OPAT File Format Specification

Version 1.0

Emily M. Boudreaux

March 2025

Abstract

The untitled 4DSSE code (hereafter "the code") will make use of the OPAT (Open Parameterized Array Table) file format for efficient storage and retrieval of double precision tabular data parameterized by some arbitrary length vector (such as a composition vector). The format is designed to be extensible, self-contained, and computationally efficient, supporting fast lookups, structured metadata, and data integrity verification. We have developed the OPAT format specifically for opacity tables; however, we are releasing it generally under GPLv3 and encourage its use for other applications. This document contains a full description of OPAT, including data types, endianness, byte arrangement, and interface tools.

1 Introduction

The OPAT format provides a structured binary format for storing tabular data sets indexed by an arbitrary floating point vector (for example hydrogen mass fraction (X) and metal mass fraction (Z)). It is designed with high-performance, data-integrity, and generalizability in mind. OPAT is the format that all radiative opacity tables are stored in for the code. The format ensures data integrity using SHA-256 checksums. All numeric values are stored as 64-bit floating point (double) values; this ensures approximately 15 decimals of precision. Data are explicitly stored in **little endian** format. Each OPAT file consists of three main sections:

1. **File Header**: Contains metadata, format version, and a lookup table offset.
2. **Data Cards**: Contains data cards. These are collections of tables and other data associated to index vectors.
3. **Card Catalog**: Maps indexing vector values to byte offsets for direct access.

A few important things to note here are that the table index is stored at the **end** of the file. Therefore when reading, the index offset will need to be used in order to move the current position to the end of the file. This is done for ease of writing since the checksums of each data card will not be known until the cards have been read in. Each data card contains its own header which outlines its structure. These are discussed in detail in §3.

2 OPAT File Structure

An OPAT file is structured as follows:

Section	Size (bytes)	Description
File Header	256 (fixed)	Metadata, format version, index offset
Data Cards	Variable	A Header, lookup index, and arbitrary numeric data
Card Catalog	Variable	Maps indexing vector to byte locations of data card

2.1 File Header

The header provides general metadata and file organization details. The header is always 256 bytes long.

Field	Type	Size (bytes)	Description
Magic Number	char[4]	4	"OPAT" identifier
Version	uint16	2	Format version (e.g., 1.0 = 0x0001)
Num Cards	uint32	4	Number of stored data cards
Header Size	uint32	4	Byte offset to the Data Cards
Catalog Offset	uint64	8	Byte offset of card catalog
Creation Date	char[16]	16	Creation Date (Feb 15, 2025)
Source Info	char[64]	64	Description of data source
Comment	char[128]	128	Units, notes, etc.
Num Indices	uint16	2	Number of index values per table
Hash Precision	uint8	1	Presicion to round index values too when calculating hash[1]
Reserved	char[23]	23	Future use (zero-filled)

2.2 Card Catalog

Each entry in the catalog provides byte offsets for locating the corresponding data card. Note that the index vector size may be variable from OPAT file to OPAT file, but within one OPAT file it must always be constant. Data cards can be indexed with vectors of size anywhere between 1 and 255 floats. This might look like a table for each X, Z pair (as in OPAL type I files) or a vector of X, Z, excess O, and excess C (as in OPAL type II files). This must be specified before adding any tables. **All tables in a**

[1] While this value can range from 0-255 numerically, modules enforce that its max value is 14 due to the max presicion of float64

single OPAT file must use an index vector of the same size or reading will fail. The python `opatio` module enforces this behavior through the `OpatIO.set_numIndex` method.
N.B. If duplicate index vectors are given then only the most recent card associated to an index vector will be stored. The python module has safe checking for this (explicitly disallowing duplicate index vectors).

Field	Type	Size (bytes)	Description
Index n `n`	float64	8	n^{th} index of file
.....	float64	8	$(n+1)^{th}$ index
Byte Start	uint64	8	Start of data card in file
Byte End	uint64	8	End of data card in file
Checksum	char[32]	32	SHA-256 checksum of data card

Each entry is 48 + 8*numIndex bytes long, allowing for efficient binary searching. The minimum possible size is 56 bytes per index, while the maximum possible size is 2088 bytes per index.

