NASA Technical Memorandum 104594, Vol. 4

MODIS Technical Report Series

Volume 4, MODIS Data Access User's Guide - Scan Cube Format

Virginia L. Kalb
Goddard Space Flight Center
Greenbelt, Maryland

Thomas E. Goff
Research and Data Systems Corporation
Greenbelt, Maryland

National Aeronautics and
Space Administration

**Scientific and Technical
Information Program**

1994

This publication is available from the NASA Center for AeroSpace Information,
800 Elkridge Landing Road, Linthicum Heights, MD 21090-2934, (301) 621-0390.

Table of Contents

List of Figures

List of Tables

INTRODUCTION

The library functions described in this document allow read and write access to the Earth Observing System (EOS) Moderate Resolution Spectroradiometer (MODIS) derived data products. They also provide access to remotely sensed data from other scanning instruments, both heritage and new instruments, that have been transformed into the MODIS scan cube format. These functions have been designed to allow all users to have easy access to all information components contained in MODIS formatted datasets. They also provide a flexible and simple method for writing MODIS formatted datasets.

The MODIS scanning technique is unique when compared with other instruments, due to its large off nadir viewing angle coupled with the simultaneous capture of multiple lines. Nevertheless, MODIS instrument data will be available as both raw detector counts and at-satellite radiances in the instrument viewing geometry. This includes all aberrations inherent in the MODIS scanning technique. Previous instruments with similar types of instrument viewing distortions have been resampled before general consumption and subsequent ingestion by science algorithms. However, highest algorithmic accuracy for science data products is maintained by using this unresampled MODIS data.

The high MODIS data volume (approaching a half terabyte per day) has been a driving factor in the library design. To avoid recopying data for user access, all I/O is performed through dynamic "C" pointer manipulation. The user has easy access to any instrument band or data product parameter in a dataset through the "C" data structure syntax. This technique is also used for creating output data products.

BACKGROUND

MODIS instrument data is used by a wide array of scientists to derive information about the Earth. In general, scientists ingest MODIS data along with ancillary data to produce MODIS Data Products. These products are then used as part of the knowledge base of information to determine long term changes in the Earth biosphere, which is then used to make policy decisions at all governmental levels. MODIS algorithms are executed in, and the resulting data products are archived by, the production EOS core system (ECS) distributed information system (EOSDIS) which also archives the resulting data products.

Data derived from the MODIS instrument can be divided into three types, corresponding to three spatial domains: the scan cube domain, an orbital rectilinear domain, and a geographically based mapped domain. The scan cube domain consists of all Data Products that are produced at MODIS instrument pixel IFOVs, or spatial aggregations thereof. (See Figure7 in Appendix A for a visualization of the MODIS scan cube.) These products are the MODIS Level-1A raw counts, MODIS Level-1A Geolocation Data Product, MODIS Level-1B at-satellite radiances, and Level-2 products that are produced in the true MODIS scan geometry. The rectilinear domain consists of MODIS scan cube data that has been resampled into a coordinate system on EOS across track and along track axes. The mapped domain encompasses all data products that have been gridded, binned, or otherwise aggregated onto a map projection or other Earth geoid based coordinate system such as the MODIS Level-3 data products.

Algorithm developers must consider the spatial domain in which the algorithm will operate. Algorithms that do not incorporate a spatial component will operate in the scan cube domain. Algorithms that incorporate spatial components or will use MISR Level -1b2 or similar data for Level-2 scan cube processing should consider processing in the rectilinear or mapped domains. Algorithms that depend on more accurate band to band registration or uniform ground field of views should also consider the rectilinear and/or mapped domains. Algorithms that depend on in situ, external ancillary inputs, other satellite or instrument data sources, or require fixed ground areas will require interactions with data in the mapped domain. Algorithm developers must be cognizant of the MODIS "bow tie" overlap effect in the scan cube domain. See the various MODIS publications listed in the reference section of this document for "bow tie" effect descriptions. In all cases, the user needs to consider the possibility of modifying the algorithm to use the "earlier" domains when possible. Transforming data products directly from the scan cube domain to the mapped domain, without going through an intermediate rectilinear resampling, is highly recommended. This will minimize data resampling, cross product data dependencies, and computer based processing power. It should be noted that the historical Landsat TM data and the EOS MISR data are resampled from the instrument geometry to scenes or orbital axes before any science algorithms are applied.

More complete descriptions of the MODIS spatial domains are given in Appendix A to this document. Considerations for each domain, and transformations among domains are given in the "MODIS Processing Spatial Domains" paper (see Reference section).

IDENTIFICATION

This document is the first in a series of User's Guides describing the library and utility functions that the SDST will be generating. This edition of the MODIS SDST Library USER'S GUIDE applies to the September 1994 delivery of the MODIS scan cube I/O access routines, version 1.0, library source code. It includes descriptions of all the pieces necessary to use the library, including sample data product header include files for AVHRR. Additional data product header include files for actual MODIS products and other data source header include files will be available via electronic means and are not a part of this edition of the guide.

Several small test datasets derived from AVHRR are included in the distribution and used for illustration purposes in this document. More complete datasets are available from the MODIS SDST or can be created using information in the appendices to this document.

This document is written and maintained by members of the MODIS Science Data and Support Team under the auspices of NASA Goddard Space Flight Center, code 920.2, in Greenbelt, MD.

SCOPE

This guide contains information applicable to both the use and maintenance of the library functions. It is directed toward four classes of users: normal MODIS algorithm creators and data users, sophisticated creators of spatially derived algorithms, those who will provide additional instrument datasets (other than MODIS), and those who will alter existing or add new data product header include files. This document does not describe the internal workings of the library source code, but provides descriptions of the methods utilized by the library to access the datasets to aid users in understanding the library's capabilities.

This edition of the MODIS SDST Library User's Guide describes the initial "C" language version of the SDST Library input and output (I/O) functions. These were created for the purpose of reading and writing currently available data formatted as MODIS data products. A future release of this document will include similar FORTRAN functions written in either FORTRAN77 or FORTRAN90. The current suite of functions at this release is limited to the MODIS Level-1A, Level-1B, and Level-2 Data Products. These are the data products that are a part of the scan cube domain. It does not cover access to Level-3 or Level-4 data products which are in the mapped domain.

Use of this library relies heavily on the user's understanding of how to use each "C" data structure as defined uniquely for each format Data Product. These are defined in data product header include files which are unique to each formal Data Product. These header include files, available to all users of this library, will be under SDST configuration management and distribution. They will be published and announced via Email to all MODIS algorithm developers. This will help to insure the synchronization of data product definitions among the MODIS Team Member responsible for the formal Data Product, the Science Product Support Office (SPSO) database entries, the data product header include files associated with this library, and all users of the data products.

The MODIS SDST proposes to write these data product header include files for algorithm developers in conjunction with the SPSO and MODIS Team Members or their programming representatives. Although the access to MODIS data is easily understood, the programming techniques utilized in the header include files are necessarily complex to accomplish the encapsulation of the data products. Documentation of these techniques is given for completeness but is not mandatory reading for library use.

PURPOSE AND OBJECTIVES

This library has been designed to allow read and write access to data from a suite of scanning type instruments whose data have been transformed into a MODIS viewing geometry and written in a MODIS scan cube data structure. This allows a common library to be used for algorithm development. Simulated and/or test data sets will be used for algorithm development and production environment testing before actual MODIS data are available. The currently available datasets emphasize AVHRR data, with TM data forthcoming, but are expected to be expanded to other instruments such as MSS, CZCS, MAS, or other data sources on a user requested basis. The internal format for the dataset contents and a simple example are given in Appendix E to this document.

The MODIS scan cube concept contains data values in a three dimensional ragged array format where the array dimensions correspond to the across track direction, the along track dimension, and the number of bands. The ragged array terminology applies to a data structure in which the sizes of each component of the array are not constant. For example, a two dimensional ragged array data structure containing the words of a sentence would use the number of words in the sentence as one dimension and the variable number of characters per word in the other dimension. In the MODIS case, the number of bands is constant, but the number of lines per scan cube in the along track dimension is a function of the band number, and the number of pixels across track varies across scan cubes.

The three dimensional scan cube array can be collapsed into a two dimensional data array consisting of across track pixels and bands by setting the along track dimension to one (1). This allows single line scanning instruments to be accommodated by the MODIS scan cube data structure. The number of elements in each dimension are included in the library structures and are explicitly available (no function call required) to the user. This allows the use of these library functions for uses beyond the scope of the MODIS effort.

The library implementation, which incorporates a "data product header include file" technique, allows user written algorithms to determine the sizes of all data array elements as a function of the instrument data source. This allows a single algorithm process (program) to be easily written in a manner such that any instrument source can be ingested. Information such as the number of bands, band central wavelengths, band widths, number of pixels, etc. can be made available from each data product header include file. Multiple data product header include files can be used within a single program, thereby allowing simultaneous access to other data products. An example of this facility comes from AVHRR in which the data has a different number of across track pixels and bands from MODIS, and from TM in which a different instrument IFOV for each band is employed. AVHRR data are currently made available to algorithm developers in raw instrument counts, the original NOAA albedo / brightness temperature data values, and also converted to radiance energy values. Actual MODIS data will be available only in radiance energy values. Utilizing the library header include files, which contain data type and structure specifications for each of the available input and output data source, allows the data types to be known to an application program without performing data type conversion within the library. Additional data types can be accommodated with a custom header file describing the new data source. The library does not need to be recompiled to add new data sources.

The function calling sequences and parameters are expected to remain constant even though the underlying implementation may change. This ensures a stable interface for algorithm developers.

The remainder of this document addresses the I/O routines and illustrates, by example, how to access the data in the MODIS scan cube. Each section of this library contains functions that allow a science algorithm to input and output MODIS data products, or other instrument data masquerading as MODIS scan cubes. Library functions that transform or resample data products from MODIS scan cubes to other forms have yet to be defined, and are not a part of this library effort.

DOCUMENTATION CONVENTIONS

The conventions used in this document are described as follows:

Table 1. Conventions Used

Times Roman font	Normal text and narrative comments added to source listings.
`Courier font`	Program source code source listings copied from the library distribution and pseudo code examples
Arial	Program output as if received at a terminal
carots (< and >)	Enclose generic items to be replaced by specific user supplied names or descriptions. This convention is also used in UNIX source listings to specify the default location for operating system library include files.
double quotes (")	Enclose a character string set that is to be considered as a unit.
Leading Caps	Are used to designate a formal noun, as in a reference to a formal registered (with the SPSO) Data Product.
UNIX meta-characters	A method of designating occurrences of characters using the UNIX conventions for metacharacters, specifically the * character to represent one or more occurrences of a character.

TOOLKIT USAGE

Four methods for illustrating the use of the library functions, and a section with technical information, are included in this section. The Quick Example presents an actual small algorithm with three parameters. The source code is included in the library distribution and can be quickly cloned to generate simple algorithms which can be expanded with time. This is the easiest method for getting started with the library function calls. The Functional Overview section goes into the philosophical aspects in the design and use of the I/O library. It includes descriptions of the functions and describes the sequence and coupling among the functions. The Advanced Example section presents a sample algorithm that exercises the library's capabilities for handling multiple scan cubes at a time with the use of moving windows. This allows two dimensional access across the boundaries of these scan cubes. This example illustrates the library's use with algorithms which need to know the relative locations of neighboring pixels for spatial operations. The final Function Reference section lists the functions in alphabetical order by function entry point names and included the details about each function call, such as error returns and ANSI function prototype definitions. The included Technical Points section presents a list of assumptions and notes concerning the use of these I/O library functions.

QUICK EXAMPLE

To illustrate the simplicity of library usage, a fully functional actual code example with make file is included in this section. It reads MODIS Level-1B radiances (actually AVHRR in MODIS form) and calculates a simple vegetation index (VI) with an average thermal temperature and a quality assurance (QA) value as additional parameters. Only those lines of source code directly applicable to a potential science algorithm are commented.

```
#include <stdio.h>
#include <string.h>
#include "AVHRR.h"                          <- define AVHRR data in MODIS format
#include "VI.h"                             <- define pseudo VI output data product
#include "MOD_IO_prototypes.h"

main()
{
  AVHRRscancube ibuf;                       <- declare an input data structure
  VIscancube obuf;                          <- declare an output data structure
  short fdin,fdout,ierr,k;
  float pixCh1, pixCh2, pixCh4, pixCh5;
  long bufid;
  long npixels, nx, ny, x, y;

  fdin=MOD_IO_openMasterInputDataset("AVHRR_Orbit543.L1b.MODISlike",
                                                        &AVHRRheader);
  fdout = MOD_IO_openOutputDataset("viOutputfile", &VIheader);
  while ((bufid = MOD_IO_readSwath(fdin, &ibuf)) > 0) { <- input the data
    ierr = MOD_IO_allocateOutputBuffer(fdout, &obuf);   <- obtain an output buffer
    npixels = ibuf.b[0].nPixels;                        <- how many pixels to process
    for (k=0; k<npixels; k++) {        (Note that we know apriori that each band of AVHRR
                                       has the same resolution, hence the same number of pixels.)
      pixCh1 = ibuf.b[0].data[k];                       <- get value for band 1
      pixCh2 = ibuf.b[1].data[k];                       <- band 2, etc..., from a band within the
      pixCh4 = ibuf.b[3].data[k];                       <- input data buffer .b[] containing
      pixCh5 = ibuf.b[4].data[k];                       <- a data array .data[]
      obuf.vegIndex.data[k] = (pixCh2 - pixCh1)/(pixCh2 + pixCh1);<-calculate the VI
      obuf.viQuality.data[k] = 0;
      if(obuf.vegIndex.data[k] < 1.0)obuf.viQuality.data[k] = -1.0;<-determine QA
      if(obuf.vegIndex.data[k] > 1.0)obuf.viQuality.data[k] = +1.0;
      obuf.avgTemp.data[k] = (pixCh4 + pixCh5)/2.0;     <- find the average temperature
    }
    ierr = MOD_IO_writeSwath(fdout);                    <- write the output buffer
  }
  MOD_IO_closeDatasets();
}
```

Figure 1. Simple algorithm example.

The three VI parameters (`vegIndex, viQuality, and avgTemp`) are treated as subelements of a complex data structure. Each element of this structure corresponds to one instance of a complex data product value. Use of the "C" data structure concept instead of separate arrays guarantees the synchronization of the various science parameters.

