Jet Propulsion Laboratory
California Institute of Technology

INTEROFFICE MEMORANDUM

MISR SDS DFM-0743-A

November 14, 2005

TO: Design File

FROM: Brian Rheingans
Charles Thompson

SUBJECT: MISR Toolkit API Design

Purpose

The purpose of the MISR Toolkit API is to simplify MISR L1B2 and L2 data product access. Access to MISR L1B2 and L2 is not intuitive and can be relatively complicated, especially for some one who is unfamiliar with the MISR "stacked-block" file format and block paradigm. There are several reasons for this complication:

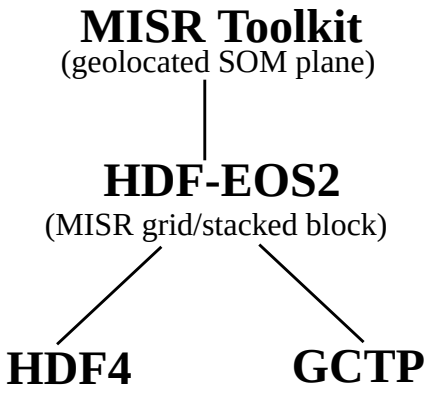
- MISR data is in non-standard "stacked-block" HDF-EOS format.
- MISR data is stored as blocks which must be "stitched" together.
- MISR data must often be unscaled or unpacked.
- MISR data has extra dimensions in some of the fields predominately in L2.
- HDF-EOS access tools that preserve geolocation (enable coordinate queries without an AGP lookup) are lacking.
- HDF-EOS support in commercial GIS is lacking, especially for the non-standard MISR "stacked-block" format.

Although, there are some applications that exist which facilitate the access of MISR data (misr_view, hdfscan), the access routines used by these tools are integral and are not separable. Currently, there is no independent simplified set of access and manipulation routines available to users of MISR data. This is the motivation for developing a MISR Toolkit API.

The MISR L1B2 and L2 products are stored in a unique MISR "stacked-block" HDF-EOS2 grid format. This unique format usually makes reading a MISR product file quite difficult because additional work needs to be performed, such as block stitching and data unpacking and unscaling to get the data into a usable form. HDF-EOS2 grid is a combination of HDF4 and GCTP. HDF4 handles the data storage and GCTP provides a means of storing geolocation and projection metadata. HDF-EOS2 files are designed to provide geolocation without the use of other data sets, such as the Ancillary Geographic Product (AGP). The problem is there are no other tools available that simultaneously make use of the GCTP geolocation metadata and are aware of the MISR "stacked-block" format.

The MISR Toolkit will provide simplified MISR data access and provide geolocation

functionality utilizing the GCTP metadata, instead of an ancillary data set lookup. It will only depend on the HDF-EOS2, HDF4 and GCTP libraries. It will not depend on the PGS Toolkit. As shown in the following diagram the MISR Toolkit will abstract the MISR grid "stacked-block" format to a geolocated SOM plane via a series of function calls. Block stitching and data unpacking and unscaling are performed automatically.



Since the plane is geolocated to a SOM projection, coordinate query routines based on the GCTP metadata are provided to map between plane line and sample, SOM x/y and geographic latitude/ longitude. In addition to reading and querying a plane of MISR data, other functions such as map reprojection and grid rescaling are provided. Other functions that operate on a data plane can be defined later to provide virtually any additional functionality. Abstracting the MISR grid "stacked-block" format to a data plane not only makes MISR data access easier, but will also provided a framework for additional functionality. The MISR Toolkit is implemented using standard C, with wrappers for languages such as C++, Java, Python, Fortran, and IDL.

Similar access routines are needed to enable reading of the MISR conventional product files. These will be added later to provide access consistency across these MISR products.

Requirements

1. Will abstract MISR L1B2 and L2 data access while automatically stitching together the MISR blocks.

2. Will access the conventional product files with the same or similar access paradigm and interface.

3. Will return any MISR grid “stack-blocked” data product or conventional data product as a 2D data plane preserving the native SOM projection, along with a mapinfo structure to allow coordinate querying of the data plane.

