

Cinema Simple Database Specification

v1.0, rev3.0

David Rogers

James Ahrens

John Patchett

LA-UR-15-20572

Cinema Database Specification

Updated: January 29, 2014

David Rogers, Jim Ahrens, John Patchett

I. Overview

Extreme scale scientific simulations are leading a charge to exascale computation, and data analytics runs the risk of being a bottleneck to scientific discovery. Due to power and I/O constraints, we expect in situ visualization and analysis will be a critical component of these workflows. Options for extreme scale data analysis are often presented as a stark contrast: write large files to disk for interactive, exploratory analysis, or perform in situ analysis to save detailed data about phenomena that a scientist knows about in advance. We present a novel framework for a third option – a highly interactive, image-based approach that promotes exploration of simulation results, and is easily accessed through extensions to widely used open source tools. This in situ approach supports interactive exploration of a wide range of results, while still significantly reducing data movement and storage.

More information about the overall design of Cinema is available in the paper, *An Image-based Approach to Extreme Scale In Situ Visualization and Analysis*, which is available at the following link:

<https://datascience.lanl.gov/data/papers/SC14.pdf>

A Cinema database is a collection of data that supports this image-based approach to interactive data exploration. It is a set of images and associated metadata, and is defined and an example given in the following sections.

Use cases

A Cinema Database supports the following three use cases:

1. Searching/querying of meta-data and samples. Samples can be searched purely on metadata, on image content, on position, on time, or on a combination of all of these.
2. Interactive visualization of sets of samples.
3. Playing interactive visualizations, allowing the user on/off control of elements in the visualization.

II. Cinema "simple" Database Specification v1.0

This document describes release v1.0 of the Cinema "simple" Database, and is intended to provide information about how to read or write data in this format.

The "simple" database is:

- a collection of images sampled by a spherical camera that satisfies the use cases described above. In particular, the database is a collection of images for all (time, theta, phi) triplets defined by the camera and time values for the database.
- a JSON file that is a specific instance of the Cinema "simple" Database schema shown in Section III.

The Cinema Database is implementation agnostic. This database specification separates the metadata description of a set of images from the implementation of how these images are stored. In particular, if the images for a specific instance of a database are stored on disk, *the design of the directory structure, metadata files, and image filenames on disk is entirely up to the person writing the data.* Instead, this specification expects a database of URIs that maps metadata to data products required by the specification.

Organization of this document.

This document includes the following sections:

Section III:	Definitions
Section IV:	Outline of a Cinema "simple" Database schema in JSON data format
Section V:	Python example for accessing the sets of image samples in this schema
Section VI:	An example of a the JSON file for a specific database instance
Appendix A:	A json-schema specification for the Cinema "simple" database schema (this section is in progress)

III. Definitions

This section describes the specific objects represented in the database.

Database

A database consists of:

- key, value properties
 - type = "simple"
 - version = "1.0"
- One or more **collections**.

Collection

A **collection** consists of:

- A "label" key, value pair
- A single set of **time values**
- A single spherical **camera**
- A set of one or more **samples**. In particular this is a set of images defined by the spherical camera and the time values.

Time values

A set of unique time values.

- A list of values. This is a list of string values having no duplicates. The interval between values need not be the same.

Camera

A camera defines a set of sample points. These, in addition to the time values, define all possible samples.

- This version of the specification supports only a spherical camera, representing images taken from a point on a sphere directed towards the center of that sphere. The points on the sphere are defined by a regular phi and theta sampling of that sphere.
- Required:
 - A list of values for phi. This is a list of string values having no duplicates. Phi can vary from 0.0 to 180.0, with 0.0 being defined as the 'south pole' and 180.0 as the 'north pole'.
 - A list of values for theta. This is a list of string values having no duplicates. Theta varies from 0.0 to 360.0.
 - Integer values for the camera's [width, height] in pixels.

Sample

A sample is a set of images matching a specific **camera**.

- "label" key, value pair
- A single image for each (theta, phi, timestep) triplet. The file format of the image should be a common format such as jpeg, tiff, png, etc.

IV. Outline of a Cinema "simple" Database schema in JSON data format

This schema encodes the information for the above definitions, and contains the required information needed to create a specific instance of a Cinema "simple" Database. Note that it requires a hierarchy of (time, theta, phi) when organizing the samples for a specific collection, but does not enforce any rules on the URIs that define the access to the files within a collection.

```
{
  "type" : "simple",
  "version" : "1.0",
```

```

"collections": [
  {
    "label": <string>,
    "time": [ list of unique string values ],
    "camera": {
      "theta": [ list of unique string values ],
      "phi": [ list of unique string values ],
      "image": {
        "width" : <integer>,
        "height": <integer>
      }
    },
    "samples": [
      {
        "label": <string>,
        "images" : {
          <time value string> : {
            <theta value string> : {
              <phi value string> : <URI>,
              ... (additional phi values)
            }
            ... (additional theta values)
          }
          ... (additional time values)
        }
      }
    ]
  }
]
}