The above example can be compiled with the MODIS I/O Utility Library via the following UNIX **make** file listed in Figure 2. This is a minimalist **make** file and does not include directory details and environmental variables. See the **make** files, README file and INSTALL file in the library distribution for details. (Note that the "`-Aa`" option flag is the ANSI switch on HP and SGI, not Sun compilers.)

```
VI: vi.o MOD_IO_lib.o
        cc -Aa -o VI vi.o MOD_IO_lib.o

vi.o: vi.c MOD_IO_lib.h AVHRR.h VI.h DataDescriptor.h
        cc -Aa -c vi.c MOD_IO_lib.h DataDescriptor.h AVHRR.h VI.h

MOD_IO_lib.o: MOD_IO_lib.c MOD_IO_lib.h DataDescriptor.h
        cc -Aa -c MOD_IO_lib.c
```

Figure 2. Simple example **make** file.

The simplicity of this source code is made possible by the predefined data product header include file. The header file for the example VI data product that defined the output data product and the parameters of this output data product is included in the library distribution. The explanation of the contents of this header include file is the subject of a Appendix D to this document.

TECHNICAL POINTS

- This initial release includes "C" bindings only. Bindings for FORTRAN functions can not be implemented due to the lack of data structures in FORTRAN77. Bindings for FORTRAN90, however, are possible. Algorithms that are written in FORTRAN77 (with POSIX and ECS approved extensions) will require the writing of "wrapper" functions that extract MODIS data for each band or parameter individually and supply this data to a calling FORTRAN function. The SDST is considering adding this capability and needs user input to generate a requirement.

- Each band or parameter in a scan cube in the data set can be accessed as either a linear array (vector) or a two dimensional array (matrix). This is the user's choice when writing the algorithm. Scan cube sizes can vary from one scan cube to the next. The bands or parameters can have different array sizes that are integer multiples of each other to accommodate multiple data resolutions.

- Output data product scan cubes are coupled to a master input scan cube. This allows the MODIS scan cube Geolocation Data Product to be applied to an output product, by reference. That is: a new geolocation data product does not need to be created for each output product, the input geolocation applies to the output product as well. This coupling is maintained in all subsequent generations of MODIS Data Products.

- Additional input data in the scan cube or rectilinear domain can be accessed as ancillary inputs by this same set of library calls. No additional functions are needed, but care must be exercised in the scan cube domain to maintain spatial registration. The onus is on the user, not this utility library, to insure proper usage of ancillary data.

- All accesses to MODIS data are assumed to be sequential. Scan cube data can not be accessed in a random manner with these library calls.

- Error returns are returned to the user as function returns. Negative numbers are an error, zero or optional positive numbers indicate a non error condition. Negative error returns from -1 to -99 are non fatal (soft) errors such as encountering an end of file (EOF) condition. Error returns from -100 to -??? indicate a fatal (hard) error such as a memory overflow or illegal dataset. This utility library will endeavor use the generic PGS error determination routines in a future release, after they are more fully understood by the developers of this library. NOTE that this current library release (1.0) produces error messages to stderr and terminates if a fatal error occurs. This is a temporary safety mechanism for algorithm development and will not be allowed in the PGS environment. Error return values and messages are documented in the Alphabetized Function Reference section of this user's guide.

- All functions use ANSI prototyping beginning with the 1.0 release. These prototypes are available in a "C" header include file, "MOD_IO_prototypes.h", included with the library distribution.

- A set of headers will be publicly available that define the Data Product data structures, file set descriptions, constant variables (typedefs), etc. Any header include files for formal Data Product structures generated by algorithm developers are required to be given back to the SDST Utility Library configured management access point for availability to all library users. A example generic header is described and included in Appendix B to this document.

- All function calls dynamically allocate space within the library and set pointers to that data space in the user's data structure. This frees the science algorithm developer (the user of the library) from concerns about memory allocation details and data array boundary overruns (memory leaks), provided the header structures are used. Most importantly, it allows variable length scan cube records for both the science and calibration portions of each MODIS scan. This ability applies to both the scan cube and rectilinear domains.

- The scan cubes are assumed to be sequential in time. That is: no global descriptor such as a scan cube counter or equivalent time based discriminator is implemented, but will be required to identify scan cubes in the production environment. Opening more than one input dataset, (granule or orbit), is allowed. Additional datasets that do not match the global dataset descriptor or master scan cube ID will product a non fatal error in future releases. The user, not the library, is then responsible for mapping or "stitching" techniques to insure geographical registration.

- More than one scan cube can be accessed (sequentially) for each opened dataset. The limit to the number of input data sets concurrently open is three (3), with user access to five (5) scan cubes per dataset. This access can be "round robin'ed" within the five (5) at a time limit. Three (3) output data sets can also be generated concurrently. These limits are parameterized in the library internal header file and could be easily changed by recompiling the library with different values. Note that larger numbers will increase the library internal table sizes.

- Users interested in the MODIS calibration and correlation with documentation supplied by Santa Barbara Research Center (SBRC , the MODIS instrument builder) will need the information in this paragraph. Science algorithm developers should not need this information but it is included here for reference purposes. External documentation with the detector numbering scheme used by SBRC will require that the user of MODIS Level-1A and Level-1B data know about the "flipping" of this detector numbering with respect to the scan cube line numbering sequence. See Figure 3. The Utility Library uses a "line within scan cube" numbering scheme that increases in the along track direction. This is opposite to the SBRC detector numbering scheme. This line numbering scheme is used in the MODIS Level-1A, Level-1B, and Level-2 Data Products.

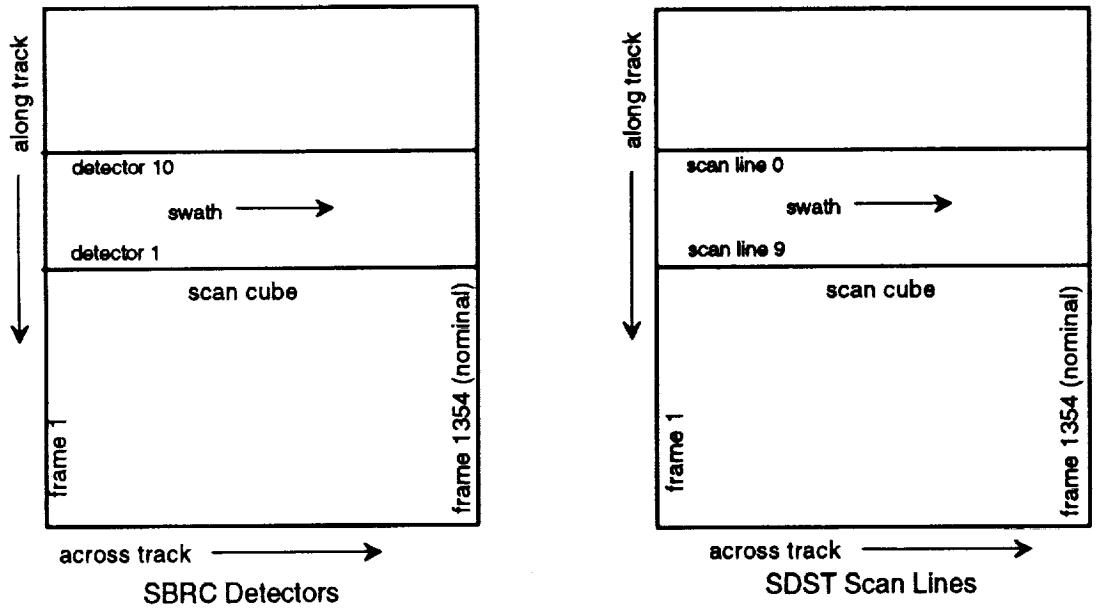

Figure 3. Detector numbering.

- The input and output MODIS data calls can be intermixed within the same program, but must be kept in the order illustrated in the included source code and pseudo code examples. This sequencing facilitates synchronization of the Geolocation Data Product with all output Data Products.

- The library is designed to enforce function call ordering and will generate error conditions otherwise.

- Fully commented sample programs that include the incorporation of heritage and/or dummy algorithms are available in the library distribution or from members of the SDST algorithm transfer team (ATT) directly. Additional help, information, and sample code is available from ATT members.

FUNCTIONAL OVERVIEW

This utility library can be used in a simple manner in which a MODIS Data Product is generated from only the MODIS Radiance input values, or a more complicated manner in which concurrent use of multiple scan cubes and more than one input source are used to generate a MODIS Data Product. The example pseudo code , listed in Figure 4, introduces the simple techniques for accessing MODIS data. A discussion of more complex capabilities of this library is contained is a later section of this user's guide.

PSEUDO CODE EXAMPLE

The following pseudo code example contains an ordered list of the basic function calls required to develop an algorithm. This sequence of calls performs the following functions. The MODIS scan cube domain input Data Products are obtained with a sequence of input library calls, allocation of output scan cubes are obtained via a library call, the algorithm results are placed into the new scan cubes, and Data Products are transferred to the file system with output library calls. The library is initialized with data set "file open" calls, scan cubes are staged with a "read cube" call, individual MODIS bands or other data product parameters are made available to the user within a "C" data structure, output scan cubes are allocated, output data product parameters are calculated, the scan cube of data product is placed into the output file, and finally the data sets are closed with a "file close" call. Multiple bands or parameters are accessible by accessing elements of each input data structure, and multiple sequential scan cubes are obtained with multiple "read cube" calls into separate input data address areas. Pseudo code with the library function names for a minimum algorithm is shown in Figure 4.

```
filein = MOD_IO_openMasterInputDataset( inputFileName,
                                        InDataDescriptorAddr )
fileout = MOD_IO_openOutputDataset( outputFileName,
                                        OutDataDescriptorAddr )
for every scan cube:
     MOD_IO_readSwath( filein, InBufferAddr )
     MOD_IO_allocateOutputBuffer( fileout, OutBufferAddr )
     npixels = InBufferAddr.band1.nPixels - 1
     compute the parameters for this algorithm, for k=0 to npixels
          OutBuffer.param1.data[k] = a function of(
                                InBuffer.band1.data[k], etc. )
     MOD_IO_writeSwath( fileout )
end of for every scan cube
MOD_IO_closeDatasets()
```

Figure 4. Pseudo code with the library function names for a minimum algorithm.

Note the use of the "C" data structure `OutBuffer.param1.data[k]` in which each derived value of an output Data Product is placed into an output buffer `'OutBuffer'`, in a specified parameter `'param1'`, in a linear array `'data[]'`. The size of the array `'data[]'`, and the corresponding range of the index `'k'`, are dynamically altered by the value of `'npixels'` for each read of a scan cube. This is necessary to prevent the placing of values outside the memory bounds of the output data product.

FUNCTION CALL SEQUENCE

The library functions must be invoked in the proper order as illustrated in Figure 4 - Pseudo Code example (previous section). The functions performed by the individual function calls, and their coupling with other functions, are given in the following paragraphs.

The '`filein = MOD_IO_openMasterInputDataset( inputFileName, InDataDescriptorAddr )`' function returns a unique identifier ('`filein`') to a dataset containing the MODIS Level-1A, MODIS Level-1B, MODIS Geolocation, or other MODIS Level-2 input Data Products. It requires the user to specify a character array '`inputFileName`' containing the fully qualified name of the MODIS or "MODIS like" input data files. The library checks the input dataset for a predefined dataset type specifier e.g. "MODIS_L1B". The user supplied parameter '`InDataDescriptorAddr`' is the address of an instance of a data structure defined in the unique product header include file for this input product. The formal name of this data structure is given in that header include file. The header include file also contains a character array with a dataset contents specifier e.g., "AVHRR_albedo_and_radiances". These characters are used as a sanity check with the dataset embedded header information to make sure the user knows the source and type of data expected. For example, an AVHRR dataset can be used as input to an algorithm for development purposes, but the user must know apriori (and specified in the AVHRR.h include file) which AVHRR bands to use in the algorithm. These bands will have different central wave numbers and band widths for other instrument data sources such as MAS or MSS. Additional information, unique to each dataset, is available to the library through the dataset contents. The header include files contain the generic dataset parameters and the specific formatting information for that dataset.

The '`fileout = MOD_IO_openOutputDataset( oputputFileName, OutDataDescriptorAddr )`' function returns a unique identifier ('`fileout`') to a dataset created by the library to contain the user derived output Data Product. It requires the user to specify a character array '`outputFileName`' containing the fully qualified name of the MODIS or "MODIS like" output data file. The library places the predefined dataset type specifier ("MODIS") into the output data product volume header. The user supplied parameter '`OutDataDescriptorAddr`' is the address of a data structure defined in the unique product header include file for this output product. The library also places a character array with a dataset contents specifier e.g. "AVHRR_derived_Locust_probability" into the output file volume header. This is obtained from the data product formal "<product.h>" file. Geolocation that has been tagged to the Master input scan cubes also applies to the output data product. The master input scan cube dataset must be open before this cal can be made.

The '`err = MOD_IO_readSwath( filein, InBufferAddr )`' function reads a scan cube's worth of data from the dataset specified by '`filein`' into the library space, not the user space. The address of the user data structure parameter '`InBufferAddr`' is passed to the library function. The library passes pointers to the science data to the user through the user's data structure '`InBuffer`'. In addition, the library places the number of Earth viewing along track elements, the number of across track elements per line, and the number of total spatial elements (across track times along track) into this user supplied pointer to the scan cube data structure. For the real MODIS instrument, the nominal number of across track spatial elements is 1354 with a fixed 10 detectors along track, thereby creating 13540 spatial elements. MODIS bands 1 through 7 will have larger numbers for these values. (Note that spatial elements are not the same as IFOVs for MODIS bands 1 through 7). For the AVHRR MODIS "look alike", these

returned numbers are 2048 pixels across, 10 detectors down, and 20480 total elements. Caution: this returned value can be different for each MODIS scan cube because the number of pixels across a MODIS scan is ground commandable. This can also vary across instrument data sources. The library handles data sources with a different number of pixels in each of the two dimensions, and a different number of pixels per band (or parameter), and can handle these within the same algorithm program.

