PSet Schema Specification

Introduction

The PSet Schema Specification defines a standard to create a schema for the property sets handled by the Trimble Connect PSetService. This document guides users to create schemas for different types for PSet data.

Table of Contents

The PSet Schema

Schema allows to describe the structure of the property set (pset) and restrict values of the properties in the pset.

Background

Json Schema and OpenAPI

The Json Schema and OpenAPI 2.0 specification were used as inspiration for defining this PSet Schema specification.

Properties in IFC

Original requirements for the PSetService came from the IFC specification. Following types of properties from the IFC specification are supported by this PSet Schema definition language.

1. single value
    a. not restricted (`IfcPropertySingleValue`)
    b. enumerated value (`IfcPropertyEnumeratedValue`)
2. list value (`IfcPropertyListValue`)
3. Complex values
4. ~~To be supported in the future: bounded value (`IfcPropertyBoundedValue`)~~
5. ~~To be supported in the future: table value (`IfcPropertyTableValue`)~~

Properties in TrimBIM

Another source of requirements was the TrimBIM format. Following are property types supported by TrimBIM (primarily driven by TS export feature):

```
enum PropertyType:byte {
 LengthMeasure = 0, // millimeters (mm)
 AreaMeasure = 1, // square meter (m2)
 VolumeMeasure = 2, // cubic meter (m3)
 MassMeasure = 3, // kilograms (kg)
 AngleMeasure = 4, // degrees
 StringValue = 5,
 IntValue = 6,
 DoubleValue = 7,
 DateTime = 8,
 Logical = 9, /* FALSE, TRUE, or UNKNOWN. */
```

```
 Boolean = 10 /* FALSE, TRUE */
}
```

Schema Level Attributes

Each pset schema has the following structure

```
 "schema" : {
 "open": true/false,
 "props": { … }
}
```

The only allowed keywords on the first level are:

- `open` - Value is boolean. Optional. Default is `false`, meaning that the pset is closed and no properties are allowed other than explicitly mentioned in the schema. If set to `true`, the pset is open and can have other not specified (and not validated properties).
- `prop` - Required. An object with descriptors for each property in the pset

Describing a property

Each property in the pset must have a unique identifier. The identifier of the property is alphanumeric STRING (max. 255 characters) which allows an individual prop to be identified. As a merely machine-readable string for identification purposes, an identifier is usually machine-generated and locale-independent.

- `type` - Value MUST be a string, see Data Types for supported data types. While relying on JSON Schema's defined types, we offer a few additional predefined types mostly related to different measurements.
- `items` - Value MUST be an object and not an array. Items MUST be present if the type is array.
- `format` - See Data Type Formats for further details.
- Additional attributes can be specified, see Validation Attributes.

Data Types

For the general type system refer to the JSON Schema spec. We have extended the standard type system with the number of measurement specific types. Note that following standard types are not supported: “`null`”. For arrays there are limitations that items can be of primitive type only.
Following data types are supported:

Type name	**Description**
`boolean`	The boolean type matches only two special values: `true` and `false`. Note that values that *evaluate* to `true` or `false`, such

	as 1 and 0, are not accepted by the schema.
array	Arrays are used for ordered elements. In JSON, each element in an array must be of the same type unlike the standard JSON. Arrays have following limitations: - Arrays of array are not supported - All items should of the same type
number	The `number` type is used for any numeric type, either integers or floating point numbers.
integer	The `integer` type is used for integral numbers.
string	The `string` type is used for strings of text. It may contain Unicode characters. The string content can be further specified with the `format` attribute (see below).
object	Objects are the mapping type. They map “keys” to “values”. This is also known as “complex” properties.
currencyMeasure	string with pattern "^[A-Z]{3} [0-9]+(.[0-9]{1,2})?$", e.g. "EUR 10.5" (units are in prefix notation, USD, EUR, CHF). Units are coming from the ISO 4217
complexNumber	A representation of a complex number expressed as an array with two elements. The first element denotes the real component which is the numerical component of a complex number whose square roots can be calculated explicitly. The second element denotes the imaginary component which is the numerical component of a complex number whose square roots cannot be determined other than through the provision of the square of the imaginary number i where i^2 = -1.

Measurement Data Types

Measurement data types correspond to data types defined as part of IFC4 standard. There are basic measure types of ISO 10303-41 and selected derived measure types.

Basic measure types

Most basic measure types coming from ISO 10303-41. Definition according to ISO/CD 10303-41:1992. The `nonNegativeLengthMeasure` is in addition to ISO 10303-41.
A measure value is a value as defined in ISO 31-0 (clause 2).

Type name	Description
`amountOfSubstanceMeasure`	An amount of substance measure is the value for the quantity of a substance when compared with the number of atoms in 0.012 kg of carbon 12. Measure in mole (mol).
`angleMeasure`	The plane angle. Floating number. degrees
`areaMeasure`	Floating number. square metre (m2)
`contextDependentMeasure`	The value of a physical quantity as defined within the exchange context. Floating number.
`countMeasure`	A count measure is the value of a count of items. Integer.
`descriptiveMeasure`	A descriptive measure is a human interpretable definition of a quantifiable value. The mode of interpretation has to be established for the exchange context. Type: `string`.
`electricCurrentMeasure`	The value for the movement of electrically charged particles. Measured in Ampere (A).
`lengthMeasure`	Floating number. millimetres (mm)
`luminousIntensityMeasure`	The value for the brightness of a body. Measured in candela (cd).
`massMeasure`	Floating number. kilograms (kg)
`nonNegativeLengthMeasure`	A non-negative length measure is a length measure that is greater than or equal to zero.
`normalisedRatioMeasure`	A dimensionless measure to express ratio values ranging from 0.0 to 1.0.
`numericMeasure`	the numeric value of a physical quantity.
`parameterValue`	The value which specifies the amount of a parameter in some parameter space.

planeAngleMeasure	Value of an angle in a plane, radian (rad, m/m = 1). Floating number.
positiveLengthMeasure	A length measure that is greater than zero.
positivePlaneAngleMeasure	A plane angle measure that is greater than zero.
positiveRatioMeasure	A ratio measure that is greater than zero.
ratioMeasure	The value of the relation between two physical quantities that are of the same kind. NOTE Input given in percent is to be divided by 100% when stored as a ratioMeasure. For example, 25% becomes 0.25.
solidAngleMeasure	The value of an angle in a solid. Measured in Steradians, (sr, m2/m2).
thermodynamicTemperatureMeasure	Definition from ISO/CD 10303-41:1992: A thermodynamic temperature measure is the value for the degree of heat of a body. Measured in degrees Kelvin (K).
angleMeasure	The plane angle. Floating number. degrees
timeMeasure	The time duration in seconds (s).
volumeMeasure	Floating number. cubic metre (m3)

Derived measure types

Type name	**Description**
angularVelocityMeasure	A measure of the velocity of a body measured in terms of angle subtended per unit time. Floating number. measured in radians/s.
curvatureMeasure	A measure for curvature, which is defined as the change of slope per length. This is typically a computed value in structural analysis. Floating number. measured in rad/m.

electricCapacitanceMeasure	A measure of the electric capacitance. Floating number. Measured in Farad (F, C/V = A s/V).
electricChargeMeasure	A measure of the electric charge. Floating number. measured in Coulomb (C, A s).
electricConductanceMeasure	A measure of the electric conductance. Floating number. measured in Siemens (S, 1/Ohm = A/V).
electricResistanceMeasure	A measure of the electric resistance. Floating number. measured in Ohm (V/A).
electricVoltageMeasure	A measure of electromotive force. Floating number. Measured in Volts (V, W/A).
energyMeasure	A measure of energy required or used. Floating number. Measured in Joules, (J, Nm).
forceMeasure	A measure of the force. Floating number. Measured in Newton (N, kg m/s2).
frequencyMeasure	A measure of the number of times that an item vibrates in unit time. Floating number. Measured in cycles/s or Hertz (Hz).
inductanceMeasure	A measure of the inductance. Floating number. Measure in Henry (H, Weber/A = V s/A).
linearVelocityMeasure	A measure of the velocity of a body measured in terms of distance moved per unit time. Floating number. Measured in m/s.
massDensityMeasure	A measure of the density of a medium. Floating number. Measured in kg/m3.
massPerLengthMeasure	A measure for mass per length. For example for rolled steel profiles the weight of an imaginary beam is expressed by kg/m length for cost calculation and structural analysis purposes. Floating number.

momentOfInertiaMeasure	A measure of moment of inertia. Floating number. Measured in m4.
monetaryMeasure	A monetary measure is the value of an amount of money without regard to its currency. Floating number.
powerMeasure	A measure of power required or used. Floating number. Measured in Watts (W, J/s).
pressureMeasure	A measure of the quantity of a medium acting on a unit area. Floating number. Measured in Pascals (Pa, N/m2).
radioActivityMeasure	A measure of activity of radionuclide. Floating number. Measured in Becquerel (Bq, 1/s).
soundPowerMeasure	A sound power measure is a measure of total radiated noise with units of watts (sonic energy per time unit). Floating number.
soundPressureMeasure	A sound pressure measure is a measure of the pressure fluctuations superimposed over the ambient pressure level with units of pascals. Floating number.
thermalExpansionCoefficientMeasure	A measure of the thermal expansion coefficient of a material, which expresses its elongation (as a ratio) per temperature difference. It is measured in 1/K. A positive elongation per (positive) rise of temperature is expressed by a positive value. Floating number.
torqueMeasure	A measure of the torque or moment of a couple. Floating number. Measured in N m
volumetricFlowRateMeasure	A measure of the volume of a medium flowing per unit time. Floating number. Measured in m3/s.
massFlowRateMeasure	A measure of the mass of a medium flowing per unit time. Floating number. Measured in kg/s.

More measurement types will be added later.

Data Type Formats

Following data formats are supported. In most cases formats conform to the Json Schema spec, but there are additional formats as well (like `user-id`, `duration`, `query`).

Type	Format	Description
string	`date`	As defined by full-date - RFC3339. for example, `2018-11-13`
	`date-time`	As defined by date-time - RFC3339, for example, `2018-11-13T20:20:39+00:00`
	`time`	RFC3339 "full-time", for example, `20:20:39+00:00`
	`duration`	This lexical representation for *datetime* is PnYnMnDTnHnMnS, where nY represents the number of years, nM the number of months, nD the number of days, 'T' is the date/time separator, nH the number of hours, nM the number of minutes and nS the number of seconds. The number of seconds can include decimal digits to arbitrary precision. EXAMPLE P2Y10M15DT10H30M20S (duration of two years, 10 months, 15 days, 10 hours, 30 minutes and 20 seconds). NOTE See extended format representation of **duration** as defined in ISO 8601.
	`email`	As defined by RFC 5322, section 3.4.1 [RFC5322]
	`uri`	A universal resource identifier (URI), according to RFC3986.
	`ipv4`	IPv4 address, according to dotted-quad ABNF syntax as defined in RFC 2673, section 3.2.
	`ipv6`	IPv6 address, as defined in RFC 2373, section 2.2.
	`user-id`	The TPaaS TID user identifier (uuid format expected). Gives a hint to the UI that the user selector is expected for this property.
	`query`	(preview) A query in following format.

		`<aggregation_method>(<prop_id> [?? <prop_id> [?? ...]])`
		
		`<aggregation_method>` – "sum", "max", "min", "avg", "count".
		`<prop_id>` – is a property identifier in one of the following forms:
		- `pset:[<libId>/]<defId>/urlencode(<propId>)` for properties stored in PSetService. The <propId> must be URL encoded to escape special characters that are allowed in the property identifier. The libId is optional to allow referencing to group of properties of the same (copied) definition. Segments are separated by slash ("/") character.
		- `trb:urlencode(<pset_name>)/urlencode(<prop_name>)` - for native model properties. There is no localization for native psets and character set is not limited (see IFC standard). To escape special characters the url encoding should be applied to names of the pset and prop itself. Segments are separated by slash ("/") character.
		`??` is a null-coalescing operator.
		
		Example:
		`sum(Calculated%20Geometry%20Values/Area ?? Tekla%20Common/Area)`

Keywords

Following attributes allow to further restrict and validate the value assigned to the property.

Data Type	Attribute	Description
string	`maxLength` `minLength`	The length of a string can be constrained using the `minLength` and `maxLength` keywords. For both keywords, the value must be a non-negative number.
	`pattern`	The `pattern` keyword is used to restrict a string to a particular regular expression. The regular expression syntax is the one defined in JavaScript (ECMA 262 specifically). See also JSON Schema spec.

Numeric types	`multipleOf`	Numbers can be restricted to a multiple of a given number, using the `multipleOf` keyword. It may be set to any positive number.
	`maximum` `minimum`	Ranges of numbers are specified using a combination of the `minimum` and `maximum` keywords
	`exclusiveMaximum` `exclusiveMinimum`	Boolean indicating if numbers in `minimum` and `maximum` keywords are exclusive.
array	`maxItems` `minItems`	The length of the array can be specified using the `minItems` and `maxItems` keywords. The value of each keyword must be a non-negative number.
	`uniqueItems`	A schema can ensure that each of the items in an array is unique. Simply set the `uniqueItems` keyword to `true`.
object	`open`	Default is `false`, meaning that the complex property (object) is closed and no properties are allowed other than explicitly mentioned in the schema. If set to `true`, the object is open and can have other not specified (and not validated properties).
Any type	`required`	default is “false”
	`enum`	The value of this keyword MUST be an array. This array SHOULD have at least one element. Elements in the array SHOULD be unique.
	`default`	The default value represents what would be assumed by the consumer of the input as the value of the schema if one is not provided. Unlike JSON Schema, the value MUST conform to the defined type for the Schema Object defined at the same level. For example, if type is string, then default can be "foo" but cannot be 1.
	`description`	Optional. CommonMark syntax MAY be used for rich text representation. But it is recommended to use the i18n translations instead of this property.

Extension Attributes

Extension attributes are ignored by the PSetService and can be used to store additional information analysed by the UI, but not used by the service to validate the data.
Must start from “`x-`” prefix. All unknown properties that don’t start from the “`x-`” prefix will be rejected when a pset definition schema is submitted.

The type of the extension attribute must be a `string`, `number`, `integer` of `boolean`. Max length for the string is 100.

Schema Examples

Simple examples

This section gives an example of the simple schemas. Details of the schema are described and specified in later sections.

Below is a schema for the simple property set with 5 pre-defined properties. The pset is open, which means any additional properties can be added to the pset without validation.

```
 "schema" : {
 "open": true,
 "props": {
        "color": {
               "type": "string",
                "enum": ["red", "blue"],
                "description": "The color."
         },
        "approvedAt" : {
               "type": "string",
                "format":"date-time"
               },
        "quantity" : {
               "type": "integer",
               "minimum": 1,
                "maximum": 20
                },
        "onSchedule" : {
               "type": "boolean",
               "default": true,
                "required" : true
       },
        "ApplicableSizes":{
               "type": "array",
               "items": {"type": "lengthMeasure"}
        }
 }
}
```

The schema below is valid, but it will not allow to store any data as no properties are allowed.

```
 "schema" : {
 "props": { }
}
```

The schema below allows to store any data in the open property set.

```
 "schema" : {
 "open": true,
 "props": { }
}
```

String Properties

In JSON schema we can specify that a document is a string by using the keyword string as the value of the name type.

```
{ "type": "string" }
```

In order to control the length of our string we can use the keywords minLength and maxLength. To restrict the shape of the string even further we can specify that it conforms to a regular expression using the keyword pattern.

Restricting the length of a string

We use the “minLength” and “maxLength” keywords to specify that the length of a string has to fall into a particular interval. For instance, the following schema

```
{
  "type": "string",
  "minLength": 3,
  "maxLength": 7
}
```

To specify that a string conforms to a regular expression we use the keyword “pattern”.

```
{
  "type": "string",
  "pattern": "^[A-Za-z]*@gmail.com$"
}
```

Number Properties

In JSON Schema we can specify that a document must be a number by using the type keyword.

```
{
   "type": "number"
}
```

In order to restrict the number to specific range we can use the keywords minimum and maximum. By default the ranges are inclusive: “minimum”: n imposes the restriction that numbers need to be greater than or equal to n, and “minimum”: m imposes that numbers need to be less than or equal to m. This means that both 0 and 150 satisfy the schema below.

```
{
   "type": "integer",
   "minimum": 0,
   "maximum": 150
}
```

We can switch the ranges to be exclusive by using the “exclusiveMinimum” and “exclusiveMaximum” keywords.For example, the following schema validates against 0, but not against 150. If the “minimum”: n keyword is present together with the keyword “exclusiveMinimum”:true then the restriction imposed that numbers must be greater than n, and if the “maximum”: m keyword is present together with “exclusiveMaximum”:true then the restriction is that the number must be lower than

```
{
   "type":"integer",
   "minimum": 0,
   "maximum": 150,
   "exclusiveMaximum":true
}
```

We can also specify that a number must be a multiple of another number, as in the following schema:

```
{
   "type": "integer",
   "multipleOf": 10
}
```

This schema validates against any number that is a multiple of 10. Note that we can also use real numbers in “multipleOf”, and we can combine it with range restrictions. For example, the schema

```
 {
   "type":"integer",
   "multipleOf": 3.3,
   "maximum": 7
  }
```

Boolean Properties

In order to specify a property that can be either true or false we use the value type boolean. A JSON Schema specifying the type boolean is given below.

```
{ "type": "boolean" }
```

Array Properties

Arrays are used to represent ordered sets of values.

```
{
 "type": "array"
}
```

Restrictions
We can restrict the contents of an array using a JSON Schema. For example, if we want to be sure that the items of the array are strings and that we could have at most 3 items we could use the following schema:

```
{
 "type": "array",
 "maxItems": 3,
 "items": {
            "type": "string"
           }
}
```

We can also specify the amount of items in an array. We use “minItems”: n to specify that the array must have at least n elements, and “maxItems”: n to specify that the array cannot have more than n elements. Here we have an example of this restriction

```
{
 "type": "array",
 "minItems": 2,
 "maxItems": 5
}
```

We use this restriction if we want all elements of an array to be different. The following schema specifies an array in which all documents are different:

```
{
```

```
 "type": "array",
 "uniqueItems": true
}
```

Currency

Currency can be stored as a `currencyMeasure` type or as a `monetaryMeasure`, compatible with the IFC type system.

```
{
  "props":
  {
      "currency":
      {
        "type": "currencyMeasure"
      }
  }
}
```

Time and Intervals

Moments of time are expressed as strings with `date-time`, `date` or `time` format in RFC3339 format.

Intervals can be expressed as strings with "`duration`" format (ISO 8601) or as a `timeMeasure`, compatible with IFC. The `timeMeasure` is better for later processing and queries as values are directly comparable and easy to operate.

Complex Properties

```
{
   "props":
   {
        "link": {
            "type": "object",
            "properties": {
                "name": {
                    "type": "string"
                },
                "url": {
                    "type": "string",
                    "required": true,
```

```
                    "format" : "uri"
                }
            }
        }
    }
}
```

Example above is a “link” property that combines a human readable name/label and url.

```
{
   "props":
   {
        "links": {
            "type": "array",
            "maxItems": 10,
            "Items":{
                "type": "object",
                "open": true,
                "properties": {
                    "name": {
                        "type": "string",
                    },
                    "url": {
                        "type": "string",
                        "required": true,
                        "format" : "uri"
                    }
                }
            }
        }
    }
}
```

Example above demonstrates that complex properties can also be used as array elements. In this example the complex property is defined as “open” and will alloy creating array elements with additional not specified properties.