizedWords". Enumeration names shall not be prefixed with type names used for specific, reusable data structures.

Enumeration data elements shall have names that match their enumeration type (e.g., `compressor_type` should be the name of a data element of the `CompressorType` enumeration) except where more than one data element of the same enumerated type is used in the same data group. In that situation, distinguishing prefix(es) shall be added to the type name. For example, if a device uses two liquids, data elements could be named `condenser_liquid_type` and `evaporator_liquid_type`.

Informative note: Regular expression pattern: (^([A-Z]([A-Z]|[a-z]|[0-9])*$)

Enumerator names shall be "UPPER_CASE_WITH_UNDERSCORES". This is the convention often used for constant values.

Informative note: Regular expression pattern: (^([A-Z]([A-Z]|[0-9])*)(_([A-Z]|[0-9])+)*$)

5.3.3.3 Array

Arrays shall be denoted in a data group by wrapping the data type of a data element in round brackets predicated by "Array". Examples: Array(Numeric) or Array(Enumeration(`CompressorType`)).

5.3.3.4 Alternative

A set of alternative data types where one, and only one corresponding value is provided. Alternatives shall be denoted in a data group by wrapping a comma separated list of alternative data types in round brackets predicated by "Alternative". Examples: Alternative `(Numeric, String)` or Alternative `(Group(RS0001), Group(RS0002))`.

5.3.3.5 Pattern
A regular expression pattern as defined by ECMA-262 (ECMA 2019). Used for characterizing data that corresponds to a group (e.g., model numbers).

Informative note: For example: CA225FB.[1-9]

5.3.3.6 References

An alternative to embedding data groups is supplying an identifying name of a data group instance. This can be useful for reducing redundant data in a serialized instance of the data model. Any data group used as a reference must have a data element called "id".

Data element "ids" shall be unique among data elements of the data group type within an instance of the data model.

References shall be denoted in a data group by wrapping the referenced data group in round brackets predicated by "Reference".

Example: "`Reference(Group(DataGroup))`".

References may be combined with embedded data groups to allow for flexible input. Example: "`(Reference(Group(DataGroup)), Group(DataGroup))`".

5.4 Units

5.4.1 Units in Machine-Readable Schema

Units of all values in all data models shall be documented using symbols defined below.

If a numeric data element does not have units, the hyphen (-) character shall be used for its units.

Symbols shall appear in lower case unless the unit name has been taken from a proper name. In this case, the first letter of the symbol shall be capitalized.
When combining base units into derived units, the following rules shall apply:

- For a symbol raised to a power, use the symbol followed by the power (e.g., m2).
- For the product of two symbols, use the hyphen (-) (e.g., N-m).
- For the quotient of two symbols, use the solidus (/) (e.g., W/m2-K)
- Use only one solidus symbol per derived unit (e.g., m/s2, not m/s/s).
- Do not use parentheses (e.g., W/m2-K, not W/(m2-K)).
- Do not use negative exponents (e.g., W/m2-K, not W-m-2-K-1).

5.4.2 Units in Human-Readable Documentation

Units of all values in all data models shall be documented using symbols defined below.

If a numeric data element does not have units, the hyphen (-) character shall be used for its units.

Symbols shall appear in lower case unless the unit name has been taken from a proper name. In this case, the first letter of the symbol shall be capitalized.

When combining base units into derived units, the following rules shall apply:

- For a symbol raised to a power, use the symbol followed by the power as a superscript (e.g., m^2).
- For the product of two symbols, use the interpunct or half-raised dot (·) (e.g., N·m).
- For the quotient of two symbols, use the solidus or forward slash (/) (e.g., W/m^2)
- Use only one solidus symbol per derived unit (e.g., m/s^2, not m/s/s).
- Use parentheses for any denominator that includes more than one symbol. (e.g., $W/(m^2 \cdot K)$, not $W/m^2 \cdot K$).
- Do not use negative exponents (e.g., $W/(m^2 \cdot K)$, not $W \cdot m^{-2} \cdot K^{-1}$).

