jHDF
Pure Java HDF5

Dr James Mudd
March 15th 2024
https://jhdf.io
james.mudd@gmail.com

My History

2010 -2014 Physics PhD
"Photoelectron Spectroscopy Investigation of CdO"

2014-2019 Diamond Light Source
Scientist / Software Engineer

2019-2023 OXMT
Software engineer - Java SAAS Application

2023-Current Osprey Charging Network
Software engineer - Java/Kotlin Microservices + iOS/Android Apps

jHDF History

Nov 2018 - First Commit

Jan, 2019 - v0.2.1 - First MVN central release

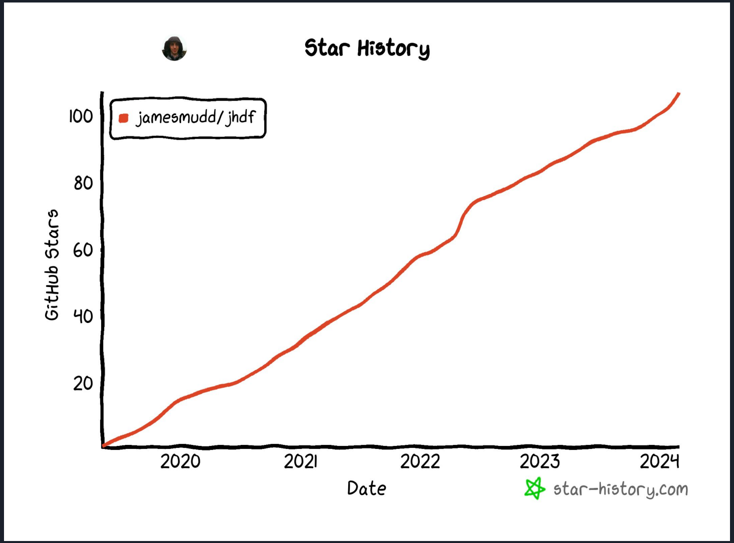
Mar 2019 - v0.4.0 - Add attributes

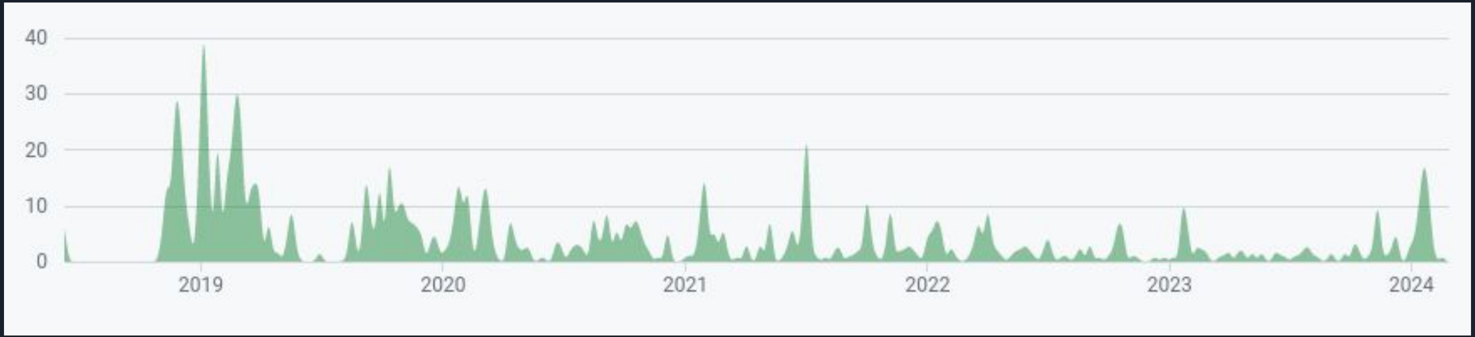
Oct 2019 - v0.5.0 - Chunked v4 datasets

Mar 2020 - First child + UK Covid Lockdown

Feb 2021 - v0.6.0 - In memory file support

Jan 2024 - v0.7.0-alpha - Preview of writing support

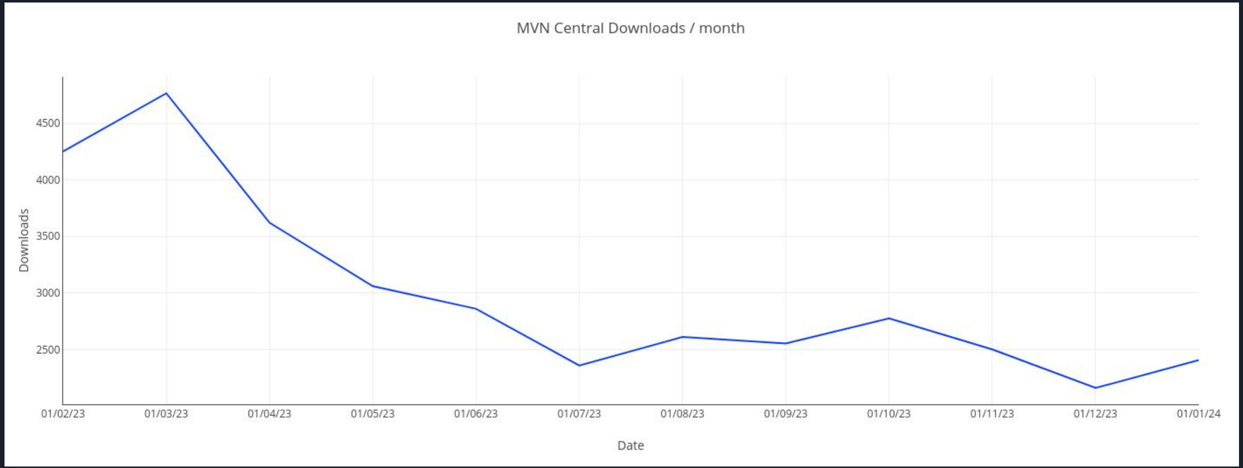




Current Support

- All data types (except **Time** WIP)
- Deflate, LZ4, LZF, Bit+Byte shuffle filters
- Attributes
- Compound and Array data types
- Reading n dimensions with reshaping
- Limited checksum support
- On disk, in memory, Java VFS
- User block

Users


MVN Central Downloads / month
Downloads
4500
4000
3500
3000
2500
Date
01/02/23
01/03/23
01/04/23
01/05/23
01/06/23
01/07/23
01/08/23
01/09/23
01/10/23
01/11/23
01/12/23
01/01/24

Why jHDF?

- Simple integration in JVM based projects.
- No requirement to load native code
- Clean API for HDF5 access
- Automatic mapping of data into Java types
- No use of JNI
- Lazy loading
- Concurrent access support

Current limitations:

- Max size of arrays is ~2B per dimension
- No Unsigned types in the language so need to be promoted
- Not quite 100% of files will work
- Lack of slicing for chunked datasets means really large datasets can be processed.

DEMO.....

Modern Java

Virtual threads (Released in Java 21)

