Cinema Database Specification
Dietrich Release v1.2

Document Version 1.0
(June 2018)
LA-UR-17-25072

by

David Rogers dhr@lanl.gov
Jon Woodring woodring@lanl.gov
James Ahrens ahrens@lanl.gov
John Patchett patchett@lanl.gov

Los Alamos National Laboratory
Bikini Atoll Rd., SM 30
Los Alamos, NM 87545
cinema@lanl.gov

Contents

1 Introduction

This document is a specification for Cinema databases. It is specification D (Dietrich), version 1.2. This is a different approach than past specifications [1] [2], but is complimentary in spirit and function to them. This version is a simple embodiment of the new approach, which can be easily adapted to a wide range of use cases, and is designed for quick adoption by scientists, programmers and others.

See the Cinema website (`http://www.cinemascience.org`) for additional information, and contact the Cinema community (`cinema-info@lanl.gov`) or the authors of this document with questions.

2 Cinema 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. Cinema represents a novel framework for a third option a highly interactive, data artifact-based approach that promotes exploration of simulation results, and is easily accessed through database specifications. This 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* [3].

A Cinema Database supports the following three use cases. Taken together, these support a novel method for interactively exploring artifacts from extremely large datasets.

1. Searching/querying of meta-data and data artifacts. Samples can be searched purely on metadata, content, position, time, or a combination of all of these.

2. Interactive visualization of sets of data artifacts.

3. Playing interactive visualizations, allowing the user on/off control of elements in the visualization.

2.1 What is a Cinema Database?

A Cinema database is a set of precomputed data artifacts that can be queried and interactively viewed. The user can decide what types of components comprise the database, based on the type of interaction that is desired with the final database. A general design philosophy of Cinema is that applications reading and viewing a Cinema database can ignore data and determine which operations to perform. This promotes a wide range of possible interactions with the data - not just the ones imagined by the creator of the database.

Previous Cinema database specifications have concentrated on the notion that a database is a set of results sampled by visualization parameters. In this specification, we abstract this to include typical use cases from experiments by scientists.

A scientist often has a spreadsheet with data about parameters for an experiment. This results in a set of parameters that map to a particular result - a graph, a sensor image, or other image-based data. These are a natural abstraction from previous Cinema databases. Figure 1 shows this mapping of parameter sets to results. A collection of these mappings can be easily expressed in a Cinema database, using this specification. This is detailed in later sections.

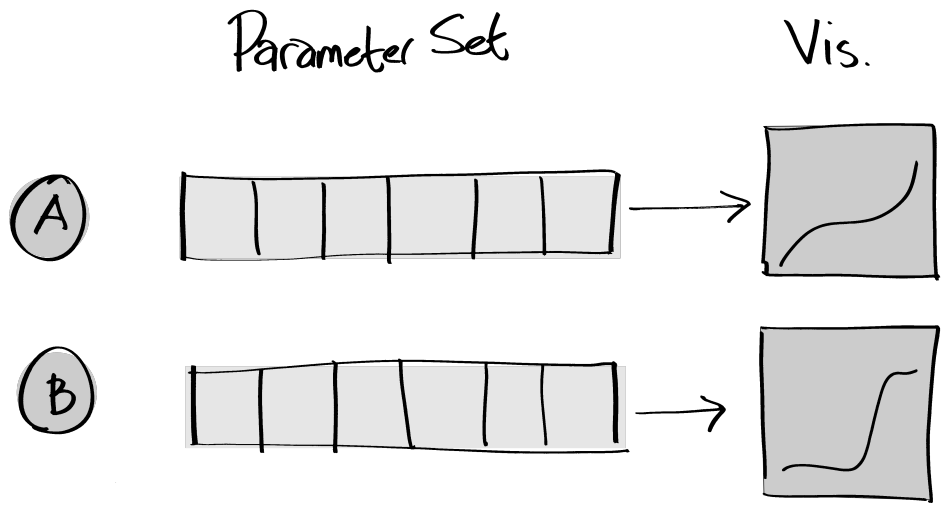

Figure 1: Diagram showing a typical set of data that a scientist has for an experiment or simulations. Some set of parameters (A or B) has been used to create a visualization - a graph, an image captured from a sensor, or other data. These parameter sets can include, for example, settings on an experimental machine, inputs to a simulation, or measurements taken by a sensor. Each one of the parameter sets thus defines a unique result. Taken together, a set of these parameter sets constitutes a database of results, and a scientist often tracks this database in a spreadsheet. These parameter set/image pairings form the basis for the simplest Spec D database.

3 The Cinema D (Dietrich) v1.2 Specification

Cinema D (Dietrich) specification is a table-based data model for scientific data. In version 1.2 of this specification, we take the simplest approach to this new model, to simplify adoption and interaction with Cinema.

3.1 Required Elements

A Specification D version 1.2 Cinema database is directory that contains all of the data in the database. The directory must have the following files. Additional files or directories may be present, but they are ignored by Cinema.

- a directory named `database_name.cdb` (*database_name* is the name of the database) containing:
  - `data.csv`, a Comma Separated Value file. This file is specified in Section 3.2. The presence of this specifically named file indicates that this is a Spec D, version 1.2 Cinema database.
  - optional data files, referenced by the `data.csv` file.
  - additional files in the directory, ignored by this specification.

3.2 The `data.csv` File

The main data file for this specification is a Comma Separated Value file that has the following requirements. The last data column(s) may reference one or more external files. These referenced files can be of any file type or file extension. Note that *new data can be added to or deleted from the `csv` file or files can be added to or removed from the database i.e., the `database_name.csv` directory, as long as the following requirements are met.* The specification for the Spec D v1.2 `csv` follows (differing from v1.0 in points 5 and 9).

1. The file is UTF-8 encoded.
2. The file follows the specification rfc4180 [4], governing how fields are defined.

3. The first line of the file (header) is **required**.
4. Header values are unique non-empty **strings** that are the labels for the columns.
5. The last header values may be or start with `FILE`. No other non-`FILE` columns can occur after the first `FILE` columns (`FILE` columns are always last).
6. All other lines in the file are data rows.
7. There must be at least one data line in the file.
8. Each data line is populated with an equal number of values (columns) as the header (i.e., the same number of separating commas per row).
9. In the data lines:
   (a) All data values must be **floats** (floating-point values), **integers**, **strings**, or **empty** (missing) values.
   (b) All data values in `FILE` columns are **required** to be **strings** or **empty** values.
   (c) All other columns are **required** to have the same value type for all data rows, either **float**, **integer**, or **string**.
   (d) A column may not have an **empty** value type, but **empty** values may appear in any column and data row.
   (e) Empty values (i.e., missing values, `NULL`s, or `None`s) are represented by an empty string.
   (f) A literal empty **string** is represented by two double-quotes: `""`, which is different from an **empty** (missing) value or **NaN**.
   (g) The **string** values in `FILE` columns represent a POSIX file path relative to the base directory (the `name.cdb` path) of the Cinema database. This is the location of the file data for that row.
   (h) The files can be of any type, where the format is indicated by MIME name extension.
      i. Consumers of Spec D databases (readers, viewers, and analytics tools) are not required to display or read all formats. This is handled on a case-by-case basis for the use cases supported by the tool. Tools must provide appropriate error messages indicating when file formats are not supported.
      ii. For clarity, we note here that browser-based viewers of Spec D are required to display the common web image file formats of JPEG, PNG, and GIF, and may ignore other file formats.

3.2.1 Examples

Example 1 This type of `data.csv` file is self-contained with no `FILE` columns. It shows the one change made for this version of the specification: allowing NULL values in the first line of the file:

```
timestep,time value,x,y,z,category
1,,1.0,,1.2,one
2,0.2,2.0,2.1,2.2,two
```

In this case, the Cinema database would contain the following files, assuming that *database_name* is the name of the database.

```
database_name.cdb/
    data.csv
```

Example 2 This type of `data.csv` file is self-contained with no `FILE` columns. It shows a set of fully populated data rows (no NULL values).

```
timestep,time value,x,y,z,category
1,0.1,1.0,1.1,1.2,one
2,0.2,2.0,2.1,2.2,two
3,0.3,3.0,3.1,3.2,three
4,0.4,4.0,4.1,4.2,four
```

In this case, the Cinema database would contain the following files, assuming that *database_name* is the name of the database.

```
database_name.cdb/
    data.csv
```

Example 3 This `data.csv` example shows the `FILE` keyword on the last column and the corresponding POSIX file paths.

```
timestep,time value,x,y,z,category,FILE
1,0.1,1.0,1.1,1.2,one,img/001.jpg
2,0.2,2.0,2.1,2.2,two,img/002.jpg
3,0.3,3.0,3.1,3.2,three,data/003.jpg
4,0.4,4.0,4.1,4.2,four,data/004.png
```

In this case, the Cinema database would contain the following files:

```
database_name.cdb/
    data.csv
    img/
        001.jpg
        002.jpg
    data/
        003.jpg
        004.png
```

Example 4 This `data.csv` example shows an example of NULL, NaN, and empty string values in the last row.

```
timestep,time value,x,y,z,category,FILE
1,0.1,1.0,1.1,1.2,one,img/001.jpg
2,0.2,2.0,2.1,2.2,two,img/002.jpg
3,0.3,3.0,3.1,3.2,three,data/003.jpg
4,0.4,4.0,4.1,4.2,four,data/004.png
,0.5,,NaN,5.2,"",
```

In this case, the Cinema database would contain the same files as Example 3.

Example 5 This `data.csv` example shows multiple `FILE`s on the last columns, and the corresponding POSIX file paths. Note that the file columns are named `FILE` and `FILE plot`, in this example.

```
timestep,time value,x,y,z,category,FILE,FILE plot
1,0.1,1.0,1.1,1.2,one,img/001.jpg,plot/a.csv
2,0.2,2.0,2.1,2.2,two,img/002.jpg,plot/b.tsv
3,0.3,3.0,3.1,3.2,three,data/003.jpg,plot/c.txt
4,0.4,4.0,4.1,4.2,four,data/004.png,plot/d.xls
,0.5,,NaN,5.2,"",,
```

In this case, the Cinema database would contain the following files:

```
database_name.cdb/
    data.csv
    img/
        001.jpg
        002.jpg
    data/
        003.jpg
```

```
        004.png
    plot/
        a.csv
        b.tsv
        c.txt
        d.xls
```

Example 6 This `data.csv` example shows string quotation (rfc4180) to create a single text vector column from the *x*, *y*, and *z* columns.

```
timestep,time value,"x,y,z",category,FILE,FILE plot
1,0.1,"1.0,1.1,1.2",one,img/001.jpg,plot/a.csv
2,0.2,"2.0,2.1,2.2",two,img/002.jpg,plot/b.tsv
3,0.3,"3.0,3.1,3.2",three,data/003.jpg,plot/c.txt
4,0.4,"4.0,4.1,4.2",four,data/004.png,plot/d.xls
,0.5,,"",,
```

In this case, the Cinema database would contain the same files as Example 5.

Example 7 This example shows a database with multiple data artifacts, and demonstrates the reason for the change made to the specification.

```
timestep,phi,theta,FILE,FILE dataset
1,45.0,45.0,01_45.png,
1,90.0,90.0,01_90.png,
2,45.0,45.0,02_45.png,
2,90.0,90.0,02_90.png,
1,,,,subset_01.vti
```

In this case, a series of `png` files have been saved at different camera angles at two different timesteps. In addition, a `vti` file has been saved for the first timestep. There are NULL values on each line, indicating that the data artifacts (the FILE columns) are associated with different sets of parameters. The files would be:

```
database_name.cdb/
    data.csv
    01_45.png
    01_90.png
    02_45.png
    02_90.png
    subset_01.vti
```

3.2.2 Data Types

The second line (first data line) is used to determine the data type of the columns. We assume that built-in parsing in modern languages, such as in Python and JavaScript, is available and can be used to test, parse, and convert values from the `csv` file.

In our previous examples **1** and **2**, the types of the columns are **integer**, four columns of **float**, and **string**. Parsing precedence for determining the type of a column is: **integer**, **float**, **string**. That is, if a column in the first data row (second line) parses as an **integer**, it is an **integer** column, otherwise if it parses as a **float**, it is a **float** column, otherwise it is a **string** column. This does mean that if you have floating-point values that are integers, they need to be written as floating point in the second line (first data row) to be able to type a column correctly, i.e., a 0 must be stored as 0.0.

For strings:

- `nan`, `NaN`, and `NAN` (or any combination of capitalization) are valid values, and should be handled by any readers and viewers according to the type of the column. If these are present in the first data row, the column is parsed as **float** values. `NaN` is not a valid **integer** value for any row.

- Whitespace is not consumed (stripped) before and after comma separators. This is part of the rfc4180 specification. Whitespace *cannot* appear before or after a comma that has a double-quoted value (an invalid rfc4180).
- Commas and double-quotes are special characters that require enclosing the value in double-quotes. A literal double-quote must be preceded by another double-quote: e.g., `""""` is the value `"` (please refer to rfc4180).
- A literal empty string must be represented as `""`, as an empty string (no value between two commas, or before the first comma or after the last comma) is `NULL` (empty value).

4 Contacts and Further Information

For further information, email the Cinema mailing list at `cinema-info@lanl.gov`, or contact the authors of this document. Additional information is available at the Cinema website, `http://www.cinemascience.org`.

5 Changes from v1.1

Version 1.2 supersedes version 1.1. All v1.1 files are compliant v1.2 files, but v1.2 files are not backwards compatible with v1.1. The following are the changes that have been made from v1.1 to v1.2:

- The first data line can contain (`NULL`) values.

6 Acknowledgements

The image used on the cover page is a publicity photo of Marlene Dietrich for the film Shanghai Express (1932). This photo is in the public domain [5].

7 Bibliography

References

[1] D. Rogers, J. Ahrens, and J. Patchett, "Cinema simple database specification," Tech. Rep. LA-UR-15-20572, Los Alamos National Laboratory, January 2015.

[2] D. Rogers, J. Woodring, J. Patchett, D. DeMarle, and B. Geveci, "Cinema database specification chaplin release," Tech. Rep. LA-UR-15-20645, Los Alamos National Laboratory, January 2017.

[3] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, and M. Petersen, "An image-based approach to extreme scale in situ visualization and analysis," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, (Piscataway, NJ, USA), pp. 424–434, IEEE Press, 2014.

[4] Y. Shafranovich, "Common Format and MIME Type for Comma-Separated Values (CSV) Files." RFC 4180, Oct. 2005.

[5] Deanlaw, "Marlene dietrich in shanghai express (1932) by don english," 2016. `https://commons.wikimedia.org/wiki/File:Marlene_Dietrich_in_Shanghai_Express_(1932)_by_Don_English.png`.