5.4.3 Unit Systems (Informative)

When selecting the unit system used in a Data Model, the following should be considered:

- Should allow for values to be represented exactly.
- Data elements should only have a single unit designated as using dual units introduces potential conversion errors that mean the exact value cannot be determined.
- SI units are preferred but other unit systems could be considered for data models that have historically used other unit systems or exclusively apply to regions adopting different unit systems.
- for data elements that have native units those should be used.
- Should be applied consistently across the data model.

5.5 Constraints

The following constraints are used for the “Constraints” *data element* attribute. When multiple constraints apply to the same data element, the constraints shall be specified as a list.

5.5.1 Range

The constraint for numerical elements with valid minimum and/or maximum values shall be expressed as numerical constants using <, <=, >=, or >. When the constraint is applied to an array data type, the constraint applies to all values in the array.

Informative note: A numerical data element that is required to be greater or equal to zero has the constraint defined as `>=0`.

5.5.2 Multiple

The constraint for numerical elements that must be multiples of a number shall be expressed using %.

Informative note: A numerical data element that is required to be a multiple of two has the constraint defined as "`%2`".

5.5.3 Set

The constraint for elements that must be one of a set of valid values shall be specified by placing the valid values in between round brackets predicated by the word "Set", separated by commas.

Informative note: A numerical data element that is required to be the one of the years 2005, 2008, and 2012 has the constraint defined as Set(`2005,2008,2012`).

Informative note: A set constraint compares the value of the data element versus the list of allowed values while an enumeration is a set of values that is converted to a list of constants (e.g., 0, 1, 2, 3, etc.)

5.5.4 Selector

The constraint for alternative data elements where the choice of alternative depends on the value of an enumeration data element shall be specific by placing the corresponding enumerator values in a comma separated list enclosed in parentheses following the enumeration data element name. In this case, the order of the enumerator values shall follow the same order of their corresponding alternative selection.

Informative note: Example: If a data element `performance_map` has the alternative data type of `(Group(PerformanceMapDiscrete), Group(PerformanceMapContinuous))` and there is an enumeration data element `performance_map_type` that has enumerators `CONTINUOUS` and `DISCRETE`, then the selector constraint on `performance_map` would be stated as `performance_map_type(CONTINUOUS, DISCRETE)`.

5.5.5 String Pattern

The constraint for string data elements that must match a specific pattern shall be specified as a regular expression of the pattern. For string data elements where all possible options are known, an enumeration shall be used instead of a string pattern (see Section 5.3.3.2).

Informative note: When a string pattern constraint is to be used on multiple data elements, then a specific string data type should be defined instead.

5.5.6 Data Element Value

The constraint for nested *data group data elements* that must have a *data element* with a specific value shall be specified by placing the name of the *data element* followed by the equals sign (=) and the required value.

Informative note: Example: If a data element `curve_fit` has a nested data group data type `{CurveFit}` having an enumeration data element `type` that is required to have a value of `LINEAR`, then the constraint on `curve_fit` is stated as `type=LINEAR`.

5.5.7 Array Length Limits

Limitations on array lengths shall be denoted using a set of square brackets containing the minimum and maximum lengths of the array separated by two periods. Example: A numeric array that must have at least one value, but no more than four values would appear as `[1..4]`.

Arrays with no minimum length (that is, it may have zero values) shall be denoted with no value before the two periods. Example: `[..4]`.

Arrays with no maximum length shall be denoted with no value after the two periods. Example: `[1..]`.

5.6 Required Data Elements

The "Required" data element attribute is used to indicate the conditions where a data element value is required if the containing data group is present. The following conditions are allowed.

5.6.1 Unconditional

The data element is unconditionally required. A data element that is unconditionally required shall be denoted with a Boolean true.

5.6.2 Prerequisite Definition

The data element value shall be required if a specific prerequisite data element is defined in the serialization regardless of the value of the prerequisite data element. A data element dependent, that is required when the prerequisite data element prerequisite is defined, has the requirement stated as `if prerequisite present`. A data element that is required when the prerequisite data element is not defined has the requirement stated as `if prerequisite not present.`

5.6.3 Prerequisite Value

The data element value shall be required if a specific prerequisite data element is defined and is equal to (or not equal to) a specific value in the serialization. A data element `option_a`, that is required when the prerequisite data element `option_type` has the value `OPTION_A`, has the requirement stated as `if option_type=OPTION_A`. Similarly, a data element `minimum_speed`, that is required when the prerequisite data element `speed_type` does not have the value `SINGLE_SPEED`, has the requirement stated as `if speed_type!=SINGLE_SPEED`. The symbol != shall be used to denote not equals to.

