COLNEO

Software // Services // BIM

Schema für Eigenschaften

Modelle erweitern, Modelle prüfen, Eigenschaften anzeigen und verändern, AIA definieren

2025.07

Für die **Eingabe, Abbildung (Kopie), Darstellung** und **Prüfung** von Eigenschaften
wird ein generisches Schema für Eigenschaften spezifiziert.

Eigenschaften werden durch Namen und Datentyp eindeutig identifiziert.
Metadaten können für die Darstellung (Vordergrund, Hintergrund, Tooltip, Anzeigename etc.) oder für die Festlegung eines Wertebereichs verwendet werden.

Eigenschaften werden gruppiert (**Property Sets**) und es können allgemeine Filterregeln in Bezug auf Objekteigenschaften bzw. Attributgruppen und Regeln festgelegt werden.
Über diese Filter können Eigenschaften bzw. Regeln für bestimmte Anwendungsfälle oder Projektphasen festgelegt werden.

Das Schema wird im JSON-Format definiert. Das Schema ist auf den folgenden Seiten erläutert.

Standardschema für Eigenschaften - JSON-Format

Das Schema für Eigenschaften wird im JSON-Format definiert.

Das Schema enthält auf oberster Ebene die Schlüssel 'metadata' und 'schema', die im rechts stehenden Textrahmen aufgeführt sind und die im Folgenden beschrieben werden.

info

Im Schlüssel 'info' sind Informationen zum Schema wie ID, Name, etc. abgelegt. Das Format und der Inhalt entsprechen dem generischen Objektmodells des COLNEO infohub.

metadata

Im Schlüssel 'metadata' sind allgemeine Informationen zum Schema abgelegt.

Dazu gehören:

- description

  Beschreibung des Schemas

SCHEMA

```
{

   "info": {
      "object_id"     : "...",
      "object_name"   : "..."
   }

   "metadata": {
      "description":  "..."
   },

   "schema": {

      "propertytypes" : {},
      "groups"        : {},
      "dependencies"  : [],
      "rules"         : {},
      "filters"       : {},
      "mappings"      : [],
      "trees"         : []

      "config"        : {
         "float_precision" : 0.0001
      }

   }

}
```

Standardschema für Eigenschaften - JSON-Format

schema

Der Schüssel 'schema' enthält das eigentliche Schema mit folgenden Schlüsseln

- propertytypes

  Attribute bzw. Attributtypen

- groups

  Gruppen (Psets, Property Sets)

- dependencies

  Abhängigkeiten von Eigenschaften von den Werten anderer Eigenschaften

- rules

  Regeldefinitionen für das Prüfen von Eigenschaften

- filters

  Filter für Attribute und Regeln

- mappings

  Regeln für das Abbilden bzw. Umkopieren von Eigenschaften.
  Z.B. für den Export von Objekteigenschaften.

```
{

   "metadata": {
      "id":          "...",
      "name":        "...",
      "description": "..."
   },

   "schema": {

      "propertytypes" : {},
      "groups"        : {},
      "dependencies"  : [],
      "rules"         : {},
      "filters"       : {},
      "mappings"      : [],
      "trees"         : []

      "config":    {
         "float_precision" : 0.0001,
      }

   }

}
```

Standardschema für Eigenschaften - JSON-Format

- `tree`

  Dieser Schlüssel enthält Regeln, um eine Baumstruktur für Objekte in Abhängigkeit ihrer Attribute aufzubauen.

- `config`

  Im Schlüssel '`config`' werden global gültige Optionen abgelegt, die in Regeln überschrieben werden können.

  Eine global gültige Option ist

  `float_precision`

  Genauigkeit, mit der Zahlenwerte verglichen werden. Ein Wert von 0.0001 prüft auf 4 gleiche Nachkommastellen (Standardwert).

```
{

   "metadata": {
      "id": "...",
      "name": "...",
      "description": "..."
   },

   "schema": {

      "propertytypes" : {},
      "groups"        : {},
      "dependencies"  : [],
      "rules"         : {},
      "filters"       : {},
      "mappings"      : [],
      "trees"         : []

      "config"        : {
         "float_precision": 0.0001,
      }

   }

}
```

1 PROPERTY TYPES

JSON-Format – Property Types

propertytypes

Der Schlüssel '**propertytypes**' definiert einen Satz von Attributen.

Jedes Attribut besteht aus einem frei wählbaren, eindeutigen Schlüssel und einem JSON-Objekt mit den Informationen, die das Attribut beschreiben.

Der Schlüssel **'$order**' ist ein festgelegter Schlüssel und darf nicht als Schlüssel für eine Eigenschaft verwendet werden.
'$order' enthält als Wert ein Array, welches Attributschlüssel enthält und die Reihenfolge der Eigenschaften wiedergibt.

```
"propertytypes": {

   "$order": ["P1", "P0"],

   "P0": {
      "name"        : "Level",
      "datatype"    : "xs:string",
      "type"        : "",
      "displayname" : "Geschoss",
      "unit"        : "",
      "domain"      : "geo",
      "comment"     : "Geschoss, Ebene in der sich das Bauteil befindet",
      "values"      : "",
      "default"     : "",
      "readonly"    : true,
      "multiple"    : false,
      "styles"      : {},
      "userdefined" : {}
   },

   "P1": {
      "name": "Anlagenbezeichnung",
      "datatype": "xs:string",
      "displayname": "Anlagenbezeichnung ",
      "unit": "",
      "domain": "geo",
      "comment": "",
      "values": ["ABC", "DEF"],
      "displayvalues": {
         "ABC": {
            "name": "Displayname für Wert ABC",
            "styles": {}
         }
         "DEF": {
            "name": "Displayname für Wert DEF",
            "styles": {}
         }
      }
      "multiple": true,
      "default": "",
      "readonly": false,
      "styles": {},
      "userdefined": {}
   }

}
```

JSON-Format – Property Types

Eine Eigenschaft ist mit folgenden Schlüsseln definiert:

- `name`: Wie im Autorensystem definiert (Pflichtfeld)
- `datatype`: Wie im Autorensystem definiert (Pflichtfeld)

- **`type`**: Semantischer Typ des Attributs, z.B. Benutzer, Bild, Projekt, etc.
- **`displayname`**: Anzeigename für die Darstellung in einer Benutzeroberfläche
- **`id`**: interne ID des Properties im json-Objekt
- **`unit`**: Die verwendete Einheit als Zeichenkette „m“, „m2“, „Stk“, etc.
- **`domain`**: Die Domäne, für die das Attribut definiert ist
  (kann auch eine Aufzählung von Domänen sein, dann durch ';' getrennt)
- **`comment`**: Beschreibung der Eigenschaft, Kommentar
- **`values`**: Wertebereich der Eigenschaft.
  Die Festlegung des Wertebereichs wird nachfolgend beschrieben.
- **`displayvalues`**: Anzeigewerte und -stile, z.B. für Werte einer Auswahlbox
- **`default`**: Der Standardwert, falls kein Wert am Attribut steht.
- **`readonly`**: Definiert, ob das Attribut editierbar ist oder nicht (default: false)
- **`multiple`**: Mehrfache Werte zugelassen, ja/nein (default: false)
- **`styles`**: Enthält Stile für die GUI-Darstellung
- **`userdefined`**: Enthält benutzerdefinierte Eigenschaften (Felder)

```
"propertytypes": {

   "$order": ["P1", "P0"],

   "P0": {
       "name"        : "Level",
       "datatype"    : "xs:string",
       “id”          : “P0”,
       "type"        : "LOCATION",
       "displayname" : "Geschoss",
       "unit": "",
       "domain": "geo",
       "comment": "Geschoss, Ebene in der sich das Bauteil befindet",
       "values": "",
       "default": "",
       "readonly": true,
       "multiple": false,
       "styles" : {},
       "userdefined": {}
   },

   "P1": {
       "name": "Anlagenbezeichnung",
       "datatype": "xs:string",
       “id”          : “P1”,
       "displayname": "Anlagenbezeichnung ",
       "unit": "",
       "domain": "geo",
       "comment": "",
       "values": ["ABC", "DEF"],
       "displayvalues": {
          "ABC": {
              "name": "Displayname für Wert ABC",
              "styles": {}
          },
          "DEF": {
              "name": "Displayname für Wert DEF",
              "styles": {}
          }
       },
       "multiple": true,
       "default": "",
       "readonly": false,
       "styles": {},
       "userdefined": {}
   }

}
```