2.3 Data Card Lookup

Recall that the purpose of the card catalog is to provide a lookup table between vectors of floats and a particular data card. Therefore we need some way to map vectors to entries in the table. The most obvious choise, comparing the value in a provided vector directly to the vector stored in the index field, has some major issues.

1. Floating point errors make looking up tables this way very prone to error.
2. Each time a table is requested the entire index may need to be iterated over to find the correct entry.

Both of these issues are resolvable while still doing direct comparisons. However, a better approach, and the one which OPAT adopts, is to use a hash map. Specifically OPAT files round all floats to the presicion given in the header field "Hash Precision" and then calculate the murmmer hash of this vector. That has is then used in a hash map to allow for average O(1) lookup efficiency while also avoiding floating point rounding errors. Note that this means that when setting a table the user needs to make sure that the Hash Presicion is greater than the minimum seperation between any values in the index vectors. The various official opatIO modules automatically enforce this constraint. By default the Hash Presicion is set to 8 which cooresponds to 8 digits of presicion in base 10.

3 Data Cards

Each data card is formatted similarly to the overall OPAT table. Each card contains a header (which is also always 256 bytes long), an index (64 bytes long per entry), and a set of tables. The structure of a data card is as follows:

1. **Card Header**: Contains metadata and an offset to the Card Index
2. **Tables**: Contains the actual data stored in the card
3. **Card Index**: Contains byte offsets, relative to the start of the data card, for the locations of tables. Tables are indexed by tags which are 8-byte unsigned character arrays.

3.1 Data Card Header

Each data card has a header not unlike the global header for the entire OPAT file. This has the following form:

Field	Type	Size (bytes)	Description
Magic Number	char[4]	4	"CARD" identifier
Num Tables	uint32	4	Number of stored tables

Header Size	uint32	4	Byte offset to the Data Tables relative to the start of the data card
Index Offset	uint64	8	Byte offset of card Index relative to the start of the data card
Card Size	uint64	8	Total byte size of the data card
Comment	char[128]	128	Units, notes, etc.
Reserved	char[100]	100	Future use (zero-filled)

Note that the header is 256 bytes.

3.2 Data Card Index

The data card index is written immediately after the header (this is in contrast to the Card Catalog for the entire OPAT file which is the last part of the file). It contains a variable number of bytes depending on how many tables are present in the file. The data card index is a table with 7 columns and a number of rows equal to the Num Tables field in the header. These data are arranged in row-major order in the file. Each row has the form:
N.B. The same tag can exist in multiple data cards; however, within one data card there may only be one instance of the same tag. Tags are case insensitive.

Field	**Type**	**Size (bytes)**	**Description**
Tag	char[8]	8	tag used to identify the table in the card.
Byte Start	uint64	8	Start byte of the table relative to the start byte of the card. The start byte of the card is byte 0.
Byte End	uint64	8	End byte of the table relative to the start byte of the card. The start byte of the card is byte 0.
Num Columns	uint16	2	Number of columns the table contains.
Num Rows	uint16	2	Number of rows the table contains.
Column Name	char[8]	8	Label for value parameterizing columns.
Row Name	char[8]	8	Label for value parameterizing rows.
Reserved	char[20]	20	Reserved for future use.

3.3 Data card tables

The actual data in an OPAT file is stored in tables. These are row-major lists of double precision values. Each table has a known size from the data card index and so reading in is simply a matter of reading the correct number of bytes and rearranging to fit the number of rows x number of columns.

4 Checksum and Data Integrity

Each data card is assigned a SHA-256 checksum stored in the Card Catalog for validation. These should be checked whenever reading in OPAT files (the official OPAT libraries handle this automatically.)

5 Creation

The python module `opatio` (in `utils/opatio`) provides a straightforward interface for the creation and reading of OPAT formatted files. There is a README in that directory which outlines its use.

6 Usage

Both the python module `opatio` and the C++ module `opatIO` allow for OPAT file generation.