INTERDISCIPLINARY RESEARCH SCENARIO TESTING OF EOSDIS

Final Report
Under
NASA Grant NAG5-1961

Covering the period of
February 1998 - August 14, 1999

Prepared by

G.D. Emmitt
Principal Investigator
University of Virginia
Charlottesville, VA 22903

12 November 1999

Overview

During the reporting period, the PI has continued to serve on numerous review panels, task forces and committees with the goal of providing input and guidance for the EOSDIS program at NASA Headquarters and NASA GSFC. In addition, the PI has worked together with personnel at the University of Virginia and the subcontractor (Simpson Weather Associates (SWA)) to continue to evaluate the latest releases of various versions of the user interfaces to the EOSDIS. Finally, as part of the subcontract, SWA has created an on-line HDF tutorial for non-HDF experts, particularly those that will be using EOSDIS and future EOS data products. A summary of these three activities is provided below.

Participation on EOSDIS Panels and Committees

During the reporting period, the PI has been involved in numerous activities related to preparing for the EOSDIS for the Terra launch, and planning for the post 2002 era:

- Reviewing prototyping proposals related to IT
- Chairing the EOS Science Data Panel
- Conducting surveys on the needs and interests of the IDS teams
- Participating on the EOSDIS Review Group
- Participating in the NewDIS activities lead by NASA HQ.
- Serving on various ad hoc panels dealing with the EOSDIS and its readiness for operations
- Attending SEC and IWG meetings.

Until recently, ESDIS was responsible for soliciting and awarding funds to develop or evaluate concepts that would have significant potential for the AM-1 platform era, as well as the period of time beyond. Approximately 20 proposals were reviewed at the request of ESDIS.

In the fall of 1998, the PI was elected chair of the Science Data Panel. A Panel meeting was held in May, 99 (see Attachment A). The Data Panel's recommendations were forwarded to the ERG that met soon after. The Data Panel's future is still an issue. However, since it serves the IWG, the Panel will remain active and will meet when it is deemed necessary.

As part of the descoping and budget cutting associated with EOSDIS, the PI was asked by ESDIS to survey the IDS teams for their reactions to the proposed OPTION A+ . The results of the survey were reported to the ERG and the SEC. A copy of the survey is included in Attachment B.

The current EOS EDC contract expires in 2002. NASA Headquarters commissioned a study to be chaired by Martha Maiden. The study was called the NewDIS. The PI was a consultant to the study team and attended two of the workshops.

During the course of getting the EOSDIS ready for support of the Terra launch, the PI was requested to serve on several ad hoc review teams. The most recent such team was the board for the Operations Readiness Review for the Terra/Sage III launches.

Evaluation and Tire Kicking of EOSDIS User Interfaces

In response to several of the defined tasks, SWA together with the PI and a student at UVA continued to evaluate the releases of various versions of the user interfaces to the EOSDIS. In particular, the student and SWA continued this advanced tire kicking to probe and evaluate the WWW version of the V0-gateway. In

addition, alternate search methods (i.e., individual Data Centers, DAACs, and Internet search engines) outside of V0 were also investigated and compared to V0 results.

Inexperienced users of V0 at both SWA and UVA were tasked with conducting searches for data sets using different types of keywords and valids as input. Searches were attempted for data sets that should contain information on winds, clouds, land use, vegetation cover, surface reflectance, lidar measurements, dust, and aerosols. The data set searches were done using the WWW V0 gateway, by contacting individual data centers and DAACs, and via Internet search engines (i.e., Infoseek, AltaVista, etc.).

One of the most telling examples was the search for AVHRR data using different valids (dust, aerosol, etc.) that were known to be covered by the AVHRR data. Inconsistent results were found under different search conditions. The valid of "dust" resulted in no "hits" of AVHRR data while using "aerosol" as a valid resulted in hits depending on the other information supplied.

One of the main findings of this exercise was that the searchable metadata for data sets doesn't always give the proper information, and is often populated by jargon that is of no use to the non-expert user or one unfamiliar with the data sets. These results helped lead to the submission and subsequent award of a proposal by a UVA colleague (Dr. Jim French) that is attempting to find better definitions for metadata and the way that they are used and searched by the interactive data systems such as V0.

An On-line HDF Tutorial

Under the subcontract, SWA has developed an on-line tutorial entitled "An HDF Tutorial for Beginners: EOSDIS Users and Small Data Providers." The tutorial was geared mainly for HDF non-experts, particularly potential future users of EOS data, with the main purpose of providing the necessary information needed to enable a user to read and write data in HDF. Information is provided in a clear an easy to understand form and includes step-by-step directions on how to work with the HDF files. Also included in the tutorial are sections on the basics of HDF and the HDF library; programming with HDF; available tools for HDF (including links), example programs, and many other features.

The tutorial was developed using, but not limited to, the following resources:

- Lessons learned by novice HDF users at SWA and UVA
- Meetings with NCSA
- Studying all existing documentation and applying/collating the most important material for novice users
- Attending annual workshops on HDF and HDF-EOS
- Learning common problems with HDF through user feedback on the tutorial and participation in various HDF newsgroups and mailing lists

Copies of the monthly reports reflecting the progress of the tutorial development are available upon request.

The tutorial has been constructed in two parts. First is what we call the "Lecture" component where we present what we think is the information necessary for a novice user to learn what HDF is, what it can be used for, and how to apply it in practice. Included in this "Lecture" material is a step-by-step outline detailing what the user must do to successfully read or write an HDF file. The second component of the tutorial is a question and answer section (what we call the "Laboratory") which tests the user's knowledge of HDF, concentrating on the information needed by the novice or average HDF user to work independently with the HDF library to read and write HDF files.

We realize that the familiarity and knowledge level of the users of this tutorial will span a wide range. As a result, we think it should be up to the users to decide how they wish to learn and navigate through the

tutorial. However, we do advise that those with very little or no knowledge of HDF should first preview and study the lecture material before testing themselves with the Laboratory section.

The tutorial was initially developed in Visual Basic and was only available for users with Windows 95/98. Early drafts of this initial version were partially described in the 16 February 1998 Progress Report, including reprints of conference papers given on the tutorial. However, the tutorial is now available via the World Wide Web and can be accessed at Simpson Weather Associate's HDF page at the following Internet address: http://cyclone.swa.com/meteorology/hdf/. A copy of this HTML version is found in Attachment C. In addition to the Internet/HTML version, a Microsoft word version of the tutorial is also available for download at the same address. A copy of the Microsoft Word version of the tutorial is found in Attachment D. It should be noted that the copies of the tutorial placed in the attachments do not contain the entire Question and Answer section of the tutorial due the interactive nature of the Laboratory. However, a few examples of the questions are included in Attachment C. The entire Laboratory can be viewed by visiting the above-mentioned site and viewing the HTML version of the tutorial.

In addition to the above tutorials, ongoing work under a follow-up proposal is expanding the tutorial to include additional HDF data types and to cover HDF-EOS. HDF-EOS is an extension of the HDF library that helps the user to deal with certain types of point, gridded, and swath data sets that will be routinely generated from EOS missions. A Beta version of this tutorial should be available shortly at the same Internet site noted above.

ATTACHMENT A

Report from the May 1999 Science Data Panel

by

Dr. G.D. Emmitt
University of Virginia
Charlottesville, VA

(In Fulfillment of NASA Contract NAG5-1961)

November 12, 1999

Report from the Data Panel

G. D. Emmitt, UVa

ERG Meeting at NASA Headquarters

4 May, 1999

Summary of Data Panel Meeting

- Held 2-3 May, 1999 in Charlottesville, VA
- 9 panel members
- ESDIS Representatives: Rama , Mike Moore, and Ken McDonald
- Headquarters: Martha Maiden
- Briefed on:
  - EMOS/EDOS
  - ECS
  - DAACs
  - IIMS
  - V0
  - Option A+
  - Federation Experiment
  - NewDISS

Panel's Comments/Recommendations

- **Mission Systems**
  - Data Panel appreciates the substantial effort and progress made since last November
  - The Panel is concerned with the failure of EDOS to complete 3 days without L0 data losses.
- **ECS**
  - Panel urges the project to ensure that the Level 1 requirements for modularity, maintainability and platform portability be fulfilled in a way that allows transfer of the ECS system to another responsible party for M & O at the conclusion of the current contract.

Panel's Comments/Recommendations

- **Instrument Teams**
  - The Panel is concerned with the late and incorrect delivery of science processing software and metadata by some Instrument Teams. In some cases, significant last minute changes to algorithms and metadata will affect many data processing interdependencies. The Panel recommends that the project ensure more serious adherence to delivery schedules for Terra and subsequent missions.

Panel's Comments/Recommendations

- **Option A+**
  - The ESE needs to develop and publicize a viable plan that will assure the user community that the processing and distribution capacity of the system will be augmented in a timely fashion to meet any demonstrated demands beyond the reduced levels associated with Option A+. This plan should also include data resource allocation board with representatives form the earth research and data applications communities.
  - The 6 month "rolling archive" should be extended to a minimum of 12 months and preferably 18 months to more appropriately accommodate annual variability analyses. Also, the 3X processing capacity of Option A+will have to absorb the additional reprocessing associated with the proposed 6 month "rolling archive".
  - The Data Panel requests a cost estimate for the 12-18 month extension of the "rolling archive" and will respond with recommendations for offsetting any additional costs.

Panel's Comments/Recommendations

- **V0 (STIIMS)**
  - The Data Panel finds the augmented V0 IMS adequate for near term post-launch operations with one exception. The current state of the valids (data access vocabulary) represents an unacceptable risk to the usability of the system. It is recommended that the project address this issue promptly. Some suggestions made by panel members include training sessions with DAAC personnel; presentations at IWG meetings; or developing simple web pages to aid in directing users to the right data sets.

Panel's Comments/Recommendations

- **V0 (LTIIMS)**
  - The panel concurs with the Project that the IIMS will need additional improvements. However, we recommend that implementation of changes be incremental and decisions on any final LTIIMS plans be deferred until 6 months after launch.
  - The Panel understands and appreciates the challenge faced by the Project to plan for anticipated future needs while dealing with many other immediate and pressing data system demands.

Panel's Comments/Recommendations

- **Federation Experiment**
  - It is too early in the federation experiment to judge its success.
  - The Panel recommends that the experiment manager provide clear success criteria for this 2-3 year effort.
- **NewDISS**
  - The Panel is concerned with the current lack of a clear exposition of the "Lessons Learned" from the current DIS experience as they relate, in particular, to the NewDISS vision.(Note: The NewDISS report was not available prior to the Data Panel meeting)
  - Currently the manager of the Federation Experiment, the Chair of the NewDISS Team and the Headquarters person responsible for information systems are the same person. Having the lead author of the NewDISS report also be the recipient of that report creates the potential for conflict of interest. The Panel agrees with the Chair's (of the NewDISS team) recommendation of an independent review prior to submitting the NewDISS report to the NRC.
  - The Data Panel is willing to provide such a review if formally requested.

ATTACHMENT B

Survey Requesting IDS Teams Input on the Descoping and Rescoping of the EOSDIS

by

Dr. G.D. Emmitt
University of Virginia
Charlottesville, VA

(In Fulfillment of NASA Contract NAG5-1961)

November 12, 1999

Request for input from the IDS teams to the process of descoping and rescoping of the EOSDIS

As you know, there is currently an intensive effort underway within NASA, the ESDIS Project in particular, to develop an EOSDIS plan that will stay within a shrinking budget over the next 4-5 years. After that period, the expectation is that there will be a transition to the "New DISS" which is being studied by a team headed by Martha Maiden. For a status report on EOSDIS see Skip Reber's article on page 37 of the Nov/Dec issue of The Earth Observer.

The bottom line is that some of the original functionality and performance of the DIS is going to have to be dropped or rescheduled. The ESDIS has presented numerous sets of options for meeting budget guidelines. There are detailed matrices of requirements, data interdependencies, and so forth for those who wish to delve into the trade space in detail (e-mail me if you want 51 viewgraphs, the latest version of those presented by Mike Moore at the IWG meeting). However, since these budget and scoping exercises seem to be nearly continuous and the details always changing, the EOSDIS Data Panel has been asked to provide some general guidance from the user community, from the IDS teams in particular. Thus I am asking for a few minutes of your time to weigh in on this matter.

First, here is my summary of Option A+ (don't be misled by the + sign) which is currently favored by ESDIS and agreed to by Ghassem.

- Rely on the existing V0 user interface (with some modest upgrades) for the early AM-1 mission era, put an indefinite hold on the development of the ECS JEST, and see how far we go with the rapidly changing data search capabilities available on the "Web".
- Eliminate automated order tracking
- Reduce system data processing capacity from 4x to 3x where "x" is the capacity required to process all data collected in 24 hours.
- Reduce the data distribution capacity from 1.6x to 1x where "x" is the capacity required to distribute all data processed in 24 hours.
- Archive levels 1 and 2 data products for only six months (with exceptions for those cases when there are no higher level products) with processing on demand thereafter. Level 3 and above data products will always be archived.
- Allow PI-led data processing to take place at PI institutions or by PI negotiated arrangements with DAACs

While you may wish to comment directly on one or more of the items above or on particular issues raised by Mike Moore's viewgraphs, there are several key trades that always need addressed when budgets are tightened. They are:

- Providing user services through several focus DAACs (LaRC, GSFC, JPL, etc) **vs.** one or two super DAACs. This trade comes down to the cost of people vs. the importance of the perception of specialized service to the user and the benefits of DAACs competing with each other for new data sets. By the way, the super DAACs were not considered for Option A+.

- Archiving all Level 2 and higher data products **vs.** producing higher level products upon demand (i.e. cost of archival vs. cost of production)
- Developing specialized EOS data search and display tools vs. relying on generally available internet data search tools (puts emphasis on making sure EOS data are visible to those search tools rather than developing a customized data/metadata format)
- Providing specialized services such as subsetting, coincidence searching, order tracking, automated version update notifications, etc vs. straightforward data listing and order functions and media options. (shifts the burden of sifting through large data sets for the target information from the DIS to the individual user).

If you do nothing more, please express your opinion, as a cross discipline data user, on:

- Multi vs. centralized user service centers
- Rapid data retrieval from archives vs. reprocessing upon demand (less timely, perhaps)
- Using generally available Internet tools to search for data sets vs. customized tools that may be more efficient.
- Simplifying the function of the DIS to advertising data sets and filling orders without special services such as subsetting, coincidence searching, granule content searches.

I need to hear from you immediately. No response will be seen by ESDIS as general agreement with their recommendations for reducing user services.

Dave Emmitt
Chair, EOSDIS Science Data Panel
Simpson Weather Associates, Inc.
809 E. Jefferson St.
Charlottesville, Va. 22902
804-979-3571
804-979-5599 fax
gde@thunder.swa.com

ATTACHMENT C

An HDF Tutorial for Beginners: EOSDIS Users and Small Data Providers (HTML Version)

by

Mr. Steven Greco
Simpson Weather Associates
Charlottesville, VA

(In Fulfillment of NASA Contract NAG5-1961)

November 12, 1999

Main Topics

An Introduction to HDF

Previous Main Topic Next Main Topic

Return to Main Topics

What is HDF?

HDF, which stands for Hierarchical Data Format, is a common data format that has been developed to aid scientists and programmers in the storing, transfer and distribution of data sets and products created on various machines and with different software. HDF has been selected by the NASA ESDIS project as the format of choice for the standard product distribution that will be part of the Earth Observing System Data and Informations System (EOSDIS).

In addition, HDF also refers to the collection of software, application interfaces, and utilities that comprise the HDF library and allows users to work with HDF files. The HDF library is discussed in detail in Section 3 - The HDF Library: Software and Hardware.

Features of HDF

HDF is a multi-object file format for the sharing and storing of scientific data. Some of the most important features of HDF are the following:

1. Self-describing: For each data object in an HDF file, there is also information (or metadata) about the data type, size, dimensions and location found within the file itself.
2. Extensibility: HDF is designed to accommodate future (new) data types and data models.
3. Versatility: Currently, HDF supports six different data types and provides software and applications to read and write these data types in HDF.
4. Flexibility: HDF lets the user group, store, and read/write different data types in the same file or in more than one file.
5. Portability: HDF software is mainly platform independent and can be shared across most computer platforms (all platforms have not been tested).

6. Standardization: HDF standardizes the formats and descriptions of many types of commonly-used data types (i.e., arrays, images, etc.).
7. HDF is available in the public domain.

Return to top

What types of data does HDF support?

As of the latest release of HDF (HDF4.1 release 3 in May 1999), the HDF library supports the working with raster images, color or gray scale palettes, multi-dimensional arrays, text strings, and statistical data (in the form of tables). The HDF library supports the following data types:

1. Scientific Data sets -- Multi-dimensional integer or floating point arrays
2. Vertex Data (Vdata and Vgroups) -- Multi-variate data stored as records in a table
3. General Raster (Gr) -- Raster images
4. Annotation -- Text strings to describe files and parts of files (metadata)
5. 8-bit Raster images
6. 24-bit Raster images
7. Palette -- 8-bit color palettes (accompany images)

In addition to these data types supported by the base HDF library, a sub-library called HDF-EOS has been developed to support the various data types anticipated from the Earth Observing System (EOS) satellite missions. The HDF-EOS data models include point data, satellite swath data, and gridded data.

As mentioned in the Welcome section, this tutorial will concentrate on the Scientific Data Model as a means of teaching the essentials of HDF. More information on the other data models can be obtained in the various documents (particularly the HDF User's Guide) provided by NCSA through their anonymous ftp server or World Wide Web home page.

Return to top

Which version of HDF should I use?

The most current version or release of HDF is the best place to begin. As of July 1999, the current version of the HDF library is HDF 4.1r3. An extension of the HDF library, called HDF-EOS, is based on this version of HDF and is designed specifically to work with data products anticipated from the upcoming EOS satellite missions. The current tutorial will focus on the releases (i.e., r1, r2 or r3) of HDF4.1. One feature of HDF4 that is important, especially to experienced users of HDF, is the backwards compatability of HDF. That is, HDF4.1r3 is compatabile with earleir versions such as HDF4.1r1 and the data sets that were generated.

It should be noted that an experimental version of HDF, called HDF5, has also recently been developed to address the shortcomings of HDF4. This new HDF library includes simpler source codes,

more consistent and fewer data models, and the ability to work with large data sets (> 2GB). However, although plans call for the HDF-EOS interface to be based on HDF5 at a later date, it is only in the experimental/prototype stage. HDF5 and the associated software will not be covered in this tutorial. The user is directed to NCSA's HDF5 Page for detailed information.

Return to top

Where can I get additional and detailed information on HDF?

The best sites or locations to find detailed information on all aspects of HDF are the NCSA HDF Information Server available through the Internet and the NCSA anonymous ftp server . Inquiries should be sent to hdfhelp@ncsa.uiuc.edu.

The following documents and information can be obtained through the sources mentioned above:

1. HDF 4.1 r3 Reference Manual
2. HDF 4.1 r3 Users Guide
3. HDF Specifications and Developers Guide v3.2 (mainly for the programmers/developers)
4. HDF Newsletters
5. HDF Frequently Asked Questions (FAQ)
6. Java Products
7. Frequently Asked Questions about Java and HDF
8. Release Notes and Man Pages provide information on items that are not covered in the above documents
9. HDF software contributions from non-NCSA users

In addition, users may wish to join the hdfnews mailing list (by emailing ncsalist@ncsa.uiuc.edu and placing subscribe hdfnews in the body of the message) for discussions and updates on HDF.

Return to top

The HDF Library: Software and Hardware

Previous Main Topic Next Main Topic

Return to Main Topics

What is the HDF library and how can it be used?

The HDF library is a collection of software routines that provides two types of interfaces which allow the user to work with HDF files. A brief capsule describing these interfaces is provided below:

Low-level Interface

Application Programming Interfaces (APIs)

Components of the HDF library include the base library, the multi-file library, the jpeg, library, and the gzip library. The most recent versions of HDF also contain a Java Products, which includes a Java HDF Interface (JHI) and a Java-based HDF viewer.

The HDF library also provides a set of command-line utilities that allow the user to work with HDF files outside of the interfaces and within the command level (such as UNIX) of a terminal session. Outside of the HDF library, there is also a large number of browsing and visualization software packages (both free and commercial) that allow the user to look at all types of HDF files. These two methods will be discussed later in the tutorial.

Return to top

Obtaining and installing the HDF library

The HDF library and utilities are public domain software and are freely available, along with documentation, from the NCSA anonymous ftp server. The latest release of the HDF library can be downloaded via ftp from the NCSA Current HDF release. Associated documentation and reference material can also be obtained from NCSA Current HDF Documentation. The source code of the HDF library and utilities are available with each "release" of HDF and can be downloaded free of charge from this ftp site. The files are available in various forms to support users of PCs, Macs, etc...

Unfortunately, the HDF library may not be accessed by every computer platform. The following sections list the platforms and operating systems on which the latest release of HDF has been tested.

NCSA provides a binary distribution for those platforms supported by HDF. For platforms that are not specifically supported, the HDF source code is provided.

HDF Binary Distribution

How do you install HDF on your computer system? Detailed directions for configuring and installing the latest version of HDF can be found in the README and INSTALL files located in the HDF_Current unpacked subdirectory of the NCSA HDF ftp server.

In order to use the HDF library through C and FORTRAN programs, the user's computer must have either a C or FORTRAN library linked with the HDF library.

For those users who wish to work with HDF using Java, Version 2.3 of the HDF Java Products has been released as part of the latest release of the HDF library. Included in these products is the Java HDF Interface (JHI) for the HDF library. The JHI provides an interface to all the functions of the HDF library and may be used by any Java application to work with HDF files. The necessary Java source code can be downloaded from ftp.ncsa.uiuc.edu/HDF/HDF/HDF_Current/java.

These are the only languages which can call HDF routines (more detailed information in "Programming languages supporting the HDF library").

Return to top

Computer platforms supporting the HDF library

The latest version of the HDF library is HDF 4.1 Release 3. Although the list of machines supported by the HDF library increases with every incremental version or release of HDF, it is still not possible to work with HDF files on every single platform or operating system. As of the current release in July 1999, the HDF library is currently supporting the following computer platforms and operating systems:

1. Sun4 (Solaris 2.6, SunOS 4.1.4)
2. SGI-Indy (IRIX v6.5)
3. SGI-Origin (IRIX64 v6.5-64/n32)

4. HP9000/735 (HP-UX 9.03)
5. HP9000/755 (HP-UX B.10.20)
6. Exemplar (HP-UX A.10.01)
7. Cray T90 (CFP, IEEE)
8. Cray C90
9. IBM SP2 (v4.2.1)
10. DEC Alpha/Digital (Unix v4.0)
11. DEC Alpha/OpenVMS (AXP v6.2 and 7.1)
12. VAX Open/VMS (v6.2)
13. IBM PC-Intel Pentium (Solarisx86, Linux (elf), FreeBSD)
14. PowerPC (C only- Mac-OS-7.6))
15. PCs with Windows NT/95
16. Windows NT/95
17. DEC Alpha NT
18. T3E (unicosmk 2.0.4.46)

As of July 1999 and the latest release of the HDF library (4.1r3), the only platforms that support the Java HDF interface (JHI) are:

1. Sun4 (Solaris 2.5)
2. SGI-Indy (IRIX5.3)
3. IBM PC - Intel Pentium (Solarisx86 (2.5) and Linux (elf) 2.0.27)
4. Windows NT/95

Earlier versions or releases of the HDF library can still be used but may not be compatible with the platforms listed above.

Return to top

Programming languages supporting the HDF library

As of the current release of HDF (HDF 4.1r3), the only programming languages which are supported by the HDF library are C and FORTRAN. Although the HDF library code is only written in C, the library provides both a FORTRAN and Java Interface which converts the code to C and allows the user to call the HDF routines. This conversion will automatically take place and requires no action by the user.

Other then the obvious differences between the programming languages, the main difference between using the different languages is the naming convention, or names used for each HDF function. In addition, to use and compile HDF application routines through C programs, an HDF header file (hdf.h) containing standard HDF data type and file access code (i.e. read, write) definitions, declarations and prototypes for the API routines must be called or included (#include "hdf.h") at the beginning of the program. These header files are not permitted in all FORTRAN versions and the needed information must be written into the FORTRAN code (taken from the HDF library file "constants.f" within "hdf.h").

One of the most recent updates to the HDF library is that it now creates free format FORTRAN

include files during the "make" process on UNIX platforms. This allows FORTRAN 90 programs to use HDF include files. The FORTRAN 90 files are designated by the ".f90" file extension.

Another recent update to the HDF library is the inclusion of the Java HDF Interface (JHI) as part of HDF version 4.1r3. The JHI provides an interface to all HDF functions and must be obtained and installed in order to use Java to work with HDF files. Please see Obtaining and installing the HDF library for further details.

Return to top

Compiling the HDF library

The following examples (for UNIX platforms) illustrate the general method of compiling the HDF library using both C and FORTRAN programs. It should be noted that, for C programs, the line "include mfhdf.h" must be included if using the mfhdf library OR the line "include hdf.h" if it is not being used. Also worth noting is that, as indicated below, the following libraries must be specified in the following order- libmfhdf.a, libdf.a, libjpeg.a, and libz.a.

C programs

> cc -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN programs

> f77 -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

Specific examples for various platforms are provided below. If the platform you use is not listed, the general instructions should be followed.

The latest platform related information can be found on the NCSA anonymous ftp server at HDF4.1r3/release_notes/compile.txt.

INSTRUCTIONS FOR SPECIFIC PLATFORMS

Cray C90 or YMP:

C:

> cc -O -s -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

cf77 -O 1 -s -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

Dec Alpha/Digital Unix:

C:

cc -Olimit 2048 -std1 -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

Dec Alpha/OpenVMS AXP:

To compile your programs, prog.c and prog1.for, with the HDF library, mfhdf.olb, df.olb, and libz.olb are required. The libjpeg.olb library is optional.

cc/opt/nodebug/define=(HDF,VMS)/nolist/include=<dir for include> prog.c

fort prog1.for

link/nodebug/notraceback/exec=prog.exe prog.obj, prog1.obj, -<dir for lib>mfhdf/lib -<dir for lib>df/lib, <dir for lib>libjpeg/lib, -<dir for lib>libz/lib, sys$library:vaxcrtl/lib

NOTE: The order of the libraries is important: mfhdf.olb first, followed by df.olb then libjpeg.olb and libz.olb.

Exemplar:

C:

cc -ext -nv -no <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

fc -sfc -72 -o <your program> <your program>.f -I<path for hdf include directory> -

L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FreeBSD:

C:

gcc -ansi -Wall -Wpointer-arith -Wcast-qual -Wcast-align -Wwrite-strings -Wmissing-prototypes -Wnested-externs -pedantic -O2 -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -O -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

HP - UX:

C:

cc -Ae -O -o <your program> <your program>. -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -O -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

IRIX 5.3:

C:

cc -ansi -O -s -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -O -s -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

IRIX 6.x with 64-bit mode:

C:

cc -ansi -64 -mips4 -O -s -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -64 -mips4 -O -s -o <your program> <your program>.f -I<path for hdf include directory>\ -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

IRIX 6.x with n32-bit mode:

C:

cc -ansi -n32 -mips3 -O -s -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -n32 -mips3 -O -s -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

Linux A.OUT And ELF:

C:

gcc -ansi -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN (a.out only):

f77 -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

Solaris:

The -lnsl is necessary in order to include the xdr library.

C:

cc -Xc -xO2 -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz -L/usr/lib -lnsl

FORTRAN:

f77 -O -o <your program> <your program>.f -I<path for hdf include directory>-L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz -L/usr/lib -lnsl

Solaris_x86 (C only):

The -lnsl is necessary in order to include the xdr library.

gcc -ansi -O -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz -L/usr/lib -lnsl

SP2 (AIX):

C:

xlc -qlanglvl=ansi -O -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -O -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

SunOS:

C:

gcc -ansi -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -f -o <your program> <your program>.f -I<path for hdf include directory>-L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

t3d:

C (only):

cc -Tcray-t3d -X1 -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

VAX OpenVMS:

To compile your programs, prog.c and prog1.for, with the HDF library, mfhdf.olb, df.olb, and libz.olb are required. The libjpeg.olb library is optional.

cc/DECC/STANDARD=VAXC/opt/nodebug/define=(HDF,VMS)/nolist/include=-<dir for include> prog.c

fort prog1.for

link/nodebug/notraceback/exec=prog.exe prog.obj, prog1.obj, -<dir for lib>mfhdf/lib -<dir for lib>df/lib, <dir for lib>libjpeg/lib, -<dir for lib>libz/lib, sys$library:deccrtl/lib

NOTE: The order of the libraries is important: mfhdf.olb first,followed by df.olb then libjpeg.olb and libz.olb.

