gegede

BV

January 23, 2016

Contents

1 Overview

- what GGD is

2 General Description of Detector Geometry Information

In GGD, the structure of geometry information is described by a defined *schema*. This schema is formal in the sense that it is a language. The overall structure is composed of specific substructures until one reaches basic data types. This structure is semantic in nature and not syntactical as the GDD supports multiple ways of "spelling" the same information. That is, it supports multiple "formats" or representations.

On the other hand, any GDD schema is represented specifically in JSON syntax and it, itself, follows a a specific structure (which could be called a meta-schema) however this schema structure is defined informally (thus avoiding infinite recursion of meta-meta-. . . -schema!). The structure of the schema and its actual definition is described in the next section.

2.1 FIXME: followed by examples of GDD info representations

2.2 FIXME: followed by examples of conversions

2.3 FIXME: followed by examples of GDD information stores, provenance tracking

3 GDD Schema

GDD geometry description information follows the GDD schema. This schema is intentionally aligned with that of the various expected applications of GDD information. Specifically, GDD schema allows the representing of concepts that are found in Geant4 and ROOT geometry systems as well as the important transportation format, GDML.

The GDD schema is represented as a JSON dictionary. The keys of this dictionary name either elements of the schema or collections of element names.

3.1 Schema elements

Elements (meaning simply "parts" not atomic elements) of the schema are represented as dictionaries. The items of these dictionaries define the low level data member names and their types. Types are expressed with the following strings:

`int` an integer number

`float` a floating point number

`str` a Unicode character string

`list` a list of some type

In the case of `list`, the sub-structure and type of items a list may hold is left unspecified here. It the expected contents of lists are specific to each instance.

3.2 Collections of elements

Schema elements which are of a like kind are collected into a top-level named list. The items in this list are the names of the elements.

3.3 The Schema

This section contains the full schema. This section is the definitive source for this schema definition. It can be extracted by loading this org-mode source file into Emacs and typing `C-c C-v t`.

The schema is expressed as an anonymous JSON dictionary so opens as:

```
{
```

It begins by describing the structure of allowed "shapes" or "solids". These are largely following the Geant4 documentation for solids which serves as documentation for individual data types of the elements. Each instance of a shape requires a unique name.

```
"shapes": ["sphere", "box", "polycone"],
"sphere": {
    "name": "string",
    "r": "float"
},
"box": {
    "name": "string",
    "x": "float",
    "y": "float",
    "z": "float"
},
"polycone": {
    "name": "string",
    "phiStart": "float",
    "phiTotal": "float",
    "zPlane": "list",
    "rInner": "list",
    "rOuter": "list"
}
```

Finally, it closes the top-level dictionary.

```
}
```

An actual set of data structures can then be created and validated against the structure. For example, in Python:

```
from collections import namedtuple
Sphere = namedtuple("Sphere","name r")
Box = namedtuple("Box","name x y z")
data = [
    Sphere(name="ball1", r=1.0),
    Box(name="box1",x=1.0,y=2.0,z=3.0)
]
```