4. Will provide map reprojection functionality supporting all of the GCTP map projections.

5. Will support a wide range of computer architectures such as recent versions of Linux, Windows, Mac OS X, Solaris and IRIX operating systems.

6. Will depend only on the HDF-EOS2, GCTP and HDF4 external libraries and not the PGS toolkit.

7. Will be a structured design to interface with other structured languages such as Fortran and IDL. Support for object-oriented languages such as C++, Java, Python will also be added.

8. Will only access MISR data, not write it out to other formats.

9. Sample applications will be provided to show how to access MISR data using this API.

10. Will be distributed in source form under the Open Channel license.

11. Will provide documentation, user guide and sample programs demonstrating and explaining API usage.

12. Will be tested using unit tests, sample programs and a group of users familiar with MISR data products.

Applications

- Comparison of MISR data with other geolocated data sets from several sensors at a specific geographic location and point in time.
- Co-register multiple data sets for a specific geographic region, resolution and projection.
- Building analysis tools which require access to MISR data sets.
- Associating geographic regions with paths, orbits and blocks and vice-versa.

Benefits

- Enscapsulates low-level data access routines into a simplified and flexible set of extraction and manipulation functions. Eliminates the necessity to depend directly on HDF-EOS routines or

other applications.
- Can be used as a common basis for developing tools and applications.
- Supports all the GCTP projections.
- Allows MISR data to be accessed without any knowledge of the “block” dimension.
- Generalized functionality can be broadened to include data sets from other EOS instruments.
- Allows access to MISR data through specification of geolocation information (latitude/longitude).
- Available within IDL as a dynamically-loaded module (DLM).
- Eliminates the need to rely on the Ancillary Geographic Product (AGP) or other pre-calculated geolocation information.
- Geolocation information available at any resolution, not just 1100m.

Usage Examples

Case 1: The sequence of steps to retrieve and optionally reproject a geolocated SOM plane of MISR data are:

1) Select a region indicating an area you would like to work with. This region defines an approximate location and area of interest. Depending on the MISR path that is to be read, the plane of data returned will be snapped to the grid defined by that path’s SOM projection. A region can be reused in many read data calls to get other parameters or paths. Distinct paths will be snapped to distinct grids. See step 4 to reproject them to the same projection.

2) Once a region, a filename, a gridname and a fieldname are determined, the data can be retrieved using a read data function. This function will snap the region of interest to the particular SOM grid of the path specified by the MISR product file. A data buffer and a map info structure will be returned. The data buffer contains an array representing the geolocated SOM plane of MISR data and the map info structure contains all the “snapped” region and map information of the data plane.

3) The map info structure can then be used to query the coordinates of the data plane. You will be able to map between line and sample, SOM x/y and geographic latitude/longitude.

4) Since distinct MISR paths are separate SOM projections, the grid cells do not align. A set of reprojection routines enable the data planes to be mapped or warped to a new map projection. This functionality will enable MISR data to be reprojected easily to other map projections for intercomparison with other map projected data.

Case 2: Locating a path/orbit/field from a region, location and/or file:

1) Using a selected region from the region selection routines or a latitude/longitude location, a function returning a list of MISR paths that cover the region can be used. This path list can aid in MISR product file selection.

2) Orbit is a function of time and path, therefore a routine given a time should return the orbit and path observed at that time.