5.6.4 Combining Prerequisite Conditions

When multiple prerequisite conditions are needed to define when a data element is required, these conditions are combined using `and` and/or `or` and grouped as needed with parentheses. Combined conditions begin with a single if.

Informative note: Example: `if (prerequisite present and option_type=OPTION_A) or option_type=OPTION_B`.

5.7 ID

The "ID" data element attribute is used to indicate when the data element value is used to reference an instance of the data group from a different data group. A data group shall have a maximum of one data element indicated as the "ID" data element. The value of an "ID" data element shall be unique from all other instances of the data group. The "ID" data element shall be denoted by a Boolean true.

6. DATA MODEL SPECIFICATIONS

The Data Model specification shall include human-readable documentation that includes all the information in this section.

6.1 Documentation

6.1.1 Identification

A string code which uniquely identifies each data model.

6.1.2 Version History

A version history for the data model shall be included that includes the version number, the date the version is published, and a description of relevant changes.

Informative note: There are many possible formats for the version history, including a simple table documenting the versions, semantic versioning, and links to repositories. A method that works best for the specific data model should be used.

6.1.3 Use Case

The intended use case for the Data Model describing how and why the Data Model is to be used.

Informative note: The description of the use case should conform to ASHRAE Guideline 20 (ASHRAE 2016).

6.1.4 Scope and Description

A narrative section providing information that defines the content covered by the data model. The following sections shall appear in each data model specification.

6.1.4.1 Applicability

A description of the intended applications covered by the data model.

6.1.4.2 Exclusions

A description of related applications explicitly not covered by the data model. Data models with no identified exclusions shall indicate this by stating "None".

6.2 Data Model

This section specifies the local data groups and local enumerations that comprise the data model.

6.2.1 Data Model Hierarchy

Each data model shall illustrate the hierarchy and relationship of data groups comprising a conforming data model instance.

6.2.2 Enumerations

Any enumerations specific to the data model shall be defined prior to any data group definitions where they are referenced.

6.2.3 Data Groups

A collection of tabular representations of the data groups that comprise the data model.

6.2.4 Verification Rules

Data model documents shall include a description of computable rules used to verify minimal data validity and accuracy.

Basic data format rules are implicit in data element data types and are enforced via automated validation against a schema. Valid value ranges shall be included in the "Constraints" attribute of the data element definitions. Rules of this type shall not be restated in Verification Rules.

Additional rules shall be included in the "Verification Rules" section specific to the application of the data model. Typical examples are logical relationships among values and physical constraints such as:

- Cross-element consistency checks.
- Physically-based tests that allow detection of impossible values. For example:
    - Psychrometric states that imply greater than 100% relative humidity
    - Geometric surface points shall be coplanar

The failure of any verification test shall indicate invalid data.

A fully verified data model instance shall be both validated against a schema and verified based on the Verification Rules.

Informative note: The set of verification tests in a data model specification may not be sufficient to detect all invalid data.

6.2.5 Publication Rules

This section shall provide instructions and advice for data publishers to create data model serializations.

Informative note: Any information that makes it easier for the data publisher to understand the data requirements (e.g., minimum grid spacings) can be included in this section.

6.2.6 Application Rules

This section shall provide instructions and advice for application developers to use the data included in data model serializations.

Informative note: Any information that makes it easier for the application developer in consuming the data (e.g., modeling assumptions and/or extrapolation methods) can be included in this section.