Each input Data Product is represented as a data structure to the user. Pointers are used within the data structure to allow access to each individual parameter. This technique allows the library to allocate memory space dynamically when the size of the scan cube changes. This is performed without user intervention. It also involves no memory transfers and is thus very efficient. The user has access to all the information and data sizes required to use the input data via a set of predefined (in the data product header include file) data structure elements. This is explained in the header file descriptions in a following section and also illustrated in examples included in the appendix to this document. IMPORTANT: Users who plan to use a scan cube call for spatial processing need to be knowledgeable of the detector numbering, band to / from detector translations, band registration over variable spatial IFOVs, and pixel line numbering schemes.

If the input file is a MODIS Level-1A dataset, additional information as defined in the MODIS Level-1A Data Product header include file is available. This allows access to the onboard calibration data. Details on how to access these components is contained in the Level-1A Data Product header include file and is not included here.

The '`err = MOD_IO_allocateOutputBuffer( fileout, OutBufferAddr )`' function creates space in the library allocated memory area for an output scan cube that corresponds to the currently specified "Master scan cube". This synchronizes the output Data Product with the Master input data Product, thereby allowing the appropriate geolocation dataset to apply to this output Data Product. The output file is designated by the user supplied '`fileout`' indicator. All information that the algorithm requires (across and along sizes, number of pixels, etc.) is now available as structure elements in the formal output Data Product data structure pointed to by the user supplied address '`OutBufferAddr`'. These sizes must be used in the algorithm code to prevent 'stepping over' or overwriting the neighboring data elements. Extreme cases will produce a memory protection fault. The function prints an error message if something goes wrong.

The '`MOD_IO_writeSwath( fileout )`' function writes the user generated data from the library memory space onto the disk. The user parameter '`fileout`' indicates to the library which output dataset is to be written. An error code is returned if unsuccessful.

The '`MOD_IO_closeDatasets()`' call, the '`MOD_IO_closeInputDataset( filein )`' call, and the '`MOD_IO_closeOutputDataset( fileout )`' call deallocate the library memory space for either the specified input file, output file, or all dataset files. It also appends a trailer record to an output dataset to indicate a logical end of file (EOF).

ADVANCED EXAMPLE

This section illustrates the use of the library for spatial processing by allowing more than one scan cube to be in memory at a time and utilizes the two dimensional access to the scan cube data. This code example performs a moving window spatial average over a 3 by 3 pixel area across scan cube boundaries. This requires the use of two input scan cube buffers for each output buffer. Figure 5 illustrates the spatial relationship for the indices in the code example. The X in the figure corresponds to the placement of the calculated average in the output data product.

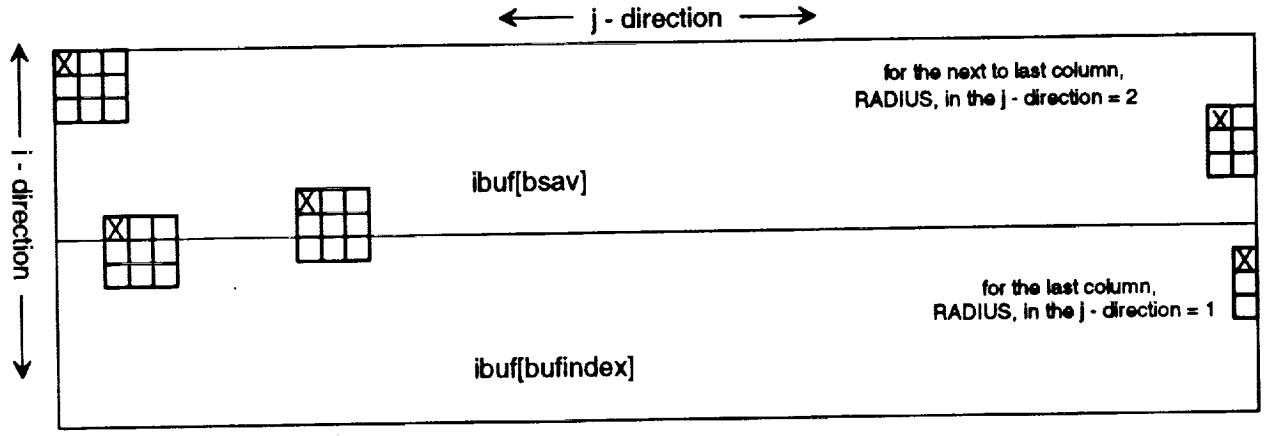

Figure 5. Multiple scan cube processing illustration.

The key to obtaining the data within the scan cube is the use of "C" data structures with dynamic pointers. The example code in this section is graphically illustrated in Figure 5. The data structure contents, including the pointers, is reinitialized on each invocation of a `MOD_IO_readSwath` or `MOD_IO_allocateOutputBuffer` function. The data structure is declared by the user, but its address is passed to the library so that the library can manage dynamic memory allocation and fill the data structure. The user gains access to the data indirectly through the pointer mechanism. Note that while the data product header include file provides both one dimensional and two dimensional access to the same scan cube data, a single algorithm can not access the same data utilizing both techniques at the same time.

```
#include <stdio.h>
#include <string.h>
#include "AVHRR.h"                          <- the MODIS header include files
#include "MOD_IO_prototypes.h"
#define RADIUS 3
#define Min(x,y) (x < y ? x : y)

main()
{
  AVHRR2Dscancube ibuf[2];                  <- declare 2 input buffers of type AVHRR2Dscancube
  AVHRR2Dscancube obuf;                     <- and one output buffer of the same type
  short fdin, fdout, ierr, bandindex;
```

Figure 6. Two dimensional access example.

```
long bufid[2];
short bufindex, bsav, j, i, kj, ki, radiusi, radiusj;
long noutput, nj, ni;
float output;

fdin=MOD_IO_openMasterInputDataset("scancubes.small", &AVHRRheader);
fdout = MOD_IO_openOutputDataset("avgOutputfile", &AVHRRheader);
if((bufid[0] = MOD_IO_readSwath(fdin, &ibuf[0])) < 0) exit(-1);
bsav=0;
bufindex=1;
while((bufid[bufindex] = MOD_IO_readSwath(fdin, &ibuf[bufindex])) > 0) {
     ^^--This while loop is executed until the readSwath function indicates no more data are available.
  fprintf(stderr,"main: read id = %d\n", bufid[bufindex]);
  ierr = MOD_IO_allocateOutputBuffer(fdout, &obuf);<- request an output buffer from
                                                     the library.
  for(bandindex=0; bandindex<AVHRRheader.nbands; bandindex++) {
    nj = ibuf[bufindex].b[bandindex].nAlongScan;<- the number of elements in
    ni = ibuf[bufindex].b[bandindex].nAlongTrack;<-    each direction.
    radiusi = RADIUS; <- how far in the i direction to average.
    for (j=0; j<nj; j++) {
                    /* Adjust the radius for rightmost columns, where a
                                  smaller window is needed.  */
      radiusj = Min(RADIUS, nj-j);   <- how far in the j direction to average.
      for (i=0; i<ni-2; i++) {
        output = 0; noutput = 0;
        for (kj=0; kj<radiusj; kj++) {
          for (ki=0; ki<radiusi; ki++) {
            output += ibuf[bsav].b[bandindex].data->xy[j+kj][i+ki];
            noutput++;
       } } ^^ sum up the pixel values in a 3 x 3 pixel area. The data structure used here addresses each
                  pixel by selecting the buffer [bsav] from an array of input buffers ibuf[bsav], indexed
                  by the band number [bandindex], further indexed to the science data .data which points
                  to the two dimensional array representation xy[ ][ ], using normally computed array
                  indices j+kj and i+ki.
        obuf.b[bandindex].data->xy[j][i] = (output/noutput+.5);
      }   ^^ the output product is contained with a single buffer obuf, using the same data structure technique
              as the input buffer.
                      /* Next to last line in the current scancube: need the
                                  top line from the next scancube: */
      output = 0; noutput = 0;
      for (kj=0; kj<radiusj; kj++) {
        output += ibuf[bsav].b[bandindex].data->xy[j+kj][ni-2]; noutput++;
        output += ibuf[bsav].b[bandindex].data->xy[j+kj][ni-1]; noutput++;
        output += ibuf[bufindex].b[bandindex].data->xy[j+kj][0]; noutput++;
      }                ^^   NOTE: the ibuf indices bsav and bufindex indicate that different scan cubes
                                  are accessed here.
```

Figure 6. Two dimensional access example (continued).

```
        obuf.b[bandindex].data->xy[j][ni-2] = (output/noutput+.5);
                        /* Last line in the current scancube: need the top 2
                                    lines from the next scancube: */
        output = 0; noutput = 0;
        for (kj=0; kj<radiusj; kj++) {
          output += ibuf[bsav].b[bandindex].data->xy[j+kj][ni-1]; noutput++;
          output += ibuf[bufindex].b[bandindex].data->xy[j+kj][0]; noutput++;
          output += ibuf[bufindex].b[bandindex].data->xy[j+kj][1]; noutput++;
        }            ^^ one scan cube from bsav and two scan cubes from bufindex.
        obuf.b[bandindex].data->xy[j][ni-1] = (output/noutput+.5); <- .5 is used to
                                                                  round to integer numbers properly.
      }
    }
    bsav = bufindex;
    bufindex ^= 1;       /* we are using only 2 scan cubes so an exclusive or is
                                    used to toggle the index */
    ierr = MOD_IO_writeSwath(fdout);
  }
  MOD_IO_closeDatasets();
}
```

Figure 6. Two dimensional access example (continued).

The above example does not include the use of more than one ancillary input MODIS data product, or auxiliary data from non MODIS sources. Ancillary data, obtained from other data sources, can be accessed via these library calls and included in the algorithm code after conversion to MODIS scan cube format. Auxiliary data is obtained via function calls that are not a part of this library and can be in any data format. The algorithm writer must insure that these extra data sources are used appropriately by checking geolocation information, spatial sizes, units, and etc. associated with these ancillary data. This information is made available to the user via the dataset header contents and accessed via the data structure pointer mechanism. MODIS output data products are kept in synchronization with the Master input data product by this library. The auxiliary data function calls can provide correct data by examining the input parameters that specify the area and units requested by the calling program.

FUNCTION REFERENCES

This section contains the alphabetical ordering of the function calls documented in this User's Guide. It is meant to be a quick reference to the individual function calls. Details of the ANSI prototyping specifications are included.

All error messages produced by the library functions are documented here. As a debugging aid, almost all errors are produced as a result of the one of the following conditions:

1 - Improper sequencing of function calls.

2 - Incompatible input data file and data product header include file (e.g., user included "AVHRR.h" but is trying to read a TM dataset).

3 - Bad data include file; perhaps an existing <product>.h file was cloned and improperly altered for a new data product.

Errors encountered by the current implementation of these functions will cause an immediate abort (with an error message to the UNIX stderr). This behavior will be altered in a future release to use the ECS error trapping and reporting functions as required in a production environment. The error messages documented here will also be passed to the MODIS log on the MODIS team leader's computer facility (TLCF). The source code contains debug statements that can be 'turned on' at compile time for additional information. These messages are not documented in this User's Guide.

MOD_IO_allocateOutputBuffer

SYNOPSIS: err = short MOD_IO_allocateOutputBuffer(short <output file designator>,
void * <returned buffer address>)

DESCRIPTION: This function requests that the library obtain memory for an output data buffer from the operating system. The space required is known from the internal data structure associated with the user supplied <output file designator> and the master scan cube.

INPUTS: short <output file designator> : The output file designator returned by the <MOD_IO_openOutputDataset function.

OUTPUTS: void * <returned buffer address> : The address of the user declared instance of the scan cube data structure, into which the library places the updated pointers to the new data.

RETURN: err : The return status value;
0 if all went well; error message and program termination if not.

EXAMPLE:

```
#include "MOD_IO_prototypes.h"
#include "AVHRR.h"
AVHRRscancube outputBuffer;

fileout = MOD_IO_openOutputDataset( ...
if( MOD_IO_allocateOutputBuffer( fileout, &outputBuffer ) != 0 )
        printf( "allocate error: %d\n", err );
```

COMMENTS: When an output buffer is allocated by the library, it inherits the scan cube number of frames from the corresponding "Master Input Scan Cube".

ERROR MESSAGES:
"No output stream to write to!"
"allocateOutputBuffer: Invalid output stream id"
"Sorry, output buffer for id = <number> has not been flushed" "can't recover"
"Sorry, cannot match next output id with master input buffers"
"Output id =<id number>, input list = <available buffer ids>" "can't recover"
"Sorry,unable to allocate output buffer for band <number>""can't recover"

MOD_IO_closeDatasets

SYNOPSIS: err = short MOD_IO_closeDatasets(void)

DESCRIPTION: This function closes all currently open files.

INPUTS: (n/a)

OUTPUTS: (n/a)

RETURN: err : The return status value is 0 if successful; error messages otherwise.

EXAMPLE:

```
#include "MOD_IO_prototypes.h"
#include "AVHRR.h"

if( MOD_IO_closeDatasets () != 0 ) printf( "Not being able to close files is bad
news\n" );
```

COMMENTS: This function just calls the MOD_IO_closeInputDataset and MOD_IO_closeOutputDataset functions. See those descriptions for details.

ERROR MESSAGES: See MOD_IO_closeInputDataset and MOD_IO_closeOutputDataset function calls.

MOD_IO_closeInputDataset

SYNOPSIS: err = short MOD_IO_closeInputDataset(short <inputFileId>)

DESCRIPTION: This function closes the user specified input dataset file and deallocates (frees) all memory (malloc'ed) associated with the file designator <inputFileId>.

INPUTS: short <inputFileId> : The file designator provided to the user by the MOD_IO_openInputDataset function.

OUTPUTS: (n/a)

RETURN: err : The return status value; 0 for normal operation.

EXAMPLE:

```
#include "MOD_IO_prototypes.h"
#include "AVHRR.h"

fileIn = MOD_IO_openMasterInputDataset( ...
if( MOD_IO_closeInputDataset ( fileIn ) != 0 ) printf( "Not being able to close files is bad news\n" );
```

COMMENTS:

ERROR MESSAGES: "closeInputDataset: Invalid input stream id"

MOD_IO_closeOutputDataset

SYNOPSIS: err = short MOD_IO_closeOutputDataset(short <outputFileId>)

DESCRIPTION: This function writes the trailer record to the output dataset file and deallocates (frees) all memory (malloc'ed) associated with the file designator <outputFileId>.