Windows NT / 95:

Using Microsoft Visual C++ version 4.x:

- Under Tools->Options, select the folder, Directories:
- Under "Show directories for", select "Include files".
- Add the following directories:

C:\MSDEV\INCLUDE

C:\MSDEV\MFC\INCLUDE

C:<path to HDF includes>\INCLUDE

- Under "Show directories for", select "Library files":
- Add the following directories:

C:\MSDEV\LIB

C:\MSDEV\MFC\LIB

C:<path to HDF libs>\LIB

- Under Build->Settings, select folder, Link:
- Add the following libraries to the beginning of the list of Object/Library Modules:

  libsrc.lib src.lib jpeg.lib zlib.lib xdr.lib getopt.lib

- The following libraries may (or may not) need to be included:

  kernel32.lib user32.lib gdi32.lib winspool.lib comdig32.lib advapi32lib
  shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib

- Under Build->Settings, select folder C/C++:
- For the Preprocessor Definitions add: INTEL86
- The following were already there: WIN32,_CONSOLE

Return to top

Low-level Interface

The so-called low-level interface provides software that enables the user to work with such file features as memory, error handling, and storage. However, these features and the software are more of interest to the experienced programmer and software developer not the HDF novice or beginner interested in learning to read and write HDF files.

Information on the low-level interface can be found in the documentation listed in Section 2 Where can I get additional and detailed information on HDF?

Return

Application Programming Interfaces (APIs)

Of more use to the average HDF user are the high-level or Application Programming Interfaces (APIs). These APIs are sets of routines that can be called in the user's FORTRAN or C program and which will allow the user to access, read, and write HDF files. There are APIs specifically created for each of the different data types supported by HDF which allow the user to work with HDF files.

Further detail is provided in Section 4 - Methods of Working with HDF Files.

Return

HDF Binary Distribution

On UNIX, VMS, and Windows NT/95, the binary distribution includes the pre-compiled libraries, utilities, include files, man pages, and release notes. The binary distribution on the Macintosh does not include the utilities.

The binaries are located in the following directories on the NCSA ftp server (ftp.ncsa.uiuc.edu):

1. /HDF/HDF_Current/bin- Unix and VMS
2. /HDF/HDF_Current/zip- Windows NT/95
3. /HDF/HDF_Current/hqx- Macintosh

If you uncompressed the binaries for a supported platform, you would (in general) find the following directories:

../bin - pre-compiled utilities

../include - include files

../lib - libraries

../man - man pages

../release_notes - release notes

The compressed source code can be found on the ftp server in /HDF/HDF_Current/tar. An uncompressed version of the source code can be found in /HDF/HDF_Current/unpacked.

To compile and install the HDF libraries from the source code, please read through the READ and INSTALL files in the top directory of the source code. In general, these are the steps you would take to compile and install HDF:

./configure -v

make >& comp.out

make test >& test.out

make install

Return to top

Methods of Working with HDF Files

There are four basic ways or methods of working with (including reading and writing) HDF files. These include two levels of programming interfaces within the HDF library, a set of command line utilities also contained in the HDF library, and a wide range of browsing and visualization software provided by both commercial vendors and non-profit organizations (NCSA, for example). Further detail on each method is given below:

- Low-level interface
- High-level interface (APIs)
- Command line utilities
- HDF browsing and visualization tools/software

Both the command line utilities and the browsing and visualization tools provide easy-to-use methods for HDF non-experts to work with HDF files. As shown above, the use of the command line utilities is rather straight forward. However, neither the command line utilities nor tools provide the user with the flexibility and means of working with the HDF files in such an encompassing fashion as permitted in the High-level APIs. For this reason, as well as the fact that information and directions regarding the use of the HDF tools are better provided by the Internet sites linked above, the following sections of the tutorial will mainly concentrate on using the APIs to work with HDF files.

Previous Main Topic Next Main Topic

Return to Main Topics

Low-level interface

The low-level interface is mainly reserved for expert HDF programmers and software developers who are interested in not only reading and writing HDF files, but also such features as error handling, memory management, and storage. A lot of the features in this interface are unnecessary for the novice HDF user. Another drawback is that routines/operation callable through this interface are only available in C and not FORTRAN.

Return to top

High-level interface (APIs)

In this interface, Application Programming Interfaces (APIs) are specifically tailored for each type of data (Images, Scientific Data arrays, etc.) supported by the HDF library. These APIs are callable routines which will allow the user to access, read and write HDF files specifically for the type of data they are interested in. Although it is necessary for the call of these APIs and associated routines to occur in either a C or FORTRAN program, the programming is usually limited to a set of call statements that access, open, operate (read, write, etc.), and terminate. All of the rest is taken care of

- **SW API (SW/sw):** The SW API is used for storing, retrieving, and manipulating time-ordered data sets such as satellite swath data. The SW API is part of the HDF-EOS sub-library.

Return

List and description of command line utilities

HDF Command line utilities can be executed at the command level (prompt) similar to UNIX. The following is a list of some of the command-line utilities available in the HDF library:

1. hdp - displays contents and data objects within an HDF file
2. hdf24to8 - converts 24-bit raster images to HDF 8-bit images
3. hdf8to24 - converts 8-bit raster images to HDF 24-bit images
4. hdfcomp - re-compresses an 8-bit raster HDF file
5. hdfls - lists basic information about an HDF file
6. hdfpack - compacts an HDF file
7. hdfunpac - unpacks an HDF file
8. hdftopal - extracts a pallete from an HDF file
9. hdftor8 - extracts 8-bit raster images and palettes from an HDF file
10. hdfed - HDF file editor
11. paltohdf - converts a raw palette to HDF
12. r8tohdf - converts 8-bit raster image to HDF
13. ristosds - converts a series of raster image HDF files into an HDF file
14. vshow - dumps out vsets from an HDF file
15. jpeg2hdf - converts jpeg images to HDF raster images
16. hdf2jpeg - converts HDF raster images to jpeg images
17. fp2hdf - converts floating point data to HDF floating point format and to HDF 8-bit raster image format
18. vmake - create Vset structures from ASCII text

The hdp command line utility is a very helpful operator, especially for the average HDF user. HDP can list the contents of HDF files at various levels and with different details. It can also dump the data of one or more specific objects in the file.

Return

Publicly Available Software

Freely available software for viewing and browsing HDF files have been developed by both NCSA and various other institutes, science or data centers, and businesses. We have broken these tools down into three categories:

- Current NCSA Tools
- Older NCSA Tools (not updated to run with latest version of HDF)
- Non-NCSA Tools

Return

Current NCSA Tools

The following are the most current and commonly used tools developed by NCSA for viewing and browsing all types of HDF files:

1. The NCSA Java-based HDF Viewer (JHV) - Java based tool that allows the user to view the contents of an HDF file.
2. The HDF WWW Scientific Data Browser - an interface program that reads HDF files by accessing teh HDF library and can visualize or format the data (in HTML) on the web.
3. The Java HDF Server (JHS) - The java based program that calls the HDF library through the Java interface and can access remote HDF files.

Older NCSA Tools

Although not updated to run with the current release of HDF (HDF 4.1r3), the following tools may still be used to work with HDF files. All of these tools are available from the NCSA anonymous ftp server

1. NCSA Collage - Collaborative visualization program
2. NCSA Polyview - Visualization and analysis of HDF files
3. NCSA Reformat - Converts to and from HDF files
4. NCSA X DataSlice - Manipulates 3-D HDF images

Non-NCSA Tools

The following tools have been developed independently from NCSA, but are still available in the public domain:

1. The Data and Dimensions Interface (DDI) - Can extract, read, write and visualize large data sets in HDF format.
2. Envision - Interactive system which provides for the management and visualization of large data sets in HDF format.
3. HDF Browser - Created by Fortner Research to provide point-and-click access to data stored in HDF. This includes viewing the data stored in arrays, images, etc.. and editing HDF files.
4. hdfv - An HDF read-only interface that is an HDF viewer with a GUI. Only supports vgroup/Vdata data types.
5. LinkWinds - A visual data analysis and exploration system designed to rapidly and interactively investigate large multivariate data sets (including HDF and HDF-EOS format).
6. SHARP - A viewer for MODIS Airborne Simulator (MAS) HDF data
7. ScaiAN - Scientific visualization and animation package.
8. VCS - Facilitates the selection, manipulation and display of scientific data. Supports the HDF format for both reading and writing.
9. EOSView - An HDF file verification tool that allows the display of most HDF and HDF-EOS data types.
10. The Data and Information Access Link (DIAL) - A server which provides tools for the searching, browsing, and visualizing of HDF and HDF-EOS files through the WWW.
11. HDFLook - A viewer used to access and view HDF and HDF-EOS files, particularly raster images and scientific data sets.

12. IRI/LDEO - A climate data library that helps in the writing of HDF files and the management of data sets.
13. Webwinds - A platform independent system written in java that acts as an interactive visualization tool for data in HDF and HDF-EOS format.
14. view_hdf - A visualization tool developed by NASA LARC that provides for the viewing, plotting, and manipulation of HDF datasets.

Return

Commercial Software

Below is a partial list of some of the more powerful and more commonly used software packages for working with HDF files:

1. AVS5/AVSExpress - Can read and write files in HDF format. Also includes a suite of data visualization and analysis techniques/tools (3-D visualization, plots, etc...).
2. Data Explorer - General purpose software package for data visualization and analysis. The data may be imported from HDF format.
3. IDL - A software package for the analysis and visualization of data. Includes advanced image processing, interactive 2-d and 3-D graphics, and flexible date input/output.
4. Noesys - A desktop software program specifically designed to easily access, view, analyze and archive data in the HDF format.
5. Plot - A package that can read, analyze and plot HDF data sets of column data using Windows, Macintosh and UNIX.
6. HDF Explorer - A visualization program that reads and views data sets in HDF format

Return

Contributed Software

In addition to the above-mentioned software, also available from the NCSA anonymous ftp server is a collection of software routines and utilities developed by HDF users who wish to share their knowledge and work with the HDF community. These software can be found in the directory pub/hdf/contrib/ of the NCSA ftp server. Most of these "contributed" routines were developed with specific platforms and operating systems in mind.

Below are a few examples:

- **readDF** - reads HDF files into IRIS Explorer
- **fits2hdf** - converts FITS files (another format) into HDF
- **iristohdf** - converts SGI image format to HDF format
- **hdfxdis** - directly displays HDF image on an X-server

These routines together with the name and address of the developer are free and publicly available to all interested users of HDF.

Return

SD API

The SD (Scientific Data) API is a collection of callable (from C or FORTRAN programs) routines which will allow the user to, among other operations, create, write, and read HDF files containing multi-dimensional arrays of scientific data. In subsequent sections, we will show how the SD API can be used for reading and writing HDF data sets. For a complete listing of all the operations permitted in the SD API, please see the HDF 4.1r3 User's Guide. As will be demonstrated shortly, FORTRAN (C) routines in the SD API begin with the prefix "sf" ("SD"). Data within a scientific data set may be of the floating real or integer type. In HDF, and in the SD API, a scientific data set (or SDS) must consist of a multi-dimensional array (called a SDS array), together with information on data type and dimension record. The SD API allows the user to work simultaneously with more than one multi-dimensional scientific data set (SDS) while the DFSD API is restricted to one multi-dimensional array.

- SDS Array
- Data Type
- Dimensions
- Optional information

Previous Main Topic Next Main Topic

Return to Main Topics

SDS Array

The SDS array is the actual data itself, an n-dimensional array which contains the floating point or integer values. Each SDS array has an SDS name (series of alphanumeric characters) that can either be assigned by the calling statement with the FORTRAN or C program or automatically assigned by the HDF library when the new data set (if writing) is created.

Return to top

Data Type

The SD API supports the following data types:

- 32-bit floating point
- 16-bit floating point
- 8-bit signed integers
- 16-bit signed integers
- 32-bit signed integers
- 8-bit unsigned integers
- 16-bit unsigned integers
- 32-bit unsigned integers

- Variable bit integers and floating point decimal values

As described later, the data type is defined in the accessing/creating function call statements within the C and FORTRAN programs.

Return to top

Dimensions

The dimensions of an SDS array identify the shape and size of the array in question. This includes the rank of the dimensions, which in HDF speak refers to the number of dimensions. One innovative feature of HDF is that one, and only one, dimension of an SDS array may be of unlimited size and referred to as an unlimited dimension.

Return to top

Optional information

When writing or creating an HDF file, the user may also wish to include information regarding the data set or array. This must be done in the calling functions of the C or FORTRAN programs.

Attributes, either predefined by NCSA or user-defined, are text strings which provide metadata about the file, data set, or dimension of interest. This includes information on what is in the file or individual SDS arrays, and how the maker of the file/data intends for the data to be used or viewed. Like most of the other routines mentioned above, attributes are defined in the function calls of the program. Attributes are further covered in section 6.

Return to top

Attributes and Metadata

The HDF library allows for several ways for the user to provide metadata (data about data) information for the HDF file, data set or image to be written or read. This information is not a requirement for HDF files. The most commonly used method or routine within the HDF library for providing metadata are "Attributes" or text- strings which describe the HDF file, data set (SDS array) or dimensions. There are two types of attributes used in HDF which can be defined in the user's calling program:

- User-defined attributes
- Predefined attributes

Both the predefined and user-defined attributes may be accessed using the general attribute routines for user-defined attributes provided by the HDF library. On the other hand, the predefined attributes may only be accessed using the routines specifically tailored for the predefined attributes (see above). As a result, in later sections, we will focus on using the general attribute routines developed for user-defined attributes.

Previous Main Topic Next Main Topic

Return to Main Topics

User-defined attributes

User-defined attributes are optional information that can be given and attached to HDF files, scientific data sets, and dimensions (only in the SD API). They are referred to, respectively, as file attributes, array attributes, and dimension attributes. These attributes are at the discretion of, and to be defined by, the user.

The SD interface uses the same functions to access all of the three types of attributes, with the difference being the use and definition of the different identifiers (i.e., file ids for file attributes, SDS ids for array attributes, and dimension ids for dimension attributes). After the proper identifier is obtained, the user can then create and define his attribute (labels, formats, coordinate system, etc.). The attributes in the GR interface work in a similar fashion with indentifiers provided for both the interface or the image in question.

More on user-defined attributes and how to define them is provided in Section 7: Writing an HDF File.

Return to top

Predefined attributes

Predefined attributes are attributes that use previously defined or reserved labels and data types. While the user-defined attributes must be defined by the user, the predefined attributes need not be defined and are already understood by the HDF library. However, in the SD API, predefined attributes can only be assigned to scientific data sets (SDS) and dimensions (not files, like is possible with user-defined attributes). In the GR API, there is only one predefined attribute, FILL_ATTR, which fills the "empty" data of an image with default values.

There are seven main predefined attributes:

For labels:	long_name
For units:	units
For formats:	format
For coordinate systems:	cordsys
For Value ranges:	valid_range
For Fill values:	FILL_ATTR, _FILL_VALUE
For Calibration:	scale_factor, scale_factor_err, add_offset, add_offset_err, calibrated_nt

The predefined attributes can be accessed by the SD interface in the same general fashion as the user-defined attributes or by using routines developed specifically for the predefined attributes. The ""general"" attribute routines are recommended in most cases.

Return to top

Writing a SDS to an HDF File

The following sections detail how a user may utilize the HDF library and the SD API within a computer program to write a data file in HDF. As a teaching tool, this tutorial will concentrate on using the FORTRAN programming language. However, examples of the appropriate C code will also be given for certain steps.

Previous Main Topic Next Main Topic

Return to Main Topics

Does the current version of HDF support your computer platform?

As outlined in Section 3, the HDF library can not be run on just any available computer platform or operating system. Before downloading the HDF library software, the user should make sure that the current release of HDF supports his/her computer and operating system. Otherwise, the user will be unable to work with the HDF library and files. There is also a possibility that previous releases of HDF may support the Users computer platform while the latest version does not. In this event, the user may wish to obtain the earlier software.

Return to top

Downloading and Installing of the HDF library

The HDF library and software is public domain software and available free to all users. The library and code can be downloaded from the (NCSA anonymous ftp server). Directions on how to install the HDF library can also be found at this location.

Return to top

Are all libraries and programs properly linked and compiled?

In order to run the HDF software, the library and the needed application routines and programs must

first be properly compiled and linked. As of the current release of HDF (4.1r3), four separate libraries must be compiled and linked. These are the libmfhdf.a, libdf.a, libjpeg.a, and libz.a libraries. Provided below are examples of the command(s) that can be used for this action. It must be noted that the order in which the libraries are linked is important and should not vary from the order shown below:

For C programs:

1. cc -o <your program> <your program>.c \
2. -I<pathf for hdf include directory> \
3. -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

For FORTRAN programs:

1. f77 -o <your program> <your program>.f \
2. -I<path for hdf include directory> \
3. -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

For the various commands needed to link and compile the HDF library on each individual platform, please see Section 3 "Compiling the HDF library".

Return to top

Writing a short program to write a scientific data set in HDF

- Select a programming language
- Make sure all include files are in place
- Make all variable and parameter declarations
- Open file containing existing non-HDF data set and store in array
- Initialize access to the SD interface and open new HDF file
- Define characteristics of new HDF data set(s)
- Write existing data set/array to a new data array in a new HDF file
- Optional operation: Provide metadata for HDF files or data sets
- Terminate / close access to all files, data sets, and APIs
- Execute program

Return to top

Select a programming language

As mentioned previously, the HDF library and programs can only be run by using either the C or FORTRAN programming language. This choice is up to the user depending on availability and the language he or she feels most familiar and comfortable with. All SD API routines which allow the user to work with scientific data sets (SDS) either have the "sf" prefix (FORTRAN) or the "SD" prefix (C). Examples of the routines used to open, create, read, write, etc. SDS are given in the following sections.

Return

Make sure all include files are in place

In section 3 - The HDF Library: Software and Hardware , it was noted that a series of standard HDF definitions and declarations of file access codes (i.e. read, write, etc.) and data types (i.e. integer, character) must be included within the programs that the user writes to utilize the various application routines. In the C programs, this is accomplished simply by adding the line #include "hdf.h" at the beginning of the program. This line effectively includes all the needed constants and definitions from the HDF software. When writing FORTRAN programs, this may also be done by simply adding an include statement that brings in only the needed definitions and declarations (constants.f) from the hdf.h header file. This is done by the following code: "include constants.f". However, all FORTRAN compilers (particularly the older ones) do not support the use of include statements. In this event, the user must type in/declare all the constants and definitions found in the constants.f file. It is advised that all declarations, whether through include statements or not, should be done at the beginning of the program.

Example:

FORTRAN:

```
      C DFACC_RDONLY is defined in hdf.h
      C if not available for FORTRAN then add
        Parameter (DFACC_RDONLY=1)
```

C:

```
        #include "hdf.h"
        main()
```

Return

Make all variable and parameter declarations

As with any program, the scientist/user should declare and initialize all variables and parameters at the beginning of the program. This includes all variables and arguments that will be used by the HDF commands to follow. The variable and parameter declarations needed for each call will be provided in the example boxes of the individual steps. These statements always belong at the top of the program.

Return

Open file containing existing non-HDF data set and store in array

Before writing any data into HDF, the actual data first has to be accessed within the program. As is normally done in non-HDF applications, the file containing the data that the user wishes to convert into HDF must first be opened. After opening the file, the user reads and stores the data into a multi-dimensional array that can be accessed by the HDF commands.

For the purpose of this tutorial, the non-HDF data set will be read from an existing file called wind.dat into a multi-dimensional real array called rwind(XL,YL) where XL= 30 and YL = 30.

Example:

C:

```
main() {

  FILE *infile;

  const int XL = 30, YL = 30;
  int i, j;
  float rwind[XL][YL];
  infile = fopen("wind.dat", "r");
  for(i=0; i<XL; i++)
    for(j=0; j<XL; j++)
      fscanf(infile, "%f", rwind[i][j]);

}
```

FORTRAN:

```
      real rwind(30,30)
      XL = 30
      YL = 30
      Open(unit=15, file='wind.dat',form='formatted')
      Do I=1,XL
       Do j=1,YL
        Read(15,25)rwind(I,J)
       Enddo
      Enddo
```

Return

Initialize access to the SD interface and open new HDF file

The first real HDF programming step actually accomplishes 2 things:

- Creates and opens a new HDF file
- Initializes and opens the SD interface.

This is done by the following command:

```
sd_id = sfstart(filename, access_mode) (FORTRAN)
```

or

```
sd_id = SDstart(filename, access_mode); ( C )
```

where

```
sd_id = HDF file id returned by the sfstart/SDstart command
filename = the name of the new HDF file (character string)
access_mode = Type of access required for this file
```

All available options for the access-mode argument are defined in the hdf.h header file mentioned previously and need only to be identified for all C and most FORTRAN operations. All options begin with the prefix "DFACC_" and include:

```
DFACC_CREATE (File Creation Access)
DFACC_RDONLY (Read Access)
DFACC_RDWR (Read and Write Access)
```

As mentioned previously, these definitions are stated in the hdf.h header file.

In the event that the user's FORTRAN compiler can not handle include statements such as those found in the hdf.h header file, the DFACC_ variable must be defined, along with its assigned value, at the beginning of the program. This is done by a code line such as:

```
parameter (DFACC_RDONLY = 1) (For FORTRAN only)
```

For the purpose of this tutorial, the new HDF file will be called wind.hdf.

Example:

FORTRAN:

```
integer*4 sd_id
integer sfstart
parameter (DFACC_CREATE = 4)
sd_id = sfstart(wind.hdf, DFACC_CREATE)
```

C:

```
#include "hdf.h"
/* Includes all the access_mode definintions */
int32 sd_id;
sd_id = SDstart(wind.hdf, DFACC_CREATE);
```

Return

Define characteristics of new HDF data set(s)

After initializing the SD interface and opening and assigning a file id (sd_id) to the HDF file to be used, the next step is to define a new HDF Scientific Data Set (SDS) to which the existing non-HDF data will be written. This is done by the following command:

```
sds id = sfcreate (sd id, name, number type, rank, dim sizes) (FORTRAN)
```

or

```
sds_id = SDcreate (sd_id, name, number_type, rank, dim_sizes); ( C )
```

It should be noted that sfselect/SDselect may also be used to write to a previously defined HDF data set.

where

```
        sds_id = HDF SDS array id returned by the sfcreate/SDcreate command
        sd_id = the new HDF file id created in the previous step (sfstart/SDstart)
name = name of new SDS (in ASCII character string)
        number_type = data type of data set
```

This argument always takes the form of DFNT_X, where X is the data type to be used. A list of all the data types supported by the API can be found in the HDF User's Guide. For most of the data types, 8,16,32 and 64-bit types are supported. A few of the available options are provided below:

HDF Data Type	Description
DFNT_FLOAT32	32 bit floating point real
DFNT_DOUBLE	double precision reals
DFNT_CHAR8	8 bit character type
DFNT_UCHAR8	8 bit unsigned character type
DFNT_INT16	16 bit integer type
DFNT_UINT16	16 bit unsigned integer type
DFNT_NINT16	16 bit native integer
DFNT_NUINT16	16 bit native unsigned integer
DFNT_NFLOAT32	32 bit native floating point real

Similar to the DFACC_ argument, all data types are defined in hdf.h. Once again, for FORTRAN compilers unable to access these include files, the DFNT_ argument, and its' assigned value, must be defined at the beginning of the program using code like this:

```
parameter (DFNT_INT16 = 22) (taken from constants.f within the hdf.h file)
```

rank = number of dimensions in array to be written (integer)

This value is best specified at the beginning of the program along with the other various declarations with a simple line of code:

rank = 2, 3,....

dim_sizes = An array defining the size of each dimension of the data array (integer)

As with the "rank" argument, this variable is best specified with the other variable declarations at the top of the program. In FORTRAN, an example for a 2-D, 30 X 30 array would be:

```
dimsizes(1) = 30 (FORTRAN)
dimsizes (2) = 30
```

or

```
dimsizes[0] = 30; ( C )
dimsizes[1] = 30;
```

EXAMPLE: For an existing data set to be written as a 2-D array of 30 (x direction) by 30(y direction), and as an 8-bit integer type, the following commands need to be used:

```
rank = 2          (FORTRAN)
dimsizes(1) = 30
dimsizes(2) = 30
sds_id = sfcreate(sd_id, newarray_1, DFNT_INT8, rank, dimsizes)
```

or

```
rank = 2;         ( C )
dimsizes[0] = 30;
dimsizes[1] = 30;
sds_id = SDcreate(sd_id, "newarray_1", DFNT_INT8, rank, dimsizes);
```

Example:

FORTRAN:

```
        integer*4 DFNT_INT16
        integer sds_id, rank
        integer dims(2), sfcreate
        rank = 2
        XL = 30
        YL = 30
        dims(1) = XL
        dims(2) = YL
        sds_id = sfcreate(sd_id, winds, DFNT_INT16, rank, dims)
```

C:

```
        int32 sds_id;
        int32 dims[2], rank;
        rank = 2;
        XL = 30;
        YL = 30;
        dims[0] = YL;
        dims[1] = XL;
        sds_id = SDcreate(sd_id, winds, DFNT_INT16, rank, dims);
```

Return

Write existing data set/array to a new data array in a new HDF file

After initializing the API and defining the new HDF file and new HDF SDS to be written to, the next step is to actually write the existing non-HDF data into the HDF file by using the SDwritedata (sfwdata) command. This command is used to write either all or part of the existing n-dimensional data set (termed a "slab") into the sds_id array with the same number of dimensions. In addition, the size of each dimension of the data "slab" must be the same or smaller then the corresponding dimension of the sds_id. The SDwritedata/sfwdata command is used in the following fashion:

```
ret=sfwdata (sds_id, start, stride, edge, data) (FORTRAN)
```

or

```
ret=SDwritedata (sds_id, start, stride, edge, data); ( C )
```

It should be noted that there are two versions of the write routine in FORTRAN, "sfwdata" is used for numeric data while "sfwcdata" is used for writing character data

where

```
        sds_id = the SDS id (identifier) determined and returned by using SDcreate
start = An array which identifies where in the SDS that the writing will begin
```

The start array identifies the location or position in the SDS where the writing of the data "slab" will begin. This array must have the same number of dimensions (rank) as the SDS and can not be larger (in each dimension) then the SDS array. The declaration of the start variables can be done at the top of the program or just preceding the call of the sfwdata (SDwritedata) command. As an example, to write the existing data set to the beginning of a new 2-dimensional SDS the following must be specified:

```
start(1) = 0 (FORTRAN)
start(2) = 0
```

Or

```
start[0] = 0; ( C )
start[1] = 0;
```

If the user wishes to begin writing the data at a location other then the beginning of the new data set, say at a first dimension (X) of 15, the declarations would be:

```
start(1) = 15 (FORTRAN)
start(2) = 0
```

Or

```
start[0] = 15; ( C )
start[1] = 0;
```

stride = An array specifying the interval between written values in each dimension

The stride argument specifies, for each dimension, the interval between consecutive written values of the data set. In other words, how many array locations are skipped with each writing of the data? Like the start array, the stride argument is predefined before calling the sfwdata (SDwritedata) command, either directly before the call or at the top of the program.

If the user does not wish to skip any array locations in a new 2-dimensional SDS, the following is to be declared:

```
stride(1) = 1 (FORTRAN)
stride(2) = 1
```

or

```
stride(0) = 1; ( C )
stride(1) = 1;
```

However, if the user wishes to skip every other X (dimension 1) location, the following would be used:

```
stride(1) = 2 (FORTRAN)
stride(2) = 1
```

Or

```
stride(0) = 2; ( C )
stride(1) = 1;
```

edge = An array defining the number of data values to be written in each dimension

The edge array defines the number of data values/elements that will be written along each dimension of the multi-dimensional SDS array. In plain terms, this argument defines the size of the data slab (all or part of the data) to be written to the new SDS array and each dimension.

edge must be specified for each dimension of the data set and SDS array, and can not be larger then the entire length of the newly defined (from sfcreate) array it is being written to.

The edge is affected by the stride. If stride = 2, then the edge will need to be divided by two, because it will be writing to every other location along a dimension.

Similar to stride and start, the edge argument needs to be defined prior to the calling of the sfwdata (SDwritedata) command, whether it be at the top of the program or directly before the routine call.

As an example, most often, the user will wish to write the entire non-HDF data set into a new array that starts from the beginning and does not contain any missing data or blanks. For a 2-dimensional array of 30X30, read and stored into the data array "rwind", this can be done, in FORTRAN, by:

```
start(1) = 0
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 30
edge(2) = 30
retn = sfwdata(sds_id, start, stride, edges, rwind)
```

Or in C by:

```
Start[0] = 0;
Start[1] = 0;
Stride[0] = 1;
Stride[1] = 1;
Edge[0] = 30;
Edge[1] = 30;
retn = SDwritedata(sds_id, start, stride, edges, rwind);
```

data = The array or buffer of data to be written

The file containing this data should be opened at the beginning of the program and the data read in and stored into the necessary arrays before beginning the HDF operations.

