* Introduction

# Overview

The Hierarchical Data Format (HDF) was designed to be an easy, straightforward, and self-describing means of sharing scientific data among people, projects, and types of computers. An extensible header and carefully crafted internal layers provide a system that can grow as scientific data-handling needs evolve.

This document, the HDF Specification and Developer's Guide, fully describes the HDF data models, the corresponding file format specifications, and library implementation, and discusses criteria employed in the library’s development. Where appropriate, this document provides limited guidelines for developers working on HDF itself or building applications that employ HDF.

This introduction provides a brief overview of HDF capabilities and design.

## Why HDF?

A fundamental requirement of scientific data management is the ability to access as much information in as many ways, as quickly and easily as possible. A data storage and retrieval system that facilitates these capabilities must provide the following features:

Support for scientific data and metadata

Scientific data is characterized by a variety of data types and representations, data sets (including images) that can be extremely large and complex, and the need to attach accompanying attributes, parameters, notebooks, and other metadata. Metadata, supplementary data that describes the basic data (sometimes referred to as the raw data), includes information such as the dimensions of an array, the number type of the elements of a record, or a color lookup table (LUT).

Support for a range of hardware platforms

Data can originate on one machine only to be used later on many different machines. Scientists must be able to access data and metadata on as many hardware platforms as possible.

Support for a range of software tools

Scientists need a variety of software tools and utilities for easily searching, analyzing, archiving, and transporting the data and metadata. These tools range from a library of routines for reading and writing data and metadata, to small utilities that simply display an image on a console, to full-blown database retrieval systems that provide multiple views of thousands of sets of data and metadata.

Rapid data transfer

Both the size and the dispersion of scientific data sets require that mechanisms exist to get the data from place to place rapidly.

Extendibility

As new types of information are generated and new kinds of science are done, a means must be provided to support them.

## What is HDF?

The HDF Structure

HDF is a self-describing extensible file format using tagged objects that have standard meanings. The idea is to store both a known format description and the data in the same file. HDF tags describe the format of the data because each tag is assigned a specific meaning; for example, the tag DFTAG\_LUT indicates a color palette, the tag DFTAG\_RI indicates an 8-bit raster image, and so on . A program that has been written to understand a certain set of tag types can scan the file for those tags and process the data. This program also can ignore any data that is beyond its scope.

Consider a data set representing a raster image in an HDF file as illustrated in Figure 1a below. The data set consists of three data objects with distinct tags representing the three types of data. The raster image object contains the basic data (or raw data) and is identified by the tag DFTAG\_RI; the palette and dimension objects contain metadata and are identified by the tags DFTAG\_LUT tags DFTAG\_ID.

* Raster Image Set in an HDF File .

The set of available data objects encompasses both basic data and metadata. Most HDF objects are machine- and medium-independent, physical representations of data and metadata.



HDF Tags

The HDF design assumes that we cannot know a priori what types of data objects will be needed in the future, nor can we know how scientists will want to view that data. As science progresses, people will discover new types of information and new relationships among existing data. New types of data objects and new tags will be created to meet these expanding needs. To avoid unnecessary proliferation of tags and to ensure that all tags are available to potential users who need to share data, a portable public domain library is available that interprets all public tags. The library contains user interfaces designed to provide views of the data that are most natural for users. As we learn more about the way scientists need to view their data, we can add user interfaces that reflect data models consistent with those views.

Types of Data and Structures

HDF currently supports the most common types of data and metadata that scientists use, including multidimensional gridded data, 2-dimensional raster images, polygonal mesh data, multivariate data sets, finite-element data, non-Cartesian coordinate data, and text.

In the future there will almost certainly be a need to incorporate new types of data, such as voice and video, some of which might actually be stored on other media than the central file itself. Under such circumstances, it may become desirable to employ the concept of a virtual file. A virtual file functions like a regular file but does not fit our normal notion of a monolithic sequence of bits stored entirely on a single disk or tape.

HDF also makes it possible for the user to include annotations, titles, and specific descriptions of the data in the file. Thus, files can be archived with human-readable information about the data and its origins.

One collection of HDF tags supports a hierarchical grouping structure called a Vgroup that allows scientists to organize data objects within HDF files to fit their views of how the objects go together, much as a person in an office or laboratory organizes information in folders, drawers, journal boxes, and on their desktops.