7. COMMON DATA GROUPS

7.1 Metadata

Name	Description	Type	Units	Constraints	Required	Notes
`schema_author`	Name of the organization that published the schema	`String`			true	Identifies the organization that defined the schema
`schema_name`	Schema name or identifier	`String`			true	Identifies the schema used to define the data content
`schema_version`	The version of the schema the data complies with	`Version`			true	
`schema_url`	The Uniform Resource Locator (url) for the schema definition and/or documentation	`String`				
`author`	Name of the entity creating the serialization	`String`			true	Identifies the organization that created the file.
`id`	Unique identifier	`UUID`				Assigned by data publisher to identify the contained data Shall remain unchanged for revised data
`description`	Description of data (suitable for display)	`String`			true	
`time_of_creation`	Timestamp indicating when the serialization was created	`Timestamp`			true	Updated anytime any data content is modified
`version`	Integer version identifier for the data	`Version`				Used by data publisher to track revisions of the data Shall be incremented for each data revision

source	Source(s) of the data	String				Used by data publisher to document methods (e.g., software and version) used to generate data Informative note: source may be different from other data source(s) included elsewhere within the data
disclaimer	Characterization of accuracy, limitations, and applicability of this data	String				
notes	Additional Information	String				

7.1.1 Versioning

There are three data elements to be used to provide versioning of a data model serialization.

• The `schema_version` indicates the version of the data model specification schema that was used to create the serialization.

• The `id` uniquely identifies the set of data covered by the serialization. An id shall be assigned when the serialization is first created.

• The `version` indicates the version of the data included in the serialization. The `version` shall be incremented whenever any data in the serialization is modified.

Informative note: The `id` and the `version` can be used in conjunction to provide versioning that both denotes what the data in the serialization covers (the `id`) and different versions of that data (the `version`). Not all data models will need all three data elements for versioning. Only `schema_version` is a required data element and the others can be used when needed to provide complete version information for the data in the serialization.

8. REUSABLE DATA GROUPS

8.1.1.1 TimeSeries

TimeSeries

Name	Description	Type	Units	Constraints	Required	Notes
display_name	The name for the time series data	String			true	
units	The units for the time series data	String			true	
value_type	The TimeSeriesType of the data	Enumeration(TimeSeriesType)			true	

value_time_intervals	The time intervals for the data	Reference(Group(TimeIntervals))			true	
values	Array with the series values	Array(Numeric)		[1..]	true	
source_time_intervals	The time intervals for the source data type for the values	Reference(Group(TimeIntervals))				
source	The DataSourceType for the data	Array(Enumeration(DataSourceType))		[1..]		
uncertainty_time_intervals	The time intervals for the uncertainty of the values	Reference(Group(TimeIntervals))				
uncertainty	The uncertainty of the values	Array(Numeric)		[1..]		
notes_time_intervals	The time intervals for the notes for the values	Reference(Group(TimeIntervals))				
notes	The notes for the values	Array(String)		[1..]		

TimeSeriesType

Enumerator	Description	Notes
INSTANTANEOUS	Instantaneous	Values reflect the instant of the current timestamp
AVERAGE	Average	Values reflect the average between the previous timestamp and the current timestamp
SUM	Sum	Values reflect the integrated sum between the previous timestamp and the current timestamp
CUMULATIVE	Cumulative	Values reflect the cumulative sum between the starting timestamp and the current timestamp

DataSourceType

Enumerator	Description	Notes
DIRECT_MEASURED	Direct Measurement	
DERIVED_MEASURED	Derived Measurement	
MODELED	Models	
INTERPOLATED	Interpolated	
ASSUMPTION	Assumption	
UNKNOWN	Unknown	

TimeIntervals

Name	Description	Type	Units	Constraints	Required	Notes
id	Reference id	ID	-		true	

`starting_time`	Beginning of the data	`Alternative(Timestamp,GenericTimestamp)`			true	The starting time for the time series.
`regular_intervals`	Duration of regular intervals	`TimeDuration`	-		If timestamps not present and intervals not present	
`intervals`	Array of time intervals	`Array(TimeDuration)`	-	`[1..]`	If timestamps not present and regular_intervals not present	Informative note: For repeating intervals, the ISO 8601 format of 'Rn/' can be used.
`timestamps`	Array of timestamps	`Alternative(Array(Timestamp),Array(GenericTimestamp))`		`[1..]`	If intervals not present and regular_intervals not present	
`labels`	Informal labels describing each interval	`Array(String)`				e.g., month names for monthly intervals
`notes`		`Array(String)`				

Informative note: Examples of TimeIntervals instances are provided in Informative Appendix C: Time Interval Examples.