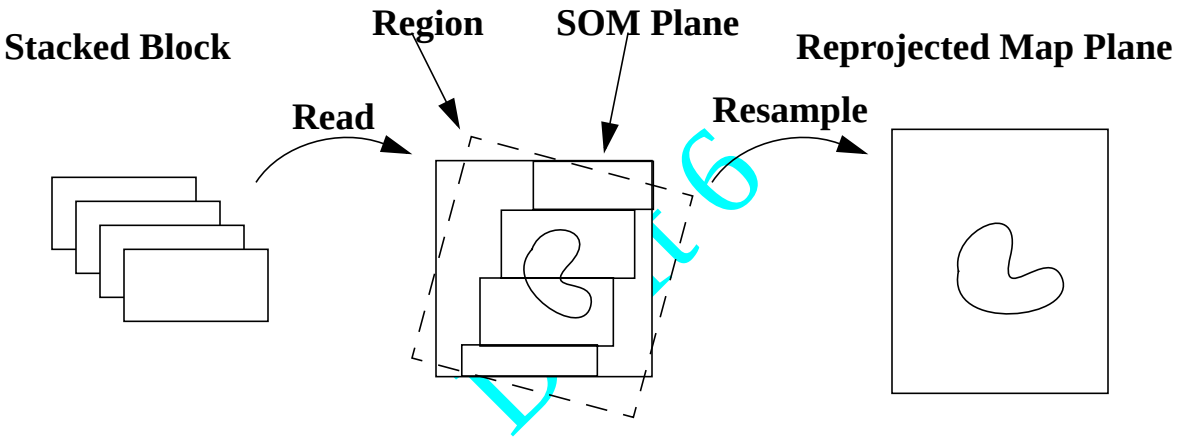
3) Once path, orbit and the particular MISR product file are determined, the file/grid/field query routines can be used to obtain lists of grids and fields. From these lists, the name of the field and grid can be obtained to be used by the read routines.

Case 3: Coordinate conversion routines:

1) Given an MISR path, coordinate conversion between block, line, sample, SOM x/y and latitude/longitude can easily be performed.

2) Given a mapinfo structure returned from the read routines, coordinate conversion between data plane, line and sample, SOM X/Y and latitude/longitude can also be performed.

Conceptual Diagram



1) Primary Routines

Primary routines to access, query and reproject MISR data.

1.1) Region Selection (SetRegion)

Before reading a geolocated plane of MISR data, a region on the globe needs to be specified. A region describes an *approximate* area of interest. There is no implied orientation of a region. It basically describes a unique location (center of the region) on the globe and two extents perpendicular to each other. The real orientation and thus the directions of the extent depends on the orbit path that is read. The region gets *snapped* to the grid of the particular path's SOM plane. Region selection can be done one of five different ways: 1) by specifying a bounding box via upper left corner (ulc) and the lower right corner (lrc) in terms of latitude and longitude in decimal degrees (the orientation implied by this method is only used to establish the two extents), 2) by specifying a center latitude and longitude in decimal degrees and an extent in the latitude and longitude direction in meters, 3) by specifying a center location and an extent in decimal degrees, 4) by specifying a center location and extent in pixels, 5) or by specifying a MISR path and block

range which will define an encapsulating bounding box and thus determine the two extents. Only one of these routines needs to be called to define a region. A region can be used more than once in the many read data calls.

```
MtkSetRegionByUlcLrc(ulc_lat_dd, ulc_lon_dd
                     lrc_lat_dd, lrc_lon_dd, &region)
MtkSetRegionByLatLonExtentMeters(ctr_lat_dd, ctr_lon_dd,
                                 lat_extent_meters, lon_extent_meters, &region)
MtkSetRegionByLatLonExtentKilometers(ctr_lat_dd, ctr_lon_dd,
                         lat_extent_kilometers, lon_extent_kilometers, &region)
MtkSetRegionByLatLonExtentDeg(ctr_lat_dd, ctr_lon_dd,
                               lat_extent_deg, lon_extent_deg, &region)
MtkSetRegionByLatLonExtentPixels(ctr_lat_dd, ctr_lon_dd, resolution,
                                 lat_extent_pixels, lon_extent_pixels, &region)
MtkSetRegionByPathBlockRange(path, start_block, end_block, &region)
```

A region is snapped to an SOM plane for a given path and resolution. This is the purpose of this routine. The Read routines below calls this routine internally, however, it may be used externally to determine various characteristics of the SOM plane, such as size, map extent, projection paramters, etc. This routine need not be called be the user unless under special circumstances.

```
MtkSnapToGrid(path, resolution, region, mapinfo)
```


1.2) Reading a Geolocated SOM Plane (ReadData)

There is one primary routine to read a geolocated SOM plane of MISR data given a file, grid, field and a region. Remember a region is an *approximate* area of interest MtkReadData is most general and in most cases will be the only routine required. It calls the MtkReadRaw and MtkReadConv as needed. MtkReadData is aware of every MISR product and parameter and implicitly unpacks and unscales as needed. It reads both MISR standard products and conventional products. For the standard product files, all block stitching is done implicitly.

```
MtkReadData(filename, gridname, fieldname, region, &databuf, &mapinfo)
```

MtkReadRaw reads any native grid/field from a MISR product file without unpacking or unscaling. MtkReadConv reads the MISR conventional product files.

```
MtkReadRaw(filename, gridname, fieldname, region, &databuf, &mapinfo)
MtkReadConv(filename, gridname, fieldname, region, &databuf, &mapinfo)
```

In addition to the above routines there is a routine to read a single block at a time.

```
MtkReadBlock(filename, gridname, fieldname, block_number, &databuf)
```

1.3) SOM Plane Coordinate Query (MapQuery)

After a plane of data is read, it may be desirable to map between line and sample in the plane and geographic latitude/longitude or SOM x/y. These routines provide this mapping given the mapinfo

structure returned by MtkReadData or MtkReadRaw. There are two sets of these routines, one set converts a single location and the other converts an array of locations.

```
MtkLSToSomXY(mapinfo, line, sample, &x, &y)
MtkSomXYToLS(mapinfo, x, y, &line, &sample)
MtkLSToLatLon(mapinfo, line, sample, &lat_dd, &lon_dd)
MtkLatLonToLS(mapinfo, lat_dd, lon_dd, &line, &sample)

MtkLSToSomXYAry(mapinfo, nelement, line[], sample[], &x[], &y[])
MtkSomXYToLSAry(mapinfo, nelement, x[], y[], &line[], &sample[])
MtkLSToLatLonAry(mapinfo, nelement, line[], sample[], &lat_dd[], &lon_dd[])
MtkLatLonToLSAry(mapinfo, nelement, lat_dd[], lon_dd[], &line[], &sample)
```

1.4) Map Reprojection

It may be desirable to reprojection a geolocated SOM plane to another map projection over another region. These routines allow the user to specify an output map projection, output region and resampling method. The specific routine names and arguments are TBD.

```
1) Specify output region selection (may use the routines above)
2) Specify output map projection using GCTP projection parameters
3) Specify any grid scale changes
4) Specify resampling method
5) Perform reprojection operation
```

2) Secondary Routines

Secondary routines are mostly used internally by the primary routines. They are, however, accessible to the toolkit user if need be.

2.1) Orbit/Path Query (OrbitPath)

These routines are used for MISR path selection. They will return a list of MISR paths that intersect a given latitude/longitude location or a given region from a region selection call. Even the block range of a region can be determined for a particular path. This will help determine which MISR paths to obtain and read with the read data routines.

```
MtkLatLonToPathList(lat_dd, lon_dd, &pathcnt, &pathlist[])
MtkRegionToPathList(region, &pathcnt, &pathlist[])
MtkRegionPathToBlockRange(region, path, &start_block, &end_block)
```

Orbit selection is also required in order to determine a MISR product filename. A single path has many orbits. Orbit is a function of time. These routines will help determine which orbits / paths are at a given time.

```
MtkTimeToOrbitPath(time, &orbit, &path)
MtkTimeRangeToOrbitList(start_time, end_time, &orbitcnt, &orbitlist)
MtkPathTimeRangeToOrbitList(path, start_time, end_time, &orbitcnt, &orbitlist)
MtkOrbitToPath(orbit, &path)
```


2.2) File/Grid/Field Query (FileQuery)

Sometimes it is useful given a product, path, orbit, camera and/or version to do a wildcard search recursively for a MISR product file in a filesystem or construct a filename string from these components.

```
MtkMakeFilename(basedir, product, camera, path, orbit, version, &filename)
MtkFindFileList(searchdir, product, camera, path, orbit, version,
                &filecnt, &filename[])
```

Also useful, are routines that when given a filename, gridname or fieldname return some attributes of the MISR product file. These attributes include a list of all the grids and/or fields in a file, the path number, block range of a file, grid resolution, field datatype, field fill value, etc. These routines can be used to query the filenames, gridnames and fieldnames in the MISR products.

```
MtkFileToPath(filename, &path)
MtkFileToBlockRange(filename, &start_block, &end_block)
MtkFileToGridList(filename, &gridcnt, &gridlist[])
MtkFileGridToFieldList(filename, gridname, &fieldcnt, &fieldlist[])
MtkFileGridToResolution(filename, gridname, &resolution)
MtkFileGridFieldToDataType(filename, gridname, fieldname, &datatype)
MtkFillValueGet(filename, gridname, fieldname, &fillvalue)
MtkFileLGID(filename, &local_granuale_id)
MtkFileType(filename, &filetype)
MtkFileVersion(filename, &fileversion)
```

2.3) Coordinate Conversion (CoordConv)

Coordinate conversion routines are provided to perform much of the underlying geolocation operations. They are not required by the end user for reading and locating MISR data, but may be useful in some circumstances. They mostly have to due with the MISR block, line and sample structure which this toolkit attempts to hide. Two particularly interesting routines to the end user are MtkLatLonToSomXY and MtkSomXYToLatLon. Given a MISR Path, conversion between SOM x/y and geographic latitude/longitude coordinates can conveniently be performed. There are also two sets of these routines, one set converts a single location and the other converts an array of locations.


```
MtkPathToProjParam(path, resolution, &projparam)

MtkLatLonToSomXY(path, lat_dd, lon_dd, &x, &y)
MtkSomXYToLatLon(path, x, y, &lat_dd, &lon_dd)
MtkBlsToSomXY(path, resolution, block, line, sample, &x, &x)
MtkSomXYToBls(path, resolution, x, y, &block, &line, &sample)
MtkLatLonToBls(path, resolution, lat, lon, &block, &line, &sample)
MtkBlsToLatLon(path, resolution, block, line, sample, &lat, &lon)

MtkLatLonToSomXYAry(path, nelement, lat_dd[], lon_dd[], &x[], &y[])
MtkSomXYToLatLonAry(path, nelement, x[], y[], &lat_dd[], &lon_dd[])
MtkBlsToSomXYAry(path, resolution, nelement, block[], line[], sample[],
                 &x[], &x)
MtkSomXYToBlsAry(path, resolution, nelement, x[], y[],
                 &block[], &line[], &sample[])
MtkLatLonToBlsAry(path, resolution, nelement, lat[], lon[],
                  &block[], &line[], &sample[])
MtkBlsToLatLonAry(path, resolution, nelement, block[], line[], sample[],
                  &lat[], &lon[])
```

2.4) Unit Conversion (UnitConv)

General routines to convert between decimal degrees, radians, and both packed and unpacked degrees, minutes, seconds.

```
MtkDmsToDd(dms, &dd)
MtkDdToDms(dd, &dms)

MtkDdToRad(dd, &rad)
MtkRadToDd(rad &dd)

MtkDmsToRad(dms, &rad)
MtkRadToDms(rad, &dms)

MtkDmsToDegMinSec(dms, &deg, &min, &sec)
MtkDegMinSecToDms(deg, min, sec, &dms)

MtkDdToDegMinSec(dd, &deg, &min, &sec)
MtkDegMinSecToDd(deg, min, sec, &dd)

MtkRadToDegMinSec(rad, &deg, &min, &sec)
MtkDegMinSecToRad(deg, min, sec, &rad)
```

2.5) Memory Management (Util)