Backward and Forward Compatibility

An important goal of HDF is to maximize backward and forward compatibility among its interfaces, and storage and object types. This is not always achievable, because data formats must sometimes change to enhance performance, to correct errors, or for other reasons. However, whenever possible, HDF files should not become out of date. For example, suppose a site falls far behind in the HDF standard so its users can only work with the portions of the specification that are three years old. Users at this site might produce files with their old HDF software then read them with newer software designed to work with more advanced data files. The newer software should still be able to read the old files.

Conversely, if the site receives files that contain objects that its HDF software does not understand, it should still be able to list the types of data in the file. It should also be able to access all of the older types of data objects that it understands, despite the fact that the older types of data objects are mixed in with new kinds of data. In addition, if the more advanced site uses the text annotation facilities of HDF effectively, the files will arrive with complete human-readable descriptions of how to decipher the new tag types.

Calling Interfaces

To present a convenient user interface made up of something more usable than a list of tag types with their associated data requirements, HDF supports multiple calling interfaces, utilities, and applications.

The low-level calling interface is used to manipulate tags and raw data, to perform error handling, and to control the physical storage of data. This interface is designed to be used by developers who are providing the higher level interfaces for applications like raster image storage or scientific data archiving. See Chapter , Low-level Interface, and in Chapter , Software Overview, see Section 3.3, "Software Organization."

The application interfaces, at the next level, include several modules specifically designed to simplify the process of storing and accessing specific types of data. For example, the palette interfaces are designed to handle color palettes and lookup tables, the general raster (GR) interface is designed to handle generalized raster images, and the scientific data (SD) interface is designed to handle arrays of scientific data. If you are primarily interested in reading data from or writing data to HDF files, you will spend most of your time working with the application interfaces. See Section 3.3, "Software Organization," for a complete list of these APIs.

The HDF utilities and NCSA applications, at the top level, are special purpose programs designed to handle specific tasks or solve specific problems. The utilities provide a command line interface for data management. The applications provide solutions for problems in specific application areas and often include a graphic user interface. Several third party applications are also available at this level.

Machine Independence

An important issue in data file design is that of machine independence or transportability. The HDF design defines standard representations for storing all data types that it supports. When data is written to a file, it is typically written in the standard HDF representation. The conversion is handled by the HDF software and need not concern the user. Users may override this convention and install their own conversion routines, or they may write data to a file in the native format of the machine on which it was generated.

## Some History

In 1987 a group of users and software developers at NCSA searched for a file format that would satisfy NCSA's data needs. There were some interesting candidates, but none that were in the public domain, were targeted to scientific data, and yet were sufficiently general and extensible. In the course of several months, borrowing concepts from several existing formats, the group designed HDF.

The first version of HDF was implemented in the spring and summer of 1988. It included a general purpose interface and an 8-bit raster image interface. In the fall of 1988, a scientific data set interface was designed and implemented, enabling HDF users to store multidimensional arrays and related data. Soon thereafter interfaces were implemented for storing color palettes, 24-bit raster images, and annotations.

In 1989, it became clear that there was a need to support a general grouping structure and unstructured data such as that used to represent polyhedra in graphical applications. This led to Vsets, whose interface routines were implemented as a separate HDF library.

Also in 1989 it became clear that the existing general purpose layer was not sufficiently powerful to meet anticipated future needs and that the coding could use a substantial overhaul. From this, the long process of redesigning the lower layers of HDF began. The first version incorporating extended tags and the new lower layers of HDF was released in the summer of 1992 as HDF Version 3.2.

In 1993, in response to the needs of flexibility in data ranges and sizes, HDF Version 3.3 was released. In this version of HDF, the new SD interface was introduced with multi-file access and an unlimited dimension feature for arrays. HDF Version 3.3 provided alternative physical storage methods (external and linked block data elements) through extended tags, JPEG data compression, changes to some Vset interface functions, access to netCDF files through a complete netCDF interface,[[1]](#footnote-1) hyperslab access routines for old-style SDS objects, and various performance improvements.