INPUTS: short <outputFileId> : The file designator provided to the user by the MOD_IO_openOutputDataset function.

OUTPUTS: (n/a)

RETURN: err : The return status value; 0 for normal operation.

EXAMPLE:

```
#include "MOD_IO_prototypes.h"
#include "AVHRR.h"

fileOut = MOD_IO_openOutputDataset( ...
if( MOD_IO_closeOutputDataset ( fileOut ) != 0 ) printf( "Not being able to close
files is bad news\n" );
```

COMMENTS:

ERROR MESSAGES: "closeOutputDataset: Invalid output stream id"
"can't write EOF"

MOD_IO_openInputDataset

SYNOPSIS: infd = short MOD_IO_openInputDataset(char * <FileName>,
DataDescriptor * <datasetType>)

DESCRIPTION: This function opens a dataset in read only mode. The file name is supplied by the user in the character string <FileName>. The dataset header is read from the disk file and compared with the contents of the data descriptor passed to this routine.

INPUTS: char * <FileName> : The user specified fully qualified input dataset file name.

DataDescriptor * <datasetType> : The address of the formal name of the data descriptor as defined in the associated data product header include file.

OUTPUTS: (n/a)

RETURN: infd : The positive data control block (dcb) number that the library uses in subsequent functions to designate which input stream is being accessed.

EXAMPLE:

```
#include "MOD_IO_prototypes.h"
#include "AVHRR.h"
#define FILENAME "/user/data/modis/Level.1B/mydata.test.data.sim"

inputFileDcb = MOD_IO_openInputDataset( FILENAME, &AVHRRheader );
```

COMMENTS: The formal name "AVHRRheader" must exist in the data product header include file "AVHRR.h". This applies to all data product header include files. The user must not alter the returned value: "inputFileDcb" in the above example. An error condition will be reported if a master input dataset is not already open. The variable "infd" can be an array element, e.g., "infd[2]".

(Error messages on next page)

ERROR MESSAGES:

"Sorry, you must specify a master scancube file first"
"Sorry, you have already used up all input DCBs; max = <number>" "can't recover"
"Sorry, unable to open input file <file name>" "can't recover"
"Can't read scancube volume header"
"Sorry, wrong instrument!"
"Sorry, wrong domain!"
"Sorry, wrong datatype!"
"Sorry, no Version=<number>!"
"compiled version: <number>, dataset version <number>" "Incompatibility between library version and dataset"
"Sorry, no nbands=<number>!"
"compiled # bands != actual datastream header # bands"
"Sorry, no nlines=<number>!"
"compiled # lines != actual datastream header # lines"

MOD_IO_openMasterInputDataset

SYNOPSIS: infd = short MOD_IO_openMasterInputDataset(char * <FileName>, DataDescriptor * <datasetType>)

DESCRIPTION: This function opens a master dataset in read only mode. The file name is supplied by the user in the character string <FileName>. The dataset header is read from the disk file and compared with the contents of the data descriptor passed to this routine.

INPUTS: char * <FileName> : The user specified fully qualified input dataset file name.

DataDescriptor * <datasetType> : The address of the formal name of the data descriptor as defined in the associated data product header include file.

OUTPUTS: (n/a)

RETURN: infd : The positive dcb number that the library uses in subsequent functions to designate which input stream is being used.

EXAMPLE:

```
#include "MOD_IO_prototypes.h"
#include "AVHRR.h"

inputFileDcb = MOD_IO_openMasterInputDataset( FILENAME, &AVHRRheader );
```

COMMENTS: This routine calls MOD_IO_openInputDataset internally to do most of the work.

ERROR MESSAGES: See MOD_IO_openInputDataset reference section, plus:
"Sorry, you cannot open more than one Master input file" "ndcb_in = <number>"
"Can't recover"

MOD_IO_openOutputDataset

SYNOPSIS: outfd = short MOD_IO_openOutputDataset(char * <character string>, DataDescriptor * <datasetType>)

DESCRIPTION: This function opens a dataset in write mode. The file name is supplied by the user in the character string <FileName>. The type of the data product is specified by the user as a pointer to the data descriptor "<datasetType>"

INPUTS: char * <FileName> : The user specified fully qualified output dataset file name.

DataDescriptor * <datasetType> : The address of the formal name of the data descriptor as defined in the associated data product header include file.

OUTPUTS: (n/a)

RETURN: outfd : The positive dcb number that the library uses in subsequent functions to designate which output dataset is being used; -1 for no more library internal dcbs available.

EXAMPLE:

```
#include "MOD_IO_prototypes.h"
#include "AVHRR.h"

outputFileDcb = MOD_IO_openOutputDataset( FILENAME, &AVHRRheader );
```

COMMENTS: The formal name "AVHRRheader" must exist in the data product header include file "AVHRR.h". This applies to all data product header include files. The user must not alter the returned value: "outputFileDcb" in the above example. The variable "outfd" can be an array element e.g., "outfd[1]".

ERROR MESSAGES

"Sorry, you have already used up all DCBs; max = <number>"
"Sorry, you must specify a master scancube file first"
"Unable to open output file <file name>" "can't recover"
"can't support writing multiple scancube types within a scancube now!"
"Can't write output header"

MOD_IO_printDataDescriptor

SYNOPSIS: void MOD_IO_printDataDescriptor(DataDescriptor * <datasetType>)

DESCRIPTION: The function prints the contents of the user specified data descriptor "<datasetType>" to stderr. It is useful for informational purposes when debugging user written algorithms and data product header include files.

INPUTS: DataDescriptor * <datasetType> : The address of the formal name of the data descriptor as defined in the associated data product header include file.

OUTPUTS: (n/a)

RETURN: (n/a)

EXAMPLE:

```
#include "MOD_IO_prototypes.h"
#include "AVHRR.h"

fdin = MOD_IO_openMasterInputDataset( "myfile", &AVHRRheader );
MOD_IO_printDataDescriptor( &AVHRRheader );
```

COMMENTS: The data descriptor contents are printed to stderr in labeled and tabulated form.

INFORMATION MESSAGES:

```
"Read Data Descriptor:"
 name = <descriptor name>"
 # data types/scancube = <number>"
 # bands = <number>"  (bands are equivalent to parameters)
 # lines = <number>"
" resolution for band <number> = <nuber>"  (multiple occurances)
" size of band <number> = <number> bytes"  (multiple occurances)
```

ERROR MESSAGES: (none)

MOD_IO_readSwath

SYNOPSIS: index = long MOD_IO_readSwath(short <inputDcb>, void * <inputBuffer>)

DESCRIPTION: This function reads a scan cube from the user specified input dataset.

INPUTS: short <inputDcb>: The input file designator returned by the MOD_IO_openMasterInputDataset or MOD_IO_openInputDataset routines.

OUTPUTS: void * <inputBuffer>: The address of the user defined scan cube data structure, into which the library places the updated pointers to the new data.

RETURN: index->if positive: a sequential index number of the scan cube that has just been read, starting from one (1).
index->if negative: an error return status value: -1 for an end of file (EOF)

EXAMPLE:

```
#include "MOD_IO_prototypes.h"
#include "AVHRR.h"
AVHRRscancube ibuf;

inputDcb = MOD_IO_openMasterInput( ...
index = MOD_IO_readSwath( inputDcb, &ibuf );
```

COMMENTS: This call brings in the next scan cube from the specified dataset contained on disk into library managed and allocated memory. Pointers to the scan cube elements are set in the user's data structure to point to the data in the library data area. The user may access any band (or parameter) and pixel through the scan cube data structure definitions. The data structure also provides access to the spatial sizes of all bands (or parameters). See code examples elsewhere in this document.

(Error messages on next page)

ERROR MESSAGES:

"No input stream to read!"
"readSwath: Invalid input stream id"
"Can't read scancube header"
"Sorry, cannot overwrite scancube with index=<number>"
"Warning: scancube header doesn't have an id"
"scancube header doesn't have # frames"
"Bad science data read; can't recover"
"Bad SD cal data read; can't recover"
"Warning: scancube header doesn't have SD mode"
"SD present, but incompatible .h file"
"Bad SRCA cal data read; can't recover"
"Warning: scancube header doesn't have SRCA mode"
"SRCA present, but incompatible .h file"
"Bad BB cal data read; can't recover"
"BB present, but incompatible .h file"
"Bad SV cal data read; can't recover"
"SV present, but incompatible .h file"
"Bad EM cal data read; can't recover"
"EM present, but incompatible .h file"
"Sorry, can't allocate space for bufindex <number>, band <number>"
"Sorry, incomplete read; wanted <number of bytes>, got <number of bytes>"
"Sorry, can't allocate space bytes data, bufindex <number>"

MOD_IO_writeSwath

SYNOPSIS: err = short MOD_IO_writeSwath(short <outputFileId>)

DESCRIPTION: This function writes the contents of the current output buffer to the dataset specified by the user supplied outputFileId, where outputFileId was obtained from the MOD_IO_openOutputDataset call.

INPUTS: short <outputFileId>: The value returned by the MOD_IO_openOutputDataset call.

OUTPUTS: (n/a)

RETURN: err : The return status value: 0 if every thing is OK, -1 if there is no output file to write to.

EXAMPLE:

```
#include "MOD_IO_prototypes.h"
#include "AVHRR.h"

OutputFileId = MOD_IO_openOutputDataset( .....
err = MOD_IO_allocateOutputBuffer( ....
   (place values in the output buffer)
err = MOD_IO_writeSwath( OutputFileId )
```

COMMENTS:

ERROR MESSAGES:

"writeSwath: Invalid output stream id"
"Sorry, no output buffer has been allocated for product #<number> (nothing to write!)" "can't recover"
"Sorry, unable to write <number> bytes; only wrote <number> bytes for output data stream #<number>" "can't recover"
"ERROR: cannot match next output id with master input buffers"
"Output id =<number>, input list = <several numbers>" "can't recover"
"arglist write too long" *(internal error)*
"Can't write output scancube header"

LIBRARY ACCESS AND INSTALLATION

The library source code, make files, and test programs are contained a UNIX **tar** file. This may be obtained from the /pub directory on modis-xl.gsfc.nasa.gov via anonymous ftp and contains a README file with the latest details and dataset contents descriptions. It also contains an INSTALL file with full installation instructions. The following list shows a sample of the files contained in the **tar** distribution:

Table 2. List of Files

FILENAME	*DESCRIPTION*
README	a file containing the latest information with a description of all the other files in the distribution
INSTALL	installation instructions for adding this library to your suite of computer based tools
MOD_IO_lib.c	the MODIS SDST utility library (all library functions)
MOD_IO_prototypes.h	ANSI function prototypes for library functions
MOD_IO_lib.h	data structures for scan cube buffer descriptions and DCBs; NOT explicitly needed by application programs
MOD_IO_headerDefs.h	defines for constants in the library specific to the dataset headers (also used by non MODIS scan cube generators)
DataDescriptor.h	data structure definition for formal Data Products
AVHRR.h	a sample AVHRR data product header file
(additional files)	sample algorithms and test cases (see the README file for details)

APPENDICES

Appendix A - MODIS Spatial Domain Descriptions

The MODIS Scan Cube Spatial Domain

The MODIS instrument generates raw (digital counts) data for each detector within the instrument. This set of detectors scans across (perpendicular to) the satellite orbit ground track to produce an amount of data designated as a scan cube (see Figure 7). Multiple scan cubes are collected together to form a data granule. Multiple granules are collected into an orbit. The sizes of this basic scan cube are summarized below. Scan cube details are found in the MODIS Data Structure, Rates, and Volumes document as referenced in the Bibliography.

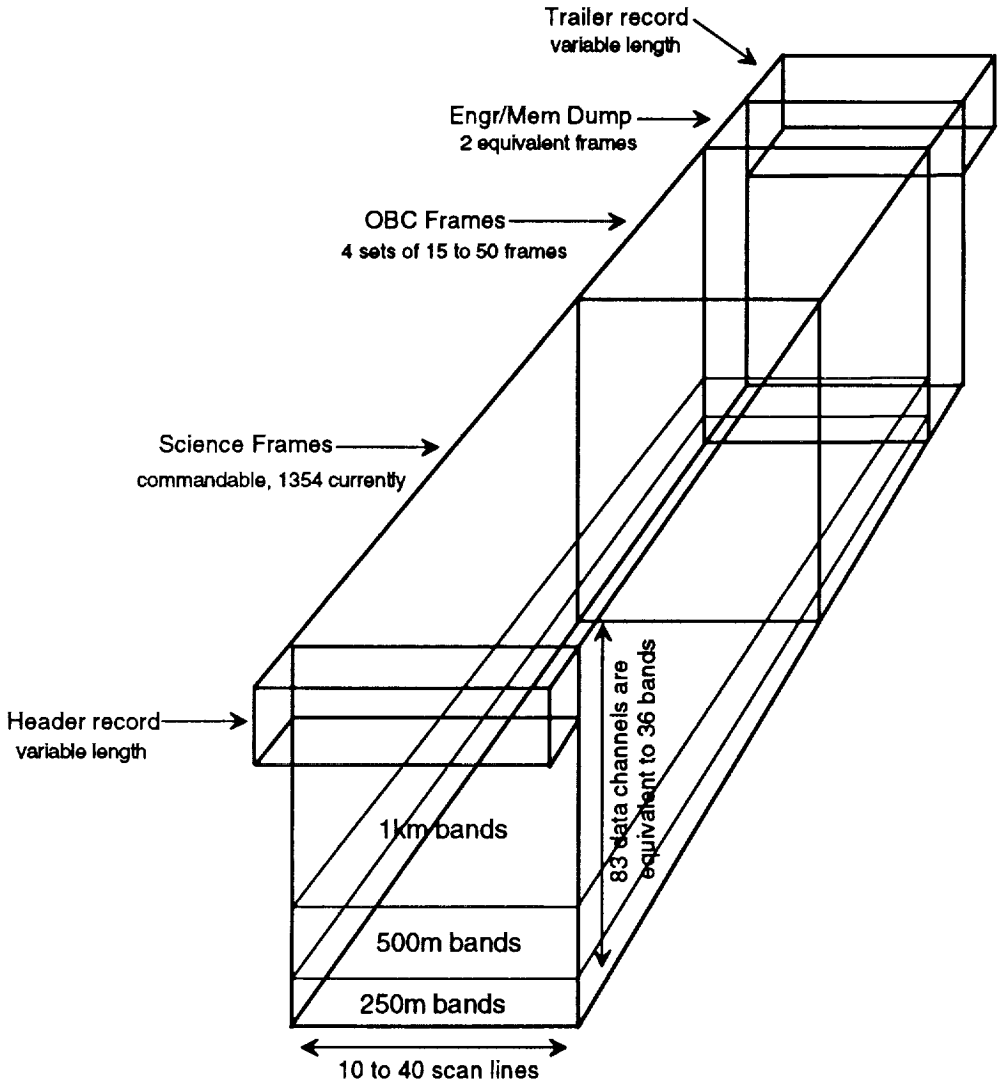

Figure 7. MODIS Level-1 simplified scan cube.