Example:

FORTRAN:

```
        integer start(2), edges(2), stride(2)
        integer retn, XL, YL
        integer sfwdata
c     Define the location, pattern and size of data set that
c     will be written to.
        XL = 30
        YL = 30
        start(1) = 0
        start(2) = 0
        edge(1) = XL
        edge(2) = YL
        stride(1) = 1
        stride(2) = 1
c     write the data
        retn = sfwdata(sds_id, start, stride, edges, rwind)

C:      int32 retn;
        int32 start[2], edges[2], stride[2];
        XL = 30;
        YL = 30;
        /*Define the location, pattern and size of the dataset*/
        For (i=0; i<rank; i++) {
        start[i] = 0;
```

```
edge[i] = dims[i];
edge(1) = 30;
/* Write the stored data to "newarray". The 5th argument must be explicitly
 a generic pointer to conform to the API definition for SDwritedata */
retn = SDwritedata(sds_id, start, NULL, edges, (VCIDP)newarray);
```

Return

Optional operation: Provide metadata for HDF files or data sets

Using the general attribute routines for user-defined attributes described in section 6, attributes can be written and attached to the file itself, the data set, and the dimension in question. This is not required, but up to the choice of the user.

After opening the file and obtaining the file id (sd_id) using the sfstart/SDstart command, the following can be done

1) FILE ATTRIBUTES:

To assign attributes to a file, the following command is used:

```
SDsetattr (sd_id,attr_name, data_type, count, value);        ( C )
sfsnatt(sd_id, attr_name, data_type, count, value)          (FORTRAN)
```

There are two FORTRAN versions of the routine, sfsnatt writes numeric attribute data while sfcatt writes character attribute data.

where

```
sd_id= file identifier
attr_name = ASCII string containing the name of the attribute (i.e., "file conten
data_type = data type of attribute values (i.e., DFNT_INT32)
count = total number values/characters in the attribute
value = text string or label
```

2) ARRAY ATTRIBUTES

After each data set identifier (sds_id) is obtained through the SDselect/sfselect command, the following is used:

```
SDsetattr (sds_id, attr_name, data_type, count, value);        ( C )
sfsnatt(sds id, attr name, data type, count, value)          (FORTRAN)
```

where

sds_id= data set identifier

rest as above

3) DIMENSION ATTRIBUTES

After getting the identifier for a dimension using the sfdimid/SDgetdimid command, the following is used:

```
SDsetattr (dim_id, attr_name, data_type, count, value);          ( C )
sfsnatt (dim_id, attr_name, data_type, count, value)           (FORTRAN)
```

where

dim_id= Dimension identifier

rest as above

4) CLOSING ATTRIBUTES

After setting/writing the attributes, the user must terminate access to the data array (using the SDendaccess/sfendacc commands) and the file and SD interface (using the SDend/sfend commands).

Example:

1) FILE ATTRIBUTES:

FORTRAN:

```
sd_id = sfstart("wind.hdf", DFACC_RDWR)
retn = sfsattr(sd_id, "Contents of file", DFNT_CHAR8, 16, "horizontal winds
```

C:

```
sd_id=SDstart ("wind.hdf", DFACC_RDWR);
retn= SDsetattr (sd_id, "Contents of file", DFNT_CHAR8, 16, "horizontal w
```

2) ARRAY ATTRIBUTES

FORTRAN:

```
sds_id=sfselect (sd_id, 0)
retn = sfsattr(sds_id, "format", DFNT_INT32, 4, "F8.2")
```

C:

```
sds_id=SDselect(sd_id, 0);
retn= SDsetattr (sds_id, "format", DFNT_INT32, 4, "F8.2");
```

3) DIMENSION ATTRIBUTES

FORTRAN:

```
dim_id=sfdimid (sds_id, 0)
retn = sfsattr(dim_id, "dim_metric", DFNT_CHAR8, 10, "meters/sec")
```

C:

```
dim_id=SDgetdimid (sds_id,0);
retn= SDsetattr (dim_id, "dim_metric", DFNT_CHAR8, 10, "meters/sec");
```

Return

Terminate / close access to all files, data sets, and APIs

After writing the data to the new SDS array within the new HDF file, it is necessary to terminate or close access to the new data set in order to prevent any possible loss of data. This is done by the following:

```
retn = sfendacc(sds_id) (FORTRAN)
```

or

```
retn = SDendaccess(sds_id); ( C )
```

In addition, the API called within the program must also be closed to prevent any data loss:

```
retn = sfend(sd_id) (FORTRAN)
```

or

```
retn = SDend(sd_id); ( C )
```

Example:

```
FORTRAN:        integer sfendacc, sfend
                retn = sfendacc(sds_id)
```

```
                retn = sfend(sd_id)
C:
                retn = SDendaccess(sds_id);
                retn = SDend(sd_id);
```

Return

Execute program

Execute like a normal FORTRAN or C program.

Return

Obtaining Information on Existing HDF Files

As mentioned previously, a single HDF file may contain more than one scientific data set (or images, tables, etc.). Within the SD interface (and other interfaces for the various data types), there are routines that can be called within short programs, C or FORTRAN, which help the user do the following:

- Determine the contents of an HDF file
- Obtain information on individual data sets or images

Previous Main Topic Next Main Topic

Return to Main Topics

Determine the contents of an HDF file

Before reading an HDF file, it might be necessary for the user to determine the number of data sets within the file, and the attributes of the file itself.

After initializing and accessing the Application interface (in this case, the SD and GR interfaces for, respectively, scientific data sets and images (with associated paettes), this can be done using the following statements:

```
        SDfileinfo(sd_id, n_datasets, n_file_attr); (C)
        GRfileinfo(gr_id,n_images,nfile_attr)
and
        sffinfo(sd_id, n_datasets, n_file_attr)      (FORTRAN)
        mgfinfo(gr_id,n_images,n_file_attr)
```

where

```
        sd_id= file id number
        gr_id= GR interface identifier
        n_datasets= Number of data sets within the file n_file_attr= number of file
        n_images= number of images within the file
```

Return to top

Obtain information on individual data sets

Before reading a particular data set or image from an HDF file, the user may need to know the rank, dimension sizes, data type, and number of attributes of the data array.

After the user has initiated and accessed the interface (for example, the GR interface for images and the SD interface for data arrays) and selected the data set by using the sfselect/SDselect (data set) or mgselct/GRselect (image) in a short FORTRAN (C) program, this information can be retrieved using the following calls:

```
        SDgetinfo (sds_id, name, rank, dim_sizes, num_type, attributes);        (C)
GRgetinfo(ri_id,name,n_comps,data_type,interlace_mode,dim_sizes,n_attrs)
and
        sfginfo (sds_id, name, rank, dim_sizes, num_type, attributes)           (FC
mgginf(ri_id,name,n_comps,data_type,interlace_mode,dim_sizes,n_attrs)
```

where

```
sds_id = data set id number
ri_id = raster image id number
name = name of corresponding data set
rank = rank of corresponding data set
dim_sizes = dimensions of corresponding data set
num_type = data type of corresponding data set
data_type = data type of corresponding image
attributes = number of attributes of corresponding data set
n_comps = number of components
interlace_mode = interlacing mode of data
n_attrs = number of sttributes
```

Return to top

Reading a Scientific Data Set from an HDF File

The following sections detail how a user may utilize the HDF library and the SD API within a computer program to read a scientific data set from an HDF file. In this section, the tutorial will concentrate on using the FORTRAN programming language and the SD API. However, examples of the appropriate C code will also be given for certain steps. For the purpose of this tutorial, we are choosing the example of reading an entire data array that is the first and only data set in the HDF file. Similar to writing an HDF file, the user should follow these simple steps:

- Does the current version of HDF support your computer platform and operating system?
- Downloading and Installing the HDF library
- Are all libraries and programs properly linked and compiled?
- Writing a short program to read an HDF data set

Previous Main Topic Next Main Topic

Return to Main Topics

Does the current version of HDF support your computer platform and operating system?

As outlined in Section 3, the HDF library can not be run on just any available computer platform or operating system. Before downloading the HDF library software, the user should make sure that the current release of HDF supports his/her computer and operating system. Otherwise, the user will be unable to work with the HDF library and files. There is also a possibility that previous releases of HDF may support the Users computer platform while the latest version does not. In this event, the user may wish to obtain the earlier software.

Return to top

Downloading and Installing the HDF library

The HDF library and software is public domain software and available free to all users. The library and code can be downloaded from the NCSA anonymous ftp server (ftp://ftp.ncsa.uiuc.edu/). Directions on how to install the HDF library can also be found at this location.

Return to top

Are all libraries and programs properly linked and compiled?

In order to eventually run the HDF software, the library and the needed application routines and programs must first be properly compiled and linked. As of the current release of HDF (4.1r1), four separate libraries must be compiled and linked. These are the libmfhdf.a, libdf.a, libjpeg.a, and libz.a libraries. Provided below are examples of the command(s) that can be used for this action. It must be noted that the order in which the libraries are linked is important and should not vary from the order shown below:

For C programs:

```
cc  -o <your program> <your program>.c \
    -I<pathf for hdf include directory>\
    -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz
```

For FORTRAN programs:

```
f77 -o <your program> <your program>.f  \
    -I<path for hdf include directory>\
    -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz
```

For the various commands needed to link and compile the HDF library on each individual platform, please see Section 3:Compiling the HDF Library.

Return to top

Writing a short program to read an HDF data set

- Select a programming language
- Make sure all include files are in place
- Make all variables and parameter declarations
- Initialize access to the SD interface and open HDF file
- Select data set to be read from the HDF file
- Read an existing data set/array
- Write non-HDF data to a file
- Optional operation: Get and Read Metadata
- Terminate/Close access to all files, data sets, and APIs
- Execute program

Return to top

Select a programming language

As mentioned previously, the HDF library and programs can only be run by using either the C or FORTRAN programming language. This choice is up to the user depending on availability and the

language he or she feels most familiar and comfortable with.

Return

Make sure all include files are in place

Earlier, it was noted that a series of standard HDF definitions and declarations of file access codes (i.e. read, write, etc.) and data types (i.e. integer, character) must be included within the programs that the user writes to utilize the various application routines. In the C programs, this is accomplished simply by adding the line #include "hdf.h" at the beginning of the program. This line effectively includes all the needed constants and definitions from the HDF software. When writing FORTRAN programs, this may also be done by simply adding an include statement that brings in only the needed definitions and declarations (constants.f) from the hdf.h header file. This is done by the following code: "include constants.f". However, all FORTRAN compilers (particularly the older ones) do not support the use of include statements. In this event, the user must type in/declare all the constants and definitions found in the constants.f file. It is advised that all declarations, whether through Include statements or not, should be done at the beginning of the program.

Return

Make all variables and parameter declarations

As with any program, the scientist/user should declare and initialize all variables and parameters at the beginning of the program. This includes all variables and arguments that will be used by the HDF commands to follow. The variable and parameter declarations needed for each call will be provided in the example boxes of the individual steps. These statements always belong at the top of the program.

Return

Initialize access to the SD interface and open HDF file

The first real HDF programming step actually accomplishes two things:

- Opens the existing HDF file
- Initializes and opens the SD interface.

This is done by the following command:

```
sd_id = sfstart(filename, access_mode)   (FORTRAN)
```

or

```
sd id = SDstart(filename, access mode)   ( C )
```

where

```
sd_id = HDF file id returned by the sfstart/SDstart command
filename = the name of the existing HDF file (character string)
access_mode = Type of access required for this file
```

All available options for the access-mode argument are defined in the hdf.h header file mentioned previously and need only to be identified for all C and most FORTRAN operations. All options begin with the prefix "DFACC_" and include:

```
DFACC_CREATE (File Creation Access)
DFACC_RDONLY (Read Access)
DFACC_RDWR (Read and Write Access)
```

As mentioned previously, these definitions are stated in the hdf.h header file.

In the event that the user's FORTRAN compiler can not handle include statements with the header file (hdf.h), the DFACC_ variable must be defined, along with its assigned value, at the beginning of the program. This is done by a code line such as:

```
parameter (DFACC_RDONLY = 1)     (For FORTRAN only)
```

Example:

FORTRAN:

```
integer*4 sd_id
integer sfstart
parameter(DFACC_RDONLY = 1)
sd_id=sfstart("wind.hdf", DFACC_RDONLY)
```

C:

```
#includehdf.h"
int32 sd_id;
sd_id=Sdstart("wind.hdf", DFACC_RDONLY);
```

Return

Select data set to be read from the HDF file

After initializing the SD interface and opening and assigning a file id (sd_id) to the HDF file to be used, the next step is to select the HDF Scientific Data Set (SDS) which will be read. This is done by the following command:

```
sds_id = sfselect (sd_id, sds_index) (FORTRAN)
```

or

```
sds id = SDselect (sd id, sds index) (  C  )
```

where

```
sds_id = HDF SDS array id returned by the sfselect/SDselect command
sd_id = the HDF file id created in the previous step (sfstart/SDstart)
sds_index = index number of data set within file                (i.e. 0 = f
```

Example:

```
FORTRAN:
        integer sds_id, sds_index, sd_id
        integer sfselect
c       sds_index = 0 represents the first data set
        sds_id = sfselect(sd_id,0)
C:
        int32 sd_id, dims[2];
        dims[0] = YL;
        dims[1] = XL;
        sds_id = Sdselect(sd_id,0);
```

Return

Read an existing data set/array

After initializing the API and selecting the HDF file and HDF SDS to be read to, the next step is to actually read the existing HDF data by using the SDreaddata (sfrdata) command. This command is used to read either all or part of the existing n-dimensional data set (termed a "slab") into the sds_id array with the same number of dimensions. In addition, the size of each dimension of the data "slab" must be the same or smaller then the corresponding dimension of the sds_id. The SDreaddata/sfrdata command is used in the following fashion

```
ret=sfrdata (sds_id, start, stride, edge, data) (FORTRAN)
```

or

```
ret=SDreaddata (sds_id, start, stride, edge, data); ( C )
```

It should be noted that there are two versions of the read routine in FORTRAN. The sfrdata routine reads numeric scientific data while sfrcdata reads character scientific data.

where

sds_id = the SDS id (identifier) determined and returned by using SDcreate or SDselect (sfcreate/sfselect)

start = An array which identifies where in the SDS that the reading will begin

The start array identifies the location or position in the SDS where the reading of the data "slab" will

begin. This array must have the same number of dimensions (rank) as the SDS and can not be larger (in each dimension) then the SDS array. The declaration of the start variables can be done at the top of the program or just preceding the call of the sfrdata (SDreaddata) command. As an example, to read the existing data set to the beginning of a new 2-dimensional SDS the following must be specified:

```
start(1) = 0 (FORTRAN)
start(2) = 0
```

or

```
start[0] = 0; ( C )
start[1] = 0;
```

If the user wishes to begin reading the data at a location other then the beginning of the data set, say at a first dimension (X) of 15, the declarations would be:

```
start(1) = 15 (FORTRAN)
start(2) = 0
```

or

```
start[0] = 15; ( C )
start[1] = 0;
```

stride = An array specifying the interval between written values in each dimension

The stride argument specifies, for each dimension, the interval between consecutive written values of the data set. In other words, how many array locations are skipped with each reading of the data. Like the start array, the stride argument is predefined before calling the sfrdata (SDreaddata) command, either directly before the call or at the top of the program.

If the user does not wish to skip any array locations in a new 2-dimensional SDS, the following is to be declared:

```
stride(1) = 1 (FORTRAN)
stride(2) = 1
```

or

```
stride[0] = 1; ( C )
stride[1] = 1;
```

However, if the user wishes to skip every other X (dimension 1) location, the following would be used:

```
stride(1) = 2 (FORTRAN)
stride(2) = 1
```

or

```
stride[0] = 2; ( C )
stride[1] = 1;
```

edge = An array defining the number of data values to be read in each dimension

The edge array defines the number of data values/elements that will be read along each dimension of the multi-dimensional SDS array. In plain terms, this argument defines the size of the data slab (all or part of the data) to be written to the new SDS array and each dimension.

The parameter edge must be specified for each dimension of the data set and SDS array, and can not be larger then the entire length of the array being read.

Similar to stride and start, the edge argument needs to be defined prior to the calling of the sfrdata (SDreaddata) command, whether it be at the top of the program or directly before the routine call. The file containing this data should be opened at the beginning of the program and the data read in and stored into the necessary arrays before beginning the HDF operations.

As an example: Most often, the user will wish to read an HDF file which contains one data set (winddata), which starts from the beginning and does not contain any missing data or blanks.

For a 2-dimensional array of 30X30, read and stored into the data array "testdata", this can be done by:

```
start(1) = 0                          (FORTRAN)
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 30
edge(2) = 30
retn = sfrdata(sds_id, start, stride, edges, winddata)
```

or

```
start[0] = 0;                         ( C )
start[1] = 0;
stride[0] = 1;
stride[0] = 1;
edge[0] = 30;
edge[1] = 30;
retn = SDreaddata(sds_id, start, stride, edges, winddata);
```

Example:

For reading the entire data set from an HDF file which contains only one 2-D array

```
FORTRAN:
        integer start(2), edges(2), stride(2)
        integer retn sfrdata
c       Define the location, pattern + size of data to be read
        YL = 30
        XL = 30 start(1) = 0    start(2) = 0    stride(1) = 1   stride(2) = 1   edg
        edge(2) = YL
        retn = sfrdata(sds_id, start,stride,edges,winddat)
C:
/* Define the location, pattern + size of data to be read */
        YL = 30;
        XL = 30;
```

```
dims[0] = YL;
dims[1] = XL;
start[0] = 0;
start[1] = 0;
stride[0] = 1;
stride[1] = 1;
edge[0] = dims[0];
edge[1] = dims[1];
retn = SDreaddata(sds_id, start,stride,edges,winddat);
```

Return

Write non-HDF data to a file

Using standard FORTRAN and C statements for writing, the non-HDF data is written into a new file (storage). In addition, the user may wish to print out all or parts of the HDF data set to view the data or as a check of the procedure/operation.

Return

Optional operation: Get and Read Metadata

After opening the HDF file using the sfstart/SDstart, the first step is to see if the file or data sets do indeed contain attributes. This is done by using the following command:

```
attr_index = SDfindattr (sd_id, attr_name);      ( C )
attr_index = sffattr (sd_id, attr_name)          (FORTRAN)
```

where

```
attr_index = valid attribute index returned if attribute exists
sd_id = file identifier
attr_name = name of attribute (i.e.,Contents of file")
```

If there is a attribute index, the name, data type (num_type), and count (number of characters) of the attribute can be obtained:

```
retn= SDattrinfo(sd_id, attr_index, attr_name, num_type, count);        (
retn= sfgainfo (sd_id, attr_index, attr_name, num_type, count)   (FORTRAN)
```

After completing these operations, the attributes can be read using the following:

```
retn= SDreadattr (sd_id, attr_index, buffer);    ( C )
retn= sfrattr (sd_id, attr_index, buffer)        (FORTRAN)
```

where

buffer is allocated to hold the attribute data

The above steps can also be followed for each data set within the file by getting the data set id (sds_id) of the data, finding a particular attribute (i.e.,"Units") and getting and reading the data.

Example:

```
FORTRAN:
        sd_id=sfstart ("wind.hdf", DFACC_RDONLY)
        attr_index= sffattr (sd_id,"file_contents")
        retn= sfgainfo (sd_id, attr_index, "file_contents", data_type, count)
        retn=  sfrattr (sd_id, attr_index, buffer)
and
        sds_id=sfselect (sd_id, 0)
        attr_index= sffattr (sds_id,"units")
        retn= sfgainfo (sds_id, attr_index,"units", data_type, count)
        retn=  sfrattr (sds_id, attr_index, buffer)
C:
        sd_id=SDstart ("wind.hdf", DFACC_RDONLY);
        attr_index= SDfindattr (sd_id,"file_contents");
        retn= SDattrinfo (sd_id, attr_index,"file_contents", data_type, count);
        retn=  SDreadattr (sd_id, attr_index, buffer);
and
        sds_id=SDselect (sd_id, 0);
        attr_index= SDfindattr (sds_id,"units");
        retn= SDattrinfo (sds_id, attr_index,"units", data_type, count);
        retn=  SDreadattr (sds_id, attr_index, buffer);
```

Return

Terminate/Close access to all files, data sets, and APIs

After writing the data to the new SDS array within the new HDF file, it is necessary to terminate or close access to the new data set in order to prevent any possible loss of data. This is done by the following:

```
retn = sfendacc(sds_id)          (FORTRAN)
```

or

```
retn = SDendaccess(sds_id)       ( C )
```

In addition, the API called within the program must also be closed to prevent any data loss:

```
retn = sfend(sd id)              (FORTRAN)
```

or

```
retn = SDend(sd_id)                ( C )
```

Example:

```
FORTRAN:
        integer sfendacc, sfend
        retn = sfendacc(sds_id)
        retn = sfend(sd_id)



C:
        retn = SDendaccess(sds_id);
        retn = SDend(sd_id);
```

Return

Execute program

Execute like a standard FORTRAN or C program.

Return

Browsing and Visualizing HDF Data

With the recent explosion of data volumes, numerous visualization and browsing tools have been developed which allow users to quickly view the contents of datasets created elsewhere. This has proven especially beneficial for users of HDF.

In fact, many visualization tools have been created specifically with HDF in mind. The NCSA anonymous ftp server provides a set of free software that enables the user to visualize and browse HDF files. Tools include theJAVA-based HDF Browser and the Scientific Data Browser. In addition, the following are available but have not been updated to run with the current version of HDF: NCSA Collage, NCSA Datascope, NCSA XDataSlice, and NCSA Polyview.

Besides NCSA, there are other sites and centers that also provide public domain (free) software that can be used to browse and visualize HDF files. This software includes, among others: LinkWinds, GRASS, FREEFORM, VISTAS, ImageMagick, and Envision. Visualization tools and software such as LinkWinds and EOSVIEW can be used for working with HDF-EOS type data (point, swath and grid data sets).

Finally, there are also commercial (for a fee) software packages that can be used to work with and browse HDF files. These include: DataExplorer, Spyglass, PV-Wave, Wavefront, IDL, AVS, IRIS Explorer, Transform, and ER Mapper.

Please see Section 4: HDF Browsing and Visualization Tools for further detail, including internet address, on the above software.

Previous Main Topic Next Main Topic

Return to Main Topics

Example Programs

The following is a list of sample programs that illustrate how the HDF library, and the SD API, can be used to work with HDF files. The example programs are given in the FORTRAN programming language. However, the detailed steps for all languages are the same. Only the syntax code particular to each language should be different.

- Writing an SDS in HDF
- Writing Attributes in HDF
- Writing the SDS and attributes in HDF
- Reading an HDF file
- Reading HDF attributes (files and data sets)

Previous Main Topic Next Main Topic

Return to Main Topics

Writing an SDS in HDF

FORTRAN:

```
      PROGRAM WRITDATA
C
      integer*4 sd_id, sds_id, rank
      integer*4 XL, YL
      integer dims(2), start (2), edges (2), stride (2)
      integer i, j, k, retn
      integer sfstart, sfcreate, sfwdata, sfendacc, sfend
      real rwind(30, 30)
C
C     DFACC_CREATE and DFNT_INT16  are defined in hdf.h but may have
C     to be defined within the program for certain FORTRAN compilers
C
      integer*4 DFACC_CREATE, DFNT_INT16
      parameter (DFACC_CREATE = 4, DFNT_INT16=22)
      rank = 2
      XL = 30
      YL = 30
C
C     Create and open a new HDF file and initiate the SD interface
C
      sd id = sfstart('wind.hdf', DFACC CREATE)
```

```
C
C     Define the rank (number of dimensions) and dimensions (size) of the
C     HDF Scientific Data Set (SDS) to be created.
C
      dims(1) = XL
      dims(2) = YL
C
C
C     Create the HDF SDS (sfselect would be used if writing to an
C     existing HDF file or data set)
C
      sds_id = sfcreate(sd_id, 'winds', DFNT_INT16, rank, dims)
C
C     Open and read the existing non-HDF data set into an array (rwind)
C
      Open (unit=10, file='wind.dat', form='formatted')
      Do j = 1,30
        Read(10, 12) (rwind(i, j),i = 1,30)
  12    Format(30(f4.1,1x))
      Enddo
C
C     Define where in the file to write the data set (start--location),
C     the pattern of the data (stride--skip any values??), and the size
C     of the data set (edges) to be written to.  This is done for each
C     dimension.   start(x) = 0 is for writing at the beginning of the
C     newly created SDS and stride(x) = 1 signifies that no data is to
C     be skipped in the writing.
C
      start(1) = 0
      start(2) = 0
      edges(1) = XL
      edges(2) = YL
      stride(1) = 1
      stride(2) = 1
C
C     Write the the stored data (in the array rwind) to the new SDS
C
      retn = sfwdata(sds_id, start, stride, edges, rwind)
C
C     Terminate access to the array
C
      retn = sfendacc(sds_id)
C
C     Terminate access to the SD interface and close the HDF file
C
      retn = sfend(sd_id)
      Stop
      End
```

Return to top

Writing Attributes in HDF

FORTRAN:

```
      PROGRAM WRITEATT
C
      integer*4 sd id, sds id, dim id, retn
```

```
      integer dims(2), start(2), edges(2), stride(2)

      integer sfstart, sfselect, sfdimid, sfscatt, sfendacc, sfend

C
C     DFACC_RDWR,  DFNT_INT16 and DFNT_CHAR8  are defined in hdf.h but
C     may have to be defined within the program for certain FORTRAN
C     compilers
C
      integer*4 DFACC_RDWR, DFNT_INT32, DFNT_CHAR8
      parameter (DFACC_RDWR = 3, DFNT_INT16 = 22, DFNT_CHAR8 = 4)
C
C     Open the HDF file,  initiate the SD interface, and get the
C     identifier for the file
C
      sd_id = sfstart('wind.hdf', DFACC_RDWR)
C
C     Set an attribute the describe the contents of the file
C
      retn = sfscatt(sd_id, 'file_contents', DFNT_CHAR8, 15,'lidar_LOS_winds')
C
C     Get the identifier for the first data set (in this example, the
C     only data set)
C
      sds_id = sfselect(sd_id, 0)
C
C     Set an attribute(s) for the data array itself. In this example, the
C     units of the data are defined
C
      retn = sfscatt(sds_id, 'units', DFNT_CHAR8, 13, 'units = m/sec')
C
C     Terminate access to the data array
C
      retn = sfendacc(sds_id)
C
C     Terminate access to the SD interface and close the HDF file
C
      retn = sfend(sd_id)
      Stop
      End
```

Return to top

Writing the SDS and attributes in HDF

FORTRAN:

```
      PROGRAM WRITESDS
C
      integer*4 sd_id, sds_id, rank, dim_id

      integer*4 XL, YL
      integer dims(2), start(2), edges(2), stride(2)
      integer i, j, k, retn
      integer sfstart, sfcreate, sfwdata, sfendacc, sfscatt, sfend
      real rwind(30, 30)
C
C     DFACC CREATE, DFACC RDWR, DFNT CHAR8 and DFNT INT16  are defined
```

```
C     in hdf.h but may have to be defined within the program for certain
C     FORTRAN compilers
C
      integer*4 DFACC_CREATE, DFNT_INT16, DFNT_CHAR8, DFACC_RDWR
      parameter (DFACC_CREATE = 4, DFACC_RDWR = 3, DFNT_INT16 = 22, DFNT_CHAR8 = 4)
C
      rank = 2
      XL = 30
      YL = 30
C
C     Create and open a new HDF file and initiate the SD interface
C
      sd_id = sfstart('wind.hdf', DFACC_CREATE)
C
C     Define the rank (number of dimensions) and dimensions (size) of the
C     HDF Scientific Data Set (SDS) to be created.
C
      dims(1) = XL
      dims(2) = YL
C
C     Create the HDF SDS (sfselect would be used if writing to an
C     existing HDF file or data set)
C
      sds_id = sfcreate(sd_id, 'winds', DFNT_INT16, rank, dims)
C
C     Open and read the existing non-HDF data set into an array (rwind)
C
     Open (unit = 10, file = 'wind.dat', form = 'formatted')
       Do j=1,30
        Read (10, 12) (rwind(i, j),i=1,30)
   12   Format (30(f4.1,1x))
      enddo
C
C     Define where in the file to write the data set (start--location),
C     the pattern of the data (stride--skip any values??), and the size
C     of the data set (edges) to be written to.  This is done for each
C     dimension.  start(x) = 0 is for writing at the beginning of the
C     newly created SDS and stride(x)=1 signifies that no data is to be
C     skipped in the writing.
C
      start(1) = 0
      start(2) = 0
      edges(1) = XL
      edges(2) = YL
      stride(1) = 1
      stride(2) = 1
C
C     Write the the stored data (in the array rwind) to the new SDS
C
      retn = sfwdata(sds_id, start, stride, edges, rwind)
C
C     For writing attributes, set an attribute the describe the
C     contents of the file
C
      retn = sfscatt(sd_id, 'file_contents', DFNT_CHAR8, 15,'lidar_LOS_winds')
C
C     Set an attribute(s) for the data array itself. In this example, the
C     units of the data are defined
C
      retn = sfscatt(sds_id, 'units', DFNT_CHAR8, 13, 'units = m/sec')
C
C     Terminate access to the data array
C
```

```
      retn = sfendacc(sds_id)
C
C     Terminate access to the SD interface and close the HDF file
C
      retn = sfend(sd_id)
      Stop
      End
```

Return to top

Reading an HDF file

FORTRAN:

```
      PROGRAM READDATA
C
      integer*4 sd_id, sds_id
      integer*4 XL, YL
      integer start(2), edges(2), stride(2)
      integer i, j, k, retn
      integer sfstart, sfselect, sfrdata, sfendacc, sfend
      real rwind(30, 30)
C
C     DFACC_RDONLY is defined in hdf.h but may have to be defined
C     within the program for certain FORTRAN compilers
C
      integer*4 DFACC_RDONLY      parameter (DFACC_RDONLY = 1)
C
C     MAX_NC_NAME (maximum # of characters) and MAX_VAR_DIMS (maximum
C     # of dimensions) are defined in netcdf.h but may have to be defined
C     here.
C
      integer*4 MAX_NC_NAME, MAX_VAR_DIMS
      parameter (MAX_NC_NAME = 256, MAX_VAR_DIMS = 32)
      integer dims(MAX_VAR_DIMS)
      XL = 30
      YL = 30
C
C     Open the HDF file and initiate the SD interface
C
      sd_id = sfstart('wind.hdf', DFACC_RDONLY)
C
C     Select the first data set in the file (In this example, the only
C     dataset).
C
      sds_id= sfselect(sd_id, 0)
C
C     To read from the data set, define the location (start--where in the
C     file), the pattern (stride--skip any values??),and the size(edges)
C     of the data.  This is done for each dimension. start(x) = 0 is for
C     reading at the beginning of the file and stride(x) = 1 signifies
C     that no data is to be skipped in the reading.
C
      dims(1) = XL
      dims(2) = YL
      start(1) = 0
      start(2) = 0
      stride(1) = 1
```

```
      stride(2) = 1
      edges(1) = dims(1)
      edges(2) = dims(2)
C
C     Read the array dataset
C
      retn = sfrdata(sds_id, start, stride, edges, rwind)
C
C     Optional - Print out data (ASCII) read from the HDF file
C     In this example we are writing to the screen (*)
C
      Do j = 1, 30
        write(*,12) (rwind(i,j),i=1,30)
 12     format(30(f4.1,1x))
      enddo
C
C     Terminate access to the array
C
      retn = sfendacc(sds_id)
C
C     Terminate access to the SD interface and close the HDF file
C
      retn = sfend(sd_id)
      Stop
      End
```

Return to top

Reading HDF attributes (files and data sets)

FORTRAN:

```
      PROGRAM READATTR
C
      integer*4 sd_id, sds_id, units_buffer
      integer attr_index, data_type, count, retn
      character attr_name * 13
      character char_buffer * 20
      integer sfstart, sfrnatt, sfrcatt, sfgainfo, sffatr, sfselect
      integer sfendacc, sfend
C
C     DFACC_RDWR is defined in hdf.h but may have to be defined
C     within the program for certain FORTRAN compilers
C
      integer*4 DFACC_RDWR, DFACC_RDONLY
      parameter (DFACC_RDWR = 3, DFACC_RDONLY = 4)
C
C     Open the HDF file and initiate the SD interface
C
      sd_id = sfstart('wind.hdf', DFACC_RDONLY)
C
C     Select the first data set in the file (In this example, the only
C     dataset).
C
      sds_id= sfselect(sd_id, 0)
C
C     Find the the attribute which describes the contents of the file
C     (usually 'file contents')
```

```
C
      attr_index = sffattr(sd_id, 'file_contents')
C
C     Get information about the file attribute
C
      retn = sfgainfo(sd_id, attr_index, attr_name, data_type, count)
C
C     Read the file attribute data
C
      retn = sfrcatt(sd_id, attr_index, char_buffer)
C
C     Read the attributes for the first data set. First step is to get
C     the identifier.
C
      sds_id = sfselect(sd_id, 0)
C
C     Find the attribute which defines the units of the data set
C     ('units')
C
      attr_index = sffattr(sds_id, 'units')
C
C     Get information about the data set attribute
C
      retn = sfgainfo(sds_id, attr_index, attr_name, data_type, count)
C
C     Read the data set attribute data
C
      retn = sfrcatt(sds_id, attr_index, units_buffer)
C
C     Terminate access to the array
C
      retn = sfendacc(sds_id)
C
C     Terminate access to the SD interface and close the HDF file
C
      retn = sfend(sd_id)
      Stop
      End
```

Return to top

HDF Laboratory

.