It may be useful to allocate a buffer of memory for temporary work. These two routines allocate and free 2D planes of memory of any specified type.


```
MtkDataBufferAllocate(nline, nsample, datatype, &databuf)
MtkDataBufferFree(&databuf)
```

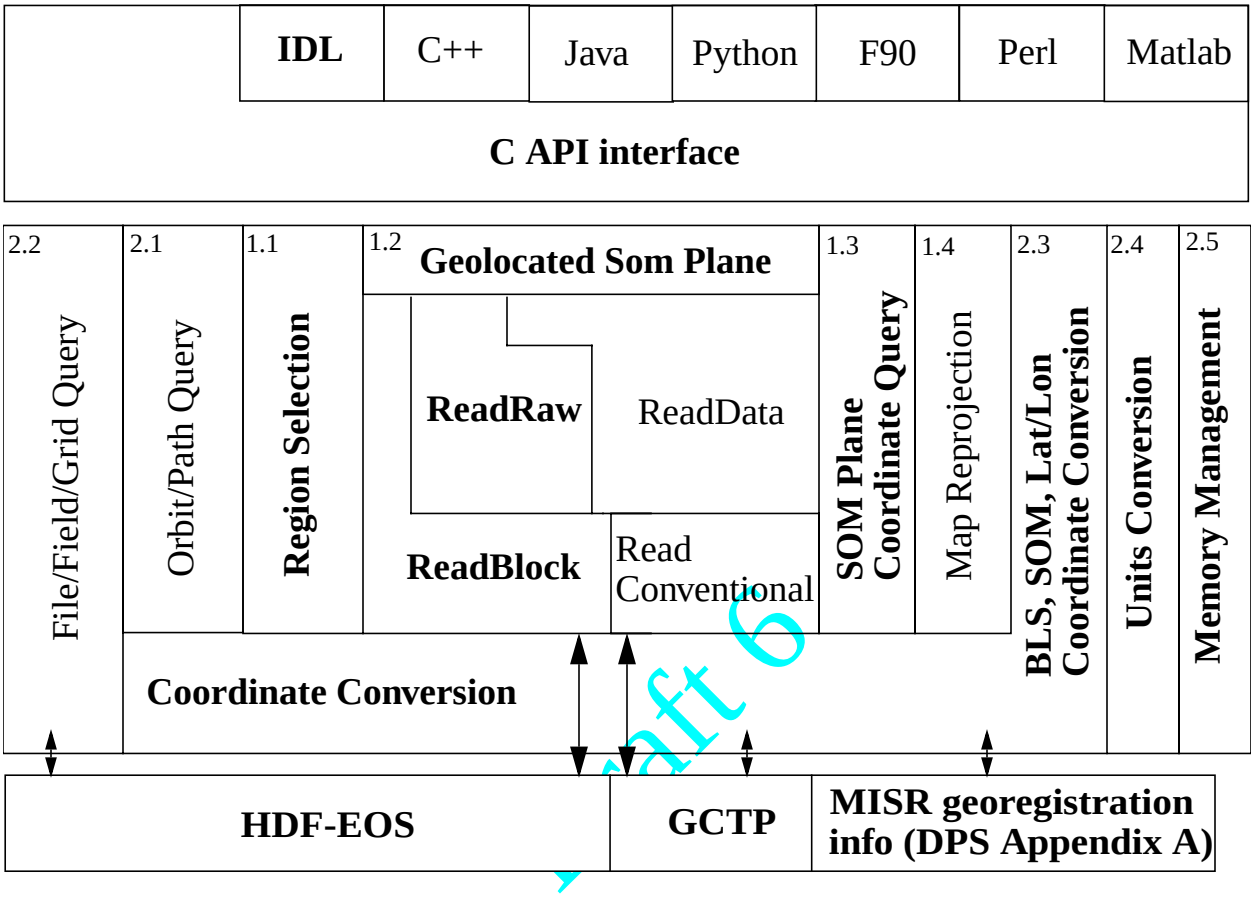
Given a fieldname of the form “SpectralOpticalDepth[1][2]” this routine will return the base fieldname “SpectralOpticalDepth” and an array of indices “1,2”. This routine is used to access MISR product field that have multiple dimensions.

```
MtkParseFieldname(nfieldname, &outfieldname, &dim[])
```