The MODIS instrument has the capability of setting the across track size of the scan cube by ground command. The current information indicates that this is not expected to be varied often, but plans are being considered to lower the data transmission rate possibly by varying the across track size as a function of latitude. In any case, the algorithm developer should be aware that the number of across track pixels is not a constant. For the scan cube domain, the algorithm developer deals with a single band of MODIS data covering an entire scan cube swath. The sizes given are in pixels, not kilometers. The pixel ground coverage sizes vary with the scan angle from nadir. The collection of detector values from all the bands that cover a single nominal (at nadir) one (1) kilometer area are called a spatial element. All bands of data are specified to be co-registered within 20 percent of the spatial element IFOV and are expected to be co-registered within 10 percent of the IFOV. Note that this specification applied to the registration knowledge. A table of band to band offsets will be published by the MODIS SDST and may be included as part of the MODIS geolocation.

Table 3. MODIS Level-1B Data Product Scan Cube Sizes

ITEM	*HOW MANY*	*FIXED OR VARIABLE*
number of bands	36	fixed
across track pixels; bands 1 and 2	5,416	variable
across track pixels; bands 3 to 7	2,708	variable
across track pixels; bands 8 to 36	1,354	variable
along track pixels; bands 1 and 2	40	fixed
along track pixels; bands 3 to 7	20	fixed
along track pixels; bands 8 to 36	10	fixed
geolocation data	8	fixed

The MODIS Level-1A Data Product consists of the raw counts with a separate Geolocation Data Product associated with each individual spatial element. The raw counts are available for the Earth view as well as the various onboard calibrator (OBC) views. This raw data product is expected to be of use only to the Level-1B calibration process. (The OBC access functions are described in an appendix.) The science algorithms are expected to use the at-satellite radiances contained within the MODIS Level-1B Data Product. This data product contains the radiance energy generated by viewing the Earth and measured at each detector within the MODIS scanning swath. These values are spatially coincident with the raw counts and have not been spatially resampled. The geolocation information associated with the MODIS Level-1A product therefore applies to, and will be associated with the Level-1B Data Product. The Level-1A dataset components will be synchronized with the Level-1A Metadata and the associated scan cube domain Geolocation Data Product. The Level-1B dataset components will similarly be transparently aligned with the Level-1B Metadata and this same scan cube domain Geolocation Data Product. All of the data sets are accessible from the identical library calls, via separate data structures.

The MODIS Level-1A Data Product also contains information derived from the onboard calibrators (OBCs) in two forms. The first data form is also in the basic structure of a scan cube, but is measured by imaging the OBC radiance sources in place of the Earth based radiances. There will also be occasional Moon viewing data in either or both of the Earth views and OBC space views. The second form of OBC data is derived from the engineering parameter measurements that are multiplexed within the packet data. These are separated into a separate calibration data product for use by the Level-1B process. The library calls to access these portions of the MODIS data are not included in this document.

The MODIS Level-2 Data Product data sets can also be accessed and generated with library calls included in this SDST utility library. This allows previously generated MODIS Level-2 products to be used as inputs for the creation of other Level-2 Data Products.

Data Products in this scan cube domain will not be temporally or spatially coincident with any other scan cubes or domains and must be resampled or otherwise aggregated into a rectified or mapped domain for true spatial registration. All true spatial operations in the scan cube domain are left to the user.

The MODIS Rectilinear Domain

Algorithms that operate on a single spatial pixel area at a time, and do not depend on the exact direction and distance to adjacent spatial pixels, can generate science values in the scan cube geometry domain. These values can then be mapped directly from the scan cube domain onto a Level-3 map projection or similar geobased database. However, algorithms that depend on spatial directions and distances can easily use either a rectilinear MODIS data domain or a mapped data domain as input when deriving science Data Products.

The rectilinear domain is defined to be a coordinate system that uses the spacecraft ground track as one axis, the scan perpendicular to the ground track as the second axis, and radiance data values resampled to a uniform distance along each of these axes..

Only the MODIS Level-1B radiance values and MODIS Level-2 Data Products are appropriate for the rectilinear domain. The MODIS Level-1A data is best represented in the scan cube domain and is not appropriate to the rectilinear domain. Note that MODIS data that uses or passes through a rectilinear domain will be resampled twice: once from scan cube to rectilinear, and again going from rectilinear to mapped. For this reason, it is best to keep algorithms in the scan cube or mapped domain.

The MODIS Mapped Domain

This designation is used for all Data Products that have been transformed, via a map projection, onto a flat surface such as a piece of paper or display pixel indices (the electronic equivalent). A map projection can be defined as the orderly transfer of Earth surface positions to corresponding points on a flat surface. This definition can be further generalized into a three dimensional hierarchical data structure such as a quad tree, where each area on the flat surface is successively divided into smaller areas. Since the surface of a sphere

cannot be forced directly onto a flat plane without simplifying approximations, several types of maps are expected to be used to represent either the entire Earth global data sets or sections of the globe such as continents.

Mapped domain library calls will be defined in a separate document on MODIS Level-3 toolkit functions.

Appendix B - Data Product header include file description

A series of data product header include files that describe the data structures for: the input and output formal Data Products, static arrays of characters describing the dataset contents, ANSI function prototypes, and global variables are included as part of the library distribution. These are for use with "C" programs only. FORTRAN equivalents are not available at the present time. Note that FORTRAN90 will enable the use of these types of headers while ANSI FORTRAN77 will not. Each formal Data Product has an accompanying header file that contains the product specific information which contains dynamic variables and is accessible to the science algorithm developers. The set of all of these Data Product header files will be created in cooperation with the algorithm writers, maintained by the MODIS SDST ATT members, and made available for use by all algorithm developers. These headers will be coordinated with the science data product office (SDPO) definitions for the formal Data Products and parameters and placed under configuration management by the SDST.

Each Data Product header include file contains the data structure definitions for each of the parameters that are a part of every formal Data Product. Data product parameters are the components of each formal Data Product such as the data product itself, values for the precision of the data product, all quality assurance (QA) parameters, and intermediate informal data products associated with the formal Data Product. The data types (short, long, float, etc.) for each parameter are defined here along with the sizes, resolutions, and dimensions of the spatial extent of each parameter. Two "C" code modules are also included; one to initialize the data structure (similar to a constructor in C++) and one to set the pointers appropriately for user access. These are "hidden" in this file so that the user need only alter the algorithm source code to include the appropriate file without the concern of linking separate code modules. There could be situations where it is desirable to put these code modules into a separate file. If so, these situations can be easily dealt with on an individual basis.

Note that the individual bands of MODIS instrument data in the Level-1A and Level-1B Data Products are equivalent (for library access) to data product parameters and are accessed by the user via the same library call mechanism.

The include files required for algorithm development are: `"MOD_IO_prototypes.h"`, `"<InputProduct.h>"`, and `"<OutputProduct.h>"`. ANSI prototypes for all library calls are included in the `"MOD_IO_prototypes.h"` file. The Data Product detailed descriptions are included in a header file that is unique to each instance of a Data Product, either input or output (`"<InputProduct.h>"` and/or `"<OutputProduct.h>"`). A fully commented sample data product header file for AVHRR data in a MODIS scan cube format is included in Appendix D. The `"DataDescriptor.h"` include file, referenced by the `<*.h>` files, defines a flexible data format which is then used by the product specific `<*.h>` files to further describe each particular formal Data Product. Each product specific header file also includes two code modules which are invoked by the library. The first module, an initialization section, is called when each input dataset is first opened. This module customizes the DataDescriptor data structure, which is then compared by the library with information in the actual dataset volume header. The second module is called each time a scan cube is read into memory, or an output scan cube is allocated. This code defines and passes the correct data pointers to the user's program for accessing the input or output scan cubes that are actually in the library memory space.

FORTRAN (FTN77) accessible function calls may be written at a later date if there is a user demand for FTN77 library functions. If written, these are expected to be "wrappers" to the "C" function calls that allow FTN77 access to data structure contents. They will allow algorithms written according to FTN77 standards to have access to the MODIS data, but not efficiently. Exceptions to the FTN77 ANSI standard (such as include files) will have to be taken to accomplish this task. Note that algorithms written in FORTRAN 90 will allow direct access to data structures via pointer mechanisms that are similar to the "C" language. This is the SDST recommended approach after approval for algorithm development in FORTRAN.

Appendix C - The DataDescriptors.h Library Include File Description

This is the include file used by the library and referenced in the formal Data Product header include files. It declares the data structure for the data description area which contains pointers to data product unique substrucure definitions.

```
/*                                                                    */
/*                        MODIS scancube i/o library                  */
/*                        SDST                                        */
/*                        Virginia Kalb, NASA/GSFC 920.2              */
/*                        Thomas Goff, RDC                            */
/*                                                                    */
/* RCS keywords:                                                      */
/* $Header: DataDescriptor.h,v 2.1 94/08/16 09:57:49 gk Exp $         */
/* $Date: 94/08/05 09:57:49 $                                         */
/* $Source: /disk5/modis/sim/MOD_IO/RCS/DataDescriptor.h,v $          */
/*                                                                    */
/*                                                                    */
#ifndef DATADESCRIPTOR
#define DATADESCRIPTOR
struct FPTR {
  void (*fptr)();
```

This declares a function pointer data type to facilitate the subsequent declaration of an array of function pointers needed for the Level-1B data. The MODIS Level-1B subcubes contain the Earth (science) view and the four sets of OBC data (SD, SRCA, BB, Space view).

```
};
typedef struct FPTR FPTR;

struct DataDescriptor {
  void (*init)();      <- Entry point to the initialization routine.
  short nptrs;         <- How many pointers to subcubes are required.
  FPTR *set;           <- Pointer to the array of subcubes.
  char *name;           <- The dataset formal name, contained within the dataset.
  long nbands;          <- The number of parameters (bands) in this formal Data Product.
  long nlines;
```

The number of lines (detectors) in the along track direction for the finest spatial resolution for this formal Data Product. For example, this corresponds to the number of along track detectors for MODIS bands 1 and 2 which is 40.

```
  long *nres;
```

For each parameter (band), this is the resolution divisor to obtain the actual number of along track detectors for that parameter (band). For example, MODIS bands 3 to 7 have an nres equal to 2 (20 detectors in the along track direction), and bands 8 to 36 have an nres value of 4 (10 detectors in the along track direction).

```
  long *nsize;
```

The size in bytes of each parameter value. For example, MODIS band 1 has 2 bytes per readout, 40 detectors along track, and 4 times the nominal 1354 samples along scan for a final value of 433280 bytes.

```
};
typedef struct DataDescriptor DataDescriptor;
#endif
```

Appendix D - Data Product Header example

This section contains an example of a generic Data Product header file with embedded comments. A machine readable copy of this file is available as part of the library distribution. The source code is in the Courier typeface and comments following each section of code are in the normal Times Roman typeface.

The formal Data Product header include file contents are divided into five (5) main sections. The first section defines header includes and internal definitions. The next two sections set up the data structures for each element of a parameter within a data product, and the order of the parameters within a scan cube. The remaining two sections are functions that populate (provide instance specific contents for) the dataset valuables and each scan cube variable respectively.

```
/*                                                                    */
/*                        MODIS scancube i/o library                  */
/*                        SDST                                        */
/*                        Virginia Kalb, NASA/GSFC 920.2              */
/*                        Thomas Goff, RDC                            */
/*                                                                    */
/* RCS keywords:                                                      */
/* $Header: PRODUCT.h,v 2.0 94/08/05 17:01:26 gk Exp $                */
/* $Date: 94/08/05 17:01:26 $                                         */
/* $Source: /disk5/modis/sim/MOD_IO/RCS/PRODUCT.h,v $                 */
/*                                                                    */
```

The RCS revision control system is used internally for configuration management during the code development stage.

```
#ifndef PRODUCT
#define PRODUCT
```

The above cpp preprocessor directives prevent duplicate compiling of this module from occurring.

```
#include <stdio.h>
#include <stdlib.h>
```

These are the standard UNIX header includes from the standard operating system location for system include files. They are used for both ANSI prototyping and function macros.

```
#include "DataDescriptor.h"
```

This header file contains the MODIS library internal structure that holds values that are fixed for a given product. It contains the data structure specifications for the PRODUCTheader.<components> data structure.

```
#define PRODUCT_HEADER_STRING "Product-specific identifer"
```

This define statement declares the formal name of each user's Data Product. No spaces are allowed. This character string will contain something like: "Level-1B_radiances" for the MODIS Level-1B Data Product. The spatial domain designation is contained within the datasets. This allows differing spatial domains containing the same data types to be used interchangeably by the library and hence the user.

```
#define DUMMY 1
```

This is a dummy number (could be any number actually) used to satisfy the compiler when declaring the variable dimension component in a multi-dimensional array. The data structures that allow for single or two dimensional access to the data are illustrated later in this listing.

```
typedef double PRODUCTparm1;
typedef long PRODUCTparm2;
typedef unsigned char PRODUCTparm3;
```

These type definitions specify the data type of the formal Data Product parameters. In this case, the Data Product has three Parameters (these are bands for Level-1A and Level-1B). The first parameter is of type double (64-bit floating point / real), the second is of type long (32-bit integer), and the last parameter is of type single byte (8-bit). The data type is user-driven.

```
/* Structure for individual bands/parameters:                    */
struct PRODUCTPARM1 {
  PRODUCTparm1 *data;
  long nAlongTrack;
  long nAlongScan;
  long nPixels;
};
typedef struct PRODUCTPARM1 PRODUCTPARM1;
```