8.1.1.2 Binary

Name	Description	Type	Units	Constraints	Required	Notes
`binary_encoding`	Encoding used for binary string	`Enumeration`			true	Identifies the encoding method used to create the binary string
`binary_string`	The binary data string	`String`			true	The binary data string

9. REFERENCES

G. van Rossum, B. Warsaw, and N. Coghlan. *PEP 8 – Style Guide for Python Code*. 2013. URL: https://peps.python.org/pep-0008/#descriptive-naming-styles

ITU. *ITU-T X.667: Information technology – Procedures for the operation of object identifier registration authorities: Generation of universally unique identifiers and their use in object identifiers*. ITU, 2012.

ISO. *ISO 8601: Date and Time Format*. ISO, 2019.

SemVer. *Semantic Versioning 2.0.0*. 2016. URL: https://semver.org/.

ECMA. *Standard ECMA-262, ECMAScript® 2019 Language Specification*. 2019. URL: https:/ /www .ecmainternational.org/publications/standards/Ecma-262.htm.

ASHRAE. *ASHRAE Guideline 20-2010 Documenting HVAC&R Work Processes and Data Exchange Requirements*. Atlanta, Georgia, 2010.

Informative Appendix A: Naming Guidelines

Readability is more important than length. Although there is the potential that long names will increase the size of serializations, accurate understanding and application of data is the overriding consideration. Unambiguous and expressive names are preferred, for example, `evaporator_pressure_drop` rather than `evap_pres_drp`. It is also anticipated that messaging infrastructure will provide data compression for efficient transfer of serializations.

Consistently specify dimensions at beginning or end of name. Determine if dimensions should be stated at the beginning (`temperature_entering`) or the end (`entering_temperature`) of the name and apply the choice consistently throughout the data model.

Avoid names that include a defined unit of measurement. For example, do not use names such as `air_cfm` or `pump_gpm`. Instead, consider names such as `air_volumetric_flow_rate` or `pump_volumetric_flow_rate`.

Do not include data types in names. Use `ahri_rated`, not `ahri_rated_boolean`.

Avoid abbreviations and acronyms. Exceptions shall be allowed where abbreviations and/or acronyms are explicitly defined within the data model.

Avoid using names that conflict with widely used programming languages. For example, do not use “case”, “switch”, “default”, etc.

Avoid names that differ only in case. Not all programming languages are case-sensitive, so it is best to avoid names differing only in case.

Names should not include a repetition of the names of containing structures or data groups. The container provides adequate context; using its name in component names is redundant and needlessly lengthens component names. For example, the capacity of a chiller included in a `chiller` data group should be called simply `capacity` rather than `chiller_capacity`.

Consider “Type” at the end of enumeration names. Clear enumeration names denote that the enumerants are a list of related choices by utilizing terms like choice, options, type, enum, etc., in their names. Examples are `ControlChoice`, `InstallChoice`, `CompressorType`, and `EnumCondenser`.

A.1 Pre-existing Names

Where possible, use names and definitions from existing relevant data dictionaries, schemas, and ontologies in priority of the order shown below.

Source	Description	URL

BEDES	Building Energy Data Exchange Specification	https://bedes.lbl.gov/
ASHRAE Terminology	Assembled by ASHRAE Technical Committee (TC) 1.6, Terminology	https://www.ashrae.org/resources--publications/free-resources/ashrae-terminology
gbXML	Green Building XML	http://www.gbxml.org/
IFC	Industry Foundation Classes	http://www.buildingsmart.org/
CEC SDD	California Energy Commission Standards Data Dictionary	http://bees.archenergy.com/software.html
BuildingSync XML	Building data exchange format focused on Commercial Building Energy Audits	https://buildingsync.net
HPXML	Schema for exchanging data related to homes	https://www.hpxmlonline.com/
CityGML	3D city model standard	Citygml.org, https://www.ogc.org/standards/citygml
DOE-2 BDL	DOE-2 Building Description Language	http://doe2.com/DOE2/
EnergyPlus IDD	EnergyPlus Input Data Dictionary	https://energyplus.net/
COBie	Construction Operations Building Information Exchange	http://www.wbdg.org/resources/cobie.php
COMNET	Commercial Energy Services Network	http://comnet.org/
BrickSchema		https://brickschema.org/
QUDT		https://www.qudt.org/
obXML	Occupant Behavior XML	https://behavior.lbl.gov/?q=obXML

Informative Appendix B: Generic Timestamp Examples

Generic timestamps are useful when data is to be conveyed in a timeseries, but it does apply to a specific year. This could include typical weather, schedules, simulation outputs, etc. This data could be for generic years with or without including leap years.