In 1994, as standard ANSI C became more commonly used, HDF shifted from K&R to ANSI C to support portability. After several beta versions, HDF Version 4.0 was released in 1996 and provided features such as support for n-bit integers and SDS compression, limited support for reading CDF files, a parallel I/O interface for the CM5, auto configuration, multi-file versions of the AN and GR interfaces, and significant improvement in I/O performance and memory usage. In addition, more options were added to existing HDF utilities and two new programs were added to the HDF utilities:

* hdp, to view the contents of HDF files
* hdfunpac, to unpack scientific datasets into external elements

HDF Version 4.1 was released in 1997. In this version, attributes were added to both the Vdata and Vgroup APIs to provide more ways for meaningfully storing data, data chunking was introduced in the SD API to improve I/O performance, and a new representation was used for storing dimensions to improve storage efficiency.

In 1998, the second release of HDF Version 4.1, called Version 4.1r2, was announced. In this release, data chunking was added for the GR API, the Java Products (the Java-based HDF Viewer, JHV, and the Java HDF interface, JHI) were incorporated into the HDF release itself, and the HDF Reference Manual and HDF User’s Guide were extensively updated. In addition, the new representation of dimensions that was introduced in the previous release became the default representation.

HDF Version 4.1r3, released in May 1999, emphasized fixing problems in the SD and GR interfaces. The HDF User's Guide accompanying the release was significantly improved and updated. The term Vset became obsolete, being replaced with the more specific terms Vgroup and Vdata.

The current release, HDF Version 4.1r4, released in October 2000, completes the enabling of all GR chunking capabilities. In addition, new options were added to the hdp utility. This document, the HDF Specification and Developer’s Guide, was largely rewritten for this release.

See the HDF website at https://portal.hdfgroup.org/display/HDF4/ for release information, lists of supported platforms, and the list of bugs fixed in the current release.

The HDF library is considered mature and complete at this time. Future work will focus on technical support, maintenance, and bug fixes; there are no plans to implement new features. All new features and tools are being implemented in the HDF5 library, a new, next-generation product from the same team that built and supports HDF. HDF5 is discussed in detail at https://www.hdfgroup.org/solutions/hdf5/.

## About This Document

This document is designed for software developers who are designing applications or routines for use with HDF files and for users who need detailed information about HDF. Users who are interested in using HDF to store or manipulate their data will not normally need the kind of detail presented in this manual. They should instead consult one of the user-level documents:[[2]](#footnote-2)

Versions 4.x

[HDF User's Guide](https://support.hdfgroup.org/release4/doc/UsrGuide_html/UG_html.htm)

[HDF Reference Manual](https://support.hdfgroup.org/release4/doc/RefMan_html/RM_Top.html)

A tutorial is available online at the following URL:  
 https://support.hdfgroup.org/services/learning.html

New material appears throughout this edition of The HDF Specification and Developer’s Guide, but the following chapters bear special mention. Chapters 7 and 8 and Appendix B are entirely new. Chapter 10 contains new compression and chunking information and some material that previously appeared in Chapter 9.

Users of third-party software that uses HDF may also have to consult a manual for that software.

## Document Contents

The HDF Specification and Developer's Guide contains the following chapters and appendix:

Chapter 1: Introduction

Introduces the document and provides an overview.

Chapter 2: Basic Structure of HDF Files

Introduces and describes the components and organization of HDF files.

Chapter 3: Software Overview

Describes the organization of the software layers that make up the basic HDF library and provides guidelines for writing HDF software.

Chapter 4: Low-level Interface

Describes the low-level HDF routines that make up the low-level interface (see also the H routines section of the HDF Reference Manual).

Chapter 5: Sets and Groups

Explains the roles of sets and groups in an HDF file, and describes raster image sets, scientific data sets, and Vgroups.

Chapter 6: Annotations

Explains the use of annotations in HDF files.

Chapter 7: Scientific Data Sets:The SD Model

Explains the role, structure, and usage of SDSs in HDF files.

Chapter 8: General Raster Images: The GR Model

Explains the role, structure, and usage of GRs in HDF files.

Chapter 9: Tag Specifications

Describes the tag identification space and the HDF-supported basic tags.

Chapter 10: Extended Tags and Special Elements

Describes the extended tag structure and the HDF-supported extended tags and special elements.