Lab Directions

Begin Lab

Previous Main Topic

Return to Main Topics

Lab Directions

The question and answer section of the tutorial was developed in Java script and is best viewed using Microsoft Internet Explorer 4.0/5.0 and the latest release of Netscape navigator. When navigating through the tutorial, individual questions will be loaded on the same window and will be controlled by the "Previous question" and "next question" buttons. However, new windows will be opened when the user attempts to look at the preview material for each question and thus allowing the user to toggle back and forth from the question and the material. To exit the tutorial, just click the "back" button from the main question screen and this will bring the user back to the Laboratory menu. When done with a "preview" window, simply close out the window and return to the question window/screen.

In this section we provide a series of questions designed for the users of the tutorial to test themselves on how well they understood the material presented in the tutorial and, more importantly, to gauge how comfortable they feel with HDF.

The questions more or less follow the order of the topics covered in the "Lecture" component of the tutorial. The Laboratory menu provides a breakdown by section of the various questions, and allows the user to select which topics they would like to focus their attention on.

Each question contains the question itself, a set of possible answers, navigation buttons ("Next Question", "Previous Question", "Lab Menu", "End Lab", "Submit Answer") and, most importantly, a feature which allows the user to review material pertaining to the question before answering or after answering incorrectly. This is done by selecting the "Preview Material" button. Since some users will like to take the "test" without any help and others will like to review material before answering (particularly those who may have skipped directly to the Laboratory), it is up to the individual user to decide how to proceed.

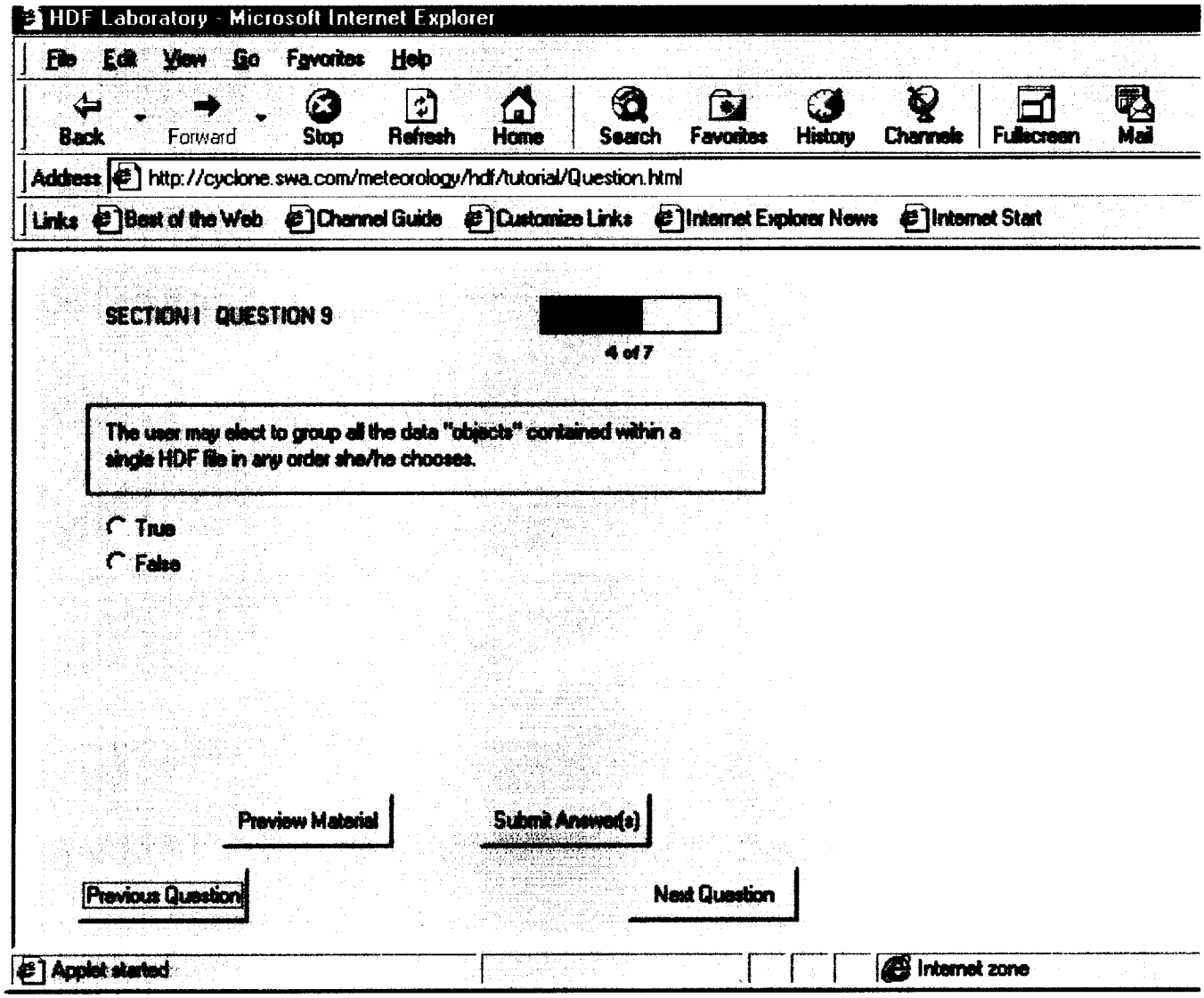
It should be noted that, in many cases, there is more than one correct answer for each individual question. The user is allowed to select more than one response and will only receive an "Answered Correctly" response if ALL correct responses have been selected.

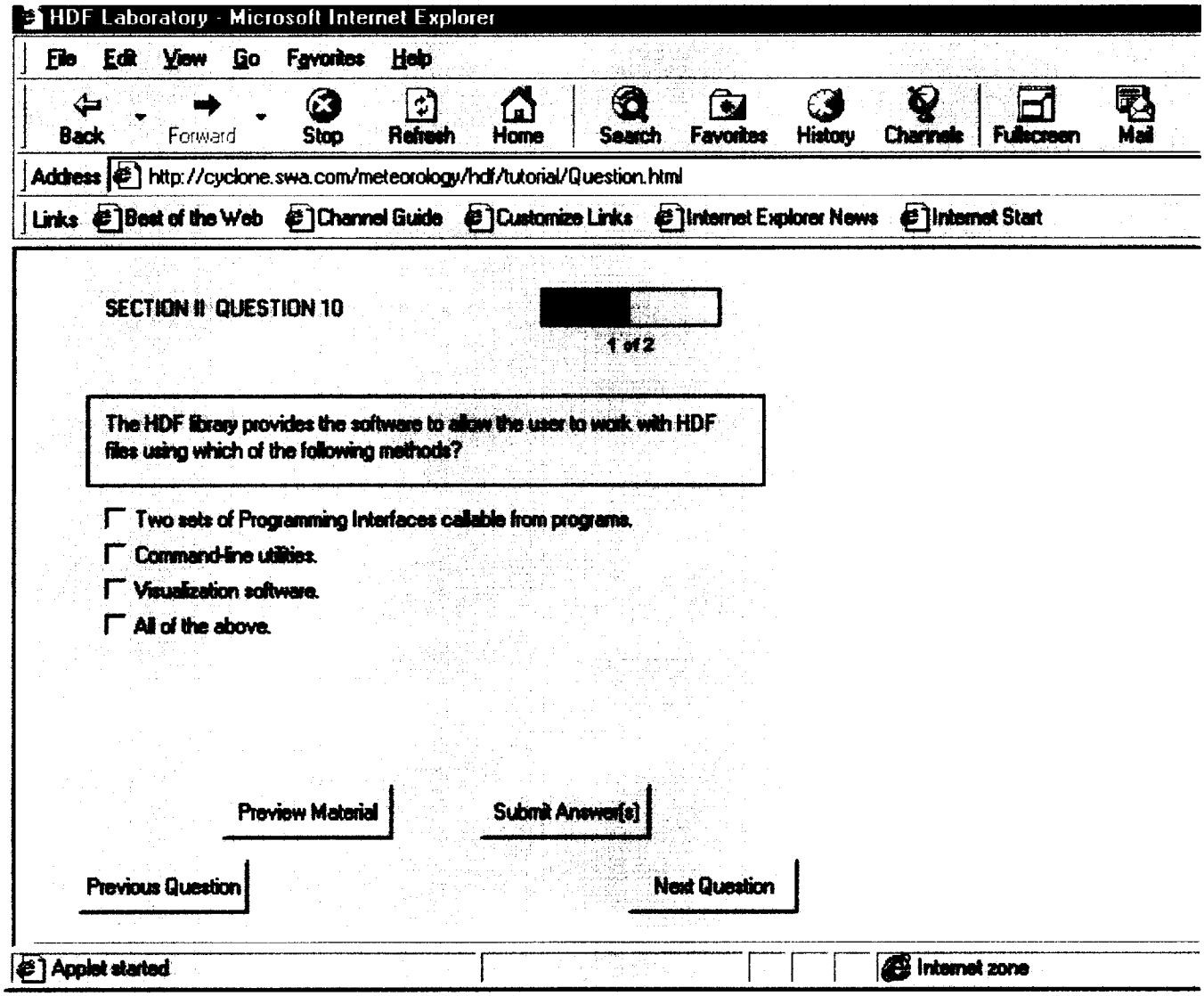
To help the users qauge their understanding of HDF and how well they are answering the questions, a "performance gauge" is provided in the upper right hand corner of each question. This quage provides both a numerical (i.e. 7 of 11) and graphical (sliding color bar for 0 - 100%) representation of user performance. The "score" found in the performance gauge only reflects the users' INITIAL answer to each question.

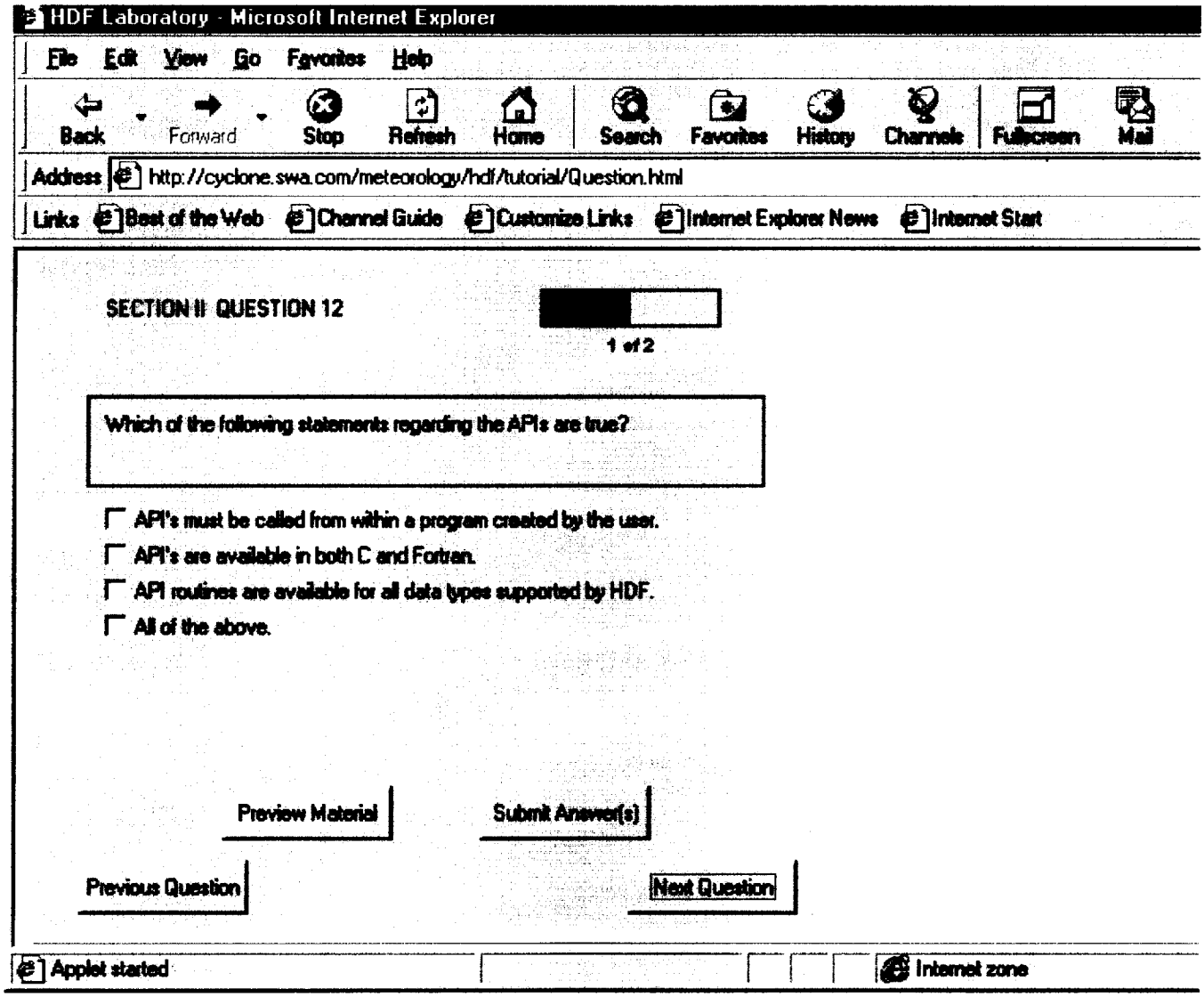
The user's performance in the Laboratory is further diagnosed in a Progress Report reached by clicking on the performance quage. Included in this report are:

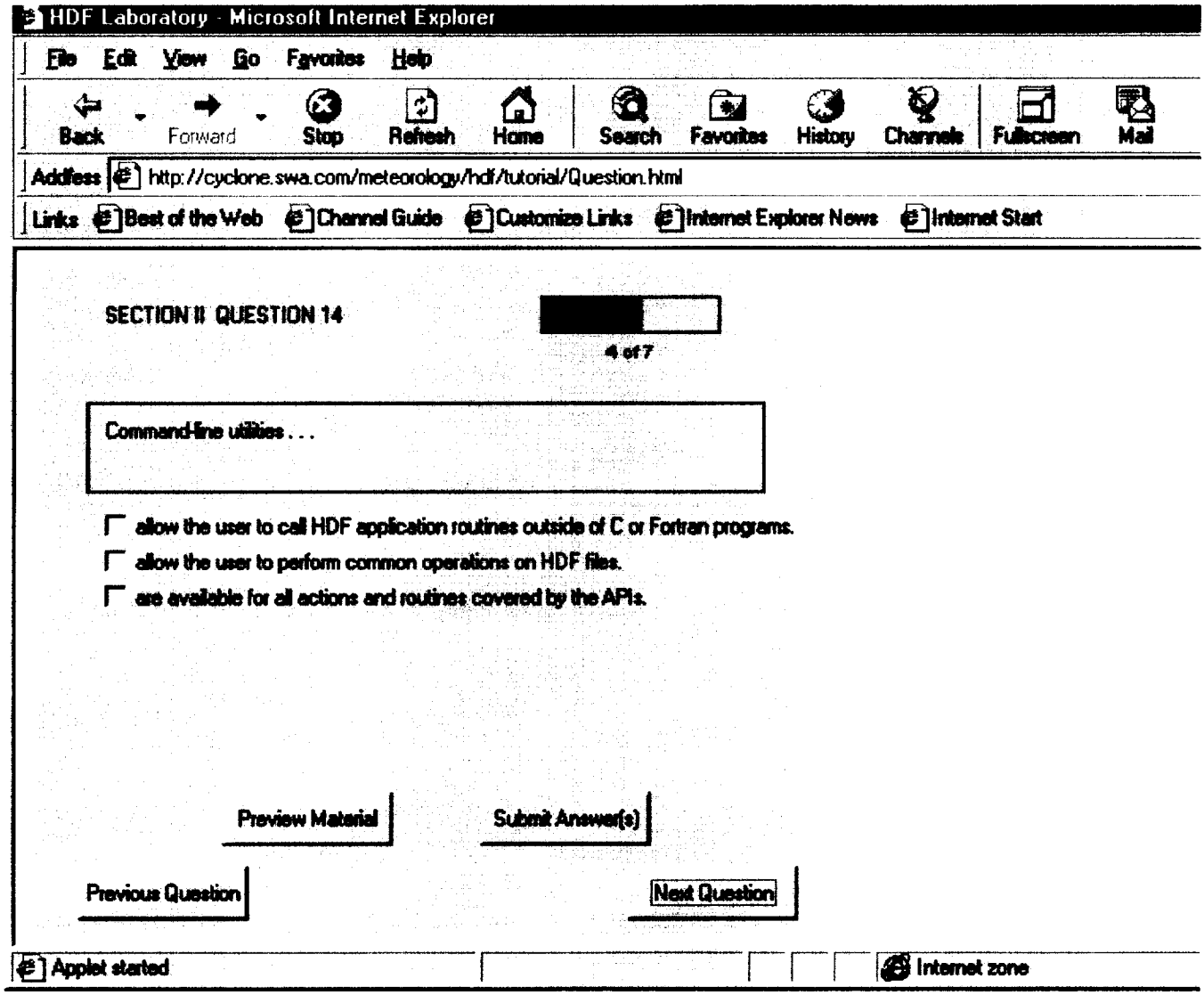
- number of questions answered correctly
- number of questions answered incorrectly
- number of questions left unanswered
- list of questions answered correctly
- list of questions answered incorrectly
- list of questions left unanswered

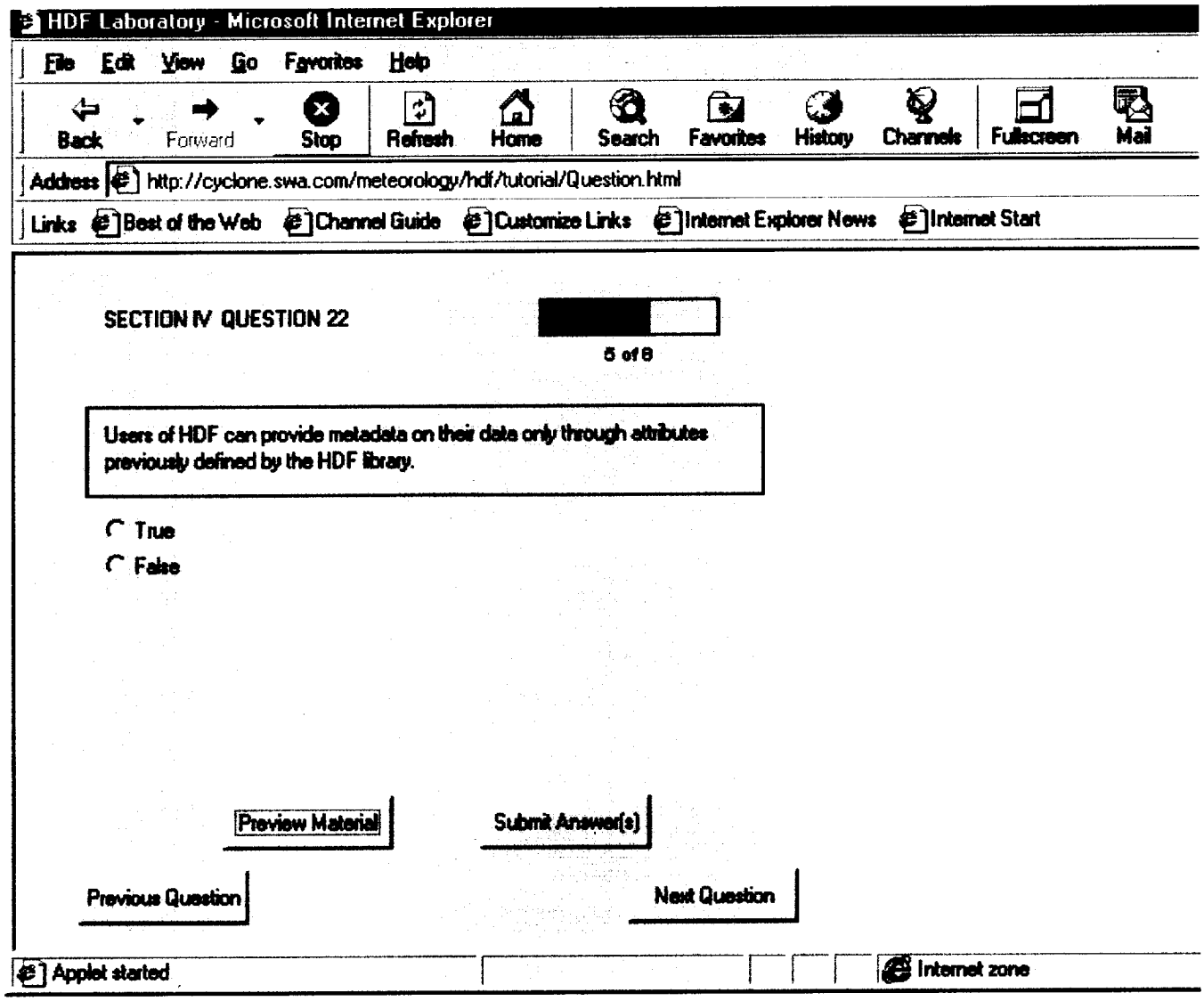
Return to top

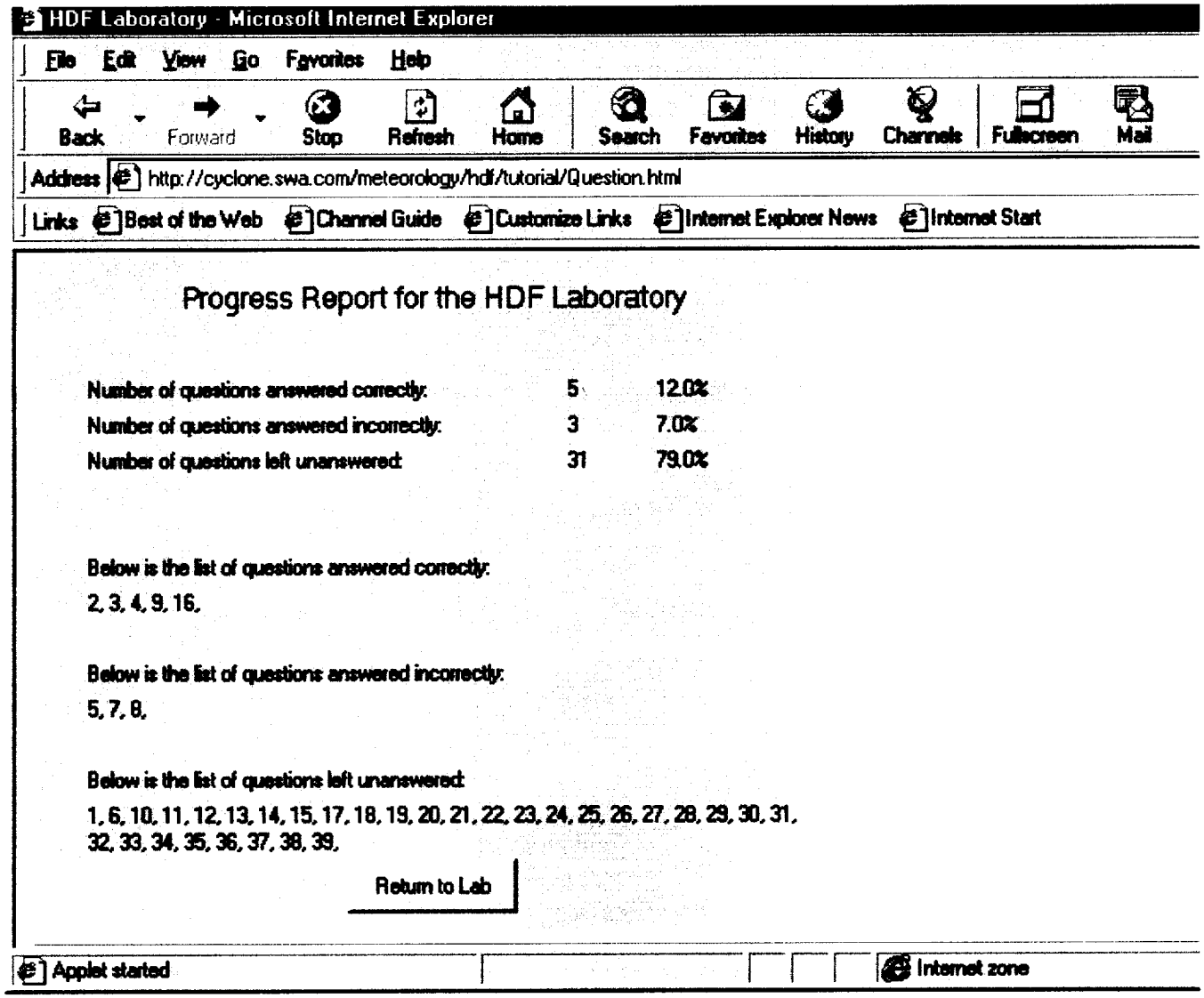

HDF Laboratory - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address http://cyclone.swa.com/meteorology/hdf/tutorial/Question.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
SECTION I QUESTION 9
4 of 7
The user may elect to group all the data "objects" contained within a single HDF file in any order she/he chooses.
True
False
Preview Material
Submit Answer(s)
Previous Question
Next Question
Applet started
Internet zone


HDF Laboratory - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address http://cyclone.swa.com/meteorology/hdf/tutorial/Question.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
SECTION II QUESTION 10
1 of 2
The HDF library provides the software to allow the user to work with HDF files using which of the following methods?
Two sets of Programming Interfaces callable from programs.
Command-line utilities.
Visualization software.
All of the above.
Preview Material
Submit Answer(s)
Previous Question
Next Question
Applet started
Internet zone


HDF Laboratory - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address http://cyclone.swa.com/meteorology/hdf/tutorial/Question.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
SECTION II QUESTION 12
1 of 2
Which of the following statements regarding the APIs are true?
API's must be called from within a program created by the user.
API's are available in both C and Fortran.
API routines are available for all data types supported by HDF.
All of the above.
Preview Material
Submit Answer(s)
Previous Question
Next Question
Applet started
Internet zone


HDF Laboratory - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address http://cyclone.swa.com/meteorology/hdf/tutorial/Question.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
SECTION II QUESTION 14
4 of 7
Command-line utilities . . .
allow the user to call HDF application routines outside of C or Fortran programs.
allow the user to perform common operations on HDF files.
are available for all actions and routines covered by the APIs.
Preview Material
Submit Answer(s)
Previous Question
Next Question
Applet started
Internet zone


HDF Laboratory - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address http://cyclone.swa.com/meteorology/hdf/tutorial/Question.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
SECTION IV QUESTION 22
5 of 8
Users of HDF can provide metadata on their data only through attributes previously defined by the HDF library.
True
False
Preview Material
Submit Answer(s)
Previous Question
Next Question
Applet started
Internet zone


HDF Laboratory - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address http://cyclone.swa.com/meteorology/hdf/tutorial/Question.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
Progress Report for the HDF Laboratory
Number of questions answered correctly: 5 12.0%
Number of questions answered incorrectly: 3 7.0%
Number of questions left unanswered: 31 79.0%
Below is the list of questions answered correctly:
2, 3, 4, 9, 16,
Below is the list of questions answered incorrectly:
5, 7, 8,
Below is the list of questions left unanswered:
1, 6, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39,
Return to Lab
Applet started
Internet zone

ATTACHMENT D

An HDF Tutorial for Beginners: EOSDIS Users and Small Data Providers (Microsoft Word Version)

by

Mr. Steven Greco
Simpson Weather Associates
Charlottesville, VA

(In Fulfillment of NASA Contract NAG5-1961)

November 12, 1999

Main Topics

1. Tutorial Overview
2. An Introduction to HDF
3. The HDF Library: Software and Hardware
4. Methods of Working with HDF Files
5. Scientific Data API
6. Attributes and Metadata
7. Writing an HDF File
8. Obtaining Information on Existing HDF Files
9. Reading an HDF Data File
10. Example Programs
11. Browsing and Visualizing HDF Data
12. Laboratory (Question and Answer)
13. Acronym List

An HDF Tutorial for Beginners: EOSDIS Users and Small Data Providers

1. Tutorial Overview

1.1 Purpose of the Tutorial

The NASA ESDIS project selected the Hierarchical Data Format (HDF) as the common data format of choice for standard product exchange and distribution. As developed by the National Center for Supercomputer Applications (NCSA), the HDF format is supported by a collection of software routines and applications needed to work with data sets in HDF. This set of software, referred to as the HDF library, is available in the public domain. To facilitate the exchange of data and data products to be generated as part of the upcoming EOS missions, a sub-library or library extension of HDF, called HDF-EOS has also been developed to deal specifically with some types of satellite and field campaign data products that will be routinely generated.

While there are many advantages to the use of HDF, a key to its' success as a common data format and software library may be dependent upon expanding the general user and science communities' awareness, knowledge, and comfort with HDF and HDF-EOS. In particular, it is the individual investigators, academia (students through researchers), the educational community (K-12 needs), and the general public that many times do not have the required knowledge, nor the resources to commit to obtaining this knowledge, to work with HDF files.

In response to this need, the NASA ESDIS project has funded the creation of this on-line HDF tutorial geared towards HDF beginners. The purpose of this tutorial is to provide the HDF non-expert, particularly potential future users of EOS data, with the necessary information to enable one to successfully write data sets into HDF and to also read data from an existing HDF file. This information will include, but not be limited to, sections on the basics of HDF and HDF files, the required software/hardware, the various ways of working with HDF files, a review of HDF commands and operations, and a step by step instruction on writing programs to work with HDF.

Some of the information presented here can also be found in further detail throughout several of the excellent HDF reference guides and manuals (more on this in Section 2 - An Introduction to HDF) written by NCSA. However, the goal of this tutorial is to present, in a concise and easy to understand form, only the information needed to help the HDF novice to read and write basic HDF files. Furthermore, the HDF library has been designed to work with many different types of data (arrays, images, etc.) and to carry out both simple and complex operations on data sets. As a teaching tool, this tutorial will concentrate on only one data type (scientific data arrays) and the simpler operations such as reading, writing, and browsing entire data sets.

Another tutorial dealing with HDF and the HDF library has been developed by NCSA (http://hdf.ncsa.uiuc.edu/tutslects.html). The current tutorial contains much more of the basic information of HDF and is geared to the HDF beginner or

novice. Though there are many places of overlap between the two tutorials, they seem to compliment each other in providing information for all types of HDF users.

1.2 How to Use the Tutorial

In support of a contract to the NASA ESDIS project, this HDF tutorial has been designed by Simpson Weather Associates, Inc. (SWA) with the goal of teaching the novice HDF users, especially potential users of EOSDIS and future EOS data products, how to use the HDF library to read and write HDF files. The tutorial has been constructed in two parts. First is what we call the "Lecture" component where we present what we think is the information necessary for a novice user to learn what HDF is, what it can be used for, and how to apply it in practice. Included in this "Lecture" material is a step-by-step outline detailing what the user must do to successfully read or write an HDF file.

The second component of the tutorial is a question and answer section (what we call the "Laboratory") which tests the user's knowledge of HDF, concentrating on the information needed by the novice or average HDF user to work independently with the HDF library to read and write HDF files.

We realize that the familiarity and knowledge level of the users of this tutorial will span a wide range. As a result, we think it should be up to the users to decide how they wish to learn and navigate through the tutorial. However, we do advise that those with very little or no knowledge of HDF should first preview and study the lecture material before testing themselves with the Laboratory section.

1.3 Future Plans of the Tutorial

The current version of this HDF tutorial concentrates on the latest release of the HDF library (HDF 4.1r3 as of July 1999) and how to use it for reading and writing HDF files containing scientific data arrays. Future versions (ongoing work) of the tutorial will be expanded to include additional operations supported by the HDF library and how they can be used to work with various other data types such as raster images, binary tables, and palettes.

While these modifications are being made, parallel work is being conducted on tutorial components devoted to working with the point, swath and grid data sets expected to be produced by EOS instruments, and supported by the HDF-EOS sub-library.

In addition, a new experimental version of HDF, called HDF5, has been designed. This new library was designed to address the main drawbacks of HDF4, particularly the inability to deal with large data sets. Once HDF5 is officially accepted, a tutorial(s) will be needed.

2. An Introduction to HDF

2.1 What is HDF?

HDF, which stands for Hierarchical Data Format, is a common data format that has been developed to aid scientists and programmers in the storing, transfer and distribution of data sets and products created on various machines and with different software. HDF has been selected by the NASA ESDIS project as the format of choice for the standard product distribution that will be part of the Earth Observing System Data and Information System (EOSDIS).

In addition, HDF also refers to the collection of software, application interfaces, and utilities that comprise the HDF library and allows users to work with HDF files. The HDF library is discussed in detail in Section 3 - The HDF Library: Software and Hardware.

2.2 Features of HDF

HDF is a multi-object file format for the sharing and storing of scientific data. Some of the most important features of HDF are the following:

1) Self-describing: For each data object in an HDF file, there is also information (or metadata) about the data type, size, dimensions and location found within the file itself.
2) Extensibility: HDF is designed to accommodate future (new) data types and data models.
3) Versatility: Currently, HDF supports six different data types and provides software and applications to read and write these data types in HDF.
4) Flexibility: HDF lets the user group, store, and read/write different data types in the same file or in more than one file.
5) Portability: HDF software is mainly platform independent and can be shared across most computer platforms (all platforms have not been tested).
6) Standardization: HDF standardizes the formats and descriptions of many types of commonly- used data types (i.e., arrays, images, etc.).
7) HDF is available in the public domain.

2.3 What types of data does HDF support?

As of the latest release of HDF (HDF4.1 release 3 as of July 1999), the HDF library supports the working with raster images, color or gray scale palettes, multi-dimensional arrays, text strings, and statistical data (in the form of tables). The HDF library supports the following data types:

1. Scientific Data sets -- Multi-dimensional integer or floating point arrays
2. Vertex Data (Vdata and Vgroups) -- Multi-variate data stored as records in a table
3. General Raster (Gr) -- Raster images

4. Annotation -- Text strings to describe files and parts of files (metadata)
5. 8-bit Raster images
6. 24-bit Raster images
7. Palette -- 8-bit color palettes (accompany images)

In addition to these data types supported by the base HDF library, a sub-library called HDF-EOS has been developed to support various data types anticipated from the Earth Observing System (EOS) satellite missions. The HDF-EOS data models include point data, satellite swath data, and gridded data.

As mentioned in the Welcome section, this tutorial will concentrate on the Scientific Data Model as a means of teaching the essentials of HDF. More information on the other data models can be obtained in the various documents (particularly the HDF User's Guide) provided by NCSA through their anonymous ftp server or World Wide Web home page.

2.4 Which version of HDF should I use?

The most current version or release of HDF is the best place to begin. As of July 1999, the current version of the HDF library is HDF 4.1r3. An extension of the HDF library, called HDF-EOS, is based on this version of HDF and is designed specifically to work with data products anticipated from the upcoming EOS satellite missions. The current tutorial will focus on the releases (i.e., r1, r2 or r3) of HDF4.1. One feature of HDF4 that is important, especially to experienced users of HDF, is the backward compatibility of HDF. That is, HDF4.1r3 is compatible with earlier versions such as HDF4.1r1 and the data sets that were generated.

It should be noted that an experimental version of HDF, called HDF5, has also recently been developed to address the shortcomings of HDF4. This new HDF library includes simpler source codes, more consistent and fewer data models, and the ability to work with large data sets (> 2GB). However, although plans call for the HDF-EOS interface to be based on HDF5 at a later date, it is only in the experimental/prototype stage. HDF5 and the associated software will not be covered in this tutorial. The user is directed to NCSA's HDF5 Page (http://hdf.ncsa.uiuc.edu/HDF5/) for detailed information.

2.5 Where can I get additional and detailed information on HDF?

The best sites or locations to find detailed information on all aspects of HDF are the NCSA HDF Information Server available through the Internet (http://hdf.ncsa.uiuc.edu/) and the NCSA anonymous ftp server (ftp://ftp.ncsa.uiuc.edu/HDF/HDF/Documentation). Inquiries should be sent to hdfhelp@ncsa.uiuc.edu.

The following documents and information can be obtained through the sources mentioned above:

- HDF 4.1 r3 Reference Manual
- HDF 4.1 r3 Users Guide
- HDF Specifications and Developers Guide v3.2 (mainly for the programmers/developers)
- HDF Newsletters

- HDF Frequently Asked Questions (FAQ)
- Java Products
- Frequently Asked Questions about Java and HDF
- Release Notes and Man Pages provide information on items that are not covered in the above documents
- HDF software contributions from non-NCSA users

In addition, users may wish to join the hdfnews mailing list (by emailing ncsalist@ncsa.uiuc.edu and placing subscribe hdfnews in the body of the message) for discussions and updates on HDF.

3. The HDF Library: Software and Hardware

The HDF library is a collection of software routines that provides two types of interfaces which allow the user to work with HDF files. A brief capsule describing these interfaces is provided below:

Low-level Interface - The so-called low-level interface provides software that enables the user to work with such file features as memory, error handling, and storage. However, these features and the software are more of interest to the experienced programmer and software developer not the HDF novice or beginner interested in learning to read and write HDF files.

Information on the low-level interface can be found in the documentation listed in Section 2 - Where can I get additional and detailed information on HDF?

Application Programming Interfaces (APIs)- Of more use to the average HDF user are the high-level or Application Programming Interfaces (APIs). These APIs are sets of routines that can be called in the user's FORTRAN or C program and which will allow the user to access, read, and write HDF files. There are APIs specifically created for each of the different data types supported by HDF, which allow the user to work with HDF files.

Further detail is provided in Section 4 - Methods of Working with HDF Files.

In addition, the HDF library also provides a set of command-line utilities that allow the user to work with HDF files outside of the interfaces and within the command level (such as UNIX) of a terminal session. Outside of the HDF library, there are also a large number of browsing and visualization software packages (both free and commercial) that allow the user to look at all types of HDF files. These two methods will be discussed later in the tutorial.

3.1 Obtaining and installing the HDF library

The HDF library and utilities are public domain software and are freely available, along with documentation, from the NCSA anonymous ftp server. The latest release of the HDF library can be downloaded from ftp://ftp.ncsa.uiuc.edu/HDF/HDF/HDF_Current. Associated documentation and reference material describing the library and its' installation can be obtained from ftp://ftp.ncsa.uiuc.edu/HDF/HDF/Documentation. The source code of the HDF library and utilities are available with each "release" of HDF and can be downloaded free of charge from this ftp site. The files are available in various forms to support users of PCs, Macs, etc.

Unfortunately, the HDF library may not be accessed by every computer platform. Following sections list the platforms and operating systems on which the latest release of HDF has been tested.

NCSA provides a binary distribution for those platforms supported by HDF. For platforms that are not specifically supported, the HDF source code is provided.

On UNIX, VMS, and Windows NT/95, the binary distribution includes the pre-compiled libraries, utilities, include files, man pages, and release notes. The binary distribution on the Macintosh does not include the utilities.

The binaries are located in the following directories on the NCSA ftp server (ftp.ncsa.uiuc.edu):

/HDF/HDF_Current/bin- Unix and VMS
/HDF/HDF_Current/zip- Windows NT/95
/HDF/HDF_Current/hqx- Macintosh

If you uncompressed the binaries for a supported platform, you would (in general) find the following directories:

../bin - pre-compiled utilities
../include - include files
../lib - libraries
../man - man pages
../release_notes - release notes

The compressed source code can be found on the ftp server in /HDF/HDF_Current/tar. An uncompressed version of the source code can be found in /HDF/HDF_Current/unpacked.

To compile and install the HDF libraries from the source code, please read through the READ and INSTALL files in the top directory of the source code. In general, these are the steps you would take to compile and install HDF:

```
./configure -v
make &gt;&amp; comp.out
make test &gt;&amp; test.out
make install
```

3.2 Installing the HDF library

How do you install HDF on your computer system? Detailed directions for configuring and installing the latest version of HDF can be found in the README and INSTALL files located in the HDF_Current unpacked subdirectory of the NCSA anonymous ftp server (ftp://ftp.ncsa.uiuc.edu/HDF/HDF/HDF_Current/unpacked).

In order to use the HDF library through C and FORTRAN programs, the user's computer must have either a C or FORTRAN library linked with the HDF library.

For those users who wish to work with HDF using Java, Version 2.3 of the HDF Java Products has been released as part of the latest release of the HDF library. Included in these products is the Java HDF Interface (JHI) for the HDF library. The JHI provides an interface to all the functions of the HDF library and may be used by any Java application to work with HDF files.

The necessary Java source code must be downloaded from ftp://ftp.ncsa.uiuc.edu/HDF/HDF/HDF_Current/unpacked/java-hdf.

These are the only languages which can call HDF routines (more detailed information in "Programming languages supporting the HDF library).

3.3 Computer platforms supporting the HDF library

The latest version of the HDF library is HDF 4.1 Release 3. Although the list of machines supported by the HDF library increases with every incremental version or release of HDF, it is still not possible to work with HDF files on every single platform or operating system. As of the current release in July 1999, the HDF library is currently supporting the following computer platforms and operating systems:

Sun4 (Solaris 2.6, SunOS 4.1.4)
SGI-Indy (IRIX v6.5)
SGI-PowerChallenge
SGI-Origin (IRIX64 v6.5-64/n32)
HP9000/735 (HP-UX 9.03)
HP9000/755 (HP-UX B.10.20)
Exemplar (HP-UX A.10.01)
Cray T90 (CFP, IEEE)
Cray C90
IBM SP2 (v4.2.1)
DEC Alpha/Digital (Unix v4.0)
DEC Alpha/OpenVMS (AXP v6.2 and 7.1)
VAX Open/VMS (v6.2)
IBM PC-Intel Pentium (Solarisx86, Linux (elf), FreeBSD)
PowerPC (C only)
PCs with Windows NT/95
Windows NT/95
T3E (unicosmk 2.0.2.16)

As of July 1999 and the latest release of the HDF library (4.1r3), the only platforms that support the Java HDF interface (JHI) are:

Sun4 (Solaris 2.5)
SGI-Indy (IRIX5.3)
IBM PC - Intel Pentium (Solarisx86 (2.5) and Linux (elf) 2.0.27)
Windows NT/95

Earlier versions or releases of the HDF library can still be used but may not be compatible with the platforms listed above.

3.4 Programming languages supporting the HDF library

As of the current release of HDF (HDF 4.1r3), the only programming languages which are supported by the HDF library are C and FORTRAN. Although the HDF library code is only written in C, the library provides both a FORTRAN and Java Interface which converts the code to C and allows the user to call the HDF routines and applications inside FORTRAN programs and Java scripts. This conversion will automatically take and requires no further action by the user.

Other then the obvious differences between the programming languages, the main difference between using the different languages is the naming convention, or names used for each HDF function. In addition, to use and compile HDF application routines through C programs, an HDF header file (hdf.h) containing

standard HDF data type and file access code (i.e. read, write) definitions, declarations and prototypes for the API routines must be called or included (#include "hdf.h") at the beginning of the program. These header files are not permitted in all FORTRAN versions and the needed information must be written into the FORTRAN code (taken from the HDF library file "constants.f" within "hdf.h").

One of the features of the HDF library is that it creates free format FORTRAN include files during the "make" process on UNIX platforms. This allows FORTRAN 90 programs to use HDF include files. The FORTRAN 90 files are designated by the ".f90" file extension.

Another recent update to the HDF library is the inclusion of the Java HDF Interface (JHI) as part of HDF version 4.1r3. The JHI provides an interface to all HDF functions and must be obtained and installed in order to use Java to work with HDF files. Please see "Obtaining and installing the HDF library" for further details.

3.5 Compiling the HDF library

The following examples (for UNIX platforms) illustrate the general method of compiling the HDF library and application programs:

C programs

cc -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN programs

f77 -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

Specific examples for various platforms are provided below. If the platform you use is not listed, the general instructions should be followed.

The latest platform related information can be found on the NCSA anonymous ftp server at HDF4.1r3/release_notes/compile.txt.

3.6 INSTRUCTIONS FOR SPECIFIC PLATFORMS

3.6.1 Cray C90 or YMP:

C:

cc -O -s -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

cf77 -O 1 -s -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.2 Dec Alpha/Digital Unix:

C:

cc -Olimit 2048 -std1 -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.3

To compile your programs, prog.c and prog1.for, with the HDF library, mfhdf.olb, df.olb, and libz.olb are required. The libjpeg.olb library is optional.

cc/opt/nodebug/define=(HDF,VMS)/nolist/include=<dir for include> prog.c

fort prog1.for

link/nodebug/notraceback/exec=prog.exe prog.obj, prog1.obj, -<dir for lib>mfhdf/lib -<dir for lib>df/lib, <dir for lib>libjpeg/lib, -<dir for lib>libz/lib, sys$library:vaxcrtl/lib

NOTE: The order of the libraries is important: mfhdf.olb first, followed by df.olb then libjpeg.olb and libz.olb.

3.6.4 Exemplar:

C:

cc -ext -nv -no <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

fc -sfc -72 -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.5 FreeBSD:

C:

gcc -ansi -Wall -Wpointer-arith -Wcast-qual -Wcast-align -Wwrite-strings -Wmissing-prototypes -Wnested-externs -pedantic -O2 -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -O -o <your program> <your program>.f -I<path for hdf include

directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.7 HP - UX:

C:

cc -Ae -O -o <your program> <your program>. -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -O -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.8 IRIX 5.3:

C:

cc -ansi -O -s -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -O -s -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.9 IRIX 6.x with 64-bit mode:

C:

cc -ansi -64 -mips4 -O -s -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN

f77 -64 -mips4 -O -s -o <your program> <your program>.f -I<path for hdf include directory>\ -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.10 IRIX 6.x with n32-bit mode:

C:

cc -ansi -n32 -mips3 -O -s -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -n32 -mips3 -O -s -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.11 Linux A.OUT And ELF:

C:

gcc -ansi -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN (a.out only):

f77 -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.12 Solaris:

The -lnsl is necessary in order to include the xdr library.

C:

cc -Xc -xO2 -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz -L/usr/lib -lnsl

FORTRAN:

f77 -O -o <your program> <your program>.f -I<path for hdf include directory>-L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz -L/usr/lib -lnsl

3.6.13 Solaris_x86 (C only):

The -lnsl is necessary in order to include the xdr library.

gcc -ansi -O -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz -L/usr/lib -lnsl

3.6.14 SP2 (AIX):

C:

xlc -qlanglvl=ansi -O -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -O -o <your program> <your program>.f -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.15 SunOS:

C:

gcc -ansi -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

FORTRAN:

f77 -f -o <your program> <your program>.f -I<path for hdf include directory>-L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.16 t3d:

C (only):

cc -Tcray-t3d -X1 -o <your program> <your program>.c -I<path for hdf include directory> -L<path for hdf libraries> -lmfhdf -ldf -ljpeg -lz

3.6.17 VAX OpenVMS:

To compile your programs, prog.c and prog1.for, with the HDF library, mfhdf.olb, df.olb, and libz.olb are required. The libjpeg.olb library is optional.

cc/DECC/STANDARD=VAXC/opt/nodebug/define=(HDF,VMS)/nolist/include=-<dir for include> prog.c

fort prog1.for

link/nodebug/notraceback/exec=prog.exe prog.obj, prog1.obj, -<dir for lib>mfhdf/lib -<dir for lib>df/lib, <dir for lib>libjpeg/lib, -<dir for lib>libz/lib, sys$library:deccrtl/lib

NOTE: The order of the libraries is important: mfhdf.olb first,followed by df.olb then libjpeg.olb and libz.olb.

3.6.18 Windows NT / 95:

Using Microsoft Visual C++ version 4.x:

Under Tools->Options, select the folder, Directories:
Under "Show directories for", select "Include files".
Add the following directories:

C:\MSDEV\INCLUDE

C:\MSDEV\MFC\INCLUDE

C:<path to HDF includes>\INCLUDE

Under "Show directories for", select "Library files":
Add the following directories:

C:\MSDEV\LIB
C:\MSDEV\MFC\LIB
C:<path to HDF libs>\LIB

Under Build->Settings, select folder, Link:

Add the following libraries to the beginning of the list of Object/Library Modules

libsrc.lib src.lib jpeg.lib zlib.lib xdr.lib getopt.lib

The following libraries may (or may not) need to be included

kernel32.lib user32.lib gdi32.lib winspool.lib comdig32.lib advapi32lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib

Under Build->Settings, select folder C/C++:
For the Preprocessor Definitions add: INTEL86

The following were already there: WIN32,_CONSOLE

4. Methods of Working with HDF Files

There are four basic ways or methods of working with (including reading and writing) HDF files. These include two levels of programming interfaces within the HDF library, a set of command line utilities also contained in the HDF library, and a wide range of browsing and visualization software provided by both commercial vendors and non-profit organizations (NCSA, for example). Further detail on each method is given below:

- Low-level interface
- High-level interface (APIs)
- Command line utilities
- HDF browsing and visualization tools

Both the command line utilities and the browsing and visualization tools provide easy-to-use methods for HDF non-experts to work with HDF files. As shown below, the use of the command line utilities is rather straightforward. However, neither the command line utilities nor tools provide the user with the flexibility and means of working with the HDF files in such an encompassing fashion as permitted in the High-level APIs. For this reason, as well as the fact that information and directions regarding the use of the HDF tools are better provided by the Internet sites indicated in section 4.4, the following sections of the tutorial will mainly concentrate on using the APIs to work with HDF files

4.1 Low-level interface

The low-level interface is mainly reserved for expert HDF programmers and software developers who are interested in not only reading and writing HDF files, but also such features as error handling, memory management, and storage. A lot of the features in this interface are unnecessary for the novice HDF user. Another drawback is that routines/operation callable through this interface are only available in C and not FORTRAN.

4.2 High-level interface (APIs)

In this interface, Application Programming Interfaces (APIs) are specifically tailored for each type of data (Images, Scientific Data arrays, etc.) supported by the HDF library. These APIs are callable routines that will allow the user to access, read and write HDF files specifically for the type of data they are interested in. Although it is necessary for the call of these APIs and associated routines to occur in either a C or FORTRAN program, the programming is usually limited to a set of call statements that access, open, operate (read, write, etc.), and terminate. All of the rest is taken care of by the interface itself. With its' availability in both C and FORTRAN, the minimal amount of programming necessary, and the independent APIs and routines for each data type, the High-level interface provides a relatively simple way for the average programmer and novice HDF users to work with HDF files.

4.2.1 Available APIs:

Through the High-level interface, the HDF library provides APIs and associated routines for all the data types supported by HDF. This includes 8- and 24-bit raster images, palettes, scientific data arrays, metadata (Annotation),

multivariate data stored as tables (Vdatas), and EOS Scientific Data (point, swath, and grid APIs contained in the HDF-EOS sub-library). In addition, there are separate APIs and routines for multi-file data sets of the various types. Each API is independent of the others and is identified by a certain prefix (different for both the FORTRAN and C program version) which is assigned to all the function calls employed by the user in his/her program for that specific data type.

The following is a list and short description of the various APIs with the C and FORTRAN prefaces for each interface given in parenthesis:

4.2.1.1 *MULTIFILE APIs*

SD API (SD/sf): For scientific data sets (multi-dimensional arrays together with a record of dimension and number type). The SD API is used to store, manage and retrieve multi-dimensional arrays (integer or floating point decimal), including their dimensions and attributes in more than one file.

GR API (GR/mg): The GR API is used to store, manage and retrieve general raster image data sets, including their dimensions and palettes. However, unlike the DF24 and DF8 APIs, this information can be in more than one file. In addition, the GR API can also manage unattached palettes.

VS API (VS/vsf): The VS API is used for reading and writing customized tables which are stored in fixed length fields (Vdata).

V API (V/vf): The V API is used to create, group, and manipulate primary HDF objects in a file (Vgroup).

VSQ API (VSQ/vsq): The VSQ API is used for querying or obtaining information on vdatas. This includes the number of records, names, and number types.

VF API (VF/not available): The VF API can be used for obtaining information on the fields in an existing vdata.

AN API (AN/af): The AN API is used to store, manage, and retrieve text strings used as metadata to describe the data file itself or any of the data elements inside the file.

4.2.1.2 SINGLE FILE APIs

DFSD API (DFSD/ds): The DFSD API is similar to the SD API, but only operates on one single file.

DFR8 API (DFR8/d8): The DFR8 API is used to store, manage and retrieve 8-bit raster images, along with their dimensions and color palettes. This information is all included in one file.

DF24 API (DF24/d24): The DF24 API is used to store, manage and retrieve 24-bit raster images, including the dimensions of the image. This information is also included in one file.

DFP API (DFP/dp): The DFP API is used to store and retrieve 8 bit palettes in one file.

DFAN API (DFAN/da): The DFAN API is used for reading and writing text string (metadata) assigned to HDF files or objects.

4.2.1.3 *HDF-EOS APIs*

PT API (PT/pt): The PT API is used for storing, retrieving, and manipulating data in point data sets. These data have associated geolocation information, but are not organized in a spatial or temporal fashion. The PT API is part of the HDF-EOS sub-library.

GD API (GD/gd): The GD API is used for storing, retrieving, and manipulating data that has been stored in a rectilinear array based on a defined map projection. The GD API is part of the HDF-EOS sub-library.

SW API (SW/sw): The SW API is used for storing, retrieving, and manipulating time-ordered data sets such as satellite swath data. The SW API is part of the HDF-EOS sub-library

Rather then providing material on all of the high-level interface APIs, we have chosen in this tutorial to use the Multi-dimensional array, multi-file interface (SD API) as an example to teach the novice how to use HDF. For detailed information on the other APIs, all of which are used in similar fashion to the SD API, the reader of the tutorial is directed to the documentation identified in Section 2 - Where can I get additional and detailed information on HDF?

4.3 Command line utilities

One method of working with HDF files is through command line utilities during a UNIX terminal session. Command line utilities allow the user to call up HDF application programs outside of formal C and FORTRAN programs.

4.3.1 List and description of command line utilities

The following is a list of command-line utilities available in the HDF library:

1) hdp - displays contents and data objects within an HDF file
2) hdf24to8 - converts 24-bit raster images to HDF 8-bit images
3) hdf8t025 - converts 8-bit raster images to HDF 24-bit images
4) hdfcomp - re-compresses an 8-bit raster HDF file
5) hdfls - lists basic information about an HDF file
6) hdfpack - compacts an HDF file
7) hdfunpac - unpacks an HDF file
8) hdftopal - extracts a palette from an HDF file
9) hdftor8 - extracts 8-bit raster images and palettes from an HDF file
10) hdfed - HDF file editor
11) paltohdf - converts a raw palette to HDF
12) r8tohdf - converts 8-bit raster image to HDF
13) ristosds - converts a series of raster image HDF files into an HDF file
14) vshow - dumps out vsets from an HDF file
15) jpeg2hdf - converts jpeg images to HDF raster images
16) hdf2jpeg - converts HDF raster images to jpeg images
17) hdfrseq - play an animation sequence through NCSA/BYU telnet
18) vmake - create Vset structures from ASCII text

The hdp command line utility is a very helpful operator, especially for the average HDF user. HDP can list the contents of HDF files at various levels and with different details. It can also dump the data of one or more specific objects in the file.

Although the command line utilities permit the user to perform common operations on HDF using a simple one-line command, the main drawback to this method is the limited number of operations supported.

4.4 HDF browsing and visualization tools

A complete and current listing of the browsing and visualization tools that can work with HDF files is provided by NCSA on their HDF home page (http://hdf.ncsa.uiuc.edu/tools.html). There are both publicly available (free) and commercial software packages that can be used to work with HDF files. A summary of some of the more useful and commonly used software, including the address of the Internet site/home page where the software may be accessed, is provided below:

4.4.1 Publicly Available Software

Freely available software for viewing and browsing HDF files have been developed by both NCSA and various other institutes, science or data centers, and businesses. We have broken these tools down into three categories:

4.4.1.1 Current NCSA Tools

The following are the most current and commonly used tools developed by NCSA for viewing and browsing all types of HDF files:

1. The NCSA Java-based HDF Viewer (JHV)(http://hdf.ncsa.uiuc.edu/java-hdf-html/jhv/) - Java based tool that allows the user to view the contents of an HDF file.
2. The HDF WWW Scientific Data Browser (http://hdf.ncsa.uiuc.edu/sdb/sdb.html) - an interface program that reads HDF files by accessing the HDF library and can visualize or format the data (in HTML) on the web.
3. The Java HDF Server (JHS) (http://hdf.ncsa.uiuc.edu/java-hdf-html/jhs/) - The java based program that calls the HDF library through the Java interface and can access remote HDF files.

4.4.1.2 Older NCSA Tools (not updated to run with latest version of HDF)

Although not updated to run with the current release of HDF (HDF 4.1r3), the following tools may still be used to work with HDF files. All of these tools are available from the NCSA anonymous ftp server (ftp://ftp.ncsa.uiuc.edu/Visualization/)

1) NCSA Collage - Collaborative visualization program
2) NCSA Mosaic -Browsing on UNIX
3) NCSA Polyview - Visualization and analysis of HDF files
4) NCSA Reformat - Converts HDF files
5) NCSA X DataSlice - Manipulates 3-D images
6) NCSA X Image

4.4.1.3 Non-NCSA Tools

The following tools have been developed independently from NCSA, but are still available in the public domain:

1) The Data and Dimensions Interface (DDI) (http://www-pcmdi.llnl.gov/williams/ddi/ddi.html) - Can extract, read, write and visualize large data sets in HDF format.

2) Envision (http://www.atmos.uiuc.edu/envision/envision.html) - Interactive system that provides for the management and visualization of large data sets in HDF format.

3) HDF Browser (http://www.fortner.com/docs/product_hdf_b.html)- Created by Fortner Research to provide point-and-click access to data stored in HDF. This includes viewing the data stored in arrays, images, etc. and editing HDF files.

4) hdfv (http://www.blueneptune.com/~yotam/hdfv.html)- An HDF read-only interface that is an HDF viewer with a GUI. Only supports vgroup/Vdata data types.

5) LinkWinds (http://linkwinds.jpl.nasa.gov) - A visual data analysis and exploration system designed to rapidly and interactively investigate large multivariate data sets (including HDF and HDF-EOS format).

6) SHARP (http://cimss.ssec.wisc.edu/~gumley/sharp/sharp.html)- A viewer for MODIS Airborne Simulator (MAS) HDF data.

7) ScaiAN (http://www.scri.fsu.edu/~lyons/scian)- Scientific visualization and animation package.

8) VCS (http://www-pcmdi.llnl.gov/software/)- Facilitates the selection, manipulation and display of scientific data and supports the HDF format for both reading and writing.

9) EOSView (http://edhs1.gsfc.nasa.gov/waisdata/toc/tp4450601toc.html) - An HDF file verification tool that allows the display of most HDF and HDF-EOS data types.

10) The Data and Information Access Link (DIAL) (http://dial.gsfc.nasa.gov/)- A server which provides tools for the searching, browsing, and visualizing of HDF and HDF-EOS files through the WWW.

11) HDFLook (http://loasys.univ-lille1.fr/Hdflook/hdflook_gb.html) - A viewer used to access and view HDF and HDF-EOS files, particularly raster images and scientific data sets.

12) IRI/LDEO (http://ingrid.ldgo.columbia.edu/) - A climate data library that helps in the writing of HDF files and the management of data sets.

13) Webwinds (http://webwinds.jpl.nasa.gov/) - A platform independent system written in java that acts as an interactive visualization tool for data in HDF and HDF-EOS format.

14) view_hdf (http://eosweb.larc.nasa.gov/HPDOCS/view_hdf.html) - A visualization tool developed by NASA LARC that provides for the viewing, plotting, and manipulation of HDF datasets

4.4.2 Commercial Software

Below is a partial list of some of the more powerful and more commonly used software packages for working with HDF files:

1)AVS5/AVSExpress (http://www.avs.com/products/index.htm) - Can read and write files in HDF format. Also includes a suite of data visualization and analysis techniques/tools (3-D visualization, plots, etc.).

2)Data Explorer (http://www.research.ibm.com/dx)- General-purpose software package for data visualization and analysis. The data may be imported from HDF format.

3) IDL (http://www.rsinc.com/idl/index.cfm) - software package for the analysis and visualization of data. Includes advanced image processing, interactive 2-d and 3-D graphics, and flexible date input/output.

4) Noesys (http://www.fortner.com/noesys)- A desktop software program specifically designed to easily access, view, analyze and archive data in the HDF format.

5) Plot (http://www.fortner.com/docs/product_plot.html)- A package that can read, analyze and plot HDF data sets of column data using Windows, Macintosh and UNIX.

6) HDF Explorer (http://www.mind.pt/hdf.explorer) - A visualization program that reads and views data sets in HDF format.

4.4.3 Contributed Software

In addition to the above-mentioned software, also available from the NCSA anonymous ftp server is a collection of software routines and utilities developed by HDF users who wish to share their knowledge and work with the HDF community. This software can be found in the directory pub/hdf/contrib/ of the anonymous ftp server (ftp://ftp.ncsa.uiuc.edu/HDF/HDF/contrib/). Most of these "contributed" routines were developed with specific platforms and operating systems in mind. Below are a few examples:

1) readDF - reads HDF files into IRIS Explorer
2) fits2hdf - converts FITS files (another format) into HDF
3) iristohdf - converts SGI image format to HDF format
4) hdfxdis - directly displays HDF image on an X-server

These routines together with the name and address of the developer are free and publicly available to all interested users of HDF

5. SD API

The SD (Scientific Data) API is a collection of callable (from C or FORTRAN programs) routines which will allow the user to, among other operations, create, write, and read HDF files containing multi-dimensional arrays of scientific data. In subsequent sections, we will show how the SD API can be used for reading and writing HDF data sets. For a complete listing of all the operations permitted in the SD API, please see the HDF 4.1r3 User's Guide (ftp://ftp.ncsa.uiuc.edu/HDF/HDF/Documentation/HDF4.1r3) As will be demonstrated shortly, FORTRAN and C routines in the SD API begin, respectively, with the prefix "sf" or "SD". Data within a scientific data set may be of the floating real or integer type. In HDF, and in the SD API, a scientific data set (or SDS) must consist of a multi-dimensional array (called a SDS array), together with information on data type and dimension record. The SD API allows the user to work simultaneously with more than one multi-dimensional scientific data set (SDS) while the DFSD API is restricted to one multi-dimensional array.

5.1 SDS Array

The SDS array is the actual data itself, an n-dimensional array which contains the floating point or integer values. Each SDS array has an SDS name (series of alphanumeric characters) that can either be assigned by the calling statement with the FORTRAN or C program or automatically assigned by the HDF library when the new data set (if writing) is created.

5.2 Data Type

The SD API supports the following data types:

1) 32-bit floating point
2) 16-bit floating point
3) 8-bit signed integers
4) 16-bit signed integers
5) 32-bit signed integers
6) 8-bit unsigned integers
7) 16-bit unsigned integers
8) 32-bit unsigned integers
9) Variable bit integers and floating point decimal values

As described later, the data type is defined in the accessing/creating function call statements within the C and FORTRAN programs.

5.3 Dimensions

The dimensions of a SDS array identify the shape and size of the array in question. This includes the rank of the dimensions, which in HDF speak refers to the number of dimensions. One innovative feature of HDF is that one, and only one, dimension of a SDS array may be of unlimited size and referred to as an unlimited dimension.

5.4 Optional information

When writing or creating an HDF file, the user may also wish to include information regarding the data set or array. This must be done in the calling functions of the C or FORTRAN programs.

Attributes, either predefined by NCSA or user-defined, are text strings which provide metadata about the file, data set, or dimension of interest. This includes information on what is in the file or individual SDS arrays, and how the maker of the file/data intends for the data to be used or viewed. Like most of the other routines mentioned above, attributes are defined in the function calls of the program. Attributes are further covered in section 6.

6. Attributes and Metadata

The HDF library allows for several ways for the user to provide metadata (data about data) information for the HDF file or data set to be written or read. This information is not a requirement for HDF files. The most commonly used method or routine within the HDF library for providing metadata are "Attributes" or text-strings which describe the HDF file, data set (SDS array) or dimensions. There are two types of attributes used in HDF that can be defined in the user's calling program:

- User-defined attributes
- Predefined attributes

Both the predefined and user-defined attributes may be accessed using the general attribute routines for user-defined attributes provided by the HDF library. On the other hand, the predefined attributes may only be accessed using the routines specifically tailored for the predefined attributes (see above). As a result, in later sections, we will focus on using the general attribute routines developed for user-defined attributes

6.1 User-defined attributes

User-defined attributes are optional information that can be given and attached to HDF files, scientific data sets, and dimensions. They are referred to, respectively, as file attributes, array attributes, and dimension attributes. These attributes are at the discretion of, and to be defined by, the user.

The SD interface uses the same functions to access all of the three types of attributes, with the difference being the use and definition of the different identifiers (i.e., file ids for file attributes, SDS ids for array attributes, and dimension ids for dimension attributes). After the proper identifier is obtained, the user can then create and define his attribute (labels, formats, coordinate system, etc.)

More on user-defined attributes and how to define them is provided in Section 7: Writing an HDF File.

6.2 Predefined attributes

Predefined attributes are attributes that use previously defined or reserved labels and data types. While the user-defined attributes must be defined by the user, the predefined attributes need not be defined and are already understood by the HDF library. However, predefined attributes can only be assigned to scientific data sets (SDS) and dimensions (not files, like is possible with user-defined attributes).

There are seven main predefined attributes:

1) For labels: long_name
2) For units: units
3) For formats: format
4) For coordinate systems: cordsys
5) For Value ranges: valid_range
6) For Fill values: _FillValue

7) For Calibration: scale_factor
scale_factor_err
add_offset
add_offset_err
calibrated_nt

The predefined attributes can be accessed by the SD interface in the same general fashion as the user-defined attributes or by using routines developed specifically for the predefined attributes. The ""general"" attribute routines are recommended in most cases.

7. Writing an HDF File

The following sections detail how a user may utilize the HDF library and various APIs within a computer program to write a data file in HDF. As a teaching tool, this tutorial will concentrate on using the FORTRAN programming language and the SD API. However, examples of the appropriate C code will also be given for certain steps.

- Does the current version of HDF support your computer platform?
- Downloading and installing of the HDF library
- Are all libraries and programs properly linked and compiled
- Writing a short program to write data in HDF

7.1 Does the current version of HDF support your computer platform?

As outlined in Section 3, the HDF library can not be run on just any available computer platform or operating system. Before downloading the HDF library software, the user should make sure that the current release of HDF supports his/her computer and operating system. Otherwise, the user will be unable to work with the HDF library and files. There is also a possibility that previous releases of HDF may support the Users computer platform while the latest version does not. In this event, the user may wish to obtain the earlier software.

7.2 Downloading and Installing of the HDF library

The HDF library and software is public domain software and available free to all users. The library and code can be downloaded from the NCSA anonymous ftp server (ftp://ftp.ncsa.uiuc.edu/HDF/HDF/HDF4.1r3). Directions on how to install the HDF library can also be found at this location.

7.3 Are all libraries and programs properly linked and compiled?

In order to run the HDF software, the library and the needed application routines and programs must first be properly compiled and linked. As of the current release of HDF (4.1r3), four separate libraries must be compiled and linked. These are the libmfhdf.a, libdf.a, libjpeg.a, and libz.a libraries. Provided below are examples of the command(s) that can be used for this action. It must be noted that the order in which the libraries are linked is important and should not vary from the order shown below:

For **C** programs:

```
cc  -o &lt;your program&gt; &lt;your program&gt;.c \
    -I&lt;pathf for hdf include directory&gt; \
    -L&lt;path for hdf libraries&gt; -lmfhdf -ldf -ljpeg -lz
```

For **FORTRAN** programs:

```
f77 -o &lt;your program&gt; &lt;your program&gt;.f \
    -I&lt;path for hdf include directory&gt; \
    -L&lt;path for hdf libraries&gt; -lmfhdf -ldf -ljpeg -lz
```

For the various commands needed to link and compile the HDF library on each individual platform, please see **Section 3 - Compiling the HDF library.**

7.4 Writing a short program to write data in HDF

The following steps (some which are rather simple and common sense) should be addressed by the user before, during or after the creation of the calling program to be used. Each step will be discussed in further detail in the sections that follow.

1) Select a programming language
2) Make sure all include files are in place
3) Make all variable and parameter declarations
4) Open file containing existing non-HDF data set and store in array
5) Initialize access to the SD interface and open new HDF file
6) Define characteristics of new HDF data set(s)
7) Write existing data set/array to a new data array in a new HDF file
8) Optional operation: Provide metadata for HDF files or data sets
9) Terminate / close access to all files, data sets, and APIs
10) Execute program

7.4.1 Select a programming language

As mentioned previously, the HDF library and programs can only be run by using either the C or FORTRAN programming language. This choice is up to the user depending on availability and the language he or she feels most familiar and comfortable with. All SD API routines which allow the user to work with scientific data sets (SDS) either have the "sf" prefix (FORTRAN) or the "SD" prefix (C). Examples of the routines used to open, create, read, and write SDS are given in the following sections.

7.4.2 Make sure all include files are in place

In section 3 - The HDF Library: Software and Hardware, it was noted that a series of standard HDF definitions and declarations of file access codes (i.e. read, write, etc.) and data types (i.e. integer, character) must be included within the user's programs. In the C programs, this is accomplished simply by adding the line #include "hdf.h" at the beginning of the program. This line effectively includes all the needed constants and definitions from the HDF software. When writing FORTRAN programs, this may also be done by simply adding an include statement that brings in only the needed definitions and declarations (constants.f) from the hdf.h header file. This is done by the following code: "include constants.f". However, all FORTRAN compilers (particularly the older ones) do not support the use of include statements. In this event, the user must type in/declare all the constants and definitions found in the constants.f file. It is advised that all declarations, whether through include statements or not, should be done at the beginning of the program.

Example:

FORTRAN:

```
C DFACC_RDONLY is defined in hdf.h
C if not available for FORTRAN then add
      Parameter (DFACC_RDONLY=1
```

C:

```
#include "hdf.h"
```

```
main() {
…}
```

7.4.3 Make all variable and parameter declarations

As with any program, the scientist/user should declare and initialize all variables and parameters at the beginning of the program. This includes all variables and arguments that will be used by the HDF commands to follow. The variable and parameter declarations needed for each call will be provided in the example boxes of the individual steps. These statements always belong at the top of the program.

7.4.4 Open file containing existing non-HDF data set and store in array

Before writing any data into HDF, the actual data first has to be accessed within the program. As is normally done in non-HDF applications, the file containing the data that the user wishes to convert into HDF must first be opened. After opening the file, the user reads and stores the data into a multi-dimensional array that can be accessed by the HDF commands.

For the purpose of this tutorial, the non-HDF data set will be read from an existing file called wind.dat into a multi-dimensional real array called rwind (XL, YL) where XL= 30 and YL = 30.

Example:

C:

```
main() {
FILE *infile;
const int
XL = 30
YL = 30;
int i, j;
float rwind[XL][YL];

infile = fopen("wind.dat", "r");
for(i=0; i&lt;XL; i++)
 for(j=0; j&lt;XL; j++)
fscanf(infile, "%f", rwind[i][j]);
}
```

FORTRAN:

```
real rwind(30,30)
XL = 30
YL = 30

Open(unit=15, file='wind.dat',form='formatted')
Do I=1,XL
 Do j=1,YL
  Read(15,25)rwind(I,J)
 Enddo
Enddo
```

7.4.5 Initialize access to the SD interface and open new HDF file

The first real HDF programming step actually accomplishes 2 things:

1) Creates and opens a new HDF file
2) Initializes and opens the SD interface.

This is done by the following command:

```
sd_id = sfstart(filename, access_mode) (FORTRAN)

                or

sd_id = SDstart(filename, access_mode); ( C )
```

where;

sd_id = HDF file id returned by the sfstart/SDstart command
filename = the name of the new HDF file (character string)
access_mode = Type of access required for this file

All available options for the access-mode argument are defined in the hdf.h header file mentioned previously and need only to be identified for all C and most FORTRAN operations. All options begin with the prefix "DFACC_" and include:

DFACC_CREATE (File Creation Access)
DFACC_RDONLY (Read Access)
DFACC_RDWR (Read and Write Access)

As mentioned previously, these definitions are stated in the hdf.h header file.

In the event that the user's FORTRAN compiler can not handle include statements such as those found in the hdf.h header file, the DFACC_ variable must be defined, along with its assigned value, at the beginning of the program. This is done by a line of code such as:

```
parameter (DFACC_RDONLY = 1) (For FORTRAN only)
```

For the purpose of this tutorial, the new HDF file will be called wind.hdf.

Example:

FORTRAN:

```
integer*4 sd_id
integer sfstart
parameter (DFACC_CREATE = 4)
sd_id = sfstart(wind.hdf, DFACC_CREATE)
```

C:

```
#include "hdf.h"
/* Includes all the access_mode definintions */
int32 sd_id;
```

```
sd_id = SDstart(wind.hdf, DFACC_CREATE);
```

7.4.6 Define characteristics of new HDF data set(s)

After initializing the SD interface and opening and assigning a file id (sd_id) to the HDF file to be used, the next step is to define a new HDF Scientific Data Set (SDS) to which the existing non-HDF data will be written. This is done by the following command:

```
sds_id = sfcreate (sd_id, name, number_type, rank, dim_sizes) (FORTRAN)

                                or

sds_id = SDcreate (sd_id, name, number_type, rank, dim_sizes); ( C )
```

It should be noted that sfselect/SDselect may also be used to write to a previously defined HDF data set.

Where

```
sds_id = HDF SDS array id returned by the sfcreate/SDcreate command
sd_id = the new HDF file id created in the previous step (sfstart/SDstart)
name = name of new SDS (in ASCII character string)
number_type = data type of data set
```

This argument always takes the form of DFNT_X, where X is the data type to be used. A list of all the data types supported by the API can be found in the HDF User's Guide. For most of the data types, 8,16,32 and 64-bit types are supported. A few of the available options are provided below:

```
DFNT_FLOAT for Floating Point Reals
DFNT_DOUBLE for Double Precision Reals
DFNT_CHAR for Character
DFNT_INT16 for 16-bit Integer Type
DFNT_UINT16 for 16-bit Unsigned Integer Type
DFNT_INT32 for 32-bit Integer Type
DFNT_UINT32 for 32-bit Unsigned Integer Type
```

Similar to the DFACC_ argument, all data types are defined in hdf.h. Once again, for FORTRAN compilers unable to access these include files, the DFNT_ argument, and its' assigned value, must be defined at the beginning of the program using code like this:

```
 parameter (DFNT_INT16 = 22) (taken from constants.f within the hdf.h file)
```

rank = number of dimensions in array to be written (integer)

This value is best specified at the beginning of the program along with the other various declarations. This can be done with a simple line of code:

```
rank = 2, 3,....
```

dim_sizes = An array defining the size of each dimension of the data array (integer)

As with the "rank" argument, this variable is best specified with the other variable declarations at the top of the program. In FORTRAN, an example for a 2-D, 30 X 30 array would be:

```
dimsizes(1) = 30          (FORTRAN)
dimsizes (2) = 30

          or

dimsizes[0] = 30;         ( C )
dimsizes[1] = 30;
```

EXAMPLE: For an existing data set to be written as a 2-D array of 30 (x direction) by 30(y direction), and as an 8-bit integer type, the following commands need to be used:

```
rank = 2                  (FORTRAN)
dimsizes(1) = 30
dimsizes(2) = 30
sds_id = sfcreate(sd_id, newarray_1, DFNT_INT8, rank, dimsizes)

                                  or

rank = 2;                 ( C )
dimsizes[0] = 30;
dimsizes[1] = 30;
sds_id = SDcreate(sd_id, "newarray_1", DFNT_INT8, rank, dimsizes);
```

Example:

FORTRAN:

```
        integer*4 DFNT_INT16
        integer sds_id, rank
        integer dims(2), sfcreate

        rank = 2
        XL = 30
        YL = 30
        dims(1) = XL
        dims(2) = YL
        sds_id = sfcreate(sd_id, winds, DFNT_INT16, rank, dims)
```

C:

```
int32 sds_id;
int32 dims[2], rank;
rank = 2;
XL = 30;
YL = 30;
dims[0] = YL;
dims[1] = XL;
sds_id = SDcreate(sd_id, winds, DFNT_INT16, rank, dims);
```

7.4.7 Write existing data set/array to a new data array in a new HDF file

After initializing the API and defining the new HDF file and new HDF SDS to be written to, the next step is to actually write the existing non-HDF data into the HDF file by using the SDwritedata (sfwdata) command. This command is used to write either all or part of the existing n-dimensional data set (termed a "slab") into the sds_id array with the same number of dimensions. In addition, the size of each dimension of the data "slab" must be the same or smaller then the corresponding dimension of the sds_id. The SDwritedata/sfwdata command is used in the following fashion:

```
ret=sfwdata (sds_id, start, stride, edge, data) (FORTRAN)

                              or

ret=SDwritedata (sds_id, start, stride, edge, data); ( C )
```

It should be noted that there are two versions of the write routine in FORTRAN, "sfwdata" is used for numeric data while "sfwcdata" is used for writing character data

Where

sds_id = the SDS id (identifier) determined and returned by using SDcreate(sfcreate)

start = An array which identifies where in the SDS that the writing will begin

The start array identifies the location or position in the SDS where the writing of the data "slab" will begin. This array must have the same number of dimensions (rank) as the SDS and can not be larger (in each dimension) then the SDS array. The declaration of the start variables can be done at the top of the program or just preceding the call of the sfwdata (SDwritedata) command. As an example, to write the existing data set to the beginning of a new 2-dimensional SDS the following must be specified:

```
start(1) = 0            (FORTRAN)
start(2) = 0

            Or

start[0] = 0;            ( C )
start[1] = 0;
```

If the user wishes to begin writing the data at a location other then the beginning of the new data set, say at a first dimension (X) of 15, the declarations would be:

```
start(1) = 15            (FORTRAN)
start(2) = 0

            Or
```

```
start[0] = 15;            ( C )
start[1] = 0;
```

stride = An array specifying the interval between written values in each dimension.

The stride argument specifies, for each dimension, the interval between consecutive written values of the data set. In other words, how many array locations are skipped with each writing of the data? Like the start array, the stride argument is predefined before calling the sfwdata (SDwritedata) command, either directly before the call or at the top of the program.

If the user does not wish to skip any array locations in a new 2-dimensional SDS, the following is to be declared:

```
stride(1) = 1             (FORTRAN)
stride(2) = 1

              or

stride(0) = 1;            ( C )
stride(1) = 1;
```

However, if the user wishes to skip every other X (dimension 1) location, the following would be used:

```
stride(1) = 2             (FORTRAN)
stride(2) = 1

              or

stride(0) = 2;            ( C )
stride(1) = 1;
```

edge = An array defining the number of data values to be written in each dimension.

The edge array defines the number of data values/elements that will be written along each dimension of the multi-dimensional SDS array. In plain terms, this argument defines the size of the data slab (all or part of the data) to be written to the new SDS array and each dimension.

edge must be specified for each dimension of the data set and SDS array, and can not be larger then the entire length of the newly defined (from sfcreate) array it is being written to.

The edge is affected by the stride. If stride = 2, then the edge will need to be divided by two, because it will be writing to every other location along a dimension.

Similar to stride and start, the edge argument needs to be defined prior to the calling of the sfwdata (SDwritedata) command, whether it be at the top of the program or directly before the routine call.

As an example, most often, the user will wish to write the entire non-HDF data set into a new array that starts from the beginning and does not contain any missing data or blanks. For a 2-dimensional array of 30X30, read and stored into the data array "rwind", this can be done, in FORTRAN, by:

```
start(1) = 0
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 30
edge(2) = 30
retn = sfwdata(sds_id, start, stride, edges, rwind)
```

Or in C by:

```
Start[0] = 0;
Start[1] = 0;
Stride[0] = 1;
Stride[1] = 1;
Edge[0] = 30;
Edge[1] = 30;
retn = SDwritedata(sds_id, start, stride, edges, rwind);
```

data = The array or buffer of data to be written

The file containing this data should be opened at the beginning of the program and the data read in and stored into the necessary arrays before beginning the HDF operations.

Example:

FORTRAN:

```
        integer start(2), edges(2), stride(2)
        integer retn, XL, YL
        integer sfwdata
c     Define the location, pattern and size of data set that
c     will be written to.
        XL = 30
        YL = 30
        start(1) = 0
        start(2) = 0
        edge(1) = XL
        edge(2) = YL
        stride(1) = 1
        stride(2) = 1
c     write the data
        retn = sfwdata(sds_id, start, stride, edges, rwind)
```

C:

```
int32 retn;
int32 start[2], edges[2], stride[2];
XL = 30;
YL = 30;
/*Define the location, pattern and size of the dataset*/
For (i=0; i&lt;rank; i++) {
start[i] = 0;
edge[i] = dims[i];
edge(1) = 30;
/* Write the stored data to "newarray". The 5th argument must be
explicitly cast to a generic pointer to conform to the API
definition for SDwritedata */
retn = SDwritedata(sds_id, start, NULL, edges, (VCIDP)newarray);
```

7.4.8 Optional operation: Provide metadata for HDF files or data sets

Using the general attribute routines for user-defined attributes described in section 6, attributes can be written and attached to the file itself, the data set, and the dimension in question. This is not required, but up to the choice of the user.

After opening the file and obtaining the file id (sd_id) using the sfstart/SDstart command, the following can be done

7.4.8.1 FILE ATTRIBUTES:

To assign attributes to a file, the following command is used:

```
SDsetattr (sd_id,attr_name, data_type, count, value);        ( C )

sfsnatt(sd_id, attr_name, data_type, count, value)          (FORTRAN)
```

There are two FORTRAN versions of the routine, sfsnatt writes numeric attribute data while sfcatt writes character attribute data.

Where

```
sd_id= file identifier
attr_name = ASCII string containing the name of the attribute (i.e., "file
contents")
data_type = data type of attribute values (i.e., DFNT_INT32)
count = total number values/characters in the attribute
value = text string or label
```

7.4.8.2 ARRAY ATTRIBUTES

After each data set identifier (sds_id) is obtained through the SDselect/sfselect command, the following is used:

```
SDsetattr (sds_id, attr_name, data_type, count, value);        ( C )

sfsnatt(sds_id, attr_name, data_type, count, value)          (FORTRAN)
```

where

sds_id= data set identifier

rest as above

7.4.8.3 DIMENSION ATTRIBUTES

After getting the identifier for a dimension using the sfdimid/SDgetdimid command, the following is used:

```
SDsetattr (dim_id, attr_name, data_type, count, value);          ( C )

sfsnatt (dim_id, attr_name, data_type, count, value)          (FORTRAN)
```

where

dim_id= Dimension identifier

rest as above

7.4.8.4 CLOSING ATTRIBUTES

After setting/writing the attributes, the user must terminate access to the data array (using the SDendaccess/sfendacc commands) and the file and SD interface (using the SDend/sfend commands).

Example:

1) FILE ATTRIBUTES:

FORTRAN:

```
sd_id = sfstart("wind.hdf", DFACC_RDWR)
retn = sfsattr(sd_id, "Contents of file", DFNT_CHAR8, 16,
"horizontal         winds")
```

C:

```
sd_id=SDstart ("wind.hdf", DFACC_RDWR);
retn= SDsetattr (sd_id, "Contents of file", DFNT_CHAR8, 16,
"horizontal winds ");
```

2) ARRAY ATTRIBUTES

FORTRAN:

```
sds_id=sfselect (sd_id, 0)
retn = sfsattr(sds_id, "format", DFNT_INT32, 4, "F8.2")
```

C:

```
sds_id=SDselect(sd_id, 0);
retn= SDsetattr (sds_id, "format", DFNT_INT32, 4,
"F8.2");</PRE></DIR>
```

3) DIMENSION ATTRIBUTES

FORTRAN:

```
dim_id=sfdimid (sds_id, 0)
retn = sfsattr(dim_id, "dim_metric", DFNT_CHAR8, 10,
"meters/sec")</PRE>
```

C:

```
dim_id=SDgetdimid (sds_id,0);
retn= SDsetattr (dim_id, "dim_metric", DFNT_CHAR8, 10,
"meters/sec");</PRE></DIR>
```

7.4.9 Terminate / close access to all files, data sets, and APIs

After writing the data to the new SDS array within the new HDF file, it is necessary to terminate or close access to the new data set in order to prevent any possible loss of data. This is done by the following:

```
retn = sfendacc(sds_id)          (FORTRAN)

                    or

retn = SDendaccess(sds_id);      ( C )
```

In addition, the API called within the program must also be closed to prevent any data loss:

```
retn = sfend(sd_id)              (FORTRAN)

                    or

retn = SDend(sd_id);             ( C )
```

Example:

FORTRAN:

```
  integer sfendacc, sfend
  retn = sfendacc(sds_id)
  retn = sfend(sd_id)
```

C:

```
retn = SDendaccess(sds_id);
retn = SDend(sd_id);
```

7.4.10 Execute program

Execute like a normal FORTRAN or C program.

8. Obtaining Information on Existing HDF Files

As mentioned previously, a single HDF file may contain more than one scientific data set (or images, tables, etc.). Within the SD interface (and other interfaces for the various data types), there are routines that can be called within short programs, C or FORTRAN, which help the user do the following:

- Determine the contents of an HDF file
- Obtain information on individual data sets
- Locate a Scientific Data Set (SDS) by name

8.1 Determine the contents of an HDF file

Before reading an HDF file, it might be necessary for the user to determine the number of data sets within the file, and the attributes of the file itself. After initializing and accessing the Application interface (in this case, the SD and GR interfaces for, respectively, scientific data sets and images (with associated palettes), this can be done using the following statements:

```
SDfileinfo(sd_id, n_datasets, n_file_attr); (C)
GRfileinfo(gr_id,n_images,nfile_attr)
```

and

```
sffinfo(sd_id, n_datasets, n_file_attr)      (FORTRAN)
mgfinfo(gr_id,n_images,n_file_attr)
```

where

```
sd_id= file id number
gr_id= GR interface identifier
n_datasets= Number of data sets within the file
n_file_attr= number of file attribute
n_images= number of images within the file
```

8.2 Obtain information on individual data sets

Before reading a particular data set or image from an HDF file, the user may need to know the rank, dimension sizes, data type, and number of attributes of the data array.

After the user has initiated and accessed the interface (for example, the GR interface for images and the SD interface for data arrays) and selected the data set by using the sfselect/SDselect (data set) or mgselct/GRselect (image) in a short FORTRAN (C) program, this information can be retrieved using the following calls:

```
SDgetinfo (sds_id, name, rank, dim_sizes, num_type, attributes);        (C)
GRgetinfo(ri_id,name,n_comps,data_type,interlace_mode,dim_sizes,n_attrs)
```

and

```
sfginfo (sds_id, name, rank, dim_sizes, num_type, attributes)     (FORTRAN)
mgginf(ri_id,name,n_comps,data_type,interlace_mode,dim_sizes,n_attrs)
```

Where

```
sds_id = data set id number
ri_id = raster image id number
name = name of corresponding data set
rank = rank of corresponding data set
dim_sizes = dimensions of corresponding data set
num_type = data type of corresponding data set
data_type = data type of corresponding image
attributes = number of attributes of corresponding data set
n_comps = number of components
interlace_mode = interlacing mode of data
n_attrs = number of sttributes
```

8.3 Locate a Scientific Data Set (SDS) by name

In most cases, when a scientific data set is created, it is assigned a unique identification number (or sds_id) so that it can be located and accessed in the future by using the sfselect (SDselect) calls in a FORTRAN (C) program. If no sds_id has been assigned, the id can be determined from the name of the data set using the following statements:

```
sds_index = SDnametoindex(sd_id, sds_name);      (C)

sds_id = SDselect(sd_id, sds_index);

sds_index = sfn2index(sd_id, sds_name)           (FORTRAN)

sds_id = sfselect(sd_id, sds_index)
```

where

```
sds_index = data set index number
sds_name = data set name
sd_id = HDF file index number
```

9. Reading Data from an HDF File

The following sections detail how a user may utilize the HDF library the SD API within a computer program to write a data file in HDF. In this section, the tutorial will concentrate on using the FORTRAN programming language and the SD API. However, examples of the appropriate C code will also be given for certain steps. For the purpose of this tutorial, we are choosing the example of reading an entire data array that is the first and only data set in the HDF file. Similar to writing an HDF file, the user should follow these simple steps:

- Does the current version of HDF support your computer platform and operating system?
- Downloading and Installing the HDF library
- Are all libraries and programs properly linked and compiled?
- Writing a short program to read an HDF data set

9.1 Does the current version of HDF support your computer platform and operating system?

As outlined in Section 3, the HDF library can not be run on just any available computer platform or operating system. Before downloading the HDF library software, the user should make sure that the current release of HDF supports his/her computer and operating system. Otherwise, the user will be unable to work with the HDF library and files. There is also a possibility that previous releases of HDF may support the Users computer platform while the latest version does not. In this event, the user may wish to obtain the earlier software.

9.2 Downloading and Installing the HDF library

The HDF library and software is public domain software and available free to all users. The library and code can be downloaded from the NCSA anonymous ftp server (ftp://ftp.ncsa.uiuc.edu/HDF/HDF/HDF Current). Directions on how to install the HDF library can also be found at this location.

9.3 Are all libraries and programs properly linked and compiled?

In order to eventually run the HDF software, the library and the needed application routines and programs must first be properly compiled and linked. As of the current release of HDF (4.1r3), four separate libraries must be compiled and linked. These are the libmfhdf.a, libdf.a, libjpeg.a, and libz.a libraries. Provided below are examples of the command(s) that can be used for this action. It must be noted that the order in which the libraries are linked is important and should not vary from the order shown below:

For C programs:

```
cc  -o &lt;your program&gt; &lt;your program&gt;.c \
    -I&lt;pathf for hdf include directory&gt;\
    -L&lt;path for hdf libraries&gt; -lmfhdf -ldf -ljpeg -lz
```

For FORTRAN programs:

```
f77 -o &lt;your program&gt; &lt;your program&gt;.f  \
    -I&lt;path for hdf include directory&gt;\
```

```
-L&lt;path for hdf libraries&gt; -lmfhdf -ldf -ljpeg -lz
```

For the various commands needed to link and compile the HDF library on each individual platform, please see Section 3- Compiling the HDF Library.

9.4 Writing a short program to read an HDF data set

9.4.1 Select a programming language

As mentioned previously, the HDF library and programs can only be run by using either the C or FORTRAN programming language. This choice is up to the user depending on availability and the language he or she feels most familiar and comfortable with.

9.4.2 Make sure all include files are in place

Earlier, it was noted that a series of standard HDF definitions and declarations of file access codes (i.e. read, write, etc.) and data types (i.e. integer, character) must be included within the programs that the user writes to utilize the various application routines. In the C programs, this is accomplished simply by adding the line #include "hdf.h" at the beginning of the program. This line effectively includes all the needed constants and definitions from the HDF software. When writing FORTRAN programs, this may also be done by simply adding an include statement that brings in only the needed definitions and declarations (constants.f) from the hdf.h header file. This is done by the following code: "include constants.f". However, all FORTRAN compilers (particularly the older ones) do not support the use of include statements. In this event, the user must type in/declare all the constants and definitions found in the constants.f file. It is advised that all declarations, whether through Include statements or not, should be done at the beginning of the program.

9.4.3 Make all variables and parameter declarations

As with any program, the scientist/user should declare and initialize all variables and parameters at the beginning of the program. This includes all variables and arguments that will be used by the HDF commands to follow. The variable and parameter declarations needed for each call will be provided in the example boxes of the individual steps. These statements always belong at the top of the program.

9.4.4 Initialize access to the SD interface and open HDF file

The first real HDF programming step actually accomplishes two things:

1) Opens the existing HDF file
2) Initializes and opens the SD interface.

This is done by the following command:

```
sd_id = sfstart(filename, access_mode)     (FORTRAN)

                                      or

sd_id = SDstart(filename, access_mode):    ( C )
```

where

```
sd_id = HDF file id returned by the sfstart/SDstart command
filename = the name of the existing HDF file (character string)
access_mode = Type of access required for this file
```

All available options for the access-mode argument are defined in the hdf.h header file mentioned previously and need only to be identified for all C and most FORTRAN operations. All options begin with the prefix "DFACC_" and include:

```
DFACC_CREATE (File Creation Access)
DFACC_RDONLY (Read Access)
DFACC_RDWR (Read and Write Access)
```

These definitions are stated in the hdf.h header file.

In the event that the user's FORTRAN compiler can not handle include statements with the header file (hdf.h), the DFACC_ variable must be defined, along with its assigned value, at the beginning of the program. This is done by a line of code such as:

```
parameter (DFACC_RDONLY = 1)        (For FORTRAN only)
```

Example:

FORTRAN:

```
integer*4 sd_id
integer sfstart
parameter(DFACC_RDONLY = 1)
sd_id=sfstart("wind.hdf", DFACC_RDONLY)
```

C:

```
#includehdf.h"
int32 sd_id;
sd_id=Sdstart("wind.hdf", DFACC_RDONLY
```

9.4.5 Select data set to be read from the HDF file

After initializing the SD interface and opening and assigning a file id (sd_id) to the HDF file to be used, the next step is to select the HDF Scientific Data Set (SDS) which will be read. This is done by the following command:

```
sds_id = sfselect (sd_id, sds_index)      (FORTRAN)

                                    or

sds_id = SDselect (sd_id, sds_index)      (  C  )
```

where

```
sds_id = HDF SDS array id returned by the sfselect/SDselect command
sd_id = the HDF file id created in the previous step (sfstart/SDstart)
sds_index = index number of data set within file
            (i.e. 0 = first data set, 1 = second data set, etc.)
```

Example:

FORTRAN:

```
integer sds_id, sds_index, sd_id
integer sfselect
sds_index = 0 represents the first data set
sds_id = sfselect(sd_id,0)
```

C:

```
int32 sd_id, dims[2];
dims[0] = YL;
dims[1] = XL;
sds_id = Sdselect(sd_id,0);
```

9.4.6 Read an existing data set/array

After initializing the API and selecting the HDF file and HDF SDS to be read to, the next step is to actually read the existing HDF data by using the SDreaddata (sfrdata) command. This command is used to read either all or part of the existing n-dimensional data set (termed a "slab") into the sds_id array with the same number of dimensions. In addition, the size of each dimension of the data "slab" must be the same or smaller then the corresponding dimension of the sds_id. The SDreaddata/sfrdata command is used in the following fashion:

```
ret=sfrdata (sds_id, start, stride, edge, data)       (FORTRAN)

                                                or

ret=SDreaddata (sds_id, start, stride, edge, data);   ( C )
```

It should be noted that there are two versions of the read routine in FORTRAN. The sfrdata routine reads numeric scientific data while sfrcdata reads character scientific data

```
sds_id = the SDS id (identifier) determined and returned by using SDcreate
or SDselect (sfcreate/sfselect)

start  = An array which identifies where in the SDS that the writing will
begin
```

The start array identifies the location or position in the SDS where the reading of the data "slab" will begin. This array must have the same number of dimensions (rank) as the SDS and can not be larger (in each dimension) then the SDS array. The declaration of the start variables can be done at the top of the

program or just preceding the call of the sfrdata (SDreaddata) command. As an example, to read the existing data set to the beginning of a new 2-dimensional SDS the following must be specified:

```
start(1) = 0            (FORTRAN)
start(2) = 0

                  or

start[0] = 0;           ( C )
start[1] = 0;
```

If the user wishes to begin reading the data at a location other then the beginning of the data set, say at a first dimension (X) of 15, the declarations would be:

```
start(1) = 15           (FORTRAN)
start(2) = 0

                  or

start[0] = 15;          (C )
start[1] = 0;
```

stride = An array specifying the interval between written values in each dimension.

The stride argument specifies, for each dimension, the interval between consecutive written values of the data set. In other words, how many array locations are skipped with each reading of the data. Like the start array, the stride argument is predefined before calling the sfrdata (SDreaddata) command, either directly before the call or at the top of the program.

If the user does not wish to skip any array locations in a new 2-dimensional SDS, the following is to be declared:

```
stride(1) = 1           (FORTRAN)
stride(2) = 1

                  or

stride[0] = 1;          ( C )
stride[1] = 1;
```

However, if the user wishes to skip every other X (dimension 1) location, the following would be used:

```
stride(1) = 2           (FORTRAN)
stride(2) = 1

                  or

stride[0] = 2;          ( C )
stride[1] = 1;
```

edge = An array defining the number of data values to be read in each dimension.

The edge array defines the number of data values/elements that will be read along each dimension of the multi-dimensional SDS array. In plain terms, this argument defines the size of the data slab (all or part of the data) to be written to the new SDS array and each dimension.

The parameter edge must be specified for each dimension of the data set and SDS array, and can not be larger then the entire length of the array being read.

Similar to stride and start, the edge argument needs to be defined prior to the calling of the sfrdata (SDreaddata) command, whether it be at the top of the program or directly before the routine call. The file containing this data should be opened at the beginning of the program and the data read in and stored into the necessary arrays before beginning the HDF operations.

As an example: Most often, the user will wish to read an HDF file which contains one data set (winddata), which starts from the beginning and does not contain any missing data or blanks.

For a 2-dimensional array of 30X30, read and stored into the data array "testdata", this can be done by:

```
start(1) = 0                    (FORTRAN)
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 30
edge(2) = 30
retn = sfrdata(sds_id, start, stride, edges, winddata)

                        or

start[0] = 0;                   ( C )
start[1] = 0;
stride[0] = 1;
stride[0] = 1;
edge[0] = 30;
edge[1] = 30;
retn = SDreaddata(sds_id, start, stride, edges, winddata);
```

Example:

For reading the entire data set from an HDF file which contains only one 2-D array:

FORTRAN:

```
      integer start(2), edges(2), stride(2)
      integer retn sfrdata
c     Define the location, pattern + size of data to be read
      YL = 30
      XL = 30
      start(1) = 0
      start(2) = 0
      stride(1) = 1
      stride(2) = 1
```

```
      edge(1) = XL
      edge(2) = YL
      retn = sfrdata(sds_id, start,stride,edges,winddat)
```

C:

```
/* Define the location, pattern + size of data to be read */
      YL = 30;
      XL = 30;
      dims[0] = YL;
      dims[1] = XL;
      start[0] = 0;
      start[1] = 0;
      stride[0] = 1;
      stride[1] = 1;
      edge[0] = dims[0];
      edge[1] = dims[1];
      retn = SDreaddata(sds_id, start,stride,edges,winddat);
```

9.4.7 Write non-HDF data to a file

Using standard FORTRAN and C statements for writing, the non-HDF data is written into a new file (storage). In addition, the user may wish to print out all or parts of the HDF data set to view the data or as a check of the procedure/operation.

9.4.8 Optional operation: Get and Read Metadata

After opening the HDF file using the sfstart/SDstart, the first step is to see if the file or data sets do indeed contain attributes. This is done by using the following command:

```
attr_index = SDfindattr (sd_id, attr_name);      (C )

attr_index = sffattr (sd_id, attr_name)         (FORTRAN)
```

where

```
attr_index = valid attribute index returned if attribute exists
sd_id = file identifier
attr_name = name of attribute (i.e.,Contents of file")
```

If there is a attribute index, the name, data type (num_type), and count (number of characters) of the attribute can be obtained:

```
retn= SDattrinfo(sd_id, attr_index, attr_name, num_type, count); (C)
retn= sfgainfo (sd_id, attr_index, attr_name, num_type, count) (FORTRAN)
```

After completing these operations, the attributes can be read using the following

```
retn= SDreadattr (sd_id, attr_index, buffer);   ( C )
retn= sfrattr (sd_id, attr_index, buffer)       (FORTRAN)
```

where

buffer is allocated to hold the attribute data

The above steps can also be followed for each data set within the file by getting the data set id (sds_id) of the data, finding a particular attribute (i.e.,"Units") and getting and reading the data.

Example:

FORTRAN:

```
sd_id=sfstart ("wind.hdf", DFACC_RDONLY)
attr_index= sffattr (sd_id,"file_contents")
retn= sfgainfo (sd_id, attr_index, "file_contents", data_type, count)
retn=  sfrattr (sd_id, attr_index, buffer)
```

and

```
sds_id=sfselect (sd_id, 0)
attr_index= sffattr (sds_id,"units")
retn= sfgainfo (sds_id, attr_index,"units", data_type, count)
retn=  sfrattr (sds_id, attr_index, buffer)
```

C:

```
sd_id=SDstart ("wind.hdf", DFACC_RDONLY);
attr_index= SDfindattr (sd_id,"file_contents");
retn= SDattrinfo (sd_id, attr_index,"file_contents", data_type, count);
retn=  SDreadattr (sd_id, attr_index, buffer);
```

and

```
sds_id=SDselect (sd_id, 0);
attr_index= SDfindattr (sds_id,"units");
retn= SDattrinfo (sds_id, attr_index,"units", data_type, count);
retn=  SDreadattr (sds_id, attr_index, buffer);
```

9.4.9 Terminate/Close access to all files, data sets, and APIs

After writing the data to the new SDS array within the new HDF file, it is necessary to terminate or close access to the new data set in order to prevent any possible loss of data. This is done by the following:

```
retn = sfendacc(sds_id)              (FORTRAN)

                                or

retn = SDendaccess(sds_id)           ( C )
```

In addition, the API called within the program must also be closed to prevent any data loss:

```
retn = sfend(sd_id)                  (FORTRAN)

                                or
```

```
retn = SDend(sd_id)                    (C )
```

Example:

FORTRAN:

```
integer sfendacc, sfend
retn = sfendacc(sds_id)
retn = sfend(sd_id)
```

C:

```
retn = SDendaccess(sds_id);
retn = SDend(sd_id);
```

9.4.10 Execute program

Execute like a standard FORTRAN or C program

10. Example Programs for Dealing with Scientific Data Sets (SDS) in HDF

The following is a list of sample programs that illustrate how the HDF library, and the SD API, can be used to work with HDF files. The example programs are given in the FORTRAN programming language. However, the detailed steps for both C and FORTRAN are the same. Only the syntax code particular to each language should be different. The following example programs are provided:

1) Writing an SDS in HDF
2) Writing Attributes in HDF
3) Writing the SDS and attributes in HDF
4) Reading an HDF file
5) Reading HDF attributes (files and data sets)

10.1 Writing an SDS in HDF

FORTRAN:

```
      PROGRAM WRITDATA

      integer*4 sd_id, sds_id, rank
      integer*4 XL, YL
      integer dims(2), start (2), edges (2), stride (2)
      integer i, j, k, retn
      integer sfstart, sfcreate, sfwdata, sfendacc, sfend
      real rwind(30, 30)
C
C     DFACC_CREATE and DFNT_INT16  are defined in hdf.h but may have
C     to be defined within the program for certain FORTRAN compilers
      integer*4 DFACC_CREATE, DFNT_INT16
      parameter (DFACC_CREATE = 4, DFNT_INT16=22)

      rank = 2
      XL = 30
      YL = 30
C
C     Create and open a new HDF file and initiate the SD interface
C
      sd_id = sfstart('wind.hdf', DFACC_CREATE)
C
C     Define the rank (number of dimensions) and dimensions (size) of the
C     HDF Scientific Data Set (SDS) to be created.
C
      dims(1) = XL
      dims(2) = YL
C
C     Create the HDF SDS (sfselect would be used if writing to an
C     existing HDF file or data set)
C
      sds_id = sfcreate(sd_id, 'winds', DFNT_INT16, rank, dims)
C
C     Open and read the existing non-HDF data set into an array (rwind)
```

```
C
      Open (unit=10, file='wind.dat', form='formatted')
C
      Do j = 1,30
        Read(10, 12) (rwind(i, j),i = 1,30)
  12    Format(30(f4.1,1x))
      Enddo
C
C     Define where in the file to write the data set (start--location),
C     the pattern of the data (stride--skip any values??), and the size
C     of the data set (edges) to be written to.  This is done for each
C     dimension.   start(x) = 0 is for writing at the beginning of the
C     newly created SDS and stride(x) = 1 signifies that no data is to
C     be skipped in the writing.
C
      start(1) = 0
      start(2) = 0
      edges(1) = XL
      edges(2) = YL
      stride(1) = 1
      stride(2) = 1
C
C     Write the stored data (in the array rwind) to the new SDS
C
      retn = sfwdata(sds_id, start, stride, edges, rwind)
C
C     Terminate access to the array
C
      retn = sfendacc(sds_id)
C
C     Terminate access to the SD interface and close the HDF file
C
      retn = sfend(sd_id)

      Stop
      End
```

10.2 Writing Attributes in HDF

FORTRAN:

```
PROGRAM WRITEATT

      integer*4 sd_id, sds_id, dim_id, retn
      integer dims(2), start(2), edges(2), stride(2)
      integer sfstart, sfselect, sfdimid, sfscatt, sfendacc, sfend
C
C     DFACC_RDWR,  DFNT_INT16 and DFNT_CHAR8  are defined in hdf.h but
C     may have to be defined within the program for certain FORTRAN
C     compilers
C
      integer*4 DFACC_RDWR, DFNT_INT32, DFNT_CHAR8
      parameter (DFACC_RDWR = 3, DFNT_INT16 = 22, DFNT_CHAR8 = 4)
C
C     Open the HDF file,  initiate the SD interface, and get the
C     identifier for the file
```

```
C
      sd_id = sfstart('wind.hdf', DFACC_RDWR)
C
C     Set an attribute the describe the contents of the file
C
      retn = sfscatt(sd_id, 'file_contents', DFNT_CHAR8, 15,
     +               'lidar_LOS_winds')
C
C     Get the identifier for the first data set (in this example, the
C     only data set)
C
      sds_id = sfselect(sd_id, 0)
C
C     Set an attribute(s) for the data array itself. In this example, the
C     units of the data are defined
C
      retn = sfscatt(sds_id, 'units', DFNT_CHAR8, 13, 'units = m/sec')
C
C     Terminate access to the data array
C
      retn = sfendacc(sds_id)
C
C     Terminate access to the SD interface and close the HDF file
C
      retn = sfend(sd_id)

      Stop
      End
```

10.3 Writing the SDS and attributes in HDF

FORTRAN:

```
PROGRAM WRITESDS

      integer*4 sd_id, sds_id, rank, dim_id
      integer*4 XL, YL
      integer dims(2), start(2), edges(2), stride(2)
      integer i, j, k, retn
      integer sfstart, sfcreate, sfwdata, sfendacc, sfscatt, sfend
      real rwind(30, 30)
C
C     DFACC_CREATE, DFACC_RDWR, DFNT_CHAR8 and DFNT_INT16  are defined
C     in hdf.h but may have to be defined within the program for certain
C     FORTRAN compilers
C
      integer*4 DFACC_CREATE, DFNT_INT16, DFNT_CHAR8, DFACC_RDWR
      parameter (DFACC_CREATE = 4, DFACC_RDWR = 3, DFNT_INT16 = 22,
     +           DFNT_CHAR8 = 4)

      rank = 2
      XL = 30
      YL = 30
C
C     Create and open a new HDF file and initiate the SD interface
C
      sd_id = sfstart('wind.hdf', DFACC_CREATE)
```

```
C
C     Define the rank (number of dimensions) and dimensions (size) of the
C     HDF Scientific Data Set (SDS) to be created.
C
      dims(1) = XL
      dims(2) = YL
C
C     Create the HDF SDS (sfselect would be used if writing to an
C     existing HDF file or data set)
C
      sds_id = sfcreate(sd_id, 'winds', DFNT_INT16, rank, dims)
C
C     Open and read the existing non-HDF data set into an array (rwind)
C
      Open (unit = 10, file = 'wind.dat', form = 'formatted')
C
      Do j=1,30
        Read (10, 12) (rwind(i, j),i=1,30)
   12   Format (30(f4.1,1x))
      enddo
C
C     Define where in the file to write the data set (start--location),
C     the pattern of the data (stride--skip any values??), and the size
C     of the data set (edges) to be written to.  This is done for each
C     dimension.  start(x) = 0 is for writing at the beginning of the
C     newly created SDS and stride(x)=1 signifies that no data is to be
C     skipped in the writing.
C
      start(1) = 0
      start(2) = 0
      edges(1) = XL
      edges(2) = YL
      stride(1) = 1
      stride(2) = 1
C
C     Write the stored data (in the array rwind) to the new SDS
C
      retn = sfwdata(sds_id, start, stride, edges, rwind)
C
C     For writing attributes, set an attribute the describe the
C     contents of the file
C
      retn = sfscatt(sd_id, 'file_contents', DFNT_CHAR8, 15,
     +               'lidar_LOS_winds')
C
C     Set an attribute(s) for the data array itself. In this example, the
C     units of the data are defined
C
      retn = sfscatt(sds_id, 'units', DFNT_CHAR8, 13, 'units = m/sec')
C
C     Terminate access to the data array
C
      retn = sfendacc(sds_id)
C
C     Terminate access to the SD interface and close the HDF file
C
      retn = sfend(sd_id)
```

```
      Stop
      End
```

10.4 Reading an HDF file

FORTRAN:

```
      PROGRAM READDATA

      integer*4 sd_id, sds_id
      integer*4 XL, YL
      integer start(2), edges(2), stride(2)
      integer i, j, k, retn
      integer sfstart, sfselect, sfrdata, sfendacc, sfend
      real rwind(30, 30)
C
C     DFACC_RDONLY is defined in hdf.h but may have to be defined
C     within the program for certain FORTRAN compilers
C
      integer*4 DFACC_RDONLY
      parameter (DFACC_RDONLY = 1)
C
C     MAX_NC_NAME (maximum # of characters) and MAX_VAR_DIMS (maximum
C     # of dimensions) are defined in netcdf.h but may have to be defined
C     here.
C
      integer*4 MAX_NC_NAME, MAX_VAR_DIMS
      parameter (MAX_NC_NAME = 256, MAX_VAR_DIMS = 32)
      integer dims(MAX_VAR_DIMS)

      XL = 30
      YL = 30
C
C     Open the HDF file and initiate the SD interface
C
      sd_id = sfstart('wind.hdf', DFACC_RDONLY)
C
C     Select the first data set in the file (In this example, the only
C     dataset).
C
      sds_id= sfselect(sd_id, 0)
C
C     To read from the data set, define the location (start--where in the
C     file), the pattern (stride--skip any values??),and the size(edges)
C     of the data.  This is done for each dimension. start(x) = 0 is for
C     reading at the beginning of the file and stride(x) = 1 signifies
C     that no data is to be skipped in the reading.
C
      dims(1) = XL
      dims(2) = YL
      start(1) = 0
      start(2) = 0
      stride(1) = 1
      stride(2) = 1
      edges(1) = dims(1)
```

```
      edges(2) = dims(2)
C
C     Read the array dataset
C
      retn = sfrdata(sds_id, start, stride, edges, rwind)
C
C     Optional - Print out data (ASCII) read from the HDF file
C     In this example we are writing to the screen (*)
C
      Do j = 1, 30
        write(*,12) (rwind(i,j),i=1,30)
 12     format(30(f4.1,1x))
      enddo
C
C     Terminate access to the array
C
      retn = sfendacc(sds_id)
C
C     Terminate access to the SD interface and close the HDF file
C
      retn = sfend(sd_id)

      Stop
      End
```

10.5 Reading HDF attributes (files and data sets

FORTRAN:

```
      PROGRAM READATTR

      integer*4 sd_id, sds_id, units_buffer
      integer attr_index, data_type, count, retn
      character attr_name * 13
      character char_buffer * 20
      integer sfstart, sfrnatt, sfrcatt, sfgainfo, sffatr, sfselect
      integer sfendacc, sfend
C
C     DFACC_RDWR is defined in hdf.h but may have to be defined
C     within the program for certain FORTRAN compilers
C
      integer*4 DFACC_RDWR, DFACC_RDONLY
      parameter (DFACC_RDWR = 3, DFACC_RDONLY = 4)
C
C     Open the HDF file and initiate the SD interface
C
      sd_id = sfstart('wind.hdf', DFACC_RDONLY)
C
C     Select the first data set in the file (In this example, the only
C     dataset).
C
      sds_id= sfselect(sd_id, 0)
C
C     Find the attribute which describes the contents of the file
C     (usually 'file_contents')
C
      attr_index = sffattr(sd_id, 'file_contents')
```

```
C
C     Get information about the file attribute
C
      retn = sfgainfo(sd_id, attr_index, attr_name, data_type, count)
C
C     Read the file attribute data
C
      retn = sfrcatt(sd_id, attr_index, char_buffer)
C
C     Read the attributes for the first data set. First step is to get
C     the identifier.
C
      sds_id = sfselect(sd_id, 0)
C
C     Find the attribute which defines the units of the data set
C     ('units')
C
      attr_index = sffattr(sds_id, 'units')
C
C     Get information about the data set attribute
C
      retn = sfgainfo(sds_id, attr_index, attr_name, data_type, count)
C
C     Read the data set attribute data
C
      retn = sfrcatt(sds_id, attr_index, units_buffer)
C
C     Terminate access to the array
C
      retn = sfendacc(sds_id)
C
C     Terminate access to the SD interface and close the HDF file
C
      retn = sfend(sd_id)

      Stop
      End
```

11. Browsing and Visualizing HDF Data

With the recent explosion of data volumes, numerous visualization and browsing tools have been developed which allow users to quickly view the contents of data sets created elsewhere. This has proven especially beneficial for users of HDF.