The Generic Timestamp data type is similar to the Timestamp data type with some differences at the front of the timestamp.

The Generic Timestamp starts with a letter denoting whether the time series will include leap years or not. If the timestamp is preceded by the letter ‘F’ for fixed calendar, then none of the years will include leap days. If the timestamp is preceded by the letter ‘G’ for Gregorian calendar, the Gregorian calendar rules for leap years apply. This means that years divisible by 4 will include a leap day, except for years evenly divisible by 100, but not by 400.

Following the letter is an optional + or – sign. Negative years indicate years before year zero and may be used to indicate timestamps before timestamps of interest (e.g., initialization periods for simulated data).

Next is a 1-to-4-digit number denoting the year. The remainder of the generic timestamp follows the same rules as the regular timestamp.

The overall format can be denoted as (G/F)(+/-)Y(YYY)-MM-DDTHH:MM:SS.f.

Using this for monthly intervals for a year without a leap day would look like this:

```
timestamps: {F1-02-01T00:00, F1-03-01T00:00, F1-04-01T00:00, …}
```

and for hourly intervals would look like this:

```
timestamps: {F1-01-01T01:00, F1-01-01T02:00, F1-01-01T03:00, …}
```

For a year with a leap day monthly intervals would look like this:

```
timestamps: {G4-02-01T00:00, G4-03-01T00:00, G4-04-01T00:00, …}
```

and for hourly intervals would look like this:

```
timestamps: {G4-01-01T01:00, G4-01-01T02:00, G4-01-01T03:00, …}
```

For a year without a leap day occurring one year prior to the period of interest would look like this:

```
timestamps: {F-1-02-01T00:00, F-1-03-01T00:00, F-1-04-01T00:00, …}
```

and for hourly intervals would look like this:

```
timestamps: {F-1-01-01T01:00, F-1-01-01T02:00, F-1-01-01T03:00, …}
```

Informative Appendix C: Time Interval Examples

Before the time series data can be specified, the time intervals for the data need to be specified. Multiple time series can use the same time interval.

The first element in defining time intervals is the `starting_time` for the data. This represents the starting time of the first interval. Knowing the starting time is required so that using the starting time and the interval length defines when the next interval starts. If the time series data starts at the beginning of the year, the starting time would look like this:

```
starting_time: 2019-01-01T00:00
```

Next the intervals for the time series data need to be specified. There are multiple methods for defining the intervals. If the data occurs in regular intervals that are exactly the same for the entire time series, the `regular_intervals` data element can be used to specify that single regular interval. For example, specifying a monthly interval would look like this:

```
regular_intervals: P1M
```

and specifying an hourly interval would look like this:

```
regular_intervals: PT1H
```

For intervals that are not regular, there are two options for defining the intervals between the data values. The first is to use `intervals` to specify the length of the interval. Using this for monthly intervals would look like this:

```
intervals: {P1M, P1M, P1M, …}
```

and for hourly intervals would look like this:

```
intervals: {PT1H, PT1H, PT1H, …}
```

The other option is to specify the `timestamps` for the end of the intervals for the data value. Using this for monthly intervals would look like this:

`timestamps: {`2019-02-01T00:00, 2019-03-01T00:00, 2019-04-01T00:00, …}

and for hourly intervals would look like this:

`timestamps: {`2019-01-01T01:00, 2019-01-01T02:00, 2019-01-01T03:00, …}

There are two optional data elements that can be included in the time intervals data groups. The labels data element allows labels to be specified for each data value. For example, a label for each month of the year would look like this:

```
labels: {"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"}
```

Likewise, the `notes` data element allows for helpful information to be included for each value.