```

V. Python example for accessing the sets of image samples in this schema

Python example to read, parse and iterate through images as defined in an instance of a Cinema "simple" Database schema - in this case found in a file named "results.json". This example access and prints the URIs for each images, but other code could perform other operations such as loading this image.

```
import json

# open and load the data
json_data = open( 'results.json' )
data = json.load(json_data)

# iterate through collections
for c in data['collections']:

    # iterate through samples
    for s in c['samples']:

        # access image URIs
        images = s['images']

        # iterate over (time, theta, phi)
        for t in c['time']:
            for theta in c['camera']['theta']:
                for phi in c['camera']['phi']:

                    print images[t][theta][phi]
```

VI. Example

This example is based on above a JSON schema outline, and shows the specific contents of the required files, and the names that result from the time, theta and phi values. For this simple demo, the files are mapped to URIs that describe paths to files on disk.

```
{
  "label" : "halo",
  "type" : "simple",
  "version" : "1.0",
  "collections": [
    {
      "label" : "earth",
      "time": ["0.0", "0.1", "0.2"],
      "camera": {
        "theta": ["0.0", "90.0", "180.0", "270.0"],
        "phi": ["0.0", "45.0", "90.0", "135.0"],
        "image": {
          "width" : 1920,
          "height": 1080
        }
      },
      "samples": [
        {
          "label": "salinity",
          "images" : {
            "0.0" : {
              "0.0" : {
                "0.0" : "file:///usr/local/data/0.0/0.0/0.0/image.png",
                "45.0" : "file:///usr/local/data/0.0/0.0/45.0/image.png",
                "90.0" : "file:///usr/local/data/0.0/0.0/90.0/image.png",
                "135.0" : "file:///usr/local/data/0.0/0.0/135.0/image.png",
                "180.0" : "file:///usr/local/data/0.0/0.0/180.0/image.png"
              },
              "90.0" : {
                "0.0" : "file:///usr/local/data/0.0/90.0/0.0/image.png",
                "45.0" : "file:///usr/local/data/0.0/90.0/45.0/image.png",
                "90.0" : "file:///usr/local/data/0.0/90.0/90.0/image.png",
                "135.0" : "file:///usr/local/data/0.0/90.0/135.0/image.png",
                "180.0" : "file:///usr/local/data/0.0/90.0/180.0/image.png"
              },
              "180.0" : {
                "0.0" : "file:///usr/local/data/0.0/180.0/0.0/image.png",
                "45.0" : "file:///usr/local/data/0.0/180.0/45.0/image.png",
                "90.0" : "file:///usr/local/data/0.0/180.0/90.0/image.png",
                "135.0" : "file:///usr/local/data/0.0/180.0/135.0/image.png",
                "180.0" : "file:///usr/local/data/0.0/180.0/180.0/image.png"
              },
              "270.0" : {
                "0.0" : "file:///usr/local/data/0.0/270.0/0.0/image.png",
                "45.0" : "file:///usr/local/data/0.0/270.0/45.0/image.png",
                "90.0" : "file:///usr/local/data/0.0/270.0/90.0/image.png",
                "135.0" : "file:///usr/local/data/0.0/270.0/135.0/image.png",
                "180.0" : "file:///usr/local/data/0.0/270.0/180.0/image.png"
              }
            },
            "0.1" : {
              "0.0" : {
                "0.0" : "file:///usr/local/data/0.0/0.0/0.0/image.png",
                "45.0" : "file:///usr/local/data/0.0/0.0/45.0/image.png",
                "90.0" : "file:///usr/local/data/0.0/0.0/90.0/image.png",
                "135.0" : "file:///usr/local/data/0.0/0.0/135.0/image.png",
                "180.0" : "file:///usr/local/data/0.0/0.0/180.0/image.png"
              },
              "90.0" : {
                "0.0" : "file:///usr/local/data/0.0/90.0/0.0/image.png",
                "45.0" : "file:///usr/local/data/0.0/90.0/45.0/image.png",
                "90.0" : "file:///usr/local/data/0.0/90.0/90.0/image.png",
                "135.0" : "file:///usr/local/data/0.0/90.0/135.0/image.png"
              }
            }
          }
        }
      ]
    }
  ]
}
```

```
"180.0" : "file:///usr/local/data/0.0/90.0/180.0/image.png",
},
"180.0" : {
  "0.0" : "file:///usr/local/data/0.0/180.0/0.0/image.png",
  "45.0" : "file:///usr/local/data/0.0/180.0/45.0/image.png",
  "90.0" : "file:///usr/local/data/0.0/180.0/90.0/image.png",
  "135.0" : "file:///usr/local/data/0.0/180.0/135.0/image.png",
  "180.0" : "file:///usr/local/data/0.0/180.0/180.0/image.png"
},
"270.0" : {
  "0.0" : "file:///usr/local/data/0.0/270.0/0.0/image.png",
  "45.0" : "file:///usr/local/data/0.0/270.0/45.0/image.png",
  "90.0" : "file:///usr/local/data/0.0/270.0/90.0/image.png",
  "135.0" : "file:///usr/local/data/0.0/270.0/135.0/image.png",
  "180.0" : "file:///usr/local/data/0.0/270.0/180.0/image.png"
}
},
"0.2" : {
  "0.0" : {
    "0.0" : "file:///usr/local/data/0.2/0.0/0.0/image.png",
    "45.0" : "file:///usr/local/data/0.2/0.0/45.0/image.png",
    "90.0" : "file:///usr/local/data/0.2/0.0/90.0/image.png",
    "135.0" : "file:///usr/local/data/0.2/0.0/135.0/image.png",
    "180.0" : "file:///usr/local/data/0.2/0.0/180.0/image.png"
  },
  "90.0" : {
    "0.0" : "file:///usr/local/data/0.2/90.0/0.0/image.png",
    "45.0" : "file:///usr/local/data/0.2/90.0/45.0/image.png",
    "90.0" : "file:///usr/local/data/0.2/90.0/90.0/image.png",
    "135.0" : "file:///usr/local/data/0.2/90.0/135.0/image.png",
    "180.0" : "file:///usr/local/data/0.2/90.0/180.0/image.png"
  },
  "180.0" : {
    "0.0" : "file:///usr/local/data/0.2/180.0/0.0/image.png",
    "45.0" : "file:///usr/local/data/0.2/180.0/45.0/image.png",
    "90.0" : "file:///usr/local/data/0.2/180.0/90.0/image.png",
    "135.0" : "file:///usr/local/data/0.2/180.0/135.0/image.png",
    "180.0" : "file:///usr/local/data/0.2/180.0/180.0/image.png"
  },
  "270.0" : {
    "0.0" : "file:///usr/local/data/0.2/270.0/0.0/image.png",
    "45.0" : "file:///usr/local/data/0.2/270.0/45.0/image.png",
    "90.0" : "file:///usr/local/data/0.2/270.0/90.0/image.png",
    "135.0" : "file:///usr/local/data/0.2/270.0/135.0/image.png",
    "180.0" : "file:///usr/local/data/0.2/270.0/180.0/image.png"
  }
}
}
}
}
```