In fact, many visualization tools have been created specifically with HDF in mind. The NCSA anonymous ftp server provides a set of free software that enables the user to visualize and browse HDF files. Tools include the JAVA-based HDF Browser (http://hdf.ncsa.uiuc.edu/java-hdf-html) and the Scientific Data Browser (http://hdf.ncsa.uiuc.edu/sdb/sdb.html). In addition, the following are available but have not been updated to run with the current version of HDF: NCSA Collage, NCSA Datascope, NCSA XDataSlice, and NCSA Polyview.

Besides NCSA, there are other sites and centers that also provide public domain (free) software that can be used to browse and visualize HDF files. This software includes, among others: LinkWinds, WebWinds, GRASS, FREEFORM, VISTAS, ImageMagick, and Envision. Visualization tools and software such as LinkWinds and EOSView (http://edhs1.gsfc.nasa.gov/waisdata/toc/tp4450601toc.html) can be used for working with HDF-EOS type data (point, swath and grid data sets).

Finally, there are also commercial (for a fee) software packages that can be used to work with and browse HDF files. These include: DataExplorer, Spyglass, PV-Wave, Wavefront, IDL, AVS, IRIS Explorer, Transform, and ER Mapper.

Please see Section 4: HDF Browsing and Visualization Tools for further detail, including internet addresses, on the above software.

12. HDF Laboratory

Due to the interactive nature of the Question and Answer session, the questions are not provided in this version of the tutorial, but may be viewed in the HTML version at http://cyclone.swa.com/meteorology/hdf/tutorial/welcome.html. However, a brief overview is provided below. The following are the general sections that are covered by the Laboratory:

Section I: General Background: HDF and the HDF Library (1-9)

Section II: Methods of Working with HDF Files (10-16)

Section III: Scientific Data Model (17-20)

Section IV: Attributes and Metadata(21-25)

Section V: Using the SD API to write an Existing Data Set in HDF (26-36)

Section VI: Querying /and Reading an HDF File (37-39)

12.1 Lab Directions

The question and answer section of the tutorial was developed in Java script and is best viewed using the latest releases of Microsoft Internet Explorer (http://microsoft.com/windows/ie/default.htm) and Netscape navigator (http://home.de.netscape.com/). When navigating through the tutorial, individual questions will be loaded on the same window and will be controlled by the "Previous question" and "next question" buttons. However, new windows will be opened when the user attempts to look at the preview material for each question and thus allowing the user to toggle back and forth from the question and the material. To exit the tutorial, just click the "back" from the main question screen and this will bring the user back to the Laboratory menu. When done with a "preview" window, simply close out the window and return to the question window/screen.

In this section we provide a series of questions designed for the users of the tutorial to test themselves on how well they understood the material presented in the tutorial and, more importantly, to gauge how comfortable they feel with HDF.

The questions more or less follow the order of the topics covered in the "Lecture" component of the tutorial. The Laboratory menu provides a breakdown by section of the various questions, and allows the user to select which topics they would like to focus their attention on.

Each question contains the question itself, a set of possible answers, navigation buttons ("Next Question", "Previous Question", "Lab Menu", "End Lab", "Submit Answer") and, most importantly, a feature which allows the user to review material pertaining to the question before answering or after answering incorrectly. This is done by selecting the "Preview Material" button. Since some users will like to take the "test" without any help and others will like to

review material before answering (particularly those who may have skipped directly to the Laboratory), it is up to the individual user to decide how to proceed.

It should be noted that, in many cases, there is more than one correct answer for each individual question. The user is allowed to select more than one response and will only receive an "Answered Correctly" response if ALL correct responses have been selected.

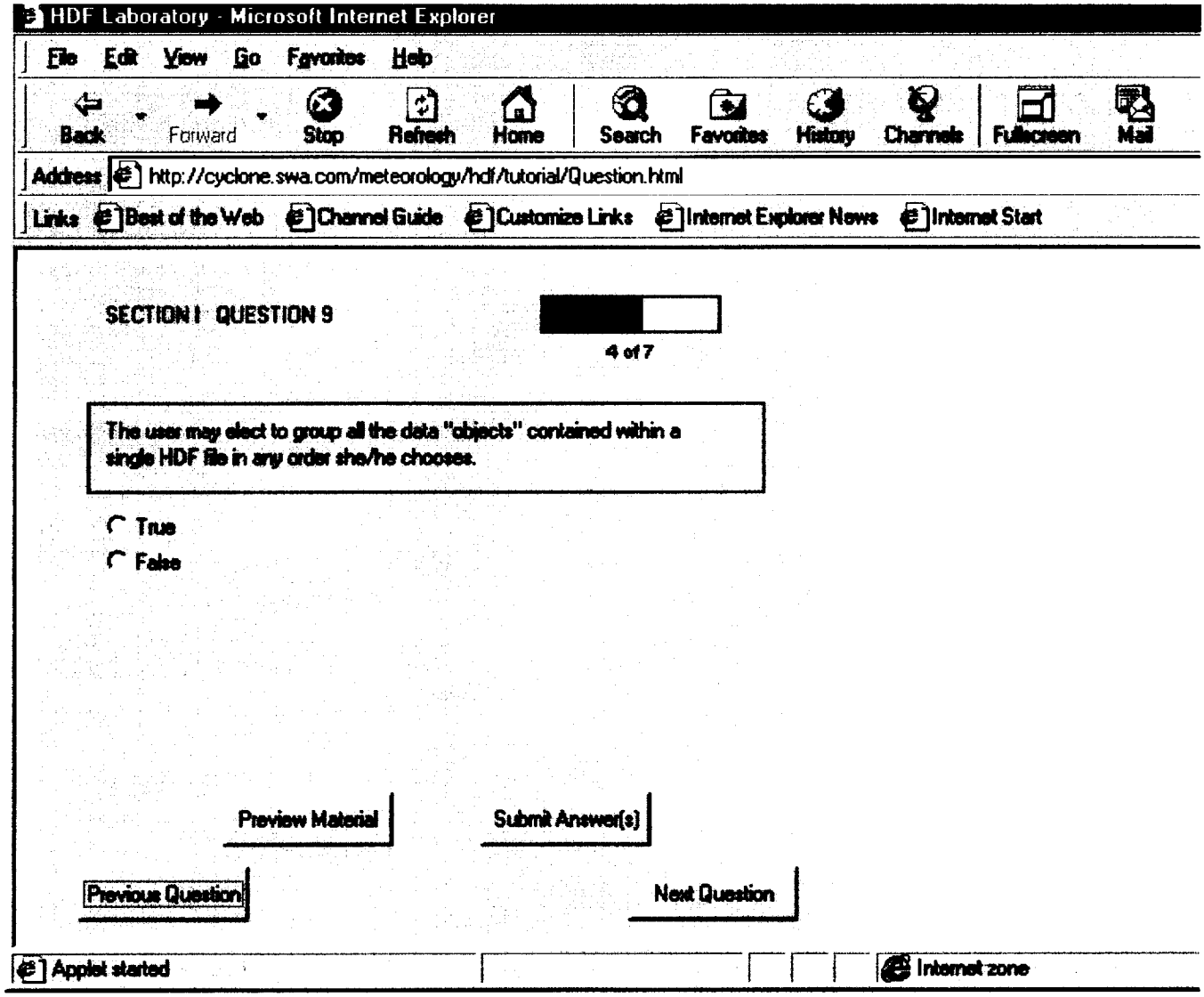
To help the users gauge their understanding of HDF and how well they are answering the questions, a "performance gauge" is provided in the upper right hand corner of each question. This gauge provides both a numerical (i.e. 7 of 11) and graphical (sliding color bar for 0 - 100%) representation of user performance. The "score" found in the performance gauge only reflects the users' initial answer to each question.

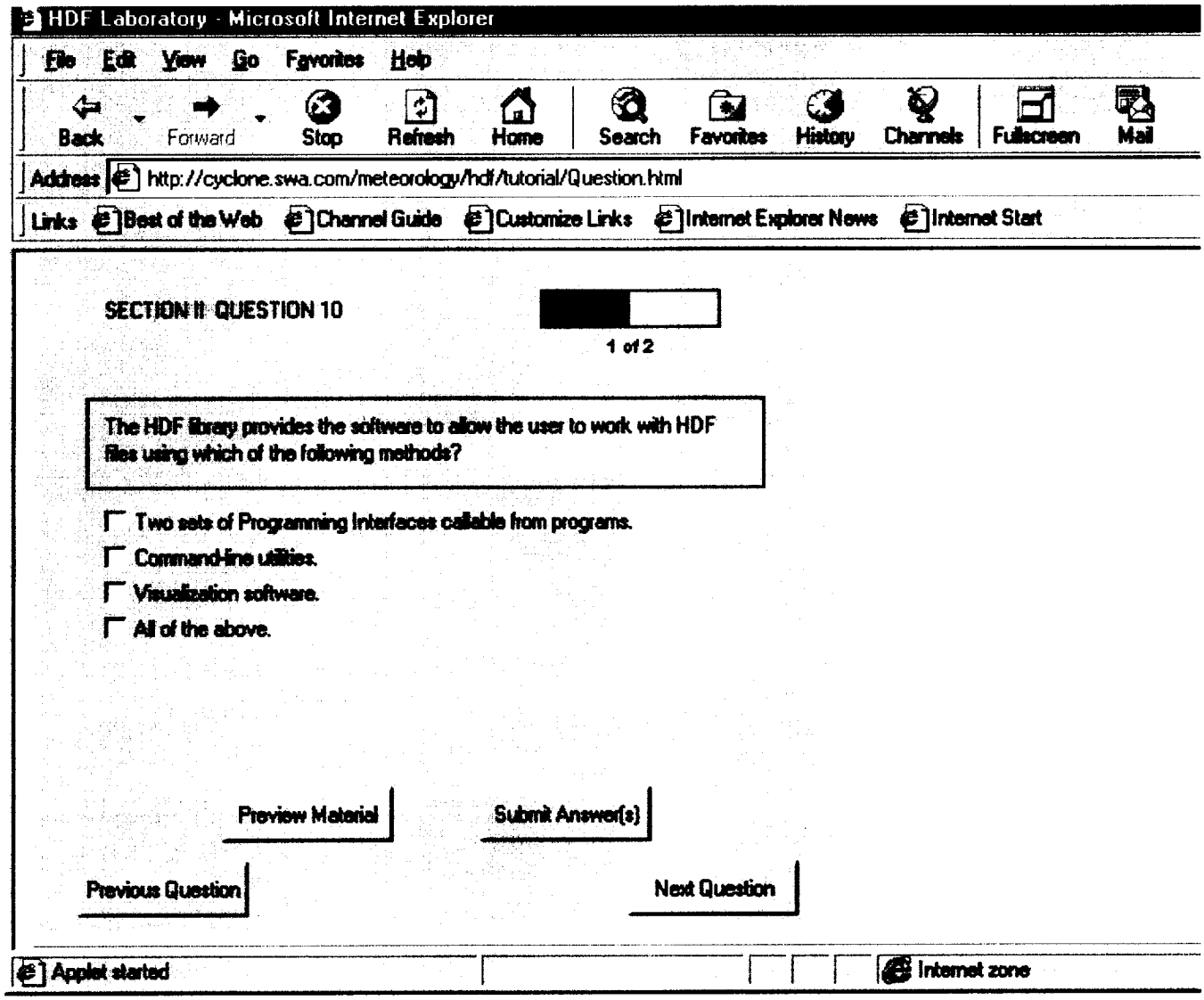
The user's performance in the Laboratory is further diagnosed in a Progress Report reached by clicking on the performance gauge. Included in this report are:

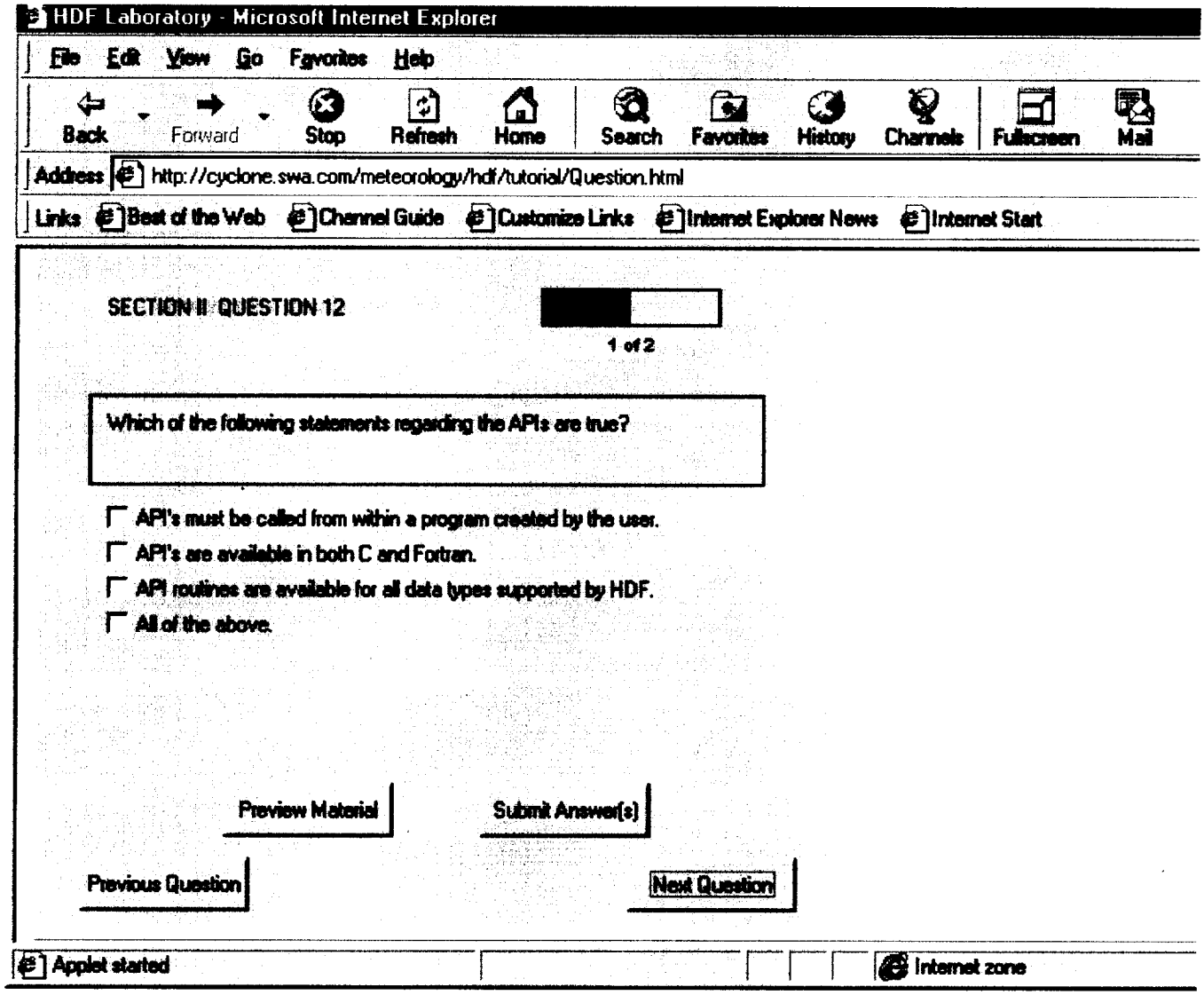
- number of questions answered correctly
- number of questions answered incorrectly
- number of questions left unanswered
- list of questions answered correctly
- list of questions answered incorrectly
- list of questions left unanswered

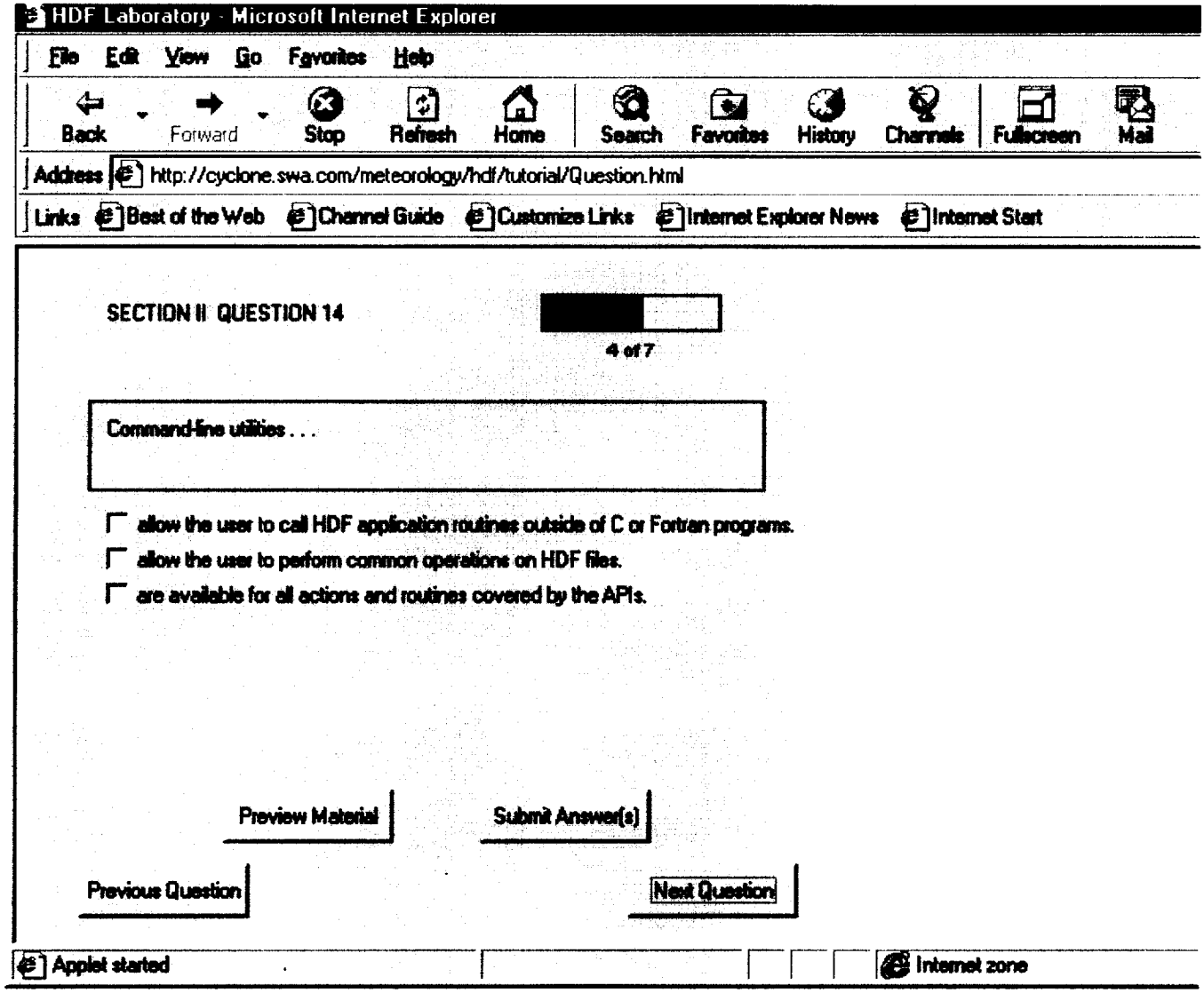
13. ACRONYM LIST

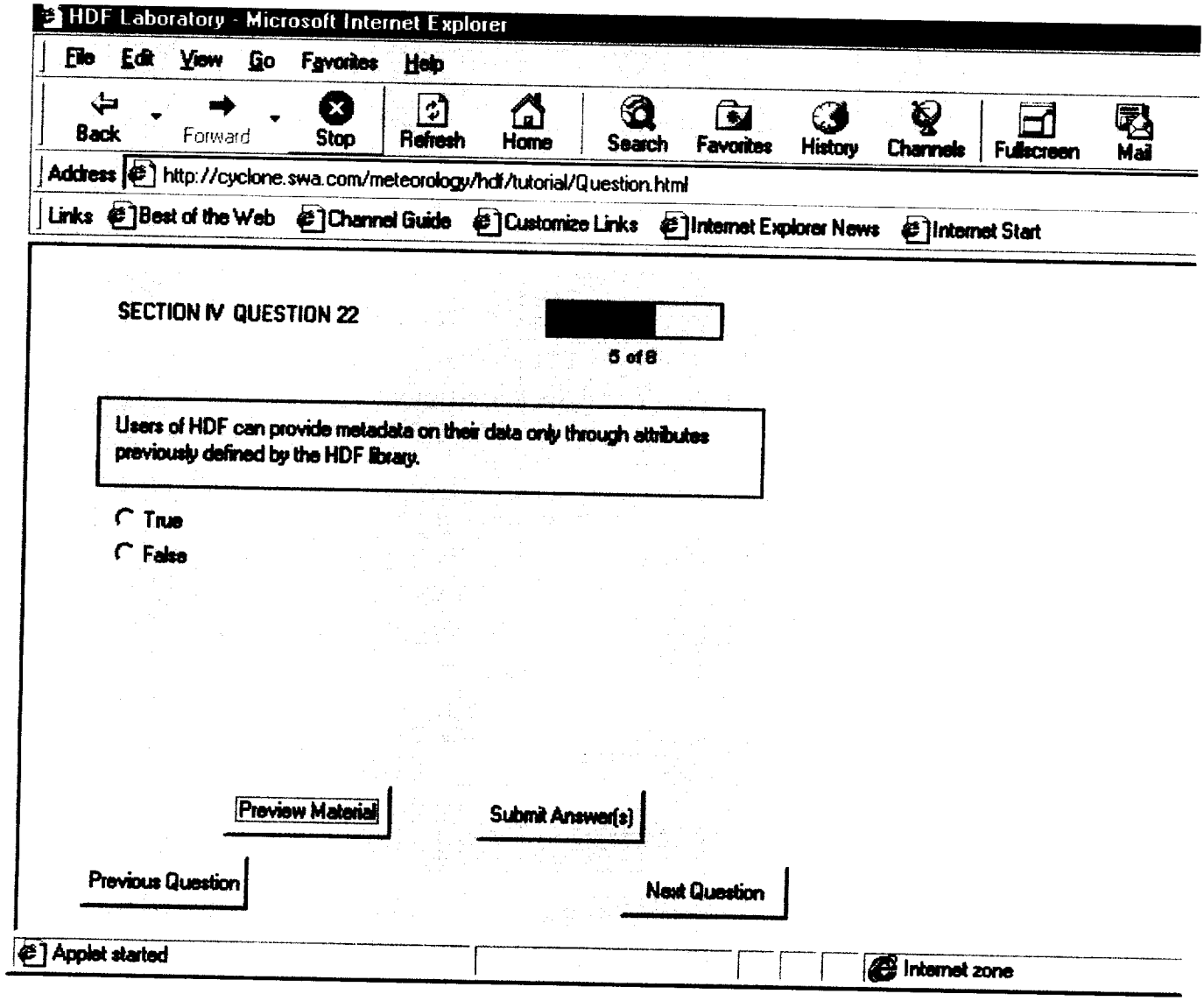
API	**Application Programming Interface**
EOS	**Earth Observing System**
EOSDIS	**EOS Data and Information System**
ESDIS	**Earth Science Data and Information System**
FAQ	**Frequently Asked Questions**
GB	**Giga-Byte**
HDF	**Hierarchical Data Format**
HDF-EOS	**Hierarchical Data Format - Earth Observing System**
JHI	**Java HDF Interface**
NASA	**National Aeronautics and Space Administration**
NCSA	**National Center for Supercomputer Applications**
SD	**Scientific Data**
SDS	**Scientific Data Set**
SWA	**Simpson Weather Associates**

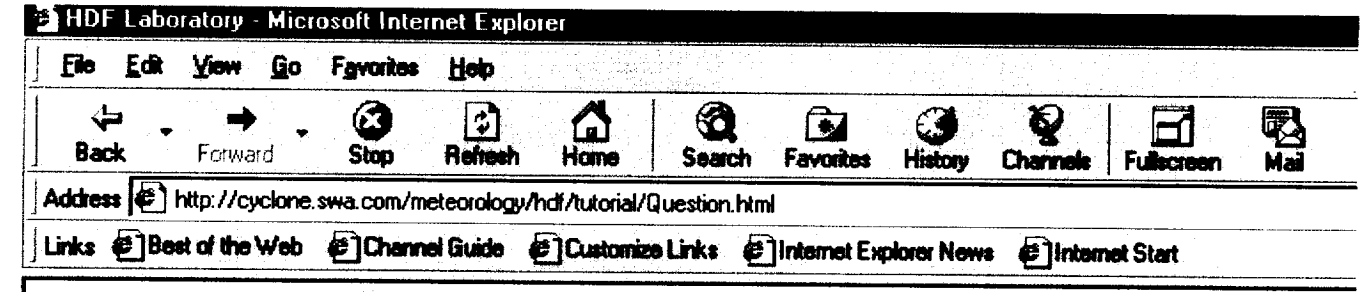

HDF Laboratory - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address http://cyclone.swa.com/meteorology/hdf/tutorial/Question.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
SECTION I QUESTION 9
4 of 7
The user may elect to group all the data "objects" contained within a single HDF file in any order she/he chooses.
True
False
Preview Material
Submit Answer(s)
Previous Question
Next Question
Applet started
Internet zone


HDF Laboratory - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address http://cyclone.swa.com/meteorology/hdf/tutorial/Question.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
SECTION II QUESTION 10
1 of 2
The HDF library provides the software to allow the user to work with HDF files using which of the following methods?
Two sets of Programming Interfaces callable from programs.
Command-line utilities.
Visualization software.
All of the above.
Preview Material
Submit Answer(s)
Previous Question
Next Question
Applet started
Internet zone


HDF Laboratory - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address http://cyclone.swa.com/meteorology/hdf/tutorial/Question.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
SECTION II QUESTION 12
1 of 2
Which of the following statements regarding the APIs are true?
API's must be called from within a program created by the user.
API's are available in both C and Fortran.
API routines are available for all data types supported by HDF.
All of the above.
Preview Material
Submit Answer(s)
Previous Question
Next Question
Applet started
Internet zone


HDF Laboratory - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address http://cyclone.swa.com/meteorology/hdf/tutorial/Question.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
SECTION II QUESTION 14
4 of 7
Command-line utilities . . .
allow the user to call HDF application routines outside of C or Fortran programs.
allow the user to perform common operations on HDF files.
are available for all actions and routines covered by the APIs.
Preview Material
Submit Answer(s)
Previous Question
Next Question
Applet started
Internet zone


HDF Laboratory - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address http://cyclone.swa.com/meteorology/hdf/tutorial/Question.html
Links Best of the Web Channel Guide Customize Links Internet Explorer News Internet Start
SECTION IV QUESTION 22
5 of 8
Users of HDF can provide metadata on their data only through attributes previously defined by the HDF library.
True
False
Preview Material
Submit Answer(s)
Previous Question
Next Question
Applet started
Internet zone



Progress Report for the HDF Laboratory

Number of questions answered correctly:	5	12.0%
Number of questions answered incorrectly:	3	7.0%
Number of questions left unanswered:	31	79.0%

Below is the list of questions answered correctly:

2, 3, 4, 9, 16,

Below is the list of questions answered incorrectly:

5, 7, 8,

Below is the list of questions left unanswered:

1, 6, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,

Return to Lab