JSON-Format – Regeln für die Attributwerte

Der Schlüssel '`values`' legt den Wertebereich eines Attributs fest und kann wie folgt verwendet werden:

- Als String / Formelausdruck

  Einfacher logischer Ausdruck (Javascript Syntax):

  `"== 'ABC'" / ">= 5.5" / "== true"`

  Beliebiger logischer Ausdruck (Javascript Syntax):

  `"([[.]] > 1 && [[.]] < 5) || [[.]] == 9"`

  `[[.]]` wird dabei durch den Wert ersetzt, den das Attribut für das jeweilige Objekt hat. Wird auf dem Wert eines anderen Attributs desselben Objekts verwiesen, wird das Attribut in der Form [[typid]] angegeben.

  Beispiel:
  `" '[[.]]' == '[[plannummer##xs:string]]' + '_abc' "`

- Als Liste (Array)

  `[ "ABC" , "CDE" ]` Liste von Zeichenketten
  `[ 1.234, 5, -9.1 ]` Liste von Zahlen

  Ein Array legt eine Liste mit zulässigen Werten fest.

```
"propertytypes": {

   "P0": {
      "name": "Bauteilbezeichnung",
      "displayname": "Beschreibung",
      "datatype": "xs:string",
      "unit": "",
      "domain": "geo",
      "comment": "",
      "values": "...",
      "default": "",
      "readOnly": true,
      "styles": {
         "tooltip": "Bezeichnung des Bauteils",
         "fgcolor": "#f3f3f3",
         "bgcolor": "#333"
      },

      "userdefined": {
         "key": true
      }

   }

}
```

JSON-Format – Regeln für die Attributwerte

Der Schlüssel '**values**' legt den Wertebereich eines Attributs fest und kann wie folgt verwendet werden:

- Als regulärer Ausdruck (/…/)

  "/^[a-zA-Z]{7}$/g"

  Beginnt der String mit ‘/’ und endet auf '/' bzw. '/g' oder '/gi', wird der Ausdruck als regulärer Ausdruck interpretiert.

Ist der Wert leer, "*" oder als JSON-Attribut nicht vorhanden, ist der Wertebereich des Attributs nicht eingeschränkt.

```
"propertytypes": {

   "P0": {
      "name": "Bauteilbezeichnung",
      "displayname": "Beschreibung",
      "datatype": "xs:string",
      "unit": "",
      "domain": "geo",
      "comment": "",
      "values": "...",
      "default": "",
      "readOnly": true,
      "styles": {
         "tooltip": "Bezeichnung des Bauteils",
         "fgcolor": "#f3f3f3",
         "bgcolor": "#333"
      },

      "userdefined": {
         "key": true
      }

   }

}
```

JSON-Format – Regeln für die Attributwerte

Mehrfache Werte für eine Eigenschaft (Wertebereich Liste)

`"multiple": true`

Wenn 'true', dann sind mehrfache Werte aus dem Wertebereich des Arrays möglich.

Die Werte müssen im zu prüfenden Ausdruck durch ein Semikolon (';') getrennt sein, z.B. "fb;rB"

```
"propertytypes": {

   "P3": {
      "name": "cn:Brandschutz",
      "displayname": "Brandschutzanforderungen",
      "datatype": "xs:string",
      "unit": "",
      "domain": "geo",
      "comment": "",
      "values": ["fb", "fh", "hfh", "rB"],
      "multiple": true,
      "default": "fb",
      "readonly": true,
      "styles": {
         "tooltip": "Please enter anything"
      }
   }

}
```

JSON-Format – Regeln für die Attributwerte

type

Der Schlüssel '`type`' enthält den Attributtyp eines Attributes.

Neben den Datentypen, die das Format für das Speichern eines Attributs definieren, kann hiermit die Verwendung eines Attributs in einer Applikation festgelegt werden.

Dies ist hilfreich bei generischen Benutzeroberflächen, die für Attribute typspezifische Eingabeelemente anbieten. Z.B. kann beim Typ 'USER' auf die verfügbaren Benutzer zurückgegriffen werden und diese in einem Auswahlelement angezeigt werden.

- `IMAGE`
  Base64 codiertes Bild
- `USER`
  Benutzer. Benutzerlisten werden über `multiple_values` abgebildet
- `PROJECT`
  Projekt-ID
- `CATALOGUE`
  Katalog-ID
- `LOCATION`
  Ort (Objekt-ID) im Ortekatalog bzw. in der Liegenschaftsstruktur
- `COORDINATES`
  Koordinaten, z.B. Gauss-Krüger-Koordinaten
  In eckigen Klammern wird das Koordinatensystem angegeben.
- `TIME`
- `COLOR`

```
[

   'IMAGE',
   'COLOR',
   'TIME',

   'USER',
   'PROJECT',
   'CATALOGUE',

   'LOCATION',
   'COORDINATES[coordinatesystem],

   'QUANTITY'[pcs,weight,length,area,volume,time]


]
```

JSON-Format – Stile für Attribute

styles

Der Abschnitt '**styles**' definiert Stile für die Darstellung von Attributen in der Benutzeroberfläche.

- tooltip (string)
  Kurzbeschreibung
- fgcolor (#RGB)
  Stiftfarbe (Schrift), RGB im Hexadezimalformat (3 oder 6 Zeichen)
  Weiss: #fff, Schwarz : #000
- bgcolor (#RGB)
  Hintergrundfarbe, RGB im Hexadezimalformat (3 oder 6 Zeichen)
- multiline (true/false)
  Wenn der Wert der Eigenschaft eine Zeichenkette ist und Zeilenumbrüche enthalten kann, ist der Wert für 'multiline' wahr.
- align (left/center/right)
  Ausrichtung der Darstellung: link / mitte / rechts
- visible (true/false)
  Attribut wird angezeigt oder nicht (wenn nicht vorhanden wird im UI true angenommen)

```
"propertytypes": {

   "P3": {
      "name": "Brandschutz",
      …
      "styles": {
         "tooltip"    : "Brandschutzklasse",
         "fgcolor"    : "#f00",
         "bgcolor"    : "#cdcdcd",
         "multiline"  : true,
         "align"      : "center",
         "visible"    : true
      },
      ...
   }

}
```

JSON-Format – Attribute mit benutzerdefinierten Eigenschaften

userdefined

Der Schlüssel '**userdefined**' definiert einen zusätzlichen Satz von benutzerdefinierten Schlüssel-Werte Paaren.

Die Werte können vom Datentyp string, number, boolean, array oder auch selbst wieder JSON-Objekte sein.