Chapter 11: Portability Issues

Describes the measures taken to maximize HDF portability across platforms and to ensure that HDF routines are available to both C and FORTRAN programs.

Appendix A: Tags and Extended Tag Labels

Presents a list of HDF-supported tags and a list of labels used with extended tags.

Appendix B: Library Calling Trees

Illustrates the calling structure of HDF library functions.

Appendix C: Function Specifications

Provides detailed specifications for selected low-level interface functions.

## Conventions Used in This Document

Most of the descriptive text in this guide is printed in 10 point Times. Other typefaces have specific meanings that will help the reader understand the functionality being described.

New concepts and newly defined terms are sometimes presented in bold italics on their first occurrence to indicate that they are defined within the paragraph.

Cross references within the specification include the title of the referenced section in quotation marks or the reference chapter in italics. (E.g., See Section 3.3, "Software Organization," in Chapter , Software Overview, for a complete list of...)

References to documents italicize the title of the document. (E.g., See the HDF User’s Guide to familiarize yourself with the basic principles of using HDF.)

Literal expressions and variables often appear in the discussion. Literal expressions are presented in Courier while variables are presented in italic Courier. A literal expression is any expression that would be entered exactly as presented, e.g., commands, command options, literal strings, and data. A variable is an expression that serves as a place holder for some other text that would be entered. Consider the expression cp file1 file2, cp is a command name and would be entered exactly as it appears, so it is printed in Courier. But file1 and file2 are variables, place holders for the names of actual files, so they are printed in italic Courier; the user would enter the actual filenames.

This guide frequently offers sample command lines. Sometimes these are examples of what might be done; other times they are specific instructions to the user. Command lines may appear within running text, as in the preceding paragraph, or on a separate line, as follows:

cp file1 file2

Command lines always include one or more literal expressions and may include one or more variables, so they are printed in Courier and italic Courier as described above.

Keys that are labeled with more than one character, such as the RETURN key, are identified with all uppercase letters. Keys that are to be pressed simultaneously or in succession are linked with a hyphen. For example, "press CONTROL-A" means to press the CONTROL key then, without releasing the CONTROL key, press the A key. Similarly, "press CONTROL-SHIFT-A " means to press the CONTROL and SHIFT keys then, without releasing either of those, press the A key.

Table 1A summarizes the use of typefaces as used in examples and illustrations of HDF code and data, such as in literal strings and on sample command lines.

|  |  |  |  |
| --- | --- | --- | --- |
| * Meaning of Entry Format Notations | | | |
| Type | Appearance | Example | Entry method |
| Literal expression (commands, literal strings, data) | Courier | do this | Enter the expression exactly as it appears. |
| Variables | Italic Courier | filename | Enter the name of the file or the specific data that this expression represents. |
| Special keys | Uppercase | RETURN | Press the key indicated. |
| Key combinations | Uppercase, with hyphens between key names | CONTROL-A | While holding down the first one or two keys, press the last key. |

Program listings and screen listings are presented in Courier typeface, as in Figure 1b. When the listing is intended as a sample that the reader will use for an exercise or model, variables that the reader will change are printed in italic Courier.

* Sample screen listing

mars\_53% ls -F

MinMaxer/ net.source

mars\_54% cd MinMaxer

mars\_55% ls -F

list.MinMaxer minmaxer.v1.04/

mars\_56% cd minmaxer.v1.04

mars\_57% ls -F

COPYRIGHT minmaxer.bin/ source.minmaxer/

README sample/ source.triangulation/

mars\_58%

1. NetCDF is a network-transparent derivative of the original CDF (Common Data Format) developed by the National Aeronautics and Space Administration (NASA). It is used widely in atmospheric sciences and other disciplines requiring very large data structures. NetCDF is in the public domain and was developed at the Unidata Program Center in Boulder, Colorado. [↑](#footnote-ref-1)
2. The user-level documents for Versions 3.2 and earlier were NCSA HDF Calling Interfaces and Utilities and NCSA HDF Vset; for Version 3.3, they were Getting Started with NCSA HDF, NCSA HDF User's Guide, and NCSA HDF Reference Manual. Library versions prior to Version 4.0 and the corresponding documents are no longer supported or available. [↑](#footnote-ref-2)