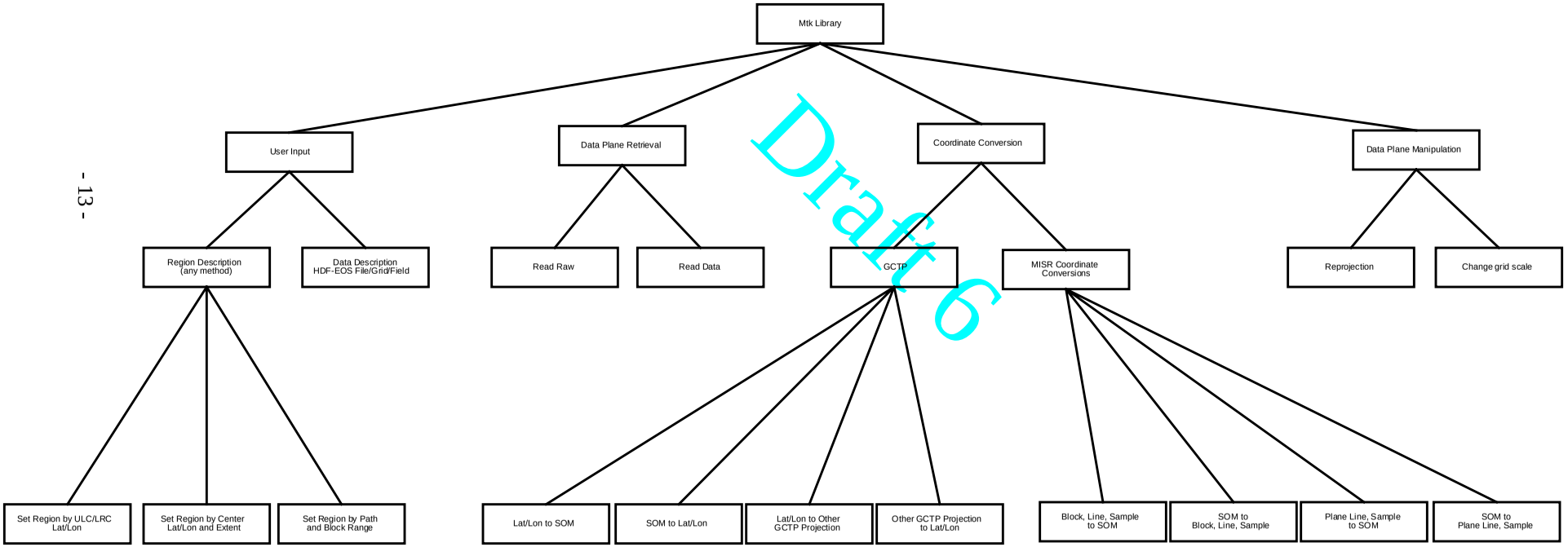
Routine to convert between HDF data types and MTK datatypes.

```
MtkHdfToMtkDataTypeConvert(hdf_datatype, &mtk_datatype)
```

Component Overview



Bold indicates functional component
Number refers to design memo section


Mtk Library
User Input
Data Plane Retrieval
Coordinate Conversion
Data Plane Manipulation
Region Description
(any method)
Data Description
HDF-EOS File/Grid/Field
Read Raw
Read Data
GCTP
MISR Coordinate
Conversions
Reprojection
Change grid scale
Set Region by ULC/LRC
Lat/Lon
Set Region by Center
Lat/Lon and Extent
Set Region by Path
and Block Range
Lat/Lon to SOM
SOM to Lat/Lon
Lat/Lon to Other
GCTP Projection
Other GCTP Projection
to Lat/Lon
Block, Line, Sample
to SOM
SOM to
Block, Line, Sample
Plane Line, Sample
to SOM
SOM to
Plane Line, Sample