This data structure declaration defines the first product parameter (the data part of which has been specified above) to be a data structure containing a pointer to the spatial data array for this parameter, an along track and across track dimension, and a total number of pixels. Note that the data dimensions are adjusted by the library to reflect the resolution of each individual parameter.

```
struct PRODUCTPARM2 {
  PRODUCTparm2 *data;
  long nAlongTrack;
  long nAlongScan;
  long nPixels;
};
typedef struct PRODUCTPARM2 PRODUCTPARM2;
```

(Same as the above description.)

```
struct PRODUCTPARM3 {
  PRODUCTparm3 *data;
  long nAlongTrack;
  long nAlongScan;
  long nPixels;
};
typedef struct PRODUCTPARM3 PRODUCTPARM3;
```

(Same as the above description.)

```
/* Structure for scancube output:                                */
struct PRODUCTscancube {
  PRODUCTPARM1 p1;
  PRODUCTPARM2 p2;
  PRODUCTPARM3 p3;
```

```
};
typedef struct PRODUCTscancube PRODUCTscancube;
/* User should declare an output structure like this:                    */
/*  PRODUCTscancube obuf;   */
```

This statement declares the data structure type for user access as a single linear array. The examples given in the main section of this document in the scan cube domain functions - introduction section, illustrate its use from a user's perspective.

```
/* Structure for 2d scancube output:                                     */
/* CAREFUL! The rightmost array dimension MUST MATCH THE                 */
/* ACTUAL # OF SPATIAL ELEMENTS for that band!                           */
/*                                                                       */

struct PRODUCT2Dparm1 {
  PRODUCTparm1 xy[DUMMY][10];
};
typedef struct PRODUCT2Dparm1 PRODUCT2Dparm1;
```

More type definitions: in this case to allow two dimensional access to the spatial data. The array name xy is used as a pointer to this two dimensional array. Note that the requirement that the varying dimension, corresponding to the MODIS variable number of frames across scan cubes, is in the first set of brackets [] with any number (DUMMY) used as a place holder. The number in the second brackets [] must correspond to the actual along track (across scan) size for each parameter. The order of the indices in "C" is reversed from those in FORTRAN.

```
struct PRODUCT2Dparm2 {
  PRODUCTparm2 xy[DUMMY][10];
};
typedef struct PRODUCT2Dparm2 PRODUCT2Dparm2;
```

(Same as the other parameters)

```
struct PRODUCT2Dparm3 {
  PRODUCTparm3 xy[DUMMY][10];
};
typedef struct PRODUCT2Dparm3 PRODUCT2Dparm3;
```

(Same as the other parameters)

```
/* Structure for individual bands/parameters (2D):                       */
struct PRODUCTPARM1_2D {
  PRODUCT2Dparm1 *data;
  long nAlongTrack;
  long nAlongScan;
  long nPixels;
};

typedef struct PRODUCTPARM1_2D PRODUCTPARM1_2D;
```

These two dimensional access methods are effectively the same as the previous typedef's for PRODUCTPARM1: `*data` is a pointer to the actual scan cube binary data.

```
struct PRODUCTPARM2_2D {
```

```
  PRODUCT2Dparm2 *data;
  long nAlongTrack;
  long nAlongScan;
  long nPixels;
};
typedef struct PRODUCTPARM2_2D PRODUCTPARM2_2D;
```

(Same as PRODUCTPARM1_2D)

```
struct PRODUCTPARM3_2D {
  PRODUCT2Dparm3 *data;
  long nAlongTrack;
  long nAlongScan;
  long nPixels;
};
typedef struct PRODUCTPARM3_2D PRODUCTPARM3_2D;
```

(Same as PRODUCTPARM1_2D)

```
struct PRODUCT2Dscancube {
  PRODUCTPARM1_2D p1;
  PRODUCTPARM2_2D p2;
  PRODUCTPARM3_2D p3;
};
typedef struct PRODUCT2Dscancube PRODUCT2Dscancube;
/* User should declare an output structure like this:                    */
/*       PRODUCT2Dscancube obuf;                                         */
```

This statement declares the data structure type for user access as a two dimensional array. The examples given in the main section of this document in the scan cube domains functions - detailed section, illustrate its use from a user's perspective. Note that ideally, one scan cube definition would be given in each header file. Both the one and two dimensional definitions are given in this "PRODUCT.h" sample header file for illustration purposes. Other definitions could be made at the user's request, but care must be exercised to propagate any changes to the intrinsic function code and other modules within the header file.

Now that the data structures and declarations are over with, we can populate instances of these complex variables with the values appropriate to this formal Data Product by invoking actual executable code.

```
void PRODUCTinit();
void PRODUCTsetptrs(void **, PRODUCTscancube *, long);
Static DataDescriptor PRODUCTheader =
                      {PRODUCTinit,PRODUCTsetptrs,NULL,0,0,NULL,NULL};
```

Lets start with the ANSI prototypes for the functions that follow, and initialize (declare space for) the data descriptor structure. This is executed for each dataset open.

```
void PRODUCTinit()
{
static short executed = 0;

if(executed > 0) return; /* don't want or need to execute this more than
once!*/
executed = 1;
```

The variable executed is used to insure that this function is executed only once.

```
PRODUCTheader.name = PRODUCT_HEADER_STRING;
PRODUCTheader.nbands = 3;
PRODUCTheader.nlines = 40;
PRODUCTheader.nres = (long *)malloc(PRODUCTheader.nbands*sizeof(long));
PRODUCTheader.nres[0] = 4;
PRODUCTheader.nres[1] = 4;
PRODUCTheader.nres[2] = 4;
PRODUCTheader.nsize = (long *)malloc(PRODUCTheader.nbands*sizeof(long));
PRODUCTheader.nsize[0] = sizeof(PRODUCTparm1);
PRODUCTheader.nsize[1] = sizeof(PRODUCTparm2);
PRODUCTheader.nsize[2] = sizeof(PRODUCTparm3);
}
```

This function populates the PRODUCTheader.<components> structure with the values appropriate to this formal Data Product. These values apply to the entire dataset. String sizes are determined at compile time, and nres and nsize arrays are allocated at run time. See the "DataDescriptors.h" section in the appendix to this document for the structure declaration which defines the data types for each structure element name.

```
void PRODUCTsetptrs(void **obuf, PRODUCTscancube *odata, long nframes)
```

This function is entered on each allocation of a scan cube. It users the library supplied pointer `odata`, and library supplied pointer `*obuf` to the user declared buffer `obuf`, along with the library supplied `nframes` to calculate the values of the scan cube dimensions `nAlongTrack` and `nAlongScan` and the scan cube size `nPixels`. The value of `nframes` is extracted by the library from each scan cube header. These values, along with the starting addresses (pointers) of the data array, are then placed into the user's instance of the formal Data Product data structure. This is repeated for each parameter separately, thereby allowing the spatial dimensions to vary across bands (parameters).

```
{
  long across, down;

  odata->p1.data = obuf[0];
  down = PRODUCTheader.nlines/PRODUCTheader.nres[0];
  odata->p1.nAlongTrack = down;
  across = nframes/PRODUCTheader.nres[0];
  odata->p1.nAlongScan = across;
  odata->p1.nPixels = down*across;

  odata->p2.data = obuf[1];
  down = PRODUCTheader.nlines/PRODUCTheader.nres[1];
```

```
  odata->p2.nAlongTrack = down;
  across = nframes/PRODUCTheader.nres[1];
  odata->p2.nAlongScan = across;
  odata->p2.nPixels = down*across;

  odata->p3.data = obuf[2];
  down = PRODUCTheader.nlines/PRODUCTheader.nres[2];
  odata->p3.nAlongTrack = down;
  across = nframes/PRODUCTheader.nres[2];
  odata->p3.nAlongScan = across;
  odata->p3.nPixels = down*across;

  return;
}
#endif
```

End of this data product header include file.

Appendix E - MODIS Dataset Internal Format

This section details the contents of the dataset that are ingested and created by the SDST utility library. The dataset contents implement a free field technique to determine the sizes and location within the Dataset for the various components of the Dataset. This allows data from multiple instrument sources and multiple input and output Data Products to be handled by the library in a manner that is transparent to the user.

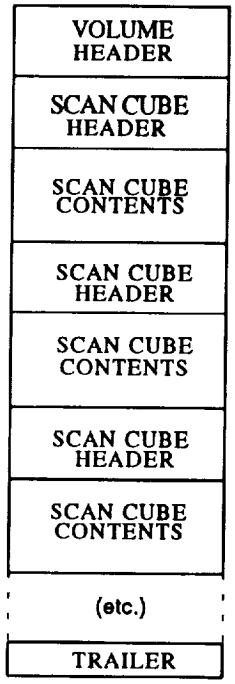

Figure 8. Overall dataset contents.

Each Dataset is composed of three main sections: a Dataset volume header, multiple instances of the scan cube header and data, and a trailer as shown in Figure 8. All headers sections and the trailer section contain information in ASCII form that can be read (carefully) with any file browser or editor. Recommended file browsers are the UNIX **od** command, the gnu **less** program, and the SDST written **fd** program. These programs can examine mixed ASCII and binary files while not creating problems with communications protocols. The **od** and **fd** programs can handle byte offsets for binary data that is not aligned on word boundaries. The UNIX **head**, **tail**, and **more** programs will cause problems with terminal based communications when examining binary data.

The Dataset is organized as a self describing data structure. Information is contained in each section that informs the library of the sizes of the following sections. This technique allows a byte stream oriented operating system (i.e. UNIX) to define variable length and mixed type (ASCII / text and binary) records. Data is organized in this byte stream by placing the along track pixels in adjacent words of memory to form a vector, vectors are then sequenced across track to form a band or parameter spatial area, then multiple areas are sequenced to form the entire scan cube. This ordering in memory allows the across track

dimension to vary among scan cubes without prohibiting two dimensional access by "C" and FORTRAN compilers. The illustration in Figure 9 assumes a data product composed of three parameters, each of type byte (unsigned character or Integer*1) with nominal spatial resolutions of 250 meters, 500 meters, and 1.5 kilometer. This is a fictitious instrument for illustration purposes only. The reader is encouraged to examine the small datasets included in the library distribution for actual parameter values and compare these with the sizes in the diagram and described in this text.

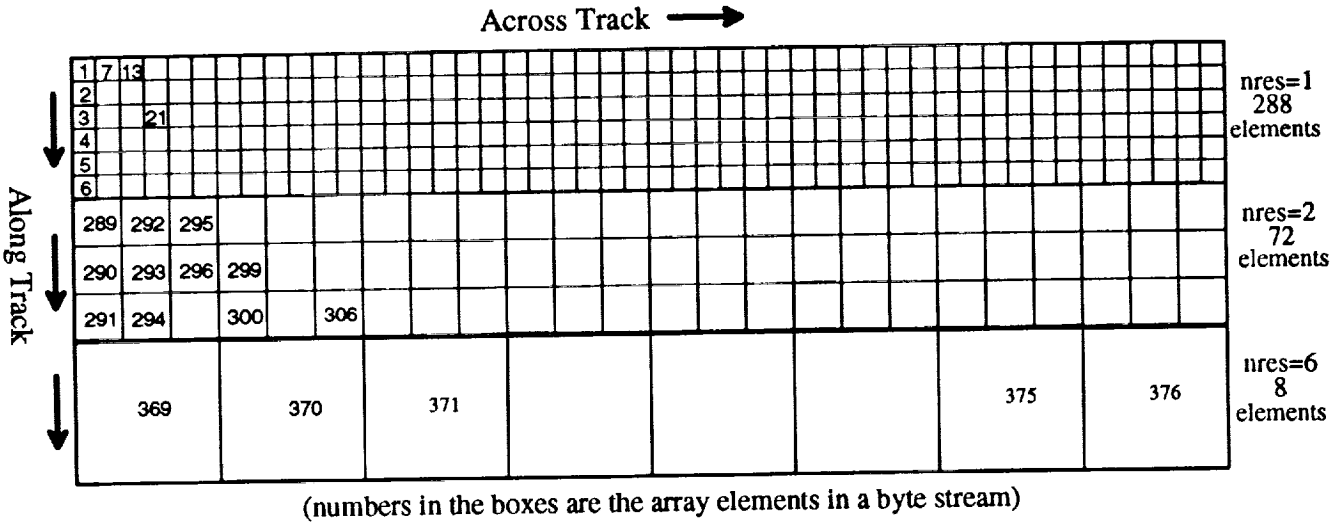

Figure 9. Memory allocation for variable coincident spatial sizes.

The numbers in this diagram represent the linear addresses of the byte stream. The three spatial areas with resolutions of 1, 2, and 6 can be spatially registered on the ground, but are sequential in computer memory. The illustration given here does not represent any of the current datasets and is used for illustration purposes only.

The Dataset volume header contains information that is required by the library to recognize and interpret the contents of the Dataset. This includes the source of the data (must be "MODIS"), the type of data (unique to each formal Data Product), and the version of the Dataset contents. This Dataset version must agree with, or otherwise be compatible with, the library. It is the library's responsibility to verify this. Other items in this header are compared with the DataDescriptors in the library as a sanity check to ensure that the library and user code are using the same data formats and types. The remainder of the volume header contains additional items unique to each Data Product and any comment fields included by the Dataset generator.

The scan cube section contains a scan cube header and data in a binary form. The scan cube header is also in the same free field format as the volume header. Dataset parameter values are reset as each scan cube is accessed. This allows the handling of binary data that can change with each scan cube.

The binary section contains the actual data values in the data type defined within each Data Product "<product.h>" include file. The data is treated by the library as a byte stream whose size is the across track size times the along track times the number of bytes in the data product, where the number of bytes in the data product is the sum of the number of bytes for each parameter.

The trailer contains the key character string "EOF" to designate a logical end of file. All other tokens in the trailer are ignored in this release, but may contain metadata information in the future.

All of the headers contain ASCII data in a free field format. Each field, called a token, consists of all the characters that are delimited by 'white space', where white space is defined to be blank (space) characters, tab characters, or commas. The first three tokens are recognized by the library as character strings that are compared with information in the "<product.h>" headers associated with each format Data Product. All tokens that contain an equal (=) sign are recognized as "C" type variables and values. These variable names are formal names used by the library to determine the sizes of Data Product parameters, spatial resolutions of the pixel elements, and the resulting sizes of the binary data that follow. All other tokens in the header are ignored by the library and can contain any user defined comments. Details are given in the following table.