- Can write procedural blocking code which can to be called multi-threaded
- Allows for many (>1M virtual threads)
- Primary targeted for web server applications but in theory any IO bound app might see performance gains
- Avoids the reactive (callback) paradigm

Foreign Function & Memory API (preview in 21, release in 22?)

- Possible replacement for JNI
- Deterministic, explicit memory allocation/deallocation API - memory segment
- Larger buffers (above 2GB)
- Better than JNI performance (especially for call back into Java methods)
- jextract tool to auto generate Java bindings for any library so you only need to write Java
- https://www.youtube.com/watch?v=iwmVbeiA42E

Modern Java VMs

GraalVM - https://www.graalvm.org/

- More advanced JIT optimizations
- Ability to AOT optimizations
- Native image generation
- Low memory usage and ultra fast startup (serverless focus)
- Polyglot - JavaScript, Python, Ruby, WebAssembly

TornadoVM - https://www.tornadovm.org/

- Java on GPU/FPGA
- OpenCL, CUDA, Intel Level Zero backends
- Automatic compute kernel generation

Future

- Get basic writing support released
- Slicing of chunked datasets
- Finish Time data type implementation
- Support for newer feature? SWMR?
- Object mapper?
- Nice API supporting combining slicing and streams
- HDFView like app maybe in Kotlin Multiplatform?

Thanks for listening!

Dr James Mudd
https://jhdf.io
james.mudd@gmail.com

Abstract

jHDF a pure Java implementation of HDF5 from the file format specification. This session will provide a quick history of the project and the current status, a walk through of some of the key APIs, a discussion of some of the key reasons to consider using jHDF, and then a look to the future on where project developer Dr. James Mudd thinks the project should go.