Appendix A: Cinema Simple Database JSON schema

(Under construction) This JSON schema uses jsonschema to validate instances of Cinema "simple" Databases. It does not validate the URIs used to map data products to this file.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "CinemaSimpleDatabase",
  "description": "Cinema Simple Database",
  "type": "object",
  "required": ["type", "version", "label", "collections"],
  "properties": {
    "type": {
      "description": "The unique type for this database",
      "type": "string",
      "enum": [ "simple" ],
      "default": "simple"
    },
    "version": {
      "description": "The version for this database",
      "type": "string",
      "enum": [ "1.0" ],
      "default": "1.0"
    },
    "label": {
      "description": "The label for this database",
      "type": "string",
      "minLength": 1
    },
    "collections": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/collection"
      },
      "minItems": 1,
      "uniqueItems": true
    }
  },
  "definitions": {
    "sample": {
      "type": "object",
      "required": ["label", "images"],
      "properties": {
        "label": {
          "type": "string"
        },
        "images": {
        }
      }
    },
    "collection": {
      "type": "object",
      "required": ["label", "time", "camera", "samples"],
      "properties": {
        "label": {
          "type": "string"
        },
        "time": {
          "$ref": "#/definitions/time"
        },
        "camera": {
          "$ref": "#/definitions/camera"
        },
        "samples": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/sample"
          }
        }
      }
    }
  }
}
```

```

    "minItems" : 1,
    "uniqueItems" : true
  }
},
"time" : {
  "type" : "array",
  "items" : {
    "type" : "string"
  },
  "minItems" : 1,
  "uniqueItems" : true
},
"camera" : {
  "type" : "object",
  "required" : ["theta", "phi", "image"],
  "properties" : {
    "theta" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      },
      "minItems" : 1,
      "uniqueItems" : true
    },
    "phi" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      },
      "minItems" : 1,
      "uniqueItems" : true
    },
    "image" : {
      "$ref" : "#/definitions/image"
    }
  }
},
"image" : {
  "type" : "object",
  "required" : ["width", "height"],
  "properties" : {
    "height" : {
      "type" : "integer",
      "minimum" : 1
    },
    "width" : {
      "type" : "integer",
      "minimum" : 1
    }
  }
}
}
}
}

```

Appendix B: Validation Example

The following python example shows how to validate an instance of a schema with jsonschema - in this case, using a files named `schema.json` and `instance.json`.

```
import jsonschema
import json

schema_file = "schema.json"
instance_file = "instance.json"

schema = open( schema_file ).read()
instance = open( instance_file ).read()

print "validating:"
print "  schema: " + schema_file
print "  instance: " + instance_file

try:
    jsonschema.validate( json.loads( instance ), json.loads( schema ) )
except jsonschema.ValidationError as e:
    print e.message
except jsonschema.SchemaError as e:
    print e
```