Table 4. Header Parameter Names

FORMAL NAME	DESCRIPTION	UNIQUE TO:
Version	The version number of the library that can access this Dataset	dataset
headerLines	The number of lines contained in this header section (terminated by \n)	dataset
nbands	Number of bands or parameters in the data structure	dataset
nlines	Number of along track lines of data within each scan cube at resolution 1	dataset
nres	Multiple of the smallest instrument IFOV (for MODIS this is 250 meters * nres)	dataset
nsize	Number of bytes per data element	dataset
nframes	number of elements across track at resolution 1 for the Earth view	scan cube
scancubeId	unique ID within this data set for each	scan cube
nSDframes	number of elements for the Solar Diffuser view	scan cube
SDmode	current SD mode: "DCrestore | Solar | Screen"	scan cube
nSRCAframes	number of elements for the SpectroRadimetric Control Assembly	scan cube
SRCAmode	"radiometric | spectral | spatialAlongScan | spatialCrossScan"	scan cube
nBBframes	number of black body viewing frames	scan cube
BBtemp	the BB temperature (an array in the future)	asynchronous

nSVframes	number of space view elements	scan cube

DATASET contents example

Here is an abbreviated example output from the UNIX **less** program of a test scan cube dataset. Line numbers have been added. Excess binary data has been deleted for illustration purposes and the dataset contents are incomplete. The reader is encouraged to look into the contents of the datasets provided in the distribution for real life examples.

line # *contents description*

```
1 - MODIS Scancubes AVHRR_albedo Version=1.0 headerLines=3
2 - nbands=5, nlines=40, nres=4 nsize=2
3 - This is a comment line
4 - nframes=64 scancubeId=1
5 - ^@^N^@^O^@^O^@^O^@^S^@^T^@^V^@^V
6 - ^A^T^A^T^A^T^A^T^A^T^A^T^A^U^A^T^A^T
7 - nframes=64 scancubeId=2
8 - ^A[m#[5m^A[m#[5m^A[m&[5m^A[m&[5m^A[m%[5m^m^A[m&[5m^A[m%[5m^A[m%[5m^A[m
9 - ^A^U^A^T^A^T^A^T^A^T^A^U^A^U^A^U^A^U^A^U^A^UU^A^U^A^U^A^U^A^U^A^U[m
10 - nframes=64 scancubeId=3
11 - ^@^T^@^W^@^W^@^W^@^V^@^U^@^U^@^U^@ @^W^@^U^@^U^@^U^@^U^@^T[m
12 - ^A^S^A^U^A^T^A^T^A^S^A^S^A^S^A^S^A^S^A^S^A^^T^A^T^A^T^A^S^A^S^A^S^A^S[m
13 - nframes=64 scancubeId=4
14 - ^@^T^@^R^@^R^@^S^@^U^@^V^@^V^@^V^U^@^V^@^V^@^V^@^V^@^W^@^W[m
15 - ^A^S^A^S^A^S^A^S^A^S^A^S^A^S^A^S^A^T^A^T^A^S^A^S^A^S^A^S^A^T^A^T^A^T[m
16 - nframes=64 scancubeId=5
17 - ^@^U^@^U^@^T^@^T^@^T^@^U^@^U^@^T^@^T^@^T^@^U^@^U^@^V^@^V[m
18 - ^A^S^A^S^A^S^A^T^A^T^A^T^A^S^A^T^A^T^A^T^A^T^A^S^A^S^A^S^A^S^A^T[m
19 - EOF
```

Figure 10. Dataset contents example.

The volume header consists of the first three lines. The key token "MODIS" is required to authenticate the dataset as a MODIS format dataset. The token "Scancubes" indicates the spatial domain of the dataset which the library uses to determine which of the "<parameter>=<value>" tokens are required. The next token "AVHRR_albedo" indicates the formal Data Product name and must agree with the corresponding string in the "<product.h>" file for this Data Product. The "Version=1.0" token validates the library version with the dataset version. The "headerLines=3" token informs the library that three text lines, terminated by a new line character, define the volume header.

The fourth line contains the two required tokens for a scan cube domain dataset. The total number of pixels for a resolution 1 spatial area for the highest resolution parameter, is given in "nframes=64". This is divided by the number of lines per scan cube "nlines=40" at each spatial resolution to give the number of across track elements (frames in the MODIS literature). The "scancubeId=1" tells the library the unique scan cube identifier. This number is written by the library into the corresponding output data product scan cube.

Lines 5 and 6 in the above example are the ASCII equivalents, in printable form, of each byte of binary data. This has been shortened for illustration purposes. Multiple scan cubes are shown in lines 7 - 9, 10-12, 13-15, and 16-18. The binary data is not terminated by a new line as shown. The next scan cube header begins directly after the binary data and can be anywhere on a displayed line. It is shown left justified for illustration purposes.

The final line, 19 in this minimal example, contains the token "EOF". This token is optional but does indicate that the dataset has been terminated gracefully and not truncated abnormally.

Each header section can optionally contain the token "headerLines=<n>" . If it is not present, a value of one (1) is assumed. If, however, this token is missing and there is more than one header line in the dataset, then the data stream will be hopelessly out of sync since the ASCII data will be interpreted as part of the binary science data. The library will eventually catch this mistake in a subsequent header read and return an error code to the user, but not at the expected processing step.

Appendix F - Library Dataset Interactions

This appendix presents an overall view of the internal workings of the library. It is not a full library documentation, but is meant to provide a feel for what is happening behind the scenes. The figures in this appendix help the reader understand the relationship between the formal Data Product header include files and the library's use of information in both the data product header include files and the dataset contents.

The diagrams in Figure 11 and Figure 12 illustrate the relationship between the input and output data sets, the utility library, and the user code. Data can be transferred, under user control, by the library from the input scan cubes to the output scan cubes without passing through the memory space of the user code. The first figure also illustrates the use of multiple input and output scan cubes within the library space and managed by the library internal DCB structures that incorporate dynamic memory allocation and release.

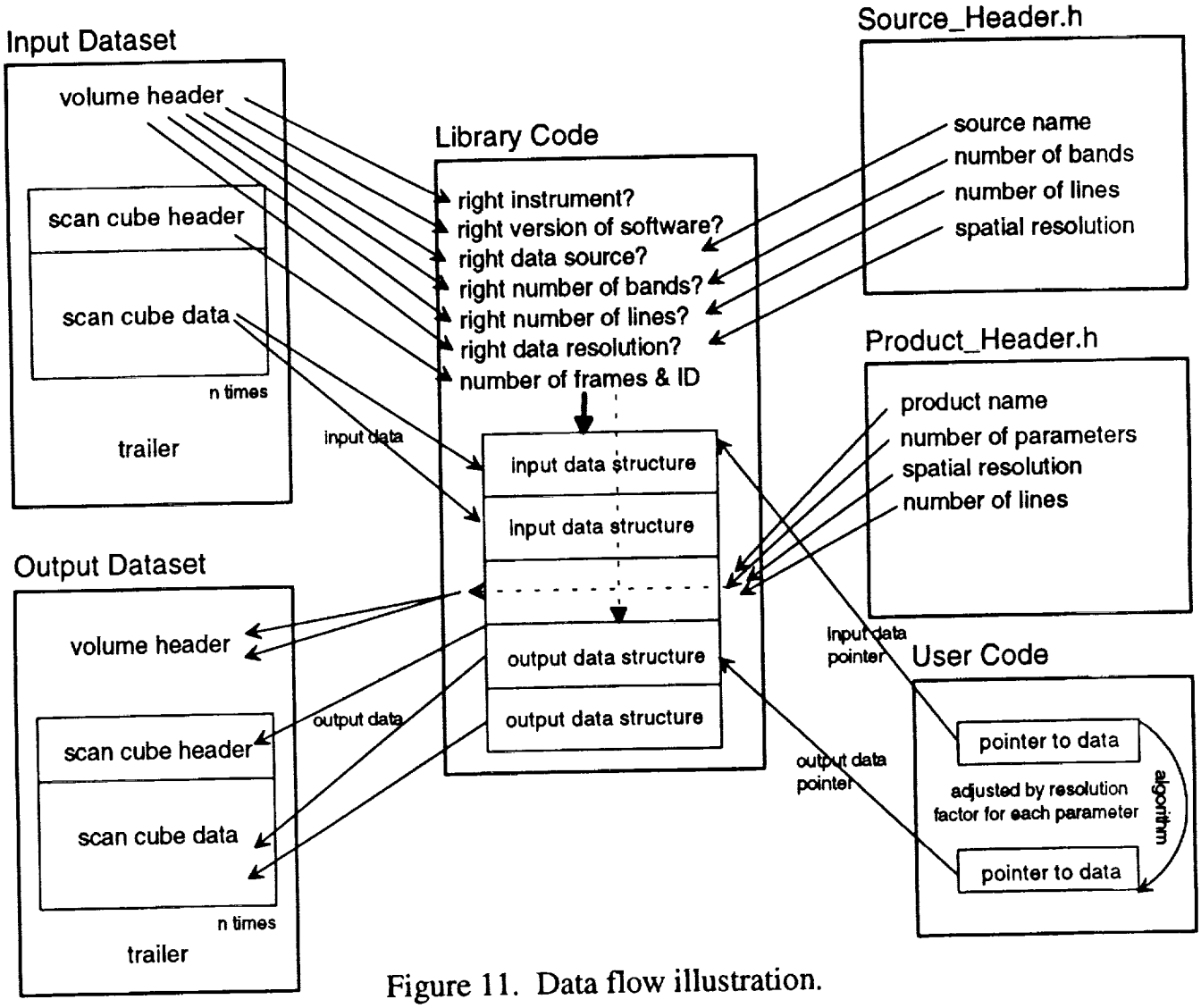

Figure 11. Data flow illustration.

The second figure illustrates dataset access by surrounding the user's code with the library functions. Dataset contents are accessed via the data structures defined in the data product header include files. Note that non-MODIS data is accessed via include files or other data structures that are not defined by this library.

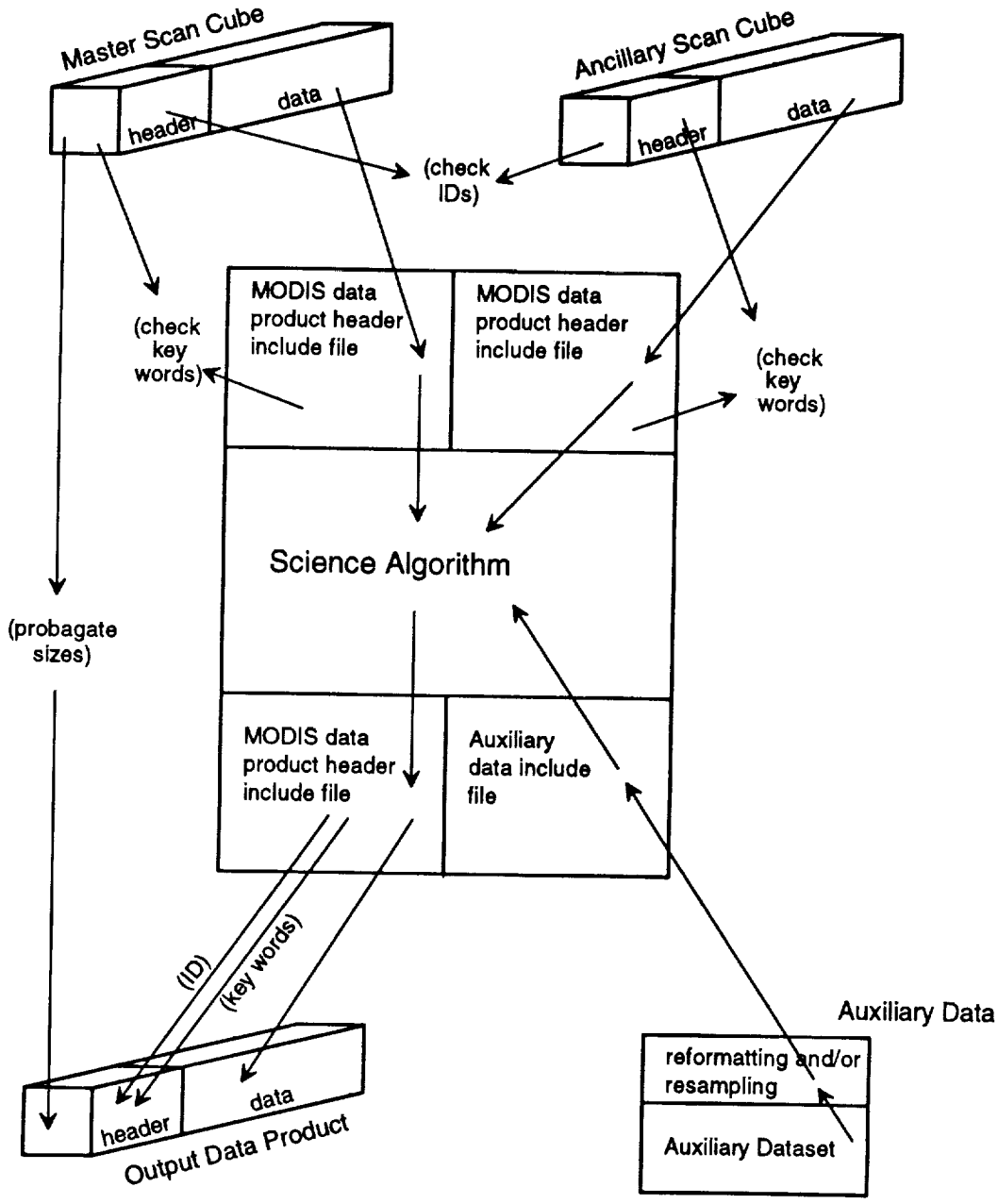

Figure 12. Algorithm dataset interactions.