Bsp:
In einer Liste von Eigenschaften sollen diejenigen markiert werden, die Schlüsselattribute für einen bestimmten Anwendungsfall darstellen. Dazu wird ein benutzerdefiniertes Attribut für die entsprechende Eigenschaft verwendet.

```
"propertytypes": {
   "P3": {
      "name": "cpName",
      "datatype": "xs:string",
      "userdefined": {
         "iskey": true
      },
   },
   "P4": {
      "name": "ifcType",
      "datatype": "xs:string",
      "userdefined": {
         "iskey": false
      },
   }
}
```

```
"propertytypes": {

   "P3": {

      "name": "Brandschutz",
      "datatype": "xs:string",

      ...

      "userdefined": {
         "Text": "Ein Text",
         "Zahl": 1.234,
         "Wahrheitswert": true,
         "Textliste": ["A", "B", "C"]
      },

      ...

   }

}
```

Unterstützte Datentypen und Domänen

Unterstützte Datentypen

```
[
    'xs:int',
    'xs:long',
    'xs:float',
    'xs:double',
    'xs:string',
    'xs:ID',
    'xs:IDREF',
    'xs:anyURI',
    'xs:boolean',
    'xs:object',
    'xs:date',
    'xs:dateTime'
]
```

Unterstützte Attributtypen:

```
[

    'IMAGE',
    'USER',
    'PROJECT',
    'CATALOGUE',
    'LOCATION',
    'COORDINATES[coordinatesystem]
    'TIME',
    'COLOR'

]
```

Unterstütze Domänen:

```
[
    'all', // all domains (e.g. for type project)
    'geo', // design (=geometry)
    'boq', // workscope (boq, costs)
    'act', // activities
    'doc', // documents
    'bst', // building structure
    'res', // resource
]
```

Der vollqualifizierte Name eines Attributs
wird durch Kettung des Namens, des Datentyps und des Attributtyps gebildet.
Der Attributtyp ist optional.

Z.B.

```
Project:picture##xs:string##IMAGE
ifcTyp##xs:string
```

2 GROUPS / PROPERTY SETS

JSON-Format - Gruppen

groups

Der Schlüssel '**groups**' definiert Mengen/Gruppen, die zur Einteilung der Attribute genutzt werden können.

Jede Gruppe besteht aus einem Schlüssel und einem Objekt mit zusätzlichen Informationen:

- `name` (Pflichtfeld)
  Der Name der Gruppe
- `Id`
  Id der Gruppe im json-Objekt
- `comment`
  Ein beliebiger Kommentar

- `styles`
  Der Abschnitt "`styles`" definiert Stile für die Darstellung in der Benutzeroberfläche (siehe Abschnitt "`propertytypes`")

  - `tooltip`
    Kann in der Anzeige in Listenform verwendet werden.
  - `fgcolor`
    Stiftfarbe (Schrift), RGB im Hexadezimalformat (3 oder 6 Zeichen nach #)
  - `bgcolor`
    Hintergrundfarbe, RGB im Hexadezimalformat (3 oder 6 Zeichen nach #)
  - `visible`
    Gruppe ist im Nutzerformular sichtbar oder nicht (wenn nicht vorhanden, wird im UI true angenommen)

- `properties`
  Weist Attribute per Schlüssel der jeweiligen Gruppe zu (Liste).

```
"groups": {

   "G0": {
      "name": "TGA",
      “id": “G0",
      "comment": "Technische Gebäudeausrüstung",
      "styles": {
         "tooltip": "Technische Gebäudeausrüstung",
         "fgcolor": "#f00",
         "bgcolor": "#0f0“,
         “visible”: true
      },
      "groups": {
         "G0.1": {
            ...
         },
         "G0.2": {
            ...
         }
      }
   },
   "G1": {
      "name": "ARC",
      “id": “G1",
      "comment": "Architektur",
      "properties": [
         "p0",
         "p1"
      ]
   },
   "G2": {
      "name": "Leere Gruppe"
   }
}
```

JSON-Format - Gruppen

groups

Die Gruppen können wiederum Gruppen enthalten, sodass eine Hierarchie entsteht.

- groups

  Definiert einen Satz von Untergruppen.

Anwendung der styles

Die styles fgcolor, bgcolor und visible werden in der Anzeige im Formular auch auf die Properties der Liste properties angewendet, sofern diese keine eigenen Definitionen dafür haben.
Beispiele:

- Alle Attribute der Gruppe ‚Architektur' werden beispielsweise in der gleichen Farbe angezeigt.
- Ein Attribut ‚Länge' ist z.B. 2 gruppen zugeordnet. Gruppe ‚Architektur' und Gruppe ‚AVA'. In der Gruppe ‚Architektur' wird es dann mit der Gruppenfarbe angezeigt. In der Gruppe ‚AVA' mit den Standardfarben.

```
"groups": {

   "G0": {
      "name": "TGA",
      "comment": "Technische Gebäudeausrüstung",
      "styles" : {
         "tooltip": "Technische Gebäudeausrüstung",
         "fgcolor": "#f00",
         "bgcolor": "#0f0“,
         “visible”: false
      },
      "groups": {
         "G0.1": {
            …
         },
         "G0.2": {
            …
         }
      }
   },
   "G1": {
      "name": "ARC",
      "comment": "Architektur",
      "properties": [
         "p0",
         "p1"
      ]
   },
   "G2": {
      "name": "Leere Gruppe"
   }
}
```

3 ABHÄNGIGKEITEN / DEPENDENCIES

Abhängigkeiten

Dependencies für Attribute

Der Schlüssel '**dependencies** ' definiert Abhängigkeiten zwischen Attributen. D.h. in Abhängigkeit von einem bestimmten Wert für ein Attribut können die Parameter für andere Attribute überschrieben werden.

Beispiele:

1. Wenn der Wert für das Attribut **P0** den Wert 'Erdbau' besitzt, dann werden die Schlüssel für die Attribute **P1** und **P3** mit den angegebenen Werten überschrieben.

2. Wenn der Wert für das Attribut 'P0' den Wert 'HydraulischGebundeneTragschicht' besitzt, dann werden die Schlüssel für die Attribut P1 mit den angegebenen Werten ('OHNE Zulage, 'MIT Zulage') überschrieben.
   Für diese beiden Werte wird wiederum eine Abhängigkeit des Attributs **P2** definiert. In beiden Fällen soll der einzige Wert 'Vlies' sein.

Bis auf den Namen und Datentyp eines Attributs können alle Schlüssel überschrieben werden, nicht nur wie im Beispiel `values`.

Schlüssel, die nicht in der Angabe enthalten sind, behalten den im Abschnitt '**propertytypes**' festgelegten Wert.

Die Bedingungen können geschachtelt werden.

```
"dependencies": [
    {
        "$ref"     : "P0", // MaterialDB
        "overrides": {} // on first level should be always empty
        "deplist"  : [
            {
                "value": "Erdbau",
                "dependencies": [
                    {
                        "$ref"     : "P1", // Art
                        "comment" : "propertytype",
                        "overrides": {
                            "values": ["Abtrag", "Auftrag"],
                            "default": "Auftrag"
                        },
                        "deplist"  : []
                    }, {
                        "$ref"     : "P3",
                        "comment" : "propertytype",
                        "overrides": {"values": 0.04},
                    }
                ]
            }, {
                "value": [ "HydrGebTragschicht" , "xyz" ]
                "dependencies": [
                    {
                        "$ref"     : "P1", // Art
                        "comment" : "propertytype",
                        "overrides": {"values": ["OHNE Zulage", "MIT Zulage"]},
                        "deplist"  : [
                            {
                                "value": "OHNE Zulage",
                                "dependencies": [
                                    {
                                        "$ref"     : "P2", // Zusatzinfo 1
                                        "comment" : "propertytypes",
                                        "overrides": {"values": ["Vlies"]},
                                    }
                                ]
                            },
                            {
                                "value": "MIT Zulage",
                                "dependencies": [
                                    {
                                        "$ref"     : "P2", // Zusatzinfo 1
                                        "comment" : "propertytypes",
                                        "overrides": {"values": ["Vlies"]},
                                    }
                ]}]}]}]}]
```

Wenn das Attribut '**P0**' den Wert '*Erdbau*' besitzt, dann werden die Attribute **P1** und **P3** überschrieben bzw. ergänzt

Wenn das Attribut '**P0**' den Wert '*HydrGebTragschicht*' oder den Wert '*xyz*' besitzt, dann wird das Attribut **P1** überschrieben bzw. ergänzt

Dependencies für Gruppen

Der Schlüssel `dependencies`, kann auch Abhängigkeiten für ganze Gruppen von Attributen definieren.

Beispiele:

1. Wie zuvor: Wenn der Wert für das Attribut **P0** den Wert 'Erdbau' besitzt, dann werden die Schlüssel für die Attribute **P1** und **P3** mit den angegebenen Werten überschrieben.

2. Gleichzeitig wird die Gruppe mit der ID „G1“ im Formular ausgeblendet.

Bis auf den Namen und Datentyp eines Attributs können alle Schlüssel überschrieben werden, nicht nur wie im Beispiel values oder visible.

Schlüssel, die nicht in der Angabe enthalten sind, behalten den im Abschnitt `propertytypes` bzw. `groups` festgelegten Wert.

Anwendung der styles

Die styles `fgcolor`, `bgcolor` und `visible` werden in der Anzeige im Formular für die Properties der Liste `properties` in folgender Reihenfolge ausgewertet. Der Letzte Wert (von Stil 1 bis 4) wird zur Anzeige gebracht:
1. Gruppenstil
2. Propertystil
3. Aus Abhängigkeit resultierender Gruppenstil
4. Aus Abhängigkeit resultierender Propertystil

```
"dependencies": [
    {
        "$ref"     : "P0", // MaterialDB
        "overrides": {} // on first level should be always empty
        "deplist"  : [
            {
                "value": "Erdbau",
                "dependencies": [
                    {
                        "$ref"       : "P1", // Art
                        "comment"    : "propertytype",
                        "overrides“  : {
                            "values“ : ["Abtrag", "Auftrag"],
                            "default": "Auftrag"
                        },
                        "deplist"  : []
                    },{
                        "$ref"     : "G1",
                        “comment"  : "group",
                        "overrides": {
                            "styles": {
                                "visible“ : false
                            }
                        }
                    }
                ]
            }
        ]
    }
]
```

Wenn das Attribut '**P0**' den Wert '*Erdbau*' besitzt, dann werden die Attribute **P1** und **P3** überschrieben bzw. ergänzt

Wenn das Attribut '**P0**' den Wert '*Erdbau*' besitzt,

4 PRÜFREGELN / RULES

JSON-Format – Regeln

rules

Der Schüssel '**rules**' definiert einen Satz von **Regeln**, die wiederum Regeln enthalten können. Somit ist es möglich, eine Hierarchie von Regeln zu definieren.

In einer Regel wird eine Liste von **Prüfregeln** ("checkrules") definiert, die von den zu prüfenden Objekten einzuhalten sind.

- **name** (Pflichtfeld)
  legt einen Namen für die Regel fest
- comment
  enthält eine Beschreibung der Regel
- config
  enthält Optionen für die Prüfung. Hier können die global für das Schema gültigen Einstellungen überschrieben werden (siehe Abschnitt 1).
- rules
  definiert einen Satz von Unterregeln
- checkrules
  definiert eine Liste mit Prüfregeln.
  Die Überprüfung erfolgt in der Reihenfolge der Definition!

```
"rules": {
    "R1": {
        "name": "Regel 1",
        "comment": "Kommentar zur 1. Regel",
        "checkrules": [],
        "rules": {
            "R1.1": {
                "name": "Regel 1.1",
                "comment": " ... ",
                "config": {
                    "float_precision": 0.01
                    "break_on_first_failure": true
                },
                "checkrules" : [
                    {
                        "id": "cr1",
                        "name": "Rule 2",
                        "comment": "...",
                        "IF": "[[ifcType]] in ['IfcWindow', 'IfcDoor']",
                        "THEN": [
                            "[[Width##xs:double]] > 0.2“,
                            "[[Type##xs:string]] match /^a*/g”
                        ]
                    },
                    {
                        "id": "cr2",
                        "name": "Rule unique ID",
                        "comment": "Check if Ids are unique in a given set of objects",
                        "IF": “*",
                        “CARDINALITY": {
                            "[[cpID##xs:ID]]": "== 1"
                        }
                    }
                ]
            }
        }
    }
    "R2": {
        "name": "Regel 2",
        "comment": "Kommentar zur 2. Regel",
        "checkrules": []
    }
}
```

Regeln - Prüfregeln

checkrules

Der Schlüssel `'checkrules'` enthält eine Liste mit **Prüfregeln**.

Eine einzelne Prüfregel besitzt eine eindeutige ID, einen Namen und einen Kommentar, einen **Objektfilter** und **Bedingungen**, die erfüllt werden müssen.

- `id` (Pflichtfeld)
  enthält einen eindeutigen Identifikator für die Prüfregel
- `name` (Pflichtfeld)
  legt einen Namen für die Prüfregel fest
- `comment` (optional)
  enthält eine Beschreibung der Prüfregel
- `warninglevel` (optional)
  definiert, ob das Ergebnis beim Scheitern der Prüfbedingungen als Fehler gewertet wird oder nur als Warnung
- `IF`
  enthält den Filterausdruck zur Bestimmung der Menge der zu prüfenden Objekte
- `THEN / PROPERTIES / CARDINALITY`
  enthalten die entsprechenden Bedingungen, die erfüllt sein müssen.
  Es ist jeweils nur einer der 3 Schlüssel in einer Prüfregel gleichzeitig zulässig.

```
"checkrules": [
    {
        "id": "cr1",
        "name": "Rule 1",
        "comment": "Check Rule #1",
        "warninglevel": “warning”,
        "IF": "{{P0}} in ['IfcWindow', 'IfcDoor']",
        "THEN": [
            "[[Width##xs:double]] >= 0.6",
            "[[Type##xs:string]] match /^TYP_[0-9]*/g"
        ]
    },
    {
        "id": "cr2",
        "name": "Rule unique ID",
        "comment": "Check if Ids are unique in a given set of objects",
        "IF": "*",
        "CARDINALITY": {
            "COUNT": {
                "[[cpID##xs:ID]]" : " == 1"
            }
        }
    },
    {
        "id":  "cr3",
        "name": "Decken",
        "comment": "Check Rule #3",
        "IF": [
            "[[ifcType##xs:string]] == 'IfcSlab'",
            "[[Typename##xs:string]] in ['Typ A', 'Typ C']"
        ],
        "PROPERTIES": {
            "Thickness##xs:double" : {
                "values" : "[[.]] > 0.25"
            }
        ]
    }
]
```

IF – Menge der zu prüfenden Objekte

checkrules

- IF

  definiert eine Bedingung bzw. eine Liste von Bedingungen, ob ein Objekt zu prüfen ist.
  Ist der Wert `"*"`, werden alle Objekte geprüft.
  Ist der Typ hinter IF ein **Array**, müssen alle einzelnen Bedingungen erfüllt sein.

- Neben den in Javascript verfügbaren Ausdrücken stehen die Operatoren '`match`' und '`in`' bzw. deren Negation '`not_match`' und '`not_in`' zur Verfügung. Diese Operationen bestehen aus einem linken Operanden, dem Vergleichsoperator und einem rechten Operanden.

  Beispiel:

  ```
  [[IfcType##xs:string]] in [ 'IfcWall', 'IfcColumn' ]
  [[IfcType##xs:string]] match /^IfcWall/g
  ```

- Die Operatoren '**`match`**' und '**not_match**' prüfen einen regulären Ausdruck.

- Die Operatoren '**`in`**' und '**`not_in`**' prüfen, ob ein Wert in einer Liste enthalten bzw. nicht enthalten ist.

- Im Ausdruck können Variablen verwendet werden bzw. Verweise auf Attribute des Objekts enthalten sein.

  In eckigen Klammern `[[ifcType##xs:string]]` wird ein beliebiges Attribut festlegt.

  Mit zwei geschweiften Klammern `{{P0}}` wird auf ein Attribut im Abschnitt `"propertytypes"` verwiesen.

```
"IF": "*"

"IF": "{{P0}} in ['IfcWall', 'IfcWallStandardCase']"

"IF": [
     "[[ifcType##xs:string]] match /^IfcWall/g",
     "[[Typename##xs:string]] in ['Typ A', 'Typ C']"
]
```

Wenn 'IF' wahr ist, dann müssen alle Bedingungen, die in THEN, PROPERTIES und CARDINALITY definiert sind, ebenfalls wahr sein, damit die Prüfung bestanden ist.

Regeln – PROPERTIES (Bedingung für Eigenschaften)

`checkrules`

- `PROPERTIES`

  legt per Verweis (ohne geschweifte Klammern) die Attribute fest, die geprüft werden sollen.
  Diese sind im Abschnitt `"propertytypes"` definiert.

  Über den Schlüssel `"values"` kann hier der im Abschnitt `"propertytypes"` ggf. definierte Wertebereich des Attributes überschrieben werden.

  Soll nur auf Existenz geprüft werden, reicht eine Referenz auf das Attribut.

  Sonderfall:
  Wird eine Referenz auf ein Attribut nicht gefunden (im Beispiel rechts `"Label##xs:string"`), so wird diese Referenz als Attribut interpretiert, sofern der Ausdruck korrekt ausgewertet werden kann (Typ-ID der Eigenschaft: name##datentyp).

```
"checkrules": [
   {
      "id": "cr2",
      "name": "Rule 2",
      "comment": "Check Rule #2",
      "IF": "[[ifcType##xs:string]] in ['IfcColumn', 'IfcDoor']",
      "PROPERTIES": {
         "P0": {},
         "P1": {
            "values": [ "01-a", "02-b" ]
         },
         "Label##xs:string": {
            "values": "*"
         }
      }
   }
}
```

Regeln – THEN (Wenn-Dann Bedingung)

`checkrules`

- `THEN`

  definiert eine Liste von Bedingungen, die für ein zu prüfendes Objekt wahr sein müssen.

  Wahlweise kann auf Attribute direkt mit einem Schlüssel aus Namen und Datentyp [[.##.]] oder per Referenz {{.}} zugegriffen werden (siehe IF).

```
"checkrules": [
   {
      "id": "cr1",
      "name": "Rule 1",
      "comment": "Check Rule #1",
      "warninglevel": "warning",
      "IF": "[[ifcType##xs:string]] == 'IfcWall'",
      "THEN": [
         "[[cpVolume##xs:double]] > 2.0",
         "{{P2}} match /^Wand_/g"
      ]
   }
}
```

Regeln – CARDINALITY (Anzahl von Objekten)

checkrules

- CARDINALITY

  gibt die Anzahl der verschiedenen Werte für ein gegebenes Attribut in der zu prüfenden Objektmenge an, gruppiert nach den verfügbaren Werten eines zu spezifizierenden Attributes.

- GROUPBY (optional)

  Attribut, nach welchem die Kardinalität gruppiert werden kann.

- COUNT

  == 0 : Der Wert darf nicht vorkommen

  == 1 : Der Wert muss eindeutig sein

  > 2 : Die Anzahl der verschiedenen Werte muss größer als 2 (mindestens 3) sein

  <= 4 : Die Anzahl der verschiedenen Werte darf höchstens 4 sein

Der Verweis auf das Attribut in GROUPBY oder COUNT muss in geschweiften Klammern (Referenz) oder eckigen Klammern (klassischer Schlüssel) stehen.

```
"checkrules": [
    {
      id: "cr3",
      "name": "Rule unique ID",
      "comment": "Check if IDs are unique in a given set of objects",
      "IF": "*",
      "CARDINALITY": {
         "COUNT": {
             "[[cpID##xs:ID]]": "== 1"
          }
      }
   },
   {
      "id": "cr4",
      "name": "Regel Anz F-Löscher",
      "comment": "Prüfen Anzahl Feuerlöscher von Hersteller je Stockwerk",
      "IF": [
         "[[IstFeuerlöscher##xs:boolean]] == true",
         "{{Hersteller}} in ['HST1', 'HST2', 'HST3']"
      ],
      "CARDINALITY": {
         "GROUPBY": "[[bs:BuildingStorey##xs:string]]",
         "COUNT": {
            "{{Hersteller}}": ">= 2"
         }
      }
   }
]
```

Regeln – Parameter

`parameters`

Am obersten Regel-Knoten können im Schlüssel '`parameters`' Parameterlisten definiert werden, die wiederum in einzelnen Prüfläufen ggf. überschrieben und ergänzt werden können.

Es besteht die Möglichkeit, Listen manuell zu definieren oder auf den Wertebereich von Attributen zu verweisen. Dazu muss allerdings der Wertebereich eines Attributes als Array definiert sein ("values" : […])

Der Zugriff erfolgt über einen eindeutigen Schlüssel, der von zwei Dollarzeichen ($$) eingerahmt ist.

```
"rules": {
   "parameters":
      {
         "liste_1": ["a", "b", "c"],
         "liste_p": "{{P0}}",
         "liste_g": "bsGeschoss##xs:string",
         "p_1": ["ABC"]
      }
   },
   "checkrules": [
      {
         "id": "cr1"
         "name": "Rule 1",
         "comment": "Check Rule #1",
         "warninglevel": “warning”,
         "IF": "[[ifcType##xs:string]] == 'IfcWall'",
         "THEN": [
             "[[Hersteller##xs:string]] in $$liste_1$$"
         ]
      }
   ]
}
```

5 FILTER

JSON-Format – Filter

`filters`

Der Schlüssel 'filters' definiert Filter, die über Filterregeln (Schlüssel 'values') eine Teilmenge von Attributen bzw. eine Teilmenge von Regeln beschreiben.

Damit können Regeln definiert werden, die z.B. von einem bestimmten Anwendungsfall oder einem LOI (**L**evel **O**f **I**nformation), etc. abhängen.

Jede **Filterregel** definiert für seine möglichen Werte

- eine Liste mit Attributen (`propertytypes`),
- eine Liste mit Attributgruppen (`groups`) und
- eine Liste mit Regeln (`rules`),

die geprüft oder ausgewertet werden sollen.

Hinweis:

Bei Prüfungen werden nicht die Attribute durch den Filter eingeschränkt, sondern nur die Regeln.

Ist eine Liste nicht vorhanden, werden alle Regeln bzw. alle Gruppen oder Attribute berücksichtigt, andernfalls werden nur die enthaltenen Elemente berücksichtigt.

Das bedeutet: Wenn die Liste vorhanden aber leer ist, werden keine Elemente berücksichtigt!

```
"filters": {
  "LOI": {
     "name": "Level of Information",
     "comment": "..."
     "values": {
        "1": {
           "name": "LOI1",
           "comment": "...",
           "propertytypes": ["p0"],
           "groups": ["g1","g2"],
           "rules": []
        },
        "2": {
           "name": "LOI2",
           "comment": "...",
           "propertytypes": ["p0"],
           "groups": ["g3"],
           "rules": []
        }
     }
  },
  "UseCases": {
     "name": "Use Cases",
     "comment": "...",
     "values": {
        "UC-1A": {
           "rules": ["R1"]
        },
        "UC-1B": {
        }
     }
  }
}
```

6 MAPPING (Abbildung, Kopie)

JSON-Format – Mapping

mappings

Im Abschnitt '`mappings`' wird eine Liste mit Regeln beschrieben, wie Attribute abgebildet, d.h. kopiert werden sollen. Zulässige Schlüssel sind nachfolgend aufgeführt.

`id`	Eindeutiger Identifikator, nicht änderbar
`pset_name`	Der Name des Propertysets, welchem das kopierte Attribut hinzugefügt wird (Attributpräfix)
`comment`	Optionaler Kommentar

config

Abschnitt für spezielle Einstellungen beim Abbilden bzw. Kopieren der Attribute (siehe nachfolgende Folien)

IF

Filterregel für Elemente. Nur diejenigen Objekte werden betrachtet, für die diese Bedingung zutrifft. Wenn der Schlüssel 'IF' fehlt, werden die Abbildungen für alle Objekte durchgeführt. Siehe dazu auch das Kapitel **PRÜFREGELN**.

```
"mappings": [
    {
       "id"          : "id12345",
       "pset_name"   : "Pset_COLNEO",
       "comment"     : "…",
       "config"      : {
          "replace_pset_name": true,
          "existing_values": "ignore"
       },

       "IF": "[[ifcType##xs:string]] in ['IfcColumn', 'IfcDoor']",

       “preprocess“ : [
          {
             “type” : “check”,
             “ref”  : “abgh/6jhGfr”,
             “options” : {
                “failed”  : [ ‘delete’ , ‘ignore’ , ‘execute’ , ‘comment’ ],
                “warning” : [ ‘delete’ , ‘ignore’ , ‘execute’ , ‘comment’],
                “ignored” : [ ‘delete’ , ‘ignore’ , ‘execute’ , ‘comment’],
                “passed”  : [ ‘execute’ , ‘comment’ ]
             }
          }
       ]

       "mapitems": [
          {
             …
          },
          {
             "id"   : "2jHgtzu%4g";
             "name" : "InnenAussen",
             "valueitems" : [
                "{{P1}}",
                …
             ],
             "valuemap" : [
                 {"I" : "innen"},
                 {"A" : "außen"},
                 {"*" : "k.A."}
             ]
          },
          …
```

JSON-Format – Mapping

`mapitems`

Der Schlüssel '`mapitems`' enthält eine Liste mit Abbildungsvorschriften. Jede Abbildungsvorschrift besteht aus den Schlüsseln

`id`	Eindeutiger Identifikator, nicht änderbar
`name`	Name des neuen Attributs
`datatype`	Datentyp des neuen Attributs (optional)
`valueitems`	"Lookup-Liste" für Werte
valuemap	Wertmodifikationen

Der Schlüssel '**`valueitems`**' legt fest, wie der Wert des neuen Attributs bestimmt wird. Es handelt sich dabei um eine Liste von Ausdrücken (Attributreferenzen, Formeln oder feste Werte), die solange nacheinander abgearbeitet werden, bis ein Ausdruck ausgewertet werden kann.

Mit dem optionalen Schlüssel '**`valuemap`**' können die über den Schlüssel '`valuemap`' ermittelte Werte modifiziert werden.
In einer Liste werden Paare `{"from_value" : "to_value"}` festgelegt, die nacheinander abgearbeitet werden.
Ein Paar mit `from_value = "*"` bildet alle Werte auf den in "to_value" angegeben Wert ab und ist im Normalfall der letzte Eintrag in der Liste.

Ist der Schlüssel '**`datatype`**' nicht definiert, bestimmt sich der Datentyp aus dem referenzierten Attribut. Für feste Werte und Formeln wird '`xs:string`' gesetzt.

```
"mappings": [
    {
       "id"          : "id12345",
       "pset_name"   : "Pset_COLNEO",
       "comment"     : "…",
       "config"      : {
          "replace_pset_name": true,
          "existing_values": "ignore"
       },

       "IF": "[[ifcType##xs:string]] in ['IfcColumn', 'IfcDoor']",

       "mapitems": [
          {
             …
          },
          {
              "id"   : "2jHgtzu%4g";
              "name" : "InnenAussen",
              "datatype" : "xs:
              "valueitems" : [
                 "{{P1}}",
                 …
              ],
              "valuemap" : [
                  {"I" : "innen"},
                  {"A" : "außen"},
                  {"*" : "k.A."}
              ]
          },
          …
       ]
    },
    …
]
```

JSON-Format – Mapping

Beispiele

Bsp. 1:

```
{
    "name"       : "Insulation",
    "valueitems": ["{{P0}}"]
}
```

Das rechts gezeigte Beispiel definiert zunächst das Attribut `Insulation` in der Attributmenge (Property Set) '`Pset_COLNEO`'. Der vollständige Name des neuen Attributs ergibt sich aus der Kettung, also `Pset_COLNEO:Insulation`.

Der Wert des neuen Attributs wird durch den Ausdruck bestimmt, der im Schlüssel 'valueitems' als erstes erfolgreich ausgewertet werden kann.

In dem gezeigten Beispiel heißt das, dass der Wert des im Schema definierten, referenzierten Attributs mit der ID '`P0`' in das neue Attribut `Pset_COLENO:Insulation`' geschrieben wird.

Bsp. 2:

```
{
    "name": "Zone",
    "valueitems": ["[[Zone##xs:string]]"]
}
```

Das Attribut `Pset_COLNEO:Zone##xs:string` wird aus dem Wert der Eigenschaft `Zone##xs:string` des Objekts bestimmt.

```
"mappings": [
    {
       "id"          : "id12345",
       "pset_name"   : "Pset_COLNEO",
       "config"      : {
          "replace_pset_name": true,
       },

       "IF": "[[ifcType##xs:string]] in ['IfcColumn', 'IfcDoor']",

       "mapitems": [
          {
             "id"          : "$abGhtui76",
             "name"        : "Insulation",
             "valueitems"  : ["{{P0}}"]
          },
          {
             "id"          : "lKj&$3hj",
             "name"        : "Zone",
             "valueitems"  : ["[[Zone##xs:string]]"]
          },
          {
             "id"          : "7uZgt%rF",
             "name"        : "Lage",
             "valueitems"  : ["Achse A-H"]
          },
          {
             "id"          : "Gu_Z1tt74",
             "name"        : "Länge_in_Metern",
             "datatype"    : "xs:double",
             "valueitems"  : ["= [[Length##xs:double]] / 1000"]
          }
       ]
    }
]
```

Präfix für die neuen Attribute, entspricht dem Namen eines Property-Sets (Pset)

Referenz auf das Attribut mit dem ID 'P0'

Wert des Attributs 'Zone##xs:string'

Wert des neuen Attributs gleich 'Achse A-H'

Formel

Pset_COLNEO:Insulation = {{P0}}

Pset_COLNEO:Zone = [[Zone##xs:string]]

Pset_COLNEO:Lage == "Achse A-H"

Pset_COLNEO:Lage == "= [[...]] ..."

JSON-Format – Mapping

Bsp. 3:

```
{
   "name": "Lage",
   "datatype": "xs:string",
   "valueitems": ["Achse A-H"]
}
```

Dem Attribut `Pset_COLNEO:Lage##xs:string` wird der **feste Wert** "Achse A-H" zugewiesen. Die Angabe des Datentyps ist erforderlich falls nicht vom Typ `xs:string`.

Bsp. 4:

```
{
   "name" : "InnenAussen",
   "value": "{{P1}}"
   "valuemap": [
      {"I" : "innen"},
      {"A" : "außen},
      {"*" : "k.A."}
   ]
}
```

`"valuemap"` definiert eine Liste von Wertezuordnungen, die in der **angegebenen Reihenfolge** abgearbeitet werden.

Das Attribut 'InnenAussen' wird auf den Wert 'innen' gesetzt, wenn das referenzierte Attribut 'P1' den Wert 'I' hat, auf den Wert 'außen' wenn das Attribut 'P1' den Wert A hat und auf den Wert 'k.A.' für alle weiteren Werte. Ist die `"*"` Bedingung nicht angegeben, entpricht dieses `"*": "{{P1}}"`

```
"mappings": [
    {
       "id"          : "id12345",
       "pset_name"   : "Pset_COLNEO",
       "comment"     : "…",
       "config"      : { … },

       "IF": "[[ifcType##xs:string]] in ['IfcColumn', 'IfcDoor']",

       "mapitems": [
          {
             "id"    : "$abGhtui76",
             "name"  : "Insulation",
             "valueitems" : ["{{P0}}"}]
          },
          {
             "id"    : "lKj&$3hj",
             "name"  : "Zone",
             "valueitems" : ["[[Zone##xs:string]]]
          },
          {
             "id"    : "7uZgt%rF",
             "name"  : "Lage",
             "datatype": "xs:string",
             "valueitems" : ["Achse A-H"]
          },
          {
             "id"    : "2jHgtzu%4g";
             "name"   : "InnenAussen",
             "valueitems" : ["{{P1}}"],
             "valuemap" : [
                 {"I" : "innen"},
                 {"A" : "außen"},
                 {"*" : "k.A."}
             ]
          }
       ]
    }
]
```

Referenz auf das Attribut mit ID 'P0'

Allgemeine Attributreferenz

Fester Wert

Attributreferenz und Abbildungs-vorschrift für Werte des Attributs

JSON-Format – Mapping

Bsp. 5:

```
{
   "name"   : "Material",
   "valueitems" : ["[ifcMaterial##xs:string"],
   "valuemap" : [
       {"StB*" : "Stahlbeton"},
       {"/^MW/g" : "Mauerwerk"}
   ]
}
```

Im Abschnitt "valuemap" sind auch **Wildcards** und **reguläre Ausdrücke** möglich. Dem Attribut Pset_COLNEO:Material wird der Wert der Eigenschaft 'ifcMaterial' zugewiesen. Falls der Wert mit 'StB' beginnt, wird der Wert in Stahlbeton umgewandelt, falls er mit 'MW' beginnt, in Mauerwerk.

Bsp. 6:

```
{
   "name": "Länge in Metern",
   "datatype": "xs:double",
   "valueitems": [
      "= [[LengthInMillimetres##xs:double]] / 1000",
      "[[BaseQuantities:Length##xs:double]]"
   ]
}
```

Beliebiger Formelausdruck (siehe THEN-Bedingung im Abschnitt Prüfregeln). Die Formel beginnt mit '=', Attributreferenzen werden durch den jeweiligen Wert für ein Objekt ersetzt.

Kann die Formel hier im Beispiel nicht ausgewertet werden, wird mit dem nächsten Ausdruck in der Liste "valueitems" fortgefahren.

```
"mappings": [
    {
       "id"        : "12345",
       "pset_name" : "Pset_COLNEO",
       "comment"   : "…",
       "config"    : { … },

       "IF": "[[ifcType##xs:string]] in ['IfcColumn', 'IfcDoor']",

       "mapitems": [
          {
              "id"    : "…";
              "name"   : "Material",
              "valueitems" : ["[ifcMaterial##xs:string"],
              "valuemap" : [
                  {"StB*" : "Stahlbeton"},
                  {"/^MW/g" : "Mauerwerk"},
              ]
          },
          {
              "id"    : "Gu_Z1tt74",
              "name"   : "Länge in Metern",
              "datatype": "xs:double",
              "valueitems" : [
                 "= [[LengthInMillimetres##xs:double]] / 1000",
                 "[[BaseQuantities:Length##xs:double]]"
              ]
          },
          …
       ]
    }
]
```

Formel

JSON-Format – Mapping

config

- **replace_pset_name**

  Ein bereits vorhandener Propertyset-Name (Attributpräfix) wird durch das im Schlüssel 'pset_name' angegebenen Namen ersetzt.

  Ist die Option nicht vorhanden oder 'false', wird der Name dem neuen Attribut vorangestellt.

  Ein Propertysetname wird dem Attributnamen durch einen Doppelpunkt getrennt vorangestellt.

  true: Pset_Alt:Attributname → Pset_COLNEO:Attributname
  false: Pset_Alt:Attributname → `Pset_COLNEO:Pset_Alt:Attributname`

- **existing_values**

  Legt fest, was bei vorhandenen Attributen geschehen soll.

`overwrite` (default)	Vorhandenes Attribut wird überschrieben
ignore	Vorhandenes Attribut bleibt unverändert
copy_with_date	Vorhandes Attribut wird umkopiert in Attribut mit Zeitstempel.

```
"mappings": [

    {
      "id": "12345",
      "pset_name": "Pset_COLNEO",
      "comment" : "…",

      "config": {
         "replace_pset_name": true,
         "existing_values": "overwrite"
      },

      "mapitems": [
         …
      ]

   },

   {
      …
   }

]
```

Der Propertyset-Name (Attributpräfix) wird ersetzt

Vorhandene Werte werden überschrieben, bleiben unverändert oder werden umkopiert

JSON-Format – Mapping

check

Im Unterabschnitt “check” können Prüfregeln aufgelistet werden, welche für jedes Objekt erfüllt sein müssen, bevor das Mapping durchgeführt wird.

```
"mappings": [

    {
      "id": "12345",
      "pset_name": "Pset_COLNEO",
      "comment" : "…",

      "IF": “...”,

      “preprocess“ : [ ... ],

      „prerequisites“: {
         „check": {
            „checkrules“: [
               „abc“, „def“
            ],
            „validcheckresults“: [„passed“, „ignored“],
            „deletevalue_ifnotpassed“: true
         }
      },

      "mapitems": [
         …
      ]

   },

   {
      …
   }

]
```

Liste von Ids von Prüfregeln

Check results in list are accepted as valid → mapping will be executed; If not defined „passed“ is asumed

Delete value if check was not valid

7 TREE STRUCTURES

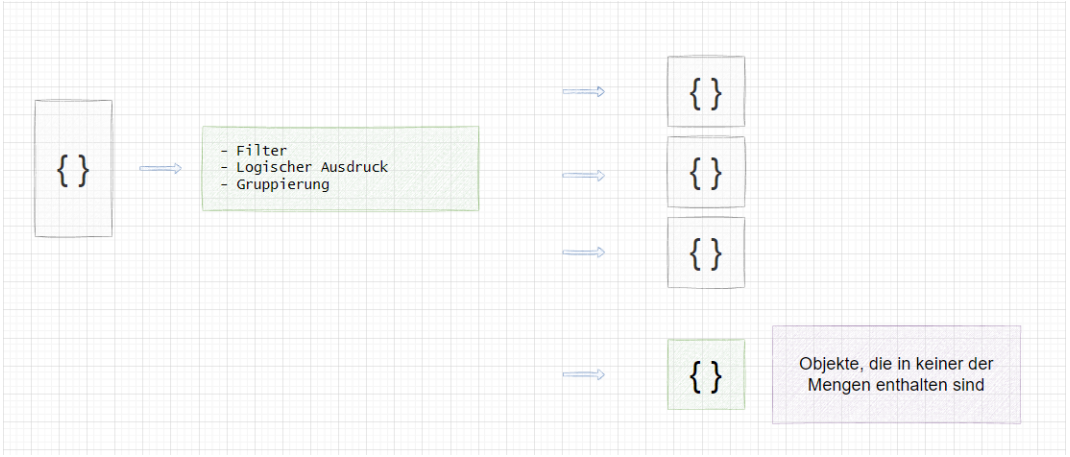
JSON-Format – Tree Structures

Für die Strukturierung von Objekten in Abhängigkeit von ihren Eigenschaften werden Regeln im Schlüssel "**`trees`**" des Schemas spezifiziert.

Hiermit wird eine formale Beschreibung der Bildungsregeln von Mengen auf der Grundlage einer Ausgangsmenge von Objekten spezifiziert.

Die Regeln können für die Ergebnismengen wiederum den Schlüssel "**`tree`**" enthalten, sodass sich im Ergebnis eine Baumstruktur ergibt.

Die Bildungsregeln berücksichtigen sowohl die Mengenbildung durch einen logischen Ausdruck als auch die Gruppierung nach den Werten von Objektattributen (analog zu Auswahlmengen).



JSON-Format – Tree Structures

trees

Der Schüssel **"trees"** enthält eine Liste von Regeln für das Erzeugen von Baumstrukturen.

Eine Regel für das Erzeugen einer Baumstruktur enthält folgende Elemente:

- metadata

  Beschreibung der Baumstruktur
  mit Identifikator, Name und Beschreibung

- properties

  Der Schlüssel "properties" erlaubt das Erzeugen von Eigenschaften am Container. Die Eigenschaften werden in einer Map durch Ihren Propertykey (Name, Datentyp) und Wert angegeben.

```
"trees": [
   {
      "metadata": {
         "id":           "...",
         "name":         "...",
         "description":  "..."
      },
      "properties": {
         "prefix##xs:string": "...",
         "cnName##xs:string": "...",
         "count##xs:long"   : 1
      },

      “filters": {
         "IF": [
            {
               "condition": "{{P0}} == 2",
               "name": "Menge, Wert gleich 2",
               "styles": {
                  "color": "#ff0000"
               }
               "tree": {}
            },
            {
               "condition":  "{{P0}} == undefined",
               "name": "-ohne-Attributwert-"
            },
         ],
         "ELSE": [
            {
               "name" : "-ohne-Attributwert-",
               "tree" : {}
            }
         ]
      }
      "group_by": {}
   }
]
```

JSON-Format – Tree Structures

trees

Der Schüssel **"trees"** enthält eine Liste von Regeln für das Erzeugen von Baumstrukturen.

Eine Regel für das Erzeugen einer Baumstruktur enthält folgende Elemente:

- metadata

  Beschreibung der Baumstruktur
  mit Identifikator, Name und Beschreibung

- properties

  Der Schlüssel "properties" erlaubt das Erzeugen von Eigenschaften am Container. Die Eigenschaften werden in einer Map durch Ihren Propertykey (Name, Datentyp) und Wert angegeben.

```
"trees": [
   {
      "metadata": {
         "id":          "...",
         "name":        "...",
         "description":  "..."
      },
      "properties": {
         "prefix##xs:string": "...",
         "cnName##xs:string": "...",
         "count##xs:long"   : 1
      }

      "filters": {},
      "group_by": {}
   }
]
```

JSON-Format – Tree Structures

- `filters`

Der Schlüssel **`"filters"`** erlaubt das Erzeugen von Mengen, die sich aus Objekten mit definierten Attributwerten ergeben.
Dazu wird eine Liste mit Filterelementen definiert.

Enthält ein Filterelement den Schlüssel **`"IF"`**, wird eine Menge mit den Objekten erzeugt, für der dort definierte Ausdruck wahr ist.

Enthält der Filter den Schlüssel **`"ELSE"`**, wird eine Objektmenge erzeugt, die in keiner der Objektmengen enthalten ist, die sich aus den Filterelementen mit dem Schlüssel **`"IF"`** ergeben.

Filterelemente mit dem Schlüssel **`"IF"`** können mehrmals in der Liste der Filterelemente vorkommen. Die Bedingungen müssen nicht disjunkt sein!

Über den Schlüssel "**condition**" im IF-Filter wird gesteuert, ob die Bedingung für ein Objekt erfüllt ist. Die Bedingung muss (nach Ersetzung der Attributreferenzen) auswertbares Java-Script enthalten.

Der Schlüssel **`"name"`** im Filter bestimmt den Namen des erzeugten Containerelementes.

Der Schlüssel "**styles**" im Filter bestimmt den Stil des erzeugten Containerelementes.

Ist im Schlüssel **`"IF"`** oder **`"ELSE"`** der Schlüssel "`tree`" enthalten, so wird die dort definierte Regel für alle erzeugten Mengen angewendet.

```
"trees": [
   {
       "metadata": { … },
       "properties": { … },

       "filters":{
          "IF": [
              {
                  "id": "...",
                  "condition": "{{P0}} == 2",
                  "name" : "Menge, Wert gleich 2",
                  "tree" : {…}
              },
              {
                  "id": "...",
                  "condition": "{{P0}} == undefined",
                  "name" : "-ohne-Attributwert-",
                  "tree" : {…}
              }
          ],
          "ELSE":
              {
                  "id": "...",
                  "name": "-ohne-Attributwert-",
                  "tree": {…}
              }
       },
        "group_by": {}
   }
]
```

JSON-Format – Tree Structures

- group_by

  Der Schlüssel "**group_by**" ermöglicht das Gruppieren von Objekten nach Attributwerten (= Auswahlmengen).

  Mit dem Schlüssel "**propertytype**" wird das Attribut festgelegt, nach dessen Wert die Gruppen erzeugt werden sollen.

  Ist im Schlüssel "**group_by**" der Schlüssel "tree" enthalten, so wird die in "**tree**" definierte Regel für alle erzeugten Mengen angewendet.

```
{
    …
     {
        "filters": {
            "IF": [
                …
                "tree": {
                    …
                }
            ],
            "ELSE": [
                …
                "tree": {
                    …
                }
            ],
        },
        "group_by": {

            "propertytype": "{{P0}}",

            "options": {
                "interval_size": 2.0,
                "interval_offset": 0.5,
                "case_sensitive": false,
                "date_grouping": "week"
            },

            "tree": {
                …
            }
        }
    }
}
```

JSON-Format – Tree Structures

- `options`

  Mit dem Schlüssel **"options"** innerhalb von **"group_by"** werden Optionen für die Gruppierung festgelegt.

  Mit **"precision"** kann die Genauigkeit (relevante Nachkommastellen) von Fließkommazahlen eingestellt werden.

- Mit **"case_sensitive"** wird gesteuert, ob bei der Gruppierung für textbasierte Datentypen Groß- und Kleinschreibung beachtet werden soll.

  Über **"skip_null_values"** kann eingestellt werden, ob Objekte ohne Wert für den Eigenschaftstyp in den Baum einsortiert werden sollen oder nicht.

  Mit **"interval_size"** kann die Intervallgröße beim Erzeugen von Gruppen spezifiziert werden, die aus einem Attributwert gebildet werden, der eine Zahl (Fließkomma- oder Ganzzahl) darstellt.

  Zusätzlich kann mit **"interval_offset"** ein Offset für das Intervall festgelegt werden.

  Im Beispiel werden die Intervalle [0,5 - 2,5], [2,5 - 4,5] usw. erzeugt, wenn Einzelwerte in diesen Bereich fallen.

  Mit **"date_grouping"** (falls vorhanden) wird festgelegt, wie Datumswerte gruppiert werden sollen. Möglich sind "day", "week" (Kalenderwoche), "month" und "year".

```
{
   …
    {
      "filters": {
         "IF": [
            …
            "tree": {
               …
            }
         ],
         "ELSE": [
            …
            "tree": {
               …
            }
         ],
       },

       "group_by": {

          "propertytype": "{{P0}}",

          "options": {
             "precision": 4,
             "skip_null_values": true,
             "case_sensitive": false,
             "interval_size": 2.0,
             "interval_offset": 0.5,
             "date_grouping": "week"
          },

          "tree": {
             …
          }

       }
    }
}
```

COLNEO

Software // Services // BIM