(This page intentionally left blank)

GLOSSARY

This section includes the definitions of selected key words used in this document.

aggregated	Any of several methods for deriving a data value as a function of two or more input values. This is used in the context of determining which value to place in a gridded bin in a map projection. For example the result could be an average, maximum, minimum, or the result of a weighting function.
algorithm	A step-by-step problem solving procedure, especially an established, recursive computational procedure for solving a problem in a finite number of steps. Used in this document to represent a section of computer source code that performs this step.
ancillary	Used in this document to mean data that has been transformed into the MODIS scan cube data format.
ANSI	American National Standards Institute. The governing body for published standards, one of which is used to standardize programming language source code and specifications.
ATT	Algorithm Transfer Team. A group within the MODIS SDST that is chartered with helping algorithm programmers with porting source code to the PGS environment.
auxiliary	Used in this document to mean external data that is not in the MODIS scan cube data format.
AVHRR	Advanced Very High Resolution Radiometer. An instrument on a series of NOAA satellites. See the "NOAA Polar Orbiter Data" by K. Kidwell, available from NOAA for details.
band	A number used to represent a range of energy wavelengths from which a single detector measures a radiance energy. This is the same as a data channel for AVHRR and several other instruments but is not the same as a data channel for the MODIS instrument or other instruments with variable spatial IFOV coverages.
Black Body	A constant temperature emissive source within the MODIS instrument used to calibrate the thermal bands.
BB View	The detector energy measurements when viewing the MODIS Black Body.
C	A computer programming language, successor to the A and B languages, the name of which is consistent with the UNIX minimalist user command interface.
CZCS	Coastal Zone Color Scanner. An ocean discipline satellite remote sensing instrument.
DAAC	Distributed Active Archiving Center. The EOS component for formal Data Product storage, production, and access.

Data Product	A formal name of a science (or instrument) value that is registered with the SPSO database.
detector	A piece of hardware in the MODIS instrument that measures radiant energy at one IFOV.
DCB	Data Control Block: a table that is used to maintain information about a dataset. This may contain file names, sizes, pointers, links to associated datasets, and dataset descriptors.
Earth View	MODIS instrument detector readouts during the portion of the scan in which the Earth is in view.
ECS	EOS Core System. The ground processing and archiving, computer based segment of the EOS
EOS	Earth Observing System. A suite of remote sensing satellites. A component of MTPE.
Frames (MODIS instrument)	All the MODIS instrument data corresponding to a nominal 1 kilometer section of an across track scan. (830 data channels)
FTN or FORTRAN	Short for Formula Translator. A computer programming language.
geolocation	The act of determining the location, on the Earth geoid, of MODIS instrument IFOVs. This is performed for each spatial element (1 kilometer nominal footprint).
global	Of, relating to, or involving the entire earth; worldwide.
header (2 defs)	A section of the dataset containing information about the dataset (applied to both the dataset volume and each scan cube). Also, a section of source code that specifies data structures and descriptors that are common to more than one module. This library uses library include files, operating system include files, generic data structure include files, and Data Product specific header include files.
heritage	Something that is passed down from preceding generations; a tradition. In the MODIS case, techniques and algorithms that were used in the past, before MODIS was invented.
hierarchical	Of or relating to a hierarchy, a series in which each element is graded or ranked. In computer terms, elements of the hierarchy are wholly contained in (a subset of) higher elements.
IFOV	Instantaneous Field Of View. Used informally to mean the instrument field of view. This is the spatial area over which energy is measured by each instrument detector.
IMS	Information Management System. A component of the ECS that performs data product selection based on metadata values.

Latitude	The angular distance north or south of the earth's equator, measured in degrees along a meridian, as on a map or globe. The origin is at the Equator, increasing to 90 degrees at the North pole.
Level-0	A formal Data Product defined to be the MODIS instrument original data in packet form with CCSDS (Consultative Committee on Space Data Systems) headers prepended.
Level-1A	A formal Data Product defined to be the MODIS instrument detector values as raw counts. This includes the Earth View, Solar Diffuser, SRCA, Black Body, Space view, and engineering / memory dump data. Level-1A is fully reversible to the Level-0 Data Product.
Level-1B	A formal Data Product defined to be the MODIS instrument at-satellite detector values as radiance energy values received at the satellite in the instrument spatial geometry. This is not the at ground radiances and therefore has no atmospheric correction applied.
Level-2	Formal science Data Products in the instrument spatial geometry.
Level-3	Formal science Data Products in a globally mapped representation, corresponding to geographic shapes and dimensions.
Limb	The circumferential edge of the apparent disk of a celestial body. Used in MODIS terminology as the outermost detector measurements at the edges of the scan (maximum scan angles).
Longitude	Angular distance on the earth's surface, measured in degrees east or west from the prime meridian at Greenwich, England, to the meridian passing through a position. Positive to the East, negative to the West and less than 180 degrees in magnitude.
mapped	A representation, usually on a plane surface, of a region of the earth or heavens. The ECS will be using many types of maps produced via several gridding, binning, and aggregating schemes.
MAS	MODIS Airborne Simulator. A 50 channel aircraft instrument with MODIS like bands.
MCST	MODIS Characterization Support Team. The MODIS entity responsible for the calibration and characterization of the MODIS instrument.
Metadata	Data about other data. E.g., a synopsis of a Data Product used for selection criteria.
MISR	An instrument on EOS with forward and backward looking scans used for stereo and bi-directional reflectance Data Products.
MODIS	Moderate Resolution Imaging Spectroradiometer.
MTPE	Mission To Planet Earth. NASA's project that focuses attention of the Earth as opposed to space or other planets.

multiplexed	Relating to, having, or consisting of multiple elements or parts. Relating to or being a system of simultaneous communication of two or more messages on the same wire or radio channel. A system in which data values are combined together, asynchronously.
nadir	A point on the celestial sphere directly below the observer, diametrically opposite the zenith. The pierce point on the Earth of the MODIS instrument to center of Earth vector.
nbands	The integer number of distinct wavelength bands for Level-1A and Level-1B products. Also used as the number of parameters in a Data Product.
nBBframes	The number of instrument frames that contain detector views of the black body
nEMbytes	The number of equivalent instrument frames that contain memory dumps from the instrument and formatter onboard computer memories. All engineering data, internal tables, and op codes are contained in these data dumps.
nframes	The integer number of sampled MODIS instrument detector elements in the across track (along scan) direction for the finest resolution bands or parameter. This is unique to each Data Product, but would correspond to the number of pixels for the 250 meter bands for the MODIS Level-1A and Level-1B Data Products and would have a nominal value of 5416.
nlines	The integer number of lines of sampled MODIS instrument detector values in the along track (across scan) direction for the finest resolution bands or parameter. This is unique to each Data Product, but would correspond to the 40 pixels for the 250 meter bands for the MODIS Level-1A and Level-1B Data Products.
npixels	The integer total number of sampled MODIS instrument detector values in both the along track (across scan) direction and the across track (along scan) direction for the finest resolution bands or parameter. This is unique to each Data Product, but would correspond to the 40 pixels times nframes for the 250 meter bands for the MODIS Level-1A and Level-1B Data Products.
nres	The integer divisor applied to the nfames and nlines values for each band or parameter of the formal Data Product. This can be considered as an array of dimension [nbands]. For MODIS, this number = 1 for bands 1 and 2, = 2 for bands 3 to 7, and = 4 for the remaining bands.
nSDframes	The number of instrument frames that contain detector views of the solar diffuser.
nsize	The number of bytes that a data element consumes. This is the total for all components of the data element. For example, a Data Product consisting of a double, a long, and a byte value would consume 13 bytes per pixel.
nSRCAframes	The number of instrument frames that contain detector views of the Spectral Radiometric Calibrator Assembly.

nSVframes	The number of instrument frames that contain detector views of deep space.
OBC	OnBoard Calibrators. The set on four calibration sources contained within the MODIS instrument. See Solar Diffuser, SRCA, Black Body, and Space View.
parameter	Used in the MODIS context to define a component of a Data Product. Several parameters constitute a formal Data Product. QA is included as a parameter to a Data Product.
PGS	Product Generation System. A component of the ECS that produces Data Products.
pixel	picture element. The smallest granule of a picture that can be represented by a unique numerical value, usually on a video display.
prototype (ANSI)	A skeleton function call definition containing all the type declarations of its arguments, but without the actual variable names.
QA	Quality Assurance. The methods applied to MODIS Data Products for, and the resulting indicators of determining the quality and validity of the Data Product.
quantization	To limit the possible values of [a magnitude or quantity] to a discrete set of values by quantum mechanical rules. The process in which a continuous physical value is divided into discrete values, thereby limiting the precision of the measurement.
radiance	The radiant energy emitted per unit time in a specified direction by a unit area of an emitting surface.
raw counts	An integer number representing the amount of energy measured at a detector. This is sometimes called a digital number (DN) in other documentation.
rectified	To set right; correct. To correct by calculation or adjustment. Used in the MODIS sense to mean a straightening of the scan geometry to orthogonal axes and uniform spatial sampling.
rectilinear	Moving in, consisting of, bounded by, or characterized by a straight line or lines. The effort of resampling pixels in the MODIS instrument geometry with the 'bow tie' effect into a linearized and parallelized orbital coordinate system.
resampling	A process in which a value at a prespecified location (spatial or temporal) is derived from the values of its surrounding values.
scan	The data acquired during one half of the scan mirror rotation, consisting of multiple instruments Frames, and assembled into a data structure called a scan cube. See the MODIS Data Structure, Volumes, and Rates document for details.
SDPO	Science Data Products Office.
SDST	MODIS Science Data Support Team. The GSFC based group responsible for the production component of the MODIS ground processing system.

Solar Diffuser	An onboard calibration white target, illuminated by the Sun. The MODIS detectors view this target during a portion of the mirror scan rotation.
Space View	MODIS instrument detector readouts during the portion of the scan in which the deep space is in view.
spatial	Of, relating to, involving, or having the nature of space, relating to geometry as opposed to time (temporal).
Spatial Element	The nominal (at nadir) 1 kilometer area covered by an ideal MODIS IFOV. Each element contains all bands of data, irrespective of spatial resolution, that occur at each 1 kilometer footprint. See the MODIS SRR, PDR, and CDR documents for exacting details.
SRCA	Spectral Radiometric Calibrator Assembly.
swath	The contiguous area viewed by a MODIS instrument scan segment. For example, the portion of the MODIS mirror scan that views the Earth.
TBD	To Be Determined
temporal	Of, relating to, or limited by time: a temporal dimension; temporal and spatial boundaries.
TM	Thematic Mapper. A satellite remote sensing instrument.
TM	Team Members. Members of the MODIS Science Team.
token	A string of characters separated by a delimiter, usually a blank character (or whitespace in UNIX terms).
typedef (C)	A "C" language facility that allows the programmer to define a new, possibly compound data type. Examples of predefined data types are int, long, short, float, and double.
UNIX	The computer operating system selected by EOS as the standard platform.

REFERENCES

Unless otherwise noted, these documents are available from the MODARCH data archiving facility, code 920, Goddard Space Flight Center, Greenbelt, Maryland. This facility is available via the Internet.

1 - "MODIS Level 1A Software Baseline Requirements", SDST, Sept. 1, 1993, NASA TM 104594, Vol 1.

2 - "MODIS Data Rates, Volumes, & Processing Performance w/ Data Structures", Thomas Goff, May 13, 1994.

3 - "MODIS Processing Spatial Domains", Virginia Kalb and Thomas Goff.

4 - "MODIS PGS Data Processing Operations Concepts,", SDST, Sept. 13, 1993.

5 - "Science Computing Facilities Plan", Edward Masuoka, Sept. 7, 1993.

6 - "Level 1A System Requirements Review (SRR)", SDST, May 11, 1993.

7 - "PGS Toolkit Users Guide for the ECS Project", EOSDIS Core System Project, Feb. 4, 1994.

8 - "MODIS Software Development Standards and Guidelines", SDST, May 3, 1994 (draft).

9 - "Geometric Correction of MODIS Data", Virginia Kalb and Thomas Goff, Sept. 23, 1993.

10 - "MODIS Level 1A Earth Location ATBD", Robert Wolfe, July 25, 1994.

11 - "MODIS Sensor Patterns and Multiresolution Pixel Registration", Al Fleig, July 27, 1994.

12 - "EOS Reference Handbook", Earth Science Support Office, Document Resource Facility, 300 D Street NW, Suite 860, Washington D.C. 20024.

13 - "The Moderate Resolution Imaging Spectrometer-Nadir (MODIS-N) Facility Instrument", Advances in Space Research vol 11(3), 231-236, 1991, V.V. Salomonson and D.L.Toll.

(This page intentionally left blank)

INDEX

N

P

R

S

T

U

V

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1994	Technical Memorandum

4. TITLE AND SUBTITLE

MODIS Technical Report Series
Volume 4, MODIS Data Access User's Guide - Scan Cube Format

5. FUNDING NUMBERS

Code 920
C-NAS5-31331

6. AUTHOR(S)

Virginia L. Kalb and Thomas E. Goff

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS (ES)

Goddard Space Flight Center
Greenbelt, Maryland 20771

8. PEFORMING ORGANIZATION REPORT NUMBER

95B00013

9. SPONSORING / MONITORING ADGENCY NAME(S) AND ADDRESS (ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING ADGENCY REPORT NUMBER

NASA TM-104594, Vol. 4

11. SUPPLEMENTARY NOTES

Kalb: Goddard Space Flight Center, Greenbelt, Maryland;
Goff: Research and Data Systems Corporation, Greenbelt, Maryland

12a. DISTRIBUTION / AVAILABILITY STATMENT

Unclassified - Unlimited
Subject Category 61
This publication is available from theNASA Center for AeroSpace Information, 800 Elkridge Landing Road, Linthicum Heights, MD 21090-2934, (301) 621-0390.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The software described in this document provides I/O functions to be used with Moderate Resolution Spectroradiometer (MODIS) level 1 and 2 data, and could be easily extended to other data sources. This data is in a scan cube data format: a 3-dimensional ragged array containing multiple bands which have resolutions ranging from 250 to 1000 meters. The complexity of the data structure is handled internally by the library. The I/O calls allow the user to access any pixel in any band through "C" structure syntax. The high MODIS data volume (approaching half a terabyte per day) has been a driving factor in the library design. To avoid recopying data for user access, all I/O is performed through dynamic "C" pointer manipulation. This manual contains background material on MODIS, several coding examples of library usage, in-depth discussions of each function, reference "man" type pages, and several appendices with details of the include files used to customize a user's data product for use with the library.

14. SUBJECT TERMS	15. NUMBER OF PAGES
MODIS; Scan Tube	72
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL