McIDAS

Programmer’s Manual

Preliminary Issue October 1995

This preliminary issue of Chapters 4, 5 and 6 is a supplement to the McIDAS Developer/Operator training sessions held October 23-25, 1995. These chapters are still in a very early stage of development and will be completed and reissued with the final version of the McIDAS Programmer’s Manual.
Data File Access and Procedures

This chapter describes how to read, write and delete ADDE (Abstract Data Distribution Environment) image datasets. It also describes how to read, write and list ADDE grid datasets.

Contents

- ADDE image object I/O .................................. Page 4-3
- Terminology ............................................. 4-3
- Reading an ADDE image dataset ....................... 4-5
- Writing an ADDE image dataset ....................... 4-22
- Deleting an ADDE image dataset ....................... 4-27
- The MCASORT function .................................. 4-29
- ADDE grid object I/O .................................. Page 4-31
- Reading an ADDE grid dataset ....................... 4-34
- Writing an ADDE grid dataset ....................... 4-36
- Listing an ADDE grid dataset ....................... 4-38
- Function descriptions for grids ....................... 4-40
ADDE image object I/O

This section describes the ADDE image object I/O. First it defines the terms used in this section. Then it provides information about reading, writing and deleting ADDE image datasets.

Terminology

The following terms are used in this section. They supplement the terms defined in the ADDE section of the McIDAS-X and -OS2 Users Guide.

**AREA**—the McIDAS image file format.

**calibration block**—the block that holds the information for transforming the image element’s sensor units to more common physical units, such as IR temperature or visible albedo.

**comment block**—a collection of 80-character text fields documenting any processing that may have altered the image elements, types of calibrations available for this image, or the latitude/longitude of the image’s center element.

**connection**—the initialization that occurs when a client determines the location of the dataset server and then issues a request for a data exchange; the server examines the request and determines its validity; if the request is valid, the connection is opened and the client is authorized to begin its transaction.

**data block**—a 2-dimensional matrix of image elements; the dimensions of the data block and size of each element are in the directory block.

**directory block**—an image object block containing a description of the physical characteristics of the image and the location of all ancillary blocks in the object.

**image element**—the individual data value produced by a sensor.

**image line**—the row dimension of a data block; image elements are ordered from left to right in the image line.
image line prefix—the prefix section of an image line that holds ancillary data defined by some image types; mapping of the line prefix entries is sensor specific and is documented in the McIDAS-X and -OS2 Users Guides.

image object—a rectangular array of elements that collectively represents an image and its collateral information.

image object blocks—a collection of image objects; each block contains either image elements or collateral information.

navigation block—the block that holds the information for determining the location of image elements in physical space; it normally includes the precise timing and attitude information of the sensor platform used to determine the earth latitude/longitude of an image element.

sort clause—a text string that specifies the spacial, temporal and spectral limits of a transaction; the server defines the number and format of the sort clauses for defining a request.

transaction—any ADDE exchange; it implies a transfer between an ADDE client and server.

---

**Reading an ADDE image dataset**

Reading an ADDE image dataset consists of reading these blocks:

- image directory block
- data block
- navigation block
- calibration block
- comment block

**Reading the image directory block**

Traditionally, McIDAS applications that access image objects (AREAs) read the directory block before accessing the image elements. The directory contains the parameters for reading and writing the image elements. READD is the subroutine that accesses the directory block. It takes an AREA number as input and returns the directory block associated with that number. The applications wanting images with specific criteria must read the directory and inspect the entries to determine if the image meets all the conditions of the search.

The following code segment, which is an old AREA directory read loop example, is typical of most applications that read/write McIDAS AREAs.
Code segment showing the AREA directory read loop.

```c
# I want an area with the following
SS = 32           : GDES-7 Visible
DAY = 95001        : Jan 1 1995
TIME = 100000     : 10 Z

# search the following range of areas
beg_area = 100
end_area = 199

# read directory for area range beg_area to end_area
do:100 area=beg_area,end_area
call readd(area,directory)

# validate area existence
if( directory(1).eq.0 ) then

# check area parameters
if( directory(3).ne.SS ) goto 100
if( directory(4).ne.DAY ) goto 100
if( directory(5).ne.TIME ) goto 100

# do something using the values in the area directory

endif
100 continue
```

The MCADIR function

The ADDE interface to the image directory is through the MCADIR function, which opens a connection based on a set of sort clauses for a given dataset name. The dataset name does not contain the position field. The position, if known, is specified with the SUBSET sort clause. For example, if the application requires the first 10 images for a given dataset, the sort clause is SUBSET 1 10. If the application requires all the image directories in a dataset, the sort clause is SUBSET ALL.

MCADIR also returns the image objects' comment block. You can specify the sort clause AUX (auxiliary) to provide an enhanced comment block. If you include the sort clause AUX YES in the sort condition list, the image object directory server will append comment entries describing the latitude and longitude of the center element of the image, the earth AREA (in kilometers) covered by the center element of the image, and the valid calibration types for the image, as shown below.

```
Center latitude = longitude
Center longitude = longitude
Latitude resolution (km) = resolution
Longitude resolution (km) = resolution
Valid calibration for band band = unit
```

Sort clauses can also restrict the search based on the image day, in the format YYYYDDD; the image start time, in the format HHMMSS; and the SSEC sensor source number, 1 to 99. You must specify these sort clauses as a range of values. Below is a list of the valid sort clauses that you can use with the MCADIR function.

```
|
AUX YES or AUX NO    | appends the center lat/lon, resolution and calibration types to the comment block (default=YES)
DAY bday eday       | image Julian day range
SS x1 x2            | SSEC sensor source number range
SUBSET hpos epos     | ADDE position range, or SUBSET ALL to the comment block
TIME btme etme      | image time range
```

The MCADDR function

Once the connection is opened with MCADIR, the application makes repeated calls to the MCADDR function. As long as the function status returns zero, MCADDR has returned an image object directory block and the associated comment block. A function status of one means all directory blocks in the dataset matching the sort conditions were returned. The MCADIR call must precede the call to MCADDR.

Code segment showing the ADDE image object directory read loop.

```c
# set sort conditions
sorts(1) = 'SS 32 32';
sorts(2) = 'DAY 95001 95001';
sorts(3) = 'TIME 100000 100000';
sorts(4) = 'SUBSET ALL';
nsort = 4

# dataset name
dataset = 'ET/GDES-7';

# turn error reporting on
error_flag = 1

# open a connection for the specified dataset
if( mcdir(dataset,nsort,sorts,error_flag).lt.0 ) return

100 continue

# read an image directory block meeting the search conditions
status = mcaddr(directory,comment_cards)

# read failed
if( status.1t.0 ) then
call edest('Failed during directory read of'/dataset,0)
return
endif

# found one
else if( status.eq.0 ) then

endif
```
The entries in the MCADDR directory block are described below; they are identical to entries 2 through 64 of the READD directory block.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>time relative position of the image object in the dataset</td>
</tr>
<tr>
<td>2</td>
<td>absolute position of the image object in the dataset</td>
</tr>
<tr>
<td>3</td>
<td>image type</td>
</tr>
<tr>
<td>4</td>
<td>SSEC sensor source number</td>
</tr>
<tr>
<td>5</td>
<td>nominal year and Julian day of the image, YYDDD</td>
</tr>
<tr>
<td>6</td>
<td>nominal time of the image, HHMMSS</td>
</tr>
<tr>
<td>7</td>
<td>upper-left line in satellite coordinates</td>
</tr>
<tr>
<td>8</td>
<td>upper-left element in satellite coordinates</td>
</tr>
<tr>
<td>9</td>
<td>upper-left z-coordinate</td>
</tr>
<tr>
<td>10</td>
<td>number of lines in the image</td>
</tr>
<tr>
<td>11</td>
<td>number of elements in the image</td>
</tr>
<tr>
<td>12</td>
<td>number of bytes per element</td>
</tr>
<tr>
<td>13</td>
<td>line resolution</td>
</tr>
<tr>
<td>14</td>
<td>element resolution</td>
</tr>
<tr>
<td>15</td>
<td>band resolution</td>
</tr>
<tr>
<td>16</td>
<td>line prefix length</td>
</tr>
<tr>
<td>17</td>
<td>project number</td>
</tr>
<tr>
<td>18</td>
<td>creation year and Julian day, YYDDD</td>
</tr>
<tr>
<td>19</td>
<td>creation time, HHMMSS</td>
</tr>
<tr>
<td>20</td>
<td>band map</td>
</tr>
<tr>
<td>21</td>
<td>image ID number</td>
</tr>
<tr>
<td>22-25</td>
<td>identification</td>
</tr>
<tr>
<td>26-33</td>
<td>memo field</td>
</tr>
<tr>
<td>34</td>
<td>reserved</td>
</tr>
<tr>
<td>35</td>
<td>byte offset to the start of the data block</td>
</tr>
<tr>
<td>36</td>
<td>byte offset to the start of the navigation block</td>
</tr>
<tr>
<td>37</td>
<td>validity code</td>
</tr>
<tr>
<td>38-45</td>
<td>PDL</td>
</tr>
<tr>
<td>46</td>
<td>source of band 8</td>
</tr>
<tr>
<td>47</td>
<td>actual image start year and Julian day, YYDDD</td>
</tr>
<tr>
<td>48</td>
<td>actual image start time, HHMMSS</td>
</tr>
<tr>
<td>49</td>
<td>actual image start scan</td>
</tr>
<tr>
<td>50</td>
<td>length of the prefix documentation</td>
</tr>
<tr>
<td>51</td>
<td>length of the prefix calibration</td>
</tr>
<tr>
<td>52</td>
<td>length of the prefix band map</td>
</tr>
<tr>
<td>53</td>
<td>source type</td>
</tr>
<tr>
<td>54</td>
<td>calibration type</td>
</tr>
<tr>
<td>55-60</td>
<td>reserved</td>
</tr>
<tr>
<td>61</td>
<td>byte offset to the start of the AUX block</td>
</tr>
<tr>
<td>62</td>
<td>byte length of the AUX block</td>
</tr>
<tr>
<td>63</td>
<td>reserved</td>
</tr>
<tr>
<td>64</td>
<td>byte offset to the start of the calibration block</td>
</tr>
<tr>
<td>65</td>
<td>number of comment cards</td>
</tr>
</tbody>
</table>

**Reading the data block**

The example code segment below is typical of the code required to read an AREA. The application opens the AREA (OPNARA), reads the data lines (REDARA) and then closes the AREA (CLSARA). The application performs all image sectorization and resolution manipulation.

**Code segment showing the AREA data read loop.**

```c
  c --- set line bounds
  beg_line = 1
  end_line = area_directory(9)

  c --- set element bounds
  beg_elem = 1
  end_elem = area_directory(10)
  nelens = end_elem - beg_elem + 1

  c --- band number
  band = lwp('BAND', 1, 8)

  c --- open the area
  call openarea(area)

  c --- declare output units (IR temperature)
  call arscept(area, 1, 'UNIT', 'JIT', 'TEMP')

  c --- declare output precision (4 bytes/element)
  call arscept(area, 1, 'SPAC', 4)

  c --- read area lines
  do 100 line = beg_line, end_line
     call readarea(area, line, beg_elem, nelens, band, data_buffer)
  c --- scan the data elements
     do 200 element = beg_elem, end_elem
        ... continue
     100 continue

  c --- close the area
  call clarea(area)
```
The MCAGET, MCALIN and MCAFREE functions

The ADDE version of the image object data block read is based on defining a request for an image sector, which is a fragment of an image object contained in a dataset. The process is initiated by specifying a set of conditions describing the desired data segment. These sort conditions form the basis of the client’s request to the server. If an image in the dataset satisfies the sort conditions, a connection is opened and the transaction proceeds. The transaction is done on a line-by-line basis until the entire request segment is transferred.

The MCAGET function passes the request from the client to the server. The return status shows if the connection is opened. If it is, the MCALIN function is called repeatedly to fetch an image line. As long as the return status is zero, an array of image data is present. A return status of one means the entire requested sector was transferred, the transaction is complete and the connection is closed.

The MCAFREE function returns the handle and the memory allocated by MCAGET. It should be called after MCALIN exhausts the transaction.

Applications use sort clauses to communicate with the data block server. The number and format of sort clauses are strictly regulated. These clauses allow the application to specify spatial, temporal and spectral limits of the transaction, eliminating the need for the application to scan the dataset for a particular image object. Below is a list of valid sort clauses for the MCAGET interface, followed by additional information for each.

<table>
<thead>
<tr>
<th>Sort clause format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUX YES or AUX NO</td>
<td>appends the unit and scale to the directory block</td>
</tr>
<tr>
<td>BAND band</td>
<td>spectral band, if the image has multiple bands</td>
</tr>
<tr>
<td>CAL QTRR</td>
<td>quick calibration switch for PORES images</td>
</tr>
<tr>
<td>DAY bday</td>
<td>image Julian day</td>
</tr>
<tr>
<td>LOCATE cor xcor</td>
<td>sets the coordinate type and the coordinate positions relative to the coordinate type</td>
</tr>
<tr>
<td>MAG imag emag</td>
<td>line and element magnification factor; positive values for blowup, negative values for blowdown</td>
</tr>
<tr>
<td>SIZE lines elems</td>
<td>number of image lines and data elements</td>
</tr>
<tr>
<td>SU name</td>
<td>stretching table name (default=no stretch)</td>
</tr>
<tr>
<td>POS pos</td>
<td>absolute position in the dataset</td>
</tr>
<tr>
<td>TIME btime etime</td>
<td>image time range</td>
</tr>
</tbody>
</table>

AUX YES—use this clause to insert the unit and scale factor of the image data into entries 58 and 59 of the directory block.

BAND—use this clause to identify a spectral band of image data. If you don’t specify BAND, the server determines a data-dependent band to use. Specify BAND ALL to return all spectral bands in the source image.

CAL—use this clause only when the source data type is TIRO. Use CAL QTRR for quick calibration.

DAY—use this clause to specify the day of the source. Using DAY implies that the image object position (POS) is not specified in the connection request. If POS is specified, the DAY clause is ignored.

LOCATE—use this clause to position the request spatially. LOCATE specifies a reference point from which the sector bounds are determined. To specify the reference point of the image sector, use one of three coordinate systems (e.g., satellite, array, earth) with one of two standard offsets (upper-left, center); for example, ECo Earth center, IU satellite upper-left. Following the reference point are two values which identify the absolute position of the reference point in the chosen coordinate system. For earth coordinates, the values are latitude and longitude; for satellite coordinates, the values are line and element; for array coordinates, the values are array row and column. If you don’t specify LOCATE, the default is a satellite upper-left (IU) reference point of the first scan line and pixel of the source image object.

MAG—use this clause to specify the resolution magnification factor for the line and element dimensions of the image object. Enter a negative integer for a blowdown; enter a positive integer for element replication (blowup). MCAGET performs all element duplication on the client. The application must reduce the size of the data request to allow for the blowup factor. You cannot store images with non-integer line/element resolutions in the McIDAS image object (AREA) data structure.

SIZE—use this clause to specify the line and element limits of the transaction. If you don’t specify SIZE, the sector limits default to 480 lines and 640 elements. If the request size exceeds the size of the image, it is trimmed to match the size of the image. Specify SIZE ALL to read the entire source image object.

SU—use this clause to specify the name of an image data stretching table. These tables are generated by the SU command.

POS—use this clause to specify the position of the image in the dataset. If POS is not specified, the server uses the most current (time relative) image in the dataset. Specify POS as either absolute or time relative. Numbers less than one imply time relative position, with zero being the most current image.

TIME—use this clause to specify a range of image start times identifying a specific image in a dataset. Using this clause implies that the image object position (POS) is not part of the connection request. If a POS clause does exist, the TIME clause is ignored.
The directory block returned by MCAGET is identical to the data structure returned by MCADRDR, except the data block contains 64 words instead of 65. The difference is the absence of the time relative pointer that occupies entry position 1 in the MCADRDR directory block. The structures parallel each other until entry 58, which specifies the units of the returned data elements. Entry 59 is the scale factor (powers of 10) applied to each of the data elements. All remaining entries are identical. The table below illustrates these differences.

<table>
<thead>
<tr>
<th>MCADRDR entry</th>
<th>MCAGET entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>time relative position of the image object</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>absolute position of the image object in the dataset</td>
</tr>
<tr>
<td>58</td>
<td>58</td>
<td>units of the returned data block elements</td>
</tr>
<tr>
<td>59</td>
<td>59</td>
<td>scale factor of the returned data block elements</td>
</tr>
<tr>
<td>60</td>
<td>64</td>
<td>number of comment cards</td>
</tr>
</tbody>
</table>

The MCAPFX function

If the line prefix is required to process the image elements, the application must fetch the prefix using the MCAPFX function. MCAPFX must be called immediately following the MCALIN call to ensure that the prefix array is the one associated with the data array.

Code segment showing an example of reading the line prefix.

```c
    c --- set up the ADDE transaction
      ... 100 continue
    c --- read the data block
      status = mcain(handle, data_buffer)
      if( status.lt.0 ) then
        call ede5('Data Read failed',0)
        return
    c --- get a line of data
      goto 100
    endif
    c --- Free the handle
      status = mcfre5( handle )
      ...}
```
Reading the navigation block

Applications rarely access the navigation block directly. In most cases, navigational operations are performed through a dedicated API. For those instances where the navigation block is read by the application, the access interface is the ARAGET subroutine. ARAGET is a generalized input routine that reads application-defined blocks from an AREA. Applications must define the location and length of the desired information in the AREA. The example below uses the ARAGET subroutine to read the navigation block from an AREA.

Code segment showing the AREA navigation block read.

```
c --- get the position of the Nav block from the area directory
    pos = areadir(35)
    if ( areadir(63), eq, 0 ) then
        length = pos - areadir(34)
    else
        length = pos - areadir(63)
    endif

    c --- read the Nav block
    if ( length gt 0 ) call araget(area,pos,length,nav_buffer)
```

The MCANAV function

The ADDE interface to the navigation block is through the MCANAV function. At any point after the connection is opened by MCAGET, the application may retrieve the navigation block using the handle returned by the preceding MCAGET call. The code segment below illustrates the use of the MCANAV function.

Code segment showing the ADDE image navigation block read.

```
c --- open a connection
    status = mcaget(dataset, nsort, sorts, unit, format, &
                   max_byte, mc_flag, directory, HANDLE)
    if ( status lt 0 ) return

    ... 

    c --- read the navigation block
    status = mcanav(HANDLE, nav_buffer)
    if ( status lt 0 ) then
        call edest('Navigation Block Read failed',0)
        return
    endif
```

Reading the calibration block

Applications use the ARAGET subroutine to read the calibration block from an AREA. ARAGET takes the position and length of the calibration block as arguments and returns the block in an array. Below is a sample code segment of a typical calibration block fetch.

Code segment showing the AREA calibration block read.

```
c --- get the position of the Cal block from the area directory
    pos = areadir(63)
    if ( pos.nn.0 ) then
        length = areadir(34) - pos
    endif

    c --- read the Cal block
    call araget(area,pos,length,cal_buffer)
```

The MCACAL function

The MCACAL function reads the calibration block of an ADDE image dataset. MCACAL can be called any time after the connection is opened by the MCAGET call. The handle returned by MCAGET is passed to MCACAL, which returns the associated calibration block.

Code segment showing the ADDE image calibration block read.

```
c --- open a connection
    status = mcaget(dataset, nsort, sorts, unit, format, &
                    max_byte, mc_flag, directory, HANDLE)
    if ( status lt 0 ) return

    ... 

    c --- read the calibration block
    status = mcacal(HANDLE, cal_buffer)
    if ( status lt 0 ) then
        call edest('Calibration Block Read failed',0)
        return
    endif
    ```
Reading the comment block

The function ICGET reads the comment block entries from an AREA. It requires no prior function calls to set up its environment. The application repeatedly calls ICGET until the return status indicates all entries are accessed. Entries are returned in an 80-byte integer array, which can be moved to a character array for output.

Code segment showing the AREA comment block read.

```
c --- read and print the comment block entries
100 continue
   if ( icget( area, comment_card ).eq.0 ) then
      call movec( comment_card, line_out )
      call apout( line_out )
      goto 100
   endif
   ...
```

The MCADRD and MCACRD functions

The ADDE image API has two interfaces for reading the comment block. The first is available through the MCADIR/MCADRD transaction interface. When a connection is opened with the MCADIR function, all MCADRD transactions return the entire comment block for an image object.

The second interface is available through connections opened by the MCAGET function. As with the navigation and calibration block read functions, MCACRD requires the handle returned by MCAGET be passed to the server, which returns the entire comment block to the application. The calling sequence is mandated. MCACRD can only be called after the MCALIN function call is exhausted. Below is an example of a comment block read using the MCAGET/MCALIN/MCACRD transaction sequence.

```
c --- open a connection
   status = mcaget( dataset, nsort, sorts, unit, format, &
                   nex_byte, msg_flag, directory, HANDLE )
   if ( status.lt.0 ) return

100 continue
   c --- read the data block
   status = mcadlin( handle, data_buffer )
   if ( status.lt.0 ) then
      call edest( 'Read failed', 0 )
      return
   endif
   c --- get a line of data
   if ( status.eq.0 ) then
      ...
      goto 100
   endif
   c --- read the comment block
   if ( mcacrd( handle, comment_buffer ).ne.0 ) then
      call edest( 'Read of Comment Block failed', 0 )
      return
   endif
   ...
```
**Function descriptions for ADDE image read**

Listed below, in alphabetical order, are the descriptions for each function used in reading ADDE image datasets.

**MCACAL**—reads the calibration block from an image object.

**Interface:**
integer function
mcal(integer handl, integer line(*))

**Input:**
handl identifies the data stream

**Input and output:**
one

**Output:**
line array containing calibration parameters

**Return values:**
0 success

**Remarks:**
This call may be made any time after MCAGET.

**MCACRD**—reads the comment block from an image object.

**Interface:**
integer function
mcacr(integer handl, integer line(*))

**Input:**
handl identifies the data stream

**Input and output:**
one

**Output:**
line array containing the comment block

**Return values:**
0 success

**Remarks:**
No other calls to other handles should come between MCALIN() and MCACRD(). This call should be made after all lines are read.

**MCADIR**—opens a connection to read the directory block from an image object.

**Interface:**
integer function
mcadir(character(*) name, integer snorts,
character(*) sorts(*), integer msgflag)

**Input:**
name dataset name or alias of the image object
snorts number of sort clauses
sorts array of sort clause text strings
msgflag error message flag
if =0, no messages, only return codes
if >0, error messages and return codes

**Input and output:**
one

**Output:**
one

**Return values:**
0 success
-53 failure, bad format of the TIME sort clause
-55 failure, request for non-image data
-other failure, return codes from server level routines

**Remarks:**
MCADIR() must be called prior to MCADR().

**MCADR**—reads the directory block from an image object.

**Interface:**
integer function
mcadr(integer data(*), integer cards(*))

**Input:**
none

**Input and output:**
one

**Output:**
data array containing the directory block
cards array containing the comment block

**Return values:**
0 success
1 success, no more directory blocks for the request

**Remarks:**
Word 1 in the directory block is the absolute position of the image object in the dataset. Word 2 is the relative position of the image object in the dataset. Words 3 to 63 are the same as words 2 to 64 from the image directory block.

Normally, word 1 of a valid image directory block is zero. Since only valid directory blocks are returned, word 1 is not sent. This word should be reset to zero if the directory block is written to disk as a part of an actual image object.

A return status of 1 means that all data satisfying the request was returned. No new data exists in the data array.

MCADR() must be called prior to MCADR().

**MCAFRE**—frees the handle and memory of a connection opened by MCAGET.

**Interface:**
integer function
mcafre(integer handl)

**Input:**
handl identifier of this data stream

**Input and output:**
one

**Output:**
one

**Return values:**
0 success

**MCAGET**—opens a connection to read the data block from an image object.

**Interface:**
integer function
mcaget(character name, integer snort, character sort(*), character unit, character form, integer msgbt, integer mgfla, integer iadtr(*), integer iadtr)

**Input:**
name dataset name or alias of the image object
snorts number of sort clauses
sorts array of sort clause text strings
unit calibration units
‘RAW’ = unprocessed instrument measurements
‘RAD’ = radiance

**Remarks:**
MCADIR() must be called prior to MCADR().
MCAGET (continued)

'TEMP' = temperature
'BRT' = visible brightness
'ALR' = albedo
' ' = as stored in image object
form individual pixel size
'1' = 1 byte/pixel
'2' = 2 bytes/pixel
'4' = 4 bytes/pixel
' ' = as stored in image object
maxbyt maximum size of the output array, in bytes
msgflag error message flag
If =0, no messages, only return codes
If >0, error messages and return codes

Input and output: none
Output: iadir array containing the directory block
(65 words)
handl identifies this data stream

Return values:
  0 success
  -35 failure, a bad sort clause encountered
  -36 failure, maxbyt is too small for lines to be returned
  -37 failure, illegal format, not 11, 12 or 14
  -39 failure, open file limit exceeded
  -other failure, return codes from server level routines

Remarks:
Word 1 in the directory block is the absolute position of the image object in the dataset. Word 2 is the relative position of the image object in the dataset. Words 3 to 65 are the same as words 2 to 64 in the image directory block.

Normally, word 1 of a valid image directory block is zero. Since only valid directory blocks are returned, word 1 is not sent. Reset this word to zero if the directory block is written to disk as a part of an actual image object.

Because more than one AREA data stream can be open simultaneously, the handle returned by MCAGET is used on calls like MCALIN to identify the data stream from which data is received. The handle is similar to an object descriptor but it is not an object descriptor, and cannot be used outside of this context. MCAGET() must be called prior to MCALIN().

MCALIN—reads the data portion of the current image line.
Interface: integer function
mcalin(integer handl, integer line(*))
Input: handl identifies the data stream
Input and output: none
Output: line line of data from the image object
Return values:
  0 success
  1 success, all data was read
  -38 failure, the handle was not legal
  -96 failure, communication has timed out
  -97 failure, communication with the server is lost

Remarks: Call this function once for each line of data in the AREA. A return status of 1 means that all data satisfying the request was returned and no new data exists in the line array. MCAGET() must be called prior to MCALIN().

MCANAV—reads the navigation block from an image object.
Interface: integer function
mcanav(integer handl, integer line(*))
Input: handl identifies the data stream
Input and output: none
Output: line array containing the navigation parameters
Return values:
  0 success

Remarks: This call may be made any time after MCAGET.

MCAPFX—reads the prefix portion of the current image line.
Interface: integer function
mcapfx(integer handl, integer line(*))
Input: handl identifies the data stream
Input and output: none
Output: line the prefix for the current line
Return values:
  0 success

Remarks: No calls to other handles should come between MCALIN() and MCAPFX().
Writing an ADDE image dataset

The code segment below illustrates the traditional sequence for writing a McIDAS AREA. The initial step is to define the essential entries of the directory block and write the block to the AREA with WRITED. The location of the navigation, calibration and data blocks are defined in the directory block in entries 35, 63 and 34 respectively. Once the directory block is written, the navigation and calibration blocks are filled and written to the AREA with ARAPUT. Next, the AREA is opened with OPNARA and the data is written to the AREA on a line-by-line basis with WRTARA. When the data block is completed, the buffers are flushed with CLSARA and the comment block is created and filled with ICPUT.

Code segment showing the AREA write.

c --- initialize the directory block
call zero(directory_block, 64)

c --- fill the essential directory block entries
directory_block(2) = 4 ; area version
directory_block(4) = 8 ; satellite number
directory_block(5) = 7 ; nominal start time of image
directory_block(6) = 2 ; starting image line number
directory_block(7) = 42 ; starting image element number
directory_block(9) = 32 ; number of lines of image data
directory_block(10) = 16 ; number of data elements per line
directory_block(11) = 10 ; number of bytes per data element
directory_block(12) = 8 ; line resolution
directory_block(13) = 4 ; element resolution
directory_block(14) = 32 ; number of bands
directory_block(15) = 4 ; band resolution
directory_block(25) = 12 ; memo field
directory_block(34) = 0 ; data offset
directory_block(35) = 0 ; byte offset to the data block
directory_block(36) = 0 ; byte offset to the nav block
directory_block(37) = 0 ; length of prefix data section
directory_block(38) = 0 ; length of prefix cal section
directory_block(39) = 0 ; length of prefix low section
directory_block(40) = 0 ; length of prefix hi section
directory_block(41) = 0 ; sensor source type

c --- write the directory block
call writd(area, directory_block)

c --- initialize the navigation block
call zero(navi_block, nav_size)

c --- fill the navigation block entries
 NOTE: "navigation_parms" is an array of navigation parameters
 which describes the geo-location of the elements of the area. 
do 10 i = 1,nav_size
 navi_block(i) = navigation_parms(i)
 10 continue

c --- write the navigation block to the area
call araput(area, nav_offset, nav_size*4, nav_array)

c --- initialize the calibration block
call zero(cal_block, cal_size)

c --- fill the calibration block entries
 NOTE: "calibration_parms" is an array of calibration
 parameters which transform the data elements to physical units.

The MCAPUT and MCAOUT functions

The process of writing an ADDE image object follows the same format as reading an ADDE image object. The application identifies a dataset, defines the sort condition list, and opens a connection with the server. The connection defines the transactions to perform before the transfer is successfully completed.

The request to open a connection is performed by the MCAPUT function. MCAPUT requires a valid dataset name, image object position, directory block navigation block and calibration block. MCAPUT does not return an object handle; therefore, only one ADDE image object can be written at a time.

The only valid sort clause defined for the ADDE image write interface is POS, which defines the location of the image object in the dataset. You must specify this clause or the request to open a connection will fail.
Once the connection is open, the server expects a specific number of bytes to be transferred. The number of bytes is defined by the entries in the directory block. Transferring too few or too many bytes results in an error. All data block write transactions are performed by the MCAOUT function. MCAOUT has only one argument, which is an array of image elements to write to the data block. MCAOUT is called as many times as necessary to transfer the desired number of bytes. MCAPUT must be called prior to MCAOUT.

The MCACOU function

The comment block can only be written after the last byte of the data block is transferred. The number of comment entries is defined during the connection phase of the transaction. Word 64 of the directory block holds the number of comment block entries for the image object. If this entry is nonzero, the number of 80-byte entries is transferred. The MCACOU function transfers the comment block. It has only one argument, which is an array holding the entire comment block.

The code segment below is a copy of the previous example re-coded to use the ADDE image object write API.

Code segment showing the ADDE image object write.

```c
# --- initialize the directory block
call zeros(directory_block, 64)

# --- create a comment card
# get day from the time card
# call gettime(time_card)
c: = ntime(time)
c: = ctime(c:)
c: = cdate(1:5)/' 'cdate(1:6)/' ' This is a comment c:
ncard = (1m_trin(comment) / 80) + 1

# --- fill the essential directory block entries
directory_block[2] = 4
! version
directory_block[3] = 66
! satellite number
directory_block[4] = 1day
directory_block[5] = start_line
! nominal start time of image
directory_block[6] = start_ale
! starting image element number
directory_block[7] = start_ale
! starting image element number
directory_block[8] = num_elements
directory_block[9] = num_bye
number of bytes
directory_block[10] = num_bytes
! line resolution
directory_block[12] = elem_res
! element resolution
directory_block[13] = num_hands
! number of hands
directory_block[14] = num_hands
! number of hands
directory_block[15] = band
! band code
call mov(mm, directory_block[25])
! mem field
directory_block[34] = data_offset
! byte offset to the data block
directory_block[35] = nav_offset
! byte offset to the nav block
directory_block[49] = doc_length
! length of prefix doc section
directory_block[50] = cal_length
! length of prefix cal section
directory_block[51] = b1w_length
! length of prefix b1w section
directory_block[52] = ln_stype
! sensor source type
directory_block[53] = ln_stype
! calibration type
directory_block[54] = cal_block
! data on the cal block
directory_block[64] = ncard
! number of comment cards
```

Code showing the ADDE image object write (continued).

```c
c --- initialize the navigation block
call zeros(nav_block, nav_size)

c --- fill the navigation block entries
! NOTE: *navigation_params* is an array of navigation parameters
! which describes the geo-location of the elements of the
! image object.
do 10 i = 1,nav_size
! nav_block[i] = navigation_params[i]
10 continue

c --- initialize the calibration block
call zeros(cal_block, cal_size)

c --- fill the calibration block entries
! NOTE: *calibration_params* is an array of calibration
! parameters which transform the data elements to physical
! units.
do 20 i = 1,cal_size
! cal_block[i] = calibration_params[i]
20 continue

c --- fill the sorts array
! nsorts = 1
! sorts(nsorts) = 'POS 'c:position'

c --- open a connection to write the image object
call mcanu( image, sorts, sorts, directory_block, nav_block,
& cal_block).ne.0 ) then
! call edest('Failed to initialize image ='/image,0 )
return endif

! --- loop to write image lines to the image object
do 100 line = 1,num_lines

c --- pack the data array
! NOTE: This assume a 4 byte to 1 byte compression of the data
call pack( num_elements, data_array, data_array)

c --- write a line of data to the image object
c! 'data_array' is a (num_lines) by (num_elements) array of data
elements each of which is (num_bytes) long. The elements
c! represent data for (band) from the sensor numbered (iss)
c! on (day) at (time).
! if( mcanu(data_array).ne.0 ) then
! call edest('Failed to write image line='line')
return endif

100 continue

c --- write the comment block
if( mcanu(comment).ne.0 ) then
! call edest('Failed to write comment block').0 )
return endif
```
Function descriptions for ADDE image write

Listed below, in alphabetical order, are the descriptions for each function used in writing ADDE image datasets.

**MCAPUT**—opens a connection to write the prefix and data blocks of an image object.

- **Interface:**
  - `integer function mcaput(char name, int sort, char sort(*), int imagdir(64), int nav(*), int call(*) )`

- **Input:**
  - `name` dataset name or alias of the image object
  - `sorts` number of sort clauses
  - `sort` array of sort clause text strings
  - `imagdir` directory block for the target image object
  - `nav` navigation block for the target image object
  - `call` calibration block for the target image object

- **Input and output:** none
- **Output:** none
- **Return values:** 0 success
- **Remarks:** Depending on the contents of imagdir, nav and/or cal arrays may be empty.

**MCAOUT**—writes the prefix and data portions of an image line.

- **Interface:**
  - `integer function mcaout(integer line(*) )`

- **Input:**
  - `line` array containing the prefix and data portions of the current line

- **Input and output:** none
- **Output:** none
- **Return values:** 0 success

**MCACOU**—writes the comment block to an image object.

- **Interface:**
  - `integer function mcacou(integer data(*) )`

- **Input:**
  - `data` array of comment card information

- **Input and output:** none
- **Output:** none
- **Return values:** 0 success
- **Remarks:** The array contains all comment cards, 80 bytes each. Call this function after all line data is transmitted. Do not call MCACOU if comment cards are not specified in the AREA directory entry.

Deleting an ADDE image dataset

McIDAS has no dedicated function to delete AREAs. The accepted mechanism is to construct a file name and then delete the file using LWD, the LW layer delete. The code segment below illustrates this technique.

**Code segment showing the AREA delete.**

```plaintext
c --- loop through area range do 10 area = beg_area, end_area
    call arnam( area, file_name )
    call lwd( file_name )
10 continue
```

**The MCAD function**

ADDE provides an interface for deleting image objects using the MCAD function, but only if you have write permission. Like MCAPUT, MCAD has only one valid sort clause, SUBSET, which you must specify during the connection phase of the transaction or the server request will fail.

The code segment below is an example of ADDE image object delete API. It deletes the image objects in the dataset that are at locations pos1 through pos2.

**Code segment showing ADDE image object delete.**

```plaintext
c --- construct sort clause
    nsort =1
    sorts(nsort) = 'SUBSET '/cfsi(pos1) '/'cfsi(pos2)
    call bequs( sorts(nsort) )

c --- delete image object
    if( mcad( dataset, nsort, sorts, negflg.1,ne.0 ) then
        call edest( 'Failed to delete image ','0')
    else
        call sdset( 'Image deleted',0)
    endif
```
The MCASORT function

The function MCASORT provides general translation of command line keyword parameters into equivalent MCAGET sort clauses. Any application level program may call MCASORT to retrieve the command line keywords and return them as MCAGET sort clauses. Below is a list of the ADDE image object access keywords and their equivalent sort clauses.

<table>
<thead>
<tr>
<th>Command line keyword</th>
<th>MCASORT translated sort clause</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAND=band</td>
<td>BAND band</td>
<td>only one spectral band in the clause</td>
</tr>
<tr>
<td>DAY=day day</td>
<td>DAY day day</td>
<td></td>
</tr>
<tr>
<td>LATLON= lat lon</td>
<td>LOCATE Eloc lat lon</td>
<td>if keyword PLACE is not specified, loc defaults to center (EC)</td>
</tr>
<tr>
<td>LINELE= line ele</td>
<td>LOCATE Iloc line ele</td>
<td>if keyword PLACE is not specified, loc defaults to center (IC)</td>
</tr>
<tr>
<td>MAG=mag emag</td>
<td>MAG mag emag</td>
<td></td>
</tr>
<tr>
<td>PLACE=loc</td>
<td>none</td>
<td>sets loc for the LATLON, LINELE and STATION keywords</td>
</tr>
<tr>
<td>RTIME=btme etime</td>
<td>TIME btme etime</td>
<td>RTIME overrides TIME</td>
</tr>
<tr>
<td>STATION= stn</td>
<td>LOCATE Eloc lat lon</td>
<td>if keyword PLACE is not specified, loc defaults to center (EC)</td>
</tr>
<tr>
<td>TIME=btme etime</td>
<td>TIME btme etime</td>
<td></td>
</tr>
</tbody>
</table>

The MCAGET function allows the application to specify the units and format of the data elements. Since these parameters are necessary to any data transaction, they are specified as separate parameters to the MCAGET function. Units may be any unit identifier valid for the image object type. A list of valid unit identifiers is available to an application through the MCADDR function by specifying the AUX YES sort clause. Use the format parameter to specify the bytes per data element in the return array. Valid formats are 11 (1 byte/element), 12 (2 bytes/element), and 14 (4 bytes/element).
Function description for MCASORT

MCASORT—gets the area sort parameters from the command line and adds them to the sort array for a future mcaget call.

**Interface:**
- integer function mcasort(integer nsorts, character(*) sort(*), integer parspos)

**Input:**
- parspos: position of the parameter within the keyword; used when information for more than one AREA is specified

**Input and output:**
- nsorts: number of sorts in the sort array passed in
- sort: array containing the sort conditions

**Output:**
- none

**Return values:**
- 0: success
- <0: error occurred

**Remarks:**
Nsorts and sort may contain data when mcasort is called; any sort conditions picked up from the command line are appended.

ADDE grid object I/O

Traditionally, McIDAS software that used information within grids was required to know the following:

- the grid file number containing the grid
- the grid number containing the grid of data

The ADDE grid software allows a user to access grids by the type of data within them. Although McIDAS grid data is still stored in grids, knowing the number of the grid and/or grid file where the data resides is not necessary, even though users can still access data that way. It is sufficient to know there is a grid in a dataset that can be retrieved. The position number of the dataset defines which grid file in that dataset will be accessed. A dataset can have many positions if it spans many grid files; it can also have one position if it consists of only one grid file.

**MCGGET, MCGPUT and MCGDIR**

These three routines facilitate the moving and listing of grid data in McIDAS:

- MCGGET requests grid(s) from a server and subsequently reads the grids matching the search conditions.
- MCGPUT writes grids to a dataset in grid form.
- MCGDIR reads information in grid file headers and grid headers.

MCGGET and MCGDIR search local or remote workstations for grids matching given search conditions. These conditions are called sorts and they are passed to the server in a sort string. As parameters are added to the sort string, the search is refined and, presumably, fewer grids are matched.

If the sort string is passed to MCGGET, the grid(s) matching the sort conditions and the corresponding grid headers are sent to the user. If the sort string is sent to MCGDIR, the grid file header and the grid headers of the grids, but not the grids themselves, are sent to the user. Entries are provided to read the data sent: MCGGRIDF and MCGGRIDC for MCGGET; MCGFDRD and MCGFDRD for MCGDIR.
MCGSORT

A utility subroutine, MCGSORT, gets the sort conditions related to information in
the grid header. These include parameter (PARM), level (LEV), day and time
(DAY/TIME), forecast hour (FHOUR), forecast day and time (FDAY/TIME), and
source of the data (SRC). MCGSORT can also pick up the sort condition GRID,
the grid number itself, and bypass all other sort conditions. The sort conditions are
read from the command line with argument fetchers. Note that users can develop
other sort clauses; for example, to distinguish between otherwise identical grids
with different projections, or numbers of rows/columns, or standard latitudes, or
units of vertical levels, or grid resolution. If these sort clauses are passed to the
server, they will do nothing unless the MCGGET and/or MCGDIR programs are
altered to search for the new sort parameters.

For some applications, you may want to limit the number of grids returned from a
search to one. Therefore, the function MCGSORT contains a flag to denote when
multiple finds of a search condition are allowed.

The valid SORT clauses for the server are described below.

LEV—use this clause to limit the search to particular levels. Both integral values,
(850, 700, 300) and character values (SFC, MSL, MAXW) are acceptable. If LEV
is not specified, any level meeting the other search conditions is valid. If rep_flag
is greater than zero, more than one level may be entered. The server searches word
10 of the grid header for a match.

PARM—use this clause to limit the search to particular parameters. Valid entries
are anything that appears in the grid header at word seven (T, U, V, THA, etc.) and
a few derived quantities (STREAML, WINDB, WINDV). If rep_flag is greater than
zero, more than one parameter may be entered. The server searches word seven
of the grid header for a match.

DAY—use this clause to limit the search to a specified day, in the format
YYYYDDD or YYDDD. If the data being searched is model data, then DAY is the
initial time of the model run. If rep_flag is greater than zero, more than one DAY
may be specified. The server searches word four of the grid header for a match.

TIME—use this clause to limit the search to a specific time, in the format
HHMMSS. If the data being searched is model data, then TIME is the initial time
of the model run, typically 0 Z or 12 Z for operational models. If rep_flag is greater
than zero, more than one time may be specified. The server searches word five of
the grid header for a match.

DRANGE—use this clause to specify a range of days to search. DRANGE is valid
only if rep_flag is greater than zero, and DAY is not specified. It allows the user to
input a beginning day, ending day, and an increment. Thus, every seven days
between 1995008 and 1995036 could be selected by entering DRANGE 1995008
1995036 7. The server searches word four of the grid header for matches.

TRANGE—use this clause to specify a range of times to search. TRANGE is valid
only if rep_flag is greater than zero and TIME is not specified. It allows the user to
input a beginning time, ending time, and an increment. For example, every hour
between 13 and 21 Z can be specified by entering TRANGE 130000 210000 10000.
The server searches word five of the grid header for matches.

SRC—use this clause to specify the source of the data. Valid entries include NMG,
ETA, MRF, and ECMF, which are models available from world meteorological
centers, and MDX and SSF, which are SSEC-generated grids. If rep_flag is greater
than zero, more than one SRC may be entered. The server searches word 33 of
the grid header for a match.

FHOUR—use this clause to specify the valid forecast hour for a model run.
FHOUR 360000, for example, tells the server to send back 36-hour forecasts.
MCGSORT allows more than one specification of FHOUR always. The server
searches word six of the grid header for a match.

FDAY—use this clause (only if FHOUR is not specified) to specify the day on
which a forecast is valid. The server uses words four through six to derive the day
on which a gridded forecast field is valid. If rep_flag is greater than zero, more than
one FDAY may be entered.

FTIME—use this clause (only if FHOUR is not specified) to specify the time at
which a forecast is valid. The server uses words four through six to derive the time
at which a gridded forecast field is valid. If rep_flag is greater than zero, more than
one FTIME may be entered.

FRANGE—use this clause to specify a range of forecast hours. Forecasts every 24
hours, for example, out to ten days is specified as FRANGE 0 240000 240000.
The user can input a beginning FHOUR, ending FHOUR, and an increment.

GRID—use this sort clause, which supersedes all the clauses above, if you know
the grid number of the data in question. If rep_flag is greater than zero, you can
enter a range of GRID numbers. GRID 7 20, for example, requests that grids 7
through 20 inclusive, be returned to the application. Because a dataset can contain
many grid files, each of which can contain grid numbers 7 through 20, use an explicit
data set position number with the GRID search keyword.
Reading an ADDE grid dataset

The example below, from the McIDAS program gridisp.pgm, displays grid information on a graphics frame. It will display all 0-hour, 850 mb temperature fields from the NGM; the dataset encompasses all grid files.

**Code to set up the call to the server.**

```c
C-----external functions

c C integer ngspec     # initiate server request
   integer nggrid     # read grid from pipe
   integer ngsort    # build sort table from command line entries

C-----local variables

c C character*100 name     # dataset name
   character*100 sorts(100)  # sort conditions array for server
   integer gridid(4)     # grid containing data to be manipulated
   integer idgridhead(4) # grid header
   integer maxgrid      # max grid size for server transaction
   integer nsorts       # number of sort conditions
   integer numbytes     # number of bytes shipped for sorting.
   integer ngridsize    # number of grids returned from server
   name='McIDAS Grid 1'
   nsorts = 0
   if (ngsort(name,sorts,rep_flag) .lt. 0) then
      print 'Error in picking up sort strings from command line!'
   end if

Assume that ngsort picks up the following parameters...

| sorts(1)    | 'PARM 1'     |
| sorts(2)    | 'LLEV 850'   |
| sorts(3)    | 'FROG 0'     |
| sorts(4)    | 'SRC NGM'    |

write(line,900)
call edout(line,0)
goto 2000 if (nggrid) goto 2000
end if

900 format(' Error in picking up sort strings from command line!'

nsorts = nsorts + 1
sorts(nsorts) = 'NUM ALL'
```

The sort conditions limit what the server returns. For example, if NUM ALL is omitted from the sort string, only the default first match is returned from the server. If SRC NGM is omitted, all 0-hour 850-mb temperature fields from any source, model or otherwise, are sent from the server. If LLEV 850 is omitted, all levels are sent.

If the user knows the grid number to access, the search conditions can be replaced by sorts(1)="GRID 7", and grid 7 from the dataset is sent from the server. To avoid a conflict in datasets spanning grid files, use an explicit naming of the dataset position when using the GRID sort. To tell the server to look in just one position, include POS (integer) in the sort clauses. If no position is specified, all are searched.

Transferring grids from the server to the applications is started with a call to MGCGGET, as shown below.

**Code showing the call to MGCGGET.**

```c
if (mgcget(name,nsorts,sorts,' ','16',4*maxgr,numfig,
          & numgrid,numbyte), .lt. 0) goto 2000 if <0, error occurred
```

where:

- `name` is the ADDE dataset name
- `nsorts` is the number of sort conditions in the sorts array
- `sorts` tells the server to send the grids with native units
- `ID` specifies that integers be sent
- `4*maxgr` is the maximum number of bytes that can be sent
- `numfig` tells MGCGGET to print error messages if numfig>1

Numgrids and numbytes are returned from the function. If numgrids is greater than zero, the server found grids matching the sort conditions, and they are sent to the application.

Reading the grids is accomplished with calls to MCGGRIDF (Fortran) or MCGGRIDC (C), as shown below.

**Code showing the call to MCGGRIDF.**

```c
do 100 i = 1,numgrid
   if (mggrid(idgrid,idgridhead) .lt. 0) goto 2000
   if <0 for errors
C      now manipulate the grid you got or get then all and manipulate then
C...
100 continue
```

If this search was done the old way with IGET, the user would have to know exactly which grid file and grid contained the desired data, so that an IOPEN/IGET could be called. This can also be achieved with these new methods by specifying the correct dataset position number and grid number.
Writing an ADDE grid dataset

MC Gupta is a routine that writes data to a grid file. The sort string passed to it defines the grid file information, which is limited to the following:

<table>
<thead>
<tr>
<th>Sort clause format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRID</td>
<td>grid number at which to start writing (default=next available)</td>
</tr>
<tr>
<td>POS</td>
<td>dataset position number (grid file) into which the grid is written</td>
</tr>
<tr>
<td>DEL YES or DEL NO</td>
<td>deletes the grid file before writing to it</td>
</tr>
<tr>
<td>MAX</td>
<td>maximum number of grids that can be written to a grid file; valid only when a grid file is created</td>
</tr>
<tr>
<td>LABEL</td>
<td>character string used to annotate a grid file directory; valid only when a grid file is created; 32 characters maximum number of grids to write (default=1)</td>
</tr>
<tr>
<td>NUM</td>
<td></td>
</tr>
</tbody>
</table>

The example below demonstrates its use. Assume the grids were retrieved by the MC GET example above, and the output is done after the manipulation of the grids, still within loop 100. The call to MC Gupta must take place after the call to MC GET and before loop 100 begins, because you must pass to the server the number of grids and bytes being written. Assume you want to start writing at grid 100 in position 6 of a dataset; the grid file will be deleted before you write to it, and you will set the maximum number of grids in the grid file to 1200, with a title of Test Grid for ADDE. The order of the sort strings is irrelevant.

Code defining the sort conditions.

```c
putsorts(1) = 'GRID 100'
putsorts(2) = 'POS 6'
putsorts(3) = 'DEL YES'
putsorts(4) = 'MAX 1200'
putsorts(5) = 'LABEL Test Grid for ADDE'
putsorts(6) = 'NUM '/cfr(numgrid)
nsorts = 6
msgflag = 1 ! msgflag = 1 turns on error msg
```

The sixth sort string, numgrid, was returned by MC GET and is not explicitly known. The call to MC Gupta opens up the pipe for the writing a set of grids to one position in a dataset; that is, to one grid file. If more than one position will be written to, you must call MC Gupta more than once. However, only one MC Gupta call can be open at a time. The call to open the pipe is shown below.

Code to call MC Gupta.

```c
if (mcgput(dataset_name, newsorts, putsorts, msgflag, numbytes) & .lt. 0) then
   stop : error if function returns < 0
endif
```

Numbytes is the number of bytes to be written. Use the numbytes from the previous MC GET example as input into MC Gupta. Otherwise, the user must compute this value.

If the call returns successfully, the pipe to write to a dataset is open, and calls to write data can be sent. These are achieved using the MCGOUTF (FORTRAN) or MCGOUTC (C) function. They have the form shown below, and are placed in loop 100 above.

Code to write data to a dataset.

```c
if (mcgoutf(grid, grid_header) .lt. 0) then
call edest('Failed in write of grid to dataset',0)
else
call ddest('Wrote grid and grid_header to dataset',0)
endif
```

Before you write data to the dataset, determine if it has enough available space for the data by using the directory listing program MC GD IR.
Listing an ADDE grid dataset

The function MCDIR moves grid headers and grid file headers from a server to an application. It is normally used to list the contents of a grid dataset. Its structure is shown below.

Code showing the call to MCDIR.

```c
if (mcdir(dataset_name, nsorts, sortstrings, msgflag) .lt. 0)
   return ! if .lt. 0 an error has occurred
```

If the function returns zero, MCDIR found grids matching the search parameters specified in the sort string. A pipe is open to the application, which is read using the functions MCGFDRD (Grid File Directory Read) and MCGHDR (Grid Directory Read). Because the specified dataset can span numerous grid files, whose headers are read by MCGFDRD, each of which can contain numerous grids, whose headers are read by MCGHDR, reading of the pipe is accomplished in a doubly nested loop as shown on the next page.

Code showing the listing of an ADDE grid dataset.

```c
if (mcdir (as above) .eq. 0) then ! successful call to mcdir
  status1 = mcgfdrd(grid_file_header)
  if (status1 .eq. 0) then ! success, header
    ! returned to application
  
  do something with grid file directory
  
  status2 = mcghdr(grid_header)
  if (status2 .eq. 0) then ! success, header
    ! returned to application.
  
  do something with grid header

  goto 150     ! get next grid header from pipe
  else (status2 .eq. 1) then

  ---if status2 is 1, there are no more grid headers associated with
  ---the grid file directory, get the next grid file directory

  goto 100

  else ! status2 is not 0 or 1

  call edest('Error occurred in mcghdr')
  stop

  endif
  else (status1 .eq. 1) then

  ---if status1 is 1, there are no more grid file directories to read, so gracefully exit

  call ddest('Done reading grid file directories from dataset',0)
  else ! status1 is not 1 or 0

  call edest('Error occurred in mcgfdrd',0)
  stop
  endif
endif
```
Function descriptions for grids

Listed below, in alphabetical order, are the function descriptions for reading, writing and listing ADDE grid datasets.

**MCGDIR**—returns the grid directory for a grid file.

**Interface:** integer function

mcgdir(character(*) name, integer nsort, character(*) sort(*), integer msgflag)

**Input:**
- name: name or nickname of the data object
- nsort: number of sort clauses
- sort: strings to modify the request
- msgflag: emit messages; 0 indicates no message

**Input and Output:** none

**Output:** none

**Return values:**
- 0: success
- < 0: failure

**Remarks:**
MCGDIR can be called to set up for calls to read the grid file headers and grid headers.

**MCGDRD**—reads grid directories.

**Interface:** integer function

mcgdrd(integer grid_dir)

**Input:** none

**Input and Output:** none

**Output:** grid_dir grid directory

**Return values:**
- 0: success
- 1: all the grid directories for the current file were read

**Remarks:**
MCGDRD returns grid directories requested in the MCGDIR function.

**MCGFDRD**—reads the grid file header.

**Interface:** integer function

mcgfdrd(integer file_header(*))

**Input:** none

**Input and Output:** none

**Output:** file_header grid file header

**Return values:**
- 0: success
- 1: end of the file directories to read

**Remarks:**
This function returns the grid directory. A call to MCGDIR must be made before MCGFDRD can be called.

**MCGGET**—requests a single grid from a grid server.

**Interface:** integer function

mcgget(string name, int nsort, string sort(*),
string unit, string form, int maxbyt, int msgflag
int numgrids, int numbytes)

**Input:**
- name: DDE true name, nickname, or grid file number
- nsort: number of sort clauses
- sort: array of clauses defining the request
- unit: requested units; blank implies native as stored
- form: currently 14 or F4
- maxbyt: largest grid size the user can handle, in bytes
- msgflag: 0 does not output error messages

**Input and Output:** none

**Output:**
- numgrids: number of grids returned
- numbytes: number of bytes returned

**Return values:**
- 0: success
- < 0: failure

**Remarks:**
The sort array contains one or more of the following clauses, which describe the subset of the data to retrieve:
- GRID: grid number
- PARM: parameter
- DAY: day
- DRANGE: day range
- TIME: time
- TRANGE: time range
- LEV: level
- F.HOUR: forecast hour
- FDAY: forecast day
- FRANGE: forecast day range
- FTIME: forecast time
- SRC: grid model
- POS: grid file position in the dataset
- NUM: YES for all grids; # for finite number requested

**MCGOUTC**—writes a C ordered grid.

**Interface:** integer function

mcgoutc(integer grid(*), integer header(64))

**Input:**
- grid: grid to write
- header: grid header

**Input and Output:** none

**Output:** none

**Return values:**
- 0: success
- < 0: failure

**Remarks:**
This writes a 2D array in C ordering, for the convenience of applications written in C.
**MCGOUTF**—outputs a grid in Fortran format.

**Interface:**
integer function mcgoutf(integer grid(*), integer header(64))

**Input:**
grid  grid to write
header  grid header

**Input and Output:**
none

**Output:**
none

**Return values:**
0  success
<0  failure

**Remarks:**
This writes a 2D array in the Fortran ordering for the convenience of applications written in Fortran.

**MCGPUT**—requests to write grids to a grid file.

**Interface:**
integer function mcgput(string name, int nsort, string sort(*), int msgflag, int numbytes)

**Input:**
name  DDE true name, nickname, or grid file number
nsort  number of sort clauses
sort  array of clauses defining the request
msgflag  0 does not output error messages
numbytes  number of bytes output

**Input and Output:**
none

**Output:**
none

**Return values:**
0  success
<0  failure

**Remarks:**
The name in MCGPUT is the dataset of the grid file to which data will be written. The sort clauses determine the structure of the grid file.Clauses are limited to:

**GRID**
grid number at which to start writing

**MAX**
maximum number of grids that can be written to a grid file; valid only if a grid file is being deleted

**APPEND**
YES appends the grids to the grid file

**DEL**
YES/NO deletes the grid file before writing to it

**NUM**
number of grids to write

**LABEL**
character string for the grid file label if a grid file is created

**MCGRIDC**—receives the grid requested by MCGGET (C ordered).

**Interface:**
integer function mcgridc(integer grid(*), integer header(64))

**Input:**
none

**Input and Output:**
grid  array to receive the grid
header  array to receive the grid header

**Output:**
none

**Return values:**
0  success
<0  failure

**Remarks:**
This returns a 2D array in C ordering. Missing values in fixed point are returned as HEX80. Missing values in floating point are returned as greater than 1.0e30. The values of the grid are converted to the units specified in the MCGGET call. If a grid is derived, all component grids are read, and the resultant grid is computed. This function is an entry point in MCGGET.

**MCGRIDF**—receives the grid requested by MCGGET (Fortran ordered).

**Interface:**
integer function mcgridf(integer grid(*), integer header(64))

**Input:**
none

**Input and Output:**
grid  array to receive the grid
header  array to receive the grid header

**Output:**
none

**Return values:**
0  success
<0  failure

**Remarks:**
This returns a 2D array in the Fortran ordering. Missing values in fixed point are returned as HEX80. Missing values in floating point are returned as greater than 1.0e30. The values of the grid are converted to the units specified in the MCGGET call. If a grid is derived, all component grids are read, and the resultant grid is computed. This function is an entry point in MCGGET.
MCGSORT—gets grid sort parameters from the command line and adds them to the sort array for future MCGGET or MCGDIR calls.

Interface: integer function mcsort(integer nsorts, character*(* sorts(*)), rep_flag)

Input: none
Input and Output: nsorts number of sorts in the sorts array passed in
sorts array containing the sort conditions
rep_flag allows multiple gets (0 no, 1 yes)

Output: none
Return values: 0 success
< 0 failure

Remarks: Nsorts and sorts may contain data when MCGSORT is called. Any sorts conditions picked up from the command line will be appended. rep_flag set to one enables multiple gets of levels, parameters, models, times, and days. Multiple gets of FHOUR are always allowed.

Categories: grid

Applications Programming Interface

This chapter lists the McIDAS Application Programming Interface routines by category and describes each routine in detail.

Contents
Categories of API routines .................................................. 5-3
Descriptions of API routines ............................................. 5-25
Categories of API routines

Use this section to help you find the API routine you need as you build applications. The routines have been divided into categories based on their functionality. On the following pages routines for each category are listed. Use the brief description of the routine to determine if it is the one you need. Find more information on each routine in the next section, Descriptions of API routines. Some routines are cross referenced. For example, if a routine draws graphics based on a grid, it is listed as both a graphics-related and grid-related routine. The types of routines are listed below. If the category name is different, it is given in parentheses.

- Calibration
- Converter
- Day and Time (day/time)
- Display
- Event
- File
- Graphic
- Grid
- Image
- Ingestor and Decoder (ingest/decode)
- Meteorology and Science (met/science)
- Navigation
- Point Source (pt_src)
- System Configuration (sys_config)
- System
- Text
- User Interface (user_interface)
- Utility
Calibration-related API routines

- areopt - sets the options and calibration for reading a McDAS area
- fixchr - handles byte swapping for nav and cal arrays
- npxel - expands/packs data values
- npxelb - expands/packs data values with a lookup table

Converter-related API routines

- mcpmap - Parse the given text into arg-fetching structure.
- mdtrndbl - Convert given numeric token to double type format.
- mdtrndhr - Convert given time token to double fractional hours hh..ffff
- mdtrndl - Convert given token to double fractional lat/lon ddd..ffff.
- mdtrndhex - Convert given hexadecimal token to integer type format.
- mdtrnths - Convert given time token to integer hours, minutes and seconds.
- mdtrnthr - Convert given time token to integer time format hhmmss.
- mdtrnths - Convert given date token to integer date format dddd.
- mcname - create a filename from a McDAS area number
- cfd - convert a double precision number, in D format, to a character*12 with leading blanks
- cff - convert a double precision number, in F format, to a character*12 with leading blanks
- cfi - convert a integer number, in I format, to a character*12 with leading blanks
- cfj - convert a integer number, in I format, to a character*12 with leading zero
- cfr - convert a integer to a character*12 with trailing blanks
- cfs - convert an integer number to a character*12 hexadecimal with leading blanks
- czf - convert a integer number to a character*12 with leading zeros
- flalo - short description of purpose/use/etc
- ftime - convert integer format time to real format
- incday - increment/decrement the date
- mcpmap - Parse the given text into arg-fetching structure.
- mcpcom - parse a list of comment cards for valid calibration keys
- mcpnav - parse a list of comment cards for valid navigation information
- mdtrndbl - Convert given numeric token to double type format.
- mdtrndhr - Convert given time token to double fractional hours hh..ffff
- mdtrndl - Convert given token to double fractional lat/lon ddd..ffff.
- mdtrndhex - Convert given hexadecimal token to integer type format.
- mdtrnths - Convert given time token to integer hours, minutes and seconds.
- mdtrnthr - Convert given time token to integer time format hhmmss.
- mdtrnths - Convert given date token to integer date format dddd.
- mcname - create a filename from a MD file number
- mdavc - returns the real-time MD file number for a specified data type and date
- tvaat - converts a tv line and element to a image line and element
### Day and time-related API routines

- **Moargdhhr**: Fetch an argument in double fractional hours format hh.ffff.
- **Moargdhhr**: Fetch an argument in integer type time format hhmms.
- **MoargiYd**: Fetch an argument in integer type date format yyyydd.
- **Mcmm’dhr**: Fetch a program argument in fractional hours format hh.ffff.
- **Mcmm’dhr**: Prints diagnostics to device for standard errors.
- **Mcmm’dhr**: Fetch a program argument in integer type time format hhmms.
- **Mcmm’dhr**: Prints diagnostics to device for standard errors.
- **Mohtrohms** - Convert double hours to hours, minutes and seconds.
- **Mohtrohms** - Convert double hours to integer hours.
- **Mohtrohms** - Convert time in hours, minutes and seconds to integer.
- **Mohtrohms** - Convert given time token to double fractional hours hh.ffff
- **Mohtrohms** - Convert given time to integer hours, minutes and seconds.
- **Mohtrohms** - Convert given time token to integer time format hhmms.
- **Mohtrohms** - Convert given date token to integer date format yyyydd.
- **Mohtrohms** - Convert integer format time to real format
- **Mohtrohms** - Increment/decrement the date
- **Mohtrohms** - Fetch an argument in double fractional hours format hh.ffff.
- **Mohtrohms** - Fetch an argument in integer type time format hhmms.
- **Mohtrohms** - Fetch an argument in integer type date format yyyydd.
- **Mohtrohms** - Fetch a program argument in fractional hours format hh.ffff.
- **Mohtrohms** - Prints diagnostics to device for standard errors.
- **Mohtrohms** - Fetch a program argument in integer type time format hhmms.
- **Mohtrohms** - Prints diagnostics to device for standard errors.
- **Mohtrohms** - Fetch a program argument in integer type date format yyyydd.
- **Mohtrohms** - Prints diagnostics to device for standard errors.
- **Mcstrrodhhr**: Convert given time token to double fractional hours hh.ffff
- **Mcstrrohms** - Convert given time to integer hours, minutes and seconds.
- **Mcstrrohms** - Convert given time token to integer time format hhmms.
- **Mcstrrohms** - Convert given date token to integer date format yyyydd.

### Display-related API routines

- **dtline**: Display of buf to iframe at iline
- **erest**: put enhancement on a frame
- **fsset**: Set font to given font name abbreviation.
- **fsiize**: obtains the number of lines and elements for a frame
- **getfrm**: obtains the frame directory entry
- **putfrm**: writes the frame directory to a file
- **tvline**: recommended line and element limits for a frame when drawing graphics

---

*Applications Programming Interface*  
1-6  
*McIDAS Programmer's Manual*  
Preliminary Issue 10/95

*McIDAS Programmer's Manual*  
Preliminary Issue 10/95  
*Applications Programming Interface*  
3-7
Event-related API routines

Currently, no event-related API routines exist.

File-related API routines

open - opens and returns a file handle
fread - reads a file
fseek - seeks in a file
fwrite - writes to a file
fclose - closes a file

lseek - seeks in a file
write - writes to a file

M_read - reads a file
M_remove - removes a file
M_write - writes to a file
M_close - closes a McIDAS area
M_copyp - copies audit trail cards from one McIDAS area to another
M_getg - gets a grid from a grid file
M_getname - creates the grid filename from the grid file number
M_read - reads a file, using byte addressing
M_write - writes a file, using byte addressing
M_copys - copies one file to another
M_remove - removes a file
M_lseek - seeks in a file
M_write - writes to a file
M_lseekw - seeks in a file
M_read - reads a file, using word addressing
M_write - writes to a file, using word addressing
M_copysf - returns the first grid number that matches the sorting conditions
M_read - reads a file
M_write - writes to a file

Mcpathname - determines the system path name of a McIDAS file

Currently, no file-related API routines exist.
Graphic-related API routines

Currently, no graphic-related API routines exist.

Grid-related API routines

igcurf - sets the current grid file number
igget - get a grid from a grid file
igmax - returns the maximum number of grids the grid file can hold
igname - create the grid filename from the grid file number
igput - write a grid to a McIDAS grid file
mcfndgrd - returns the first grid number that matches the sorting conditions
mggrid - request a single grid from a grid server
mgslast - returns the number of the last grid in a grid file
mgro2 - receive grid & grid header in c-ordering
**Image-related API routines**

- **MCALIN** - receive a single line of data from a remote server.
- **arabox** - obtains a rectangular array of data from a McIDAS area file.
- **araget** - obtains bytes from a McIDAS area file.
- **arename** - creates a filename from a McIDAS area number.
- **araopt** - sets the options and calibration for reading a McIDAS area.
- **araput** - writes area parameters into an area file.
- **clda** - closes a McIDAS area.
- **cppc** - copies audit trail cards from one McIDAS area to another.
- **dfline** - display linebuf if frame at line.
- **erest** - put an enhancement on a frame.
- **fixchr** - handles byte swapping for nav and cal arrays.
- **iget** - obtains an audit trail card from a McIDAS area.
- **ifband** - check if the specified band exists in a McIDAS area.
- **mcacal** - reads the calibration parameters from the current AREA.
- **mcacrd** - reads the comment cards from the AREA.
- **mcadel** - delete image files in a ADDS object.
- **mcaen** - return McIDAS AREAS directory structure from image objects.
- **mcadrl** - return one AREA directory and comment cards.
- **mcget** - start the transaction of receiving AREA data.
- **mcname** - construct image file data set name and position from frame directory entry.
- **mcnav** - reads the navigation from the current AREA.
- **mcnum** - return number of images in a dataset.
- **mcpfx** - reads the prefix from the current line.
- **mcdnum** - return number of positions in a dataset name.
- **mcpar** - parse a list of comment cards for valid calibration keys.
- **mcnav** - parse a list of comment cards for valid navigation information.
- **mcblob** - samples/moves blocks of bytes from one array to another array with both source and destination byte offsets.
- **mvblist** - samples/moves bytes from one array to another array with both source and destination byte offsets.
- **mpixel** - expands/packs data values.
- **mvxat** - converts a tv line and element to a image line and element.

**Ingestor and decoder-related API routines**

Currently, no ingestor and decoder-related API routines exist.
Meteorology and science-related API routines

Currently, no meteorology and science-related API routines exist.

Navigation-related API routines

Mcdargd1 - Fetch argument in double fractional lat/lon format ddd.fffffff.
Mcdargg1 - Fetch an argument in integer type lat/lon format dddmmmss.
Mcdmdd11 - Fetch a program argument in fractional lat/lon format dd.fffffff.
Mcdmdd1 - Prints diagnostics to edest device for standard errors.
Mcdmddll - Fetch a program argument in integer type lat/lon form dd:dd:dd.
Mcdmntod1 - Prints diagnostics to edest device for standard errors.
Mcdntntod1 - Convert given token to double fractional lat/lon ddd.fffffff.
Mcdntntoll - Convert given lat/lon token to integer type format dddmmmss.
Flachr - handles byte swapping for new and cal arrays
Flatio - short description of purpose/uses/etc
Mcdargd1 - Fetch argument in double fractional lat/lon format ddd.fffffff.
Mcdargg1 - Fetch an argument in integer type lat/lon format dddmmmss.
Mcdmdd11 - Fetch a program argument in fractional lat/lon format dd.fffffff.
Mcdmdd1 - Prints diagnostics to edest device for standard errors.
Mcdmddll - Fetch a program argument in integer type lat/lon form dd:dd:dd.
Mcdntntod1 - Convert given token to double fractional lat/lon ddd.fffffff.
Mcdntntoll - Convert given lat/lon token to integer type format dddmmmss.
treat - converts a tv line and element to a image line and element
Point source-related API routines

ndclos - closes an ND file
ndcode - returns the missing data code for the ND file
ndrmt - sets the current ND file
ndinfo - returns the ND header
ndname - creates a filename from a ND file number
ndopen - opens a ND file
ndsvc - returns the real-time ND file number for a specified data type and date

System configuration-related API routines

ndev2uc - Convert given DEV= character value to its numeric UC value.
nc1uc - Lookup user common for given index.
nc2ucdev - Update user common with the given DEV= arguments.
nc3ucdev - Convert given DEV= numeric UC value to its character value.
nftsetc - Set font to given font name abbreviation.
nuc - Lookup user common for given index.
nc4ucdev - Convert given DEV= numeric UC value to its character value.
System-related API routines

- `mcatget` - get an attribute string given a key string
- `mcatrnt` - gets the next attribute pair from the list
- `mcatrput` - puts an attribute pair (key-value) to a file
- `mcluc` - Lookup user common for given index.
- `mcisleep` - sleep for a given number of milliseconds
- `abort` - terminates the execution of a routine
- `iniset` - Set font to given font name abbreviation.
- `luc` - Lookup user common for given index.
- `main` - Set up state before calling users main0.
- `mcadr` - returns one AREA directory and comment cards
- `mcatget` - get an attribute string given a key string
- `mcatrnt` - gets the next attribute pair from the list
- `mcatrput` - puts an attribute pair (key-value) to a file
- `mcisleep` - sleep for a given number of milliseconds
- `prese` - Pre-scanning of command line, doing what can be done.
- `sleep` - sleep for a given number of milliseconds

Currently, no text-related API routines exist.
User interface-related API routines

**Mocrgmd** - Build and return a McIDAS command line for the given handle.

**Mocrgdb** - Fetch an argument in double type format.

**Mocrgdh** - Fetch an argument in double fractional hours format hh..fffff.

**Mocrgdhp** - Get keyword parameter in double precision form.

**Mocrgdhp** - Display parsed arg-fetching to McIDAS debug destination.

**Mocrgfree** - Free parsed arg-fetching structure for the given handle.

**Mocrgih** - Fetch an argument in integer type time format hhmmss.

**Mocrgi** - Fetch an argument in integer type format.

**Mocrgip** - Fetch an argument in integer type date format yyyydd.

**Mocrgkey** - Validate defined and command line keywords.

**Mocrgnam** - Fetch all keyword names within parsed arg-fetching text.

**Mocrgpar** - Return # args for given keyword in parsed arg-fetching text.

**Mocrgpar** - Parse the given text into arg-fetching structure.

**Mocrgqu** - Fetch the quote field string argument.

**Mocrgc** - Fetch an argument in character form.

**Mccc** - Fetch the current McIDAS command line.

**Moccdscl** - Fetch a program command line argument in double type format.

**Mccc** - Fetch an argument in double type format.

**Mccddhr** - Fetch an argument in double fractional hours format hh..fffff.

**Mccddh** - Display parsed arg-fetching to McIDAS debug destination.

**Mccdfree** - Free parsed arg-fetching structure for the given handle.

**Mccg** - Fetch an argument in integer type time format hhmmss.

**Mccg** - Fetch an argument in integer type format.

**Mccgip** - Fetch an argument in integer type date format yyyydd.

**Mccgkey** - Validate defined and command line keywords. Edest diagnostics.

**Mccdnam** - Fetch all keyword names occurring in the command line.

**Mccdr** - Return # values associated with given command line keyword.

**Mccdqu** - Fetch the quote field string argument.

**Mccdstr** - Fetch a program command line argument in character form.

**Mccddhr** - Fetch a program command line argument in double type format.

**Mccddhr** - Prints diagnostics to edest device for standard errors.

**Mccddhr** - Display parsed arg-fetching to McIDAS debug destination.

**Mccdfree** - Free parsed arg-fetching structure for the given handle.

**Mccgid** - Fetch an argument in integer type date format yyyydd.

**Mccgip** - Fetch an argument in integer type date format yyyydd.

**Mccgkey** - Validate defined and command line keywords. Edest diagnostics.

**Mccd** - Fetch all keyword names within parsed arg-fetching text.

**Mccddhr** - Parse the given text into arg-fetching structure.

**Mccddhr** - Fetch the quote field string argument.

**Mccgstr** - Fetch an argument in character form.

**Mccddhr** - Fetch a program command line argument in double type format.

**Mccddhr** - Prints diagnostics to edest device for standard errors.

**Mccddh** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.

**Mccddhr** - Fetch a program argument in fractional lat/lon format dd..fffff.
Utility-related API routines

Modprintf - print to McIDAS debug destination
Mceprintf - print to McIDAS error destination
Mcprintf - print to McIDAS standard destination
Mcsleep - sleep for a given number of milliseconds
Mcstrtodbl - Convert given numeric token to double type format.
Mcstrtodh - Convert given time token to double fractional hours hh.fffff.
Mcstrtohex - Convert given hexadecimal token to integer type format.
Mcstrtohm - Convert given time token to integer hours.
Mcstrtol - Convert given time token to integer time format.
Mcstrtolh - Convert given time token to integer time format.
Mcstrtolhms - Convert given time token to integer time format hh.mmm.
Mcstrtolms - Convert given time token to integer time format.
Mcstrtolmsms - Convert given lat/lon token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Mcstrtolmsms - Convert given date token to integer type format.
Utility-related API routines

- `ncsleep` - sleep for a given number of milliseconds
- `ncsttodbh` - Convert given numeric token to double type format.
- `ncsttodbhr` - Convert given time token to double fractional hour/lon dddd.dddd.
- `ncsttodbll` - Convert given time token to double fractional lon/lat dddd.dddd.
- `ncsttoboeh` - Convert given hexadecimal token to integer type format.
- `ncsttoboehr` - Convert given time token to integer hours, minutes and seconds.
- `ncsttoboehms` - Convert given time token to integer time format hhmmss.
- `ncsttoboell` - Convert given lat/lon token to integer type format ddddmmss.
- `ncsttobooy` - Convert given date token to integer type format.
- `ncsttobyoyd` - Convert given date token to integer date format yyyymmm.
- `movb` - moves bytes from one array to another array with a destination byte offset.
- `movblk` - moves blocks of bytes from one array to another array with both source and destination byte offsets.
- `move` - moves bytes from one array to another array with source and destination offsets.
- `movpix` - moves pixels from one array to another array with both source and destination byte offsets.
- `putfrm` - writes the frame directory to a file.
- `rpixel` - replicates elements of an array in place.
- `sleep` - sleeps for a given number of milliseconds.
- `tvline` - recommended line and element limits for a frame when drawing graphics.
- `tvset` - converts a TV line and element to a line and element.
**Name:**

mcname - a brief description of the routine

**Interface:**

the calling syntax for the Fortran or C routine, including the declarations of the arguments and the return type, if appropriate

The formats for Fortran and C routines are different:

**Fortran example:**

```fortran
integer function mcname(integer param1, character (*) param2, integer param3(64))
```

**C example:**

```c
#include 'mcidas.h'

int mcname(int param1, char *param2, int *param3)
```

**Input:**

the arguments that specify data to the routine

**Input and Output:**

the arguments that both specify data to the routine and return data from the routine

**Output:**

the arguments that return data from the routine

**Return values:**

the information returned by the routine

**Remarks:**

additional explanation of the routine

**Categories:**

the McIDAS classification of routine: calibration, converters, day/time, display, events, file, graphics, grid, image, ingest/decode, met/science, navigation, pt_src, sys_config, system, text, userInterface, utilities

**Filename:**

the file containing the routine

---

**Name:**

MCALIN - receive a single line of data from an area server

**Interface:**

integer function

```fortran
mcalin(integer integerl, integer line(*))
```

**Input:**

`handl` - identifier of this data stream

**Input and Output:**

none

**Output:**

`line` - line of data from the AREA

**Return values:**

0 - success

-1 - failure, all data has already been read

-38 - failure, the handle was not legal

-96 - failure, communication has timed out

-97 - failure, communication with the server has died

**Remarks:**

Call this once for each line of data in the area.

**Categories:**

IMAGE

**Filename:**

mcaget.f
**Mcontrol**

**Name:**
Mcontrol - Build and return a McIDAS command line for the given handle.

**Interface:**
```c
#include "mcidas.h"

char* Mcontrol(int arg_handle)
```

**Input:**
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.

**Input and Output:**
none

**Output:**
none

**Return values:**
Allocated McIDAS command line, or NULL.

**Remarks:**
Builds a McIDAS command line, from the parsed arg-fetching structure.
The returned text will be reformatted in McIDAS command line syntax.

**Categories:**
USER_INTERFACE

**Filename:**
argutil.c

---

**Mcargdbl**

**Name:**
Mcargdbl - Fetch an argument in double type format.

**Interface:**
```c
#include "mcidas.h"

int Mcargdbl(int arg_handle, const char* keyword, int position, double def, double min, double max, double* value, const char** arg)
```

**Input:**
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name in key,word format, or null/blank for positional arg, where key=minimum keyword name, .word=maximum name.
position - Given argument position within the keyword, or positional argument's place. 1 indicates 1st positional arg or 1st arg for keyword.
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range. Tests for min=value<max. min=max defaults to no given range checking.

**Input and Output:**
none

**Output:**
value - The desired argument's double value, or 0 if invalid.
arg - Pointer to global argument text string, or NULL.

**Return values:**
200 - given double default argument returned
1200 - keyedin double argument returned
1210 - keyedin hexadecimal argument returned
-204 - given double default argument < given min
-205 - given double default argument > given max
-1201 - keyedin argument is illegal decimal format
-1203 - keyedin arg exceeds system double limits
-1204 - keyedin double argument < given min
-1205 - keyedin double argument > given max
-1211 - keyedin arg is illegal hexadecimal format
-1213 - keyedin hex exceeds system integer limits
-1214 - keyedin hexadecimal argument < given min
-1215 - keyedin hexadecimal argument > given max
< -9999 - failure status
Mcargdb

Remarks:
Accepts numeric arguments in standard McIDAS formats:
[-]i.F[=+-]s , $k[xxxxxxxx]

Ignores leading blanks, and from any trailing blank on.
Where i = integer part
f = fractional part
E = exponent of E, e, D or d
s = exponential scalar
x = hexadecimal digit 0-9 or A-F

Acceptable Formats:
[-i].i , $k[xxxxxxxx]
[=]+i.f
[=-]i.F[=+-]s , $k[xxxxxxxx]
[-i].f
[-]i.F

Categories:
USEA_INTERFACE

Filename:
argfetch.c

Mcargdhr

Name:
Mcargdhr - Fetch an argument in double fractional hours format hh.fffff.

Interface:
#include "mcidac.h"

int Mcargdhr(int arg_handle, const char* keyword, int int position, double def,
double min, double max, double* value, const char** arg)

Input:
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name, in keyword format, or null/blank for
positional arg, where key=minimum keyword name,
.word=maximum name.
position - Given argument position within the keyword, or
positional argument's place.
1 indicates 1st positional arg or 1st arg for keyword.
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for
min=value<=max.
min=max defaults to no given range checking.

Input and Output:
none

Output:
value - Double time argument in format [-]hh.fffff, or 0.
arg - Pointer to global argument text string, or NULL.

Return values:
450 - given hh.ffff time default arg returned
1450 - keyedin time argument returned as hh.ffff
1460 - keyedin current hh.ffff default returned
-454 - given time default argument < given min
-455 - given time default argument > given max
-1451 - keyedin argument is illegal time format
-1453 - keyedin time arg exceeds system double limits
-1454 - keyedin time argument < given min
-1455 - keyedin time argument > given max
-1464 - keyedin current hh.ffff default < given min
-1465 - keyedin current hh.ffff default > given max
-1471 - keyedin time has illegally formatted hours
-1473 - keyedin hours exceeds system integer limits
-1481 - keyedin time has illegal minutes format
-1483 - keyedin minutes > 59
-1491 - keyedin time has illegal seconds format
-1493 - keyedin seconds > 59
<-9999 - failure statuses
Remarks:
Accept time arguments in formats:

\[ \text{[-]}hh:mm:ss, \text{[-]}hh:mm\text{[:][.]}ff\text{ft} \]

Ignores leading blanks, and from any trailing blank on.

Where \( h \) = hours  (default: 0)
\( m \) = minutes <= 59 (default: 0)
\( s \) = seconds <= 59 (default: 0)
\( ffff \) = fractions of an hour.
\( E \) = exponent of E, e, E, or d
\( n \) = exponential scalar

Acceptable Formats:

\[ \text{hh:mm, hh:mm:ss, hh:ss, .ffffff} \]

\[ \text{[-]} \text{hh, h:mm, .hh, .hh:mm, .hh:mm:ss, .hh:ss} \]

\[ \text{[-]} \text{hh, hh:mm:ss, .hh:mm:ss, .hh:ss} \]

Categories:
USER_INTERFACE, DAY/TIME

Filename:
argcfetch.c

Name:

Mcargdil - Fetch argument in double fractional lat/lon format ddd.dddd.

Interface:

\#include "mcidas.h"

int
Mcargdil(int arg, char* keyword, int position, double def, double min, double max, double* value, char* const char);

Input:

arg, handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name, in keyword format, or null/blank for positional arg, where key-minimum keyword name, .word=maximum name.
position - Given argument position within the keyword, or positional argument's place.
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for min<=value<=max.

min=max defaults to no given range checking.

Output and Output:

None

Output:

value - Double lat/lon argument in format [-]ddd.dddd, or 0.
arg - Pointer to global argument text string, or NULL.

Return values:

-550 - Given ddd.dddd lat/lon default arg returned
-1550 - keyed lat/lon arg returned as ddd.dddd
-554 - Given lat/lon default argument < given min
-1555 - keyed lat/lon default argument > given max
-1551 - keyed lat/lon argument is illegal lat/lon format
-1553 - keyed lat/lon argument > given min
-1554 - keyed lat/lon argument < given min
-1555 - keyed lat/lon argument > given max
-1571 - keyed lat/lon has illegal degrees format
-1573 - keyed degrees exceeds system integer limits
-1581 - keyed lat/lon has illegal minutes format
-1583 - keyed lat/lon minutes > 59
-1591 - keyed lat/lon has illegal seconds format
-1593 - keyed lat/lon seconds > 59
-9999 - failure status
Mcargdump

Name:
Mcargdump – Display parsed arg-fetching to McIDAS debug destination.

Interface:
#include "mcidas.h"

void
Mcargdump(int arg_handle)

Input:
arg_handle – Given argument-fetching handle to be displayed, or 0 to display
the current McIDAS command’s structure.

Input and Output:
none

Output:
none

Return values:
none

Remarks:
none

Categories:
USER_INTERFACE

Filename:
argutil.c

Mcargdump

Name:
Mcargdump – Display parsed arg-fetching to McIDAS debug destination.

Interface:
#include "mcidas.h"

void
Mcargdump(int arg_handle)

Input:
arg_handle – Given argument-fetching handle to be displayed, or 0 to display
the current McIDAS command’s structure.

Input and Output:
none

Output:
none

Return values:
none

Remarks:
none

Categories:
USER_INTERFACE

Filename:
argutil.c
Mcargfree

Name:
Mcargfree - Free parsed arg-fetching structure for the given handle.

Interface:
#include "mcidas.h"

int Mcargfree(int arg_handle)

Input:
arg_handle - Given argument-fetching handle to be freed, or 0 to free the current
McIDAS command's structure.

Input and Output:
none

Output:
none

Return values:
0 - success
< 0 - failure

Remarks:
Frees the arg handle, the arg structure, all of the keyword structures, and all
of the argument strings.

Categories:
USER_INTERFACE

Filename:
argutil.c

Mcargihr

Name:
Mcargihr - Fetch an argument in integer type time format hhmmss.

Interface:
#include "mcidas.h"

int Mcargihr(int arg_handle, const char* keyword, int position,
int def, int min, int max, int* value, const char** arg)

Input:
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name, in key.word format, or null/blank
for positional arg, where key=minimum keyword name,
.word=maximum name.
position - Given argument position within the keyword, or
positional argument's place.
def - Given default value to be returned if arg not found.
If not valid, will use it, but returns an error status.
min, max - Given acceptable arg range, tests for min<value<max.
min-max defaults to no given range checking.

Input and Output:
none

Output:
value - Integer time argument in format [-]hhmmss, or 0.
arg - Pointer to global argument text string, or NULL.

Return values:
400 - given hhmmss time default argument returned
1400 - keyedin time argument returned as hhmmss
1410 - keyedin current hhmmss default arg returned
-404 - given time default argument < given min
-405 - given time default argument > given max
-433 - given time default argument minutes > 59
-443 - given time default argument seconds > 59
-1401 - keyedin argument is illegal time format
-1403 - keyedin time arg exceeds system int limits
-1404 - keyedin time argument < given min
-1405 - keyedin time argument > given max
-1414 - keyedin current hhmmss default arg < given min
-1415 - keyedin current hhmmss default arg > given max
-1421 - keyedin time has illegally formatted hours
-1423 - keyedin hours exceeds system integer limits
-1431 - keyedin time has illegal minutes format
-1433 - keyedin minutes > 59
-1441 - keyedin time has illegal seconds format
-1443 - keyedin seconds > 59
< -9999 - failure statuses

Remarks:
Accepts time arguments in formats:
[-]hh:mm:ss, [+-]hh:mm:ss, [-]hh:mm
Ignore leading blanks, and from any trailing blank on.
Where hh = hours (default:0)
mm = minutes <= 59 (default:0)
ss = seconds <= 59 (default:0)
fff = fractions of an hour.
E = exponent of E, e, D or d
n = exponential scalar
; = defaults to current time

Acceptable Formats:
; => .. HMMSS (current time)
hh:mm, hh:mm; => .. hh:mm
hh:mm:ss, hh:mm:ss => .. hh:mm:ss
hh, hh:, hh; => .. hh000
hh:ss, hh:ss => .. hh:ss
mm, mm; => .. mm0
mm:ss, .fff => .. mm:ss
1:00 => .. 1
;

Categories:
USER_INTERFACE, DAY/TIME

Mcargill

Name:
Mcargill - Fetch an argument in integer type lat/lon format dddmm.

Interface:
#include "mcidas.h"

int
Mcargill(int arg_handle, const char* keyword, int position, int def, int min, int max, int* value, const char** arg);

Input:
arg_handle - Given arg-fetched handle, or 0 for McIDAS command.
keyword - Given keyword name, in key.word format, or null/blank
          for positional arg, where key=minimum keyword name,
          .word=maximum name.
position - Given argument position within the keyword, or
          positional argument's place.
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for min<=value<=max.
           min=max defaults to no given range checking.

Input and Output:
none

Output:
value - Integer lat/lon value in format [-]dd:ddmm, or 0.
arg - Pointer to global argument text string, or NULL.

Return values:
500 - given dd:ddmm lat/lon default arg returned
1500 - keyedin lat/lon argument returned as dd:ddmm
-504 - given lat/lon default argument < given min
-505 - given lat/lon default argument > given max
-533 - given lat/lon default argument minutes > 59
-543 - given lat/lon default argument seconds > 59
-1501 - keyedin argument is illegal lat/lon format
-1503 - keyedin lat/lon arg exceeds integer limits
-1504 - keyedin lat/lon argument < given min
-1505 - keyedin lat/lon argument > given max
-1511 - keyedin lat/lon arg exceeds integer limits
-1521 - keyedin lat/lon arg exceeds integer limits
-1533 - keyedin lat/lon has illegal minutes format
-1533 - keyedin lat/lon has illegal minutes format
-1541 - keyedin lat/lon has illegal seconds format
-1543 - keyedin lat/lon has illegal seconds format
< -9999 - failure statuses

Filename:
argfetch.c
Remarks:
Accepts lat/lon arguments in formats:
\(+\)-ddd:mm:ss, \(+\)-|ddd|:mm:mm
Ignores leading blanks, and from any trailing blank on.
Where ddd = degrees (default:0)
mm = minutes <= 59 (default:0)
ss = seconds <= 59 (default:0)
ffff = fractions of a degree.
E = exponent of E, e, D or d
n = exponential scalar
Acceptable Formats:
ddd:mm:ss, ddd.fffff --> [-] dddmm00
ddd:mm:ss, ddd.dddd --> [-] dddmmss
ddd, dddi, dddi: --> [-] ddd0000
ddd:ssss, ddd.mm, mmss, mm:ss, mmm.mmm, mm:sss, mmss.xxx, mm:ss.xx, mm:ss.x, mm:ss --> [-] ddd0000
fff, ffff --> [-] mm00
fff:fff, .fff --> [-] mmss
fff:fff:fff, .fff:fff, .fff:fff:fff --> [-] ss
Input and Output:
one
Output:
value = The desired argument's integer value, or 0 if invalid.
arg = Pointer to global argument text string, or NULL.

Return values:
100 = given integer default argument returned
1100 = keyedin integer argument returned
1110 = keyedin hexadecimal argument returned
-104 = given integer default argument >= given min
-106 = given integer default argument > given max
-1101 = keyedin argument is illegal integer format
-1103 = keyedin integer arg has illegal fraction
-1105 = keyedin integer argument >= given max
-1111 = keyedin integer argument > given max
-1115 = keyedin hexadecimal argument < given min
-1117 = keyedin hexadecimal argument > given max
-1119 = keyedin argument exceeds system integer limits
-1121 = keyedin argument is illegal hexadecimal format
-1123 = keyedin hex exceeds system integer limits
-1125 = keyedin hexadecimal argument < given min
-1127 = keyedin hexadecimal argument > given max
< -9999 = failure statuses

Name:
Mcargint - Fetch an argument in integer type format.

interface:
#include "mcidas.h"
int Mcargint(int arg.handle, const char* keyword, int int.position, int def, int min, int max, int* value, const char** arg)
Mcargint

Remarks:
Accepts integer arguments in standard McIDAS formats:
[-]i, i.E[,]s, Sx[xxxxxxxx]

Ignores leading blanks, and from any trailing blank on.

Where i = integer part
f = fractional part
E = exponent of E, e, D or d
s = exponential scalar
x = hexadecimal digit 0-9 or A-F

Acceptable Formats:
[-]i, i, iE[,]s, i.E[,]s, i.E[,]s, Sx[xxxxxxxx] ==> [-] i

Categories:
USER_INTERFACE

Filename:
argfetch.c

Mcargiyd

Name:
Mcargiyd - Fetch an argument in integer type date format yyyyddd.

Interface:

#include "mcidas.h"

int
Mcargiyd(int arg_handle, const char* keyword, int position,
int def, int min, int max, int* value, const char** arg)

Input:
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name, in key.word format, or null/blank
for positional arg, where key-minimum keyword name,
.word-maximum name.
position - Given argument position within the keyword, or
positional argument's place.
    1 indicates 1st positional arg or 1st arg for keyword.
def - Given default value to be returned if arg not found.
    If not valid, will use it, but returns an error status.
min, max - Given acceptable arg range, tests for min=value<max.
    min-max defaults to no given range checking.

Input and Output:
none

Output:
value - Integer date argument in format [-]yyyyddd, or 0.
arg - Pointer to global argument text string, or NULL.

Return values:
300 - given date default argument returned
1300 - keyedi date argument returned as yyyyddd
1310 - keyedi current date default arg returned
-304 - given date default argument < given min
-305 - given date default argument > given max
-363 - given day default is illegal for given year
-1301 - keyedi argument is illegal date format
-1303 - keyedi date arg exceeds system int limits
-1304 - keyedi date argument < given min
-1305 - keyedi date argument > given max
-1314 - keyedi current date default arg < given min
-1315 - keyedi current date default arg > given max
-1321 - keyedi date has illegally formatted year
-1323 - keyedi year exceeds system integer limits
Mcargykd

-1331 - keyedin date has illegal month (mon) format
-1341 - keyedin date has illegal month (mm) format
-1343 - keyedin month (mm) not 1-12
-1351 - keyedin date has illegally formatted day
-1353 - keyedin day is illegal for given month
-1363 - keyedin day is illegal for given year
< -9999 - failure statuses

Remarks:
Accepts date arguments in formats:

yyyy/mm/dd , mm/dd , dd/mon/yyyy , dd/mon , yyyy/ddd ,
yyyy-mm-dd , mm-dd , dd-mon-yyyy , dd-mon
(leading + or - sign allowed)

Ignores leading blanks, and from any trailing blank on.

Where yyyy- optional year, defaults to current year,
      mm = optional valid month of 1 thru 12, defaults to current month
      mon = valid calendar month (at least first 3 chars),
      dd = required valid day of the given month
      ddd = required valid day of year (1-365/366)

Acceptable formats:

/ => [-] YYYDDD (today's date)
/ddd , dd/mon , ddd , /mm/dd , dd/mon/ => [-] YYYDDD (current year)
/mm/dd , dd/mon/yyyy , yyyy/ddd => [-] YYYDDD (current century)
/year/mm/dd , dd/mon/yyyy , yyyy/ddd => [-] yyyyddd (given date)

Categories:
USER_INTERFACE , DAY/TIME

Filename:
argfetch.c

Mcargkey

Name:
Mcargkey - Validate defined and command line keywords.

Interface:
#include "mcidas.h"

int Mcargkey(int arg_handle, int numkey, const char * const valid_keywords[],
int printflag)

Input:
arg_handle - Handle of argument fetching structure.
numkey - Number of keywords in valid_keywords array.
valid_keywords - Array of defined keywords, in keyword format, where
key = minimum keyword name, .words = maximum keyword name.
printflag - Print error messages if not equal to zero.

Input and Output:
none

Output:
none

Return values:
0 - successful
-1 - defined keywords are illegal or ambiguous
-2 - command line keywords are invalid or ambiguous
< -9999 - failure statuses

Remarks:
The array of defined keywords, valid_keywords, is used to verify
keywords entered on the command line. If printflag is TRUE, a
diagnostic message is printed for each specific error, prior to
returning an error status code. Checks that all command line keywords
are unambiguously within the given array of defined keywords, and that
the given defined keywords themselves are unambiguously legal.

Categories:
USER_INTERFACE

Filename:
argkey.c
**McArgNum**

**Name:**
McArgNum - Fetch all keyword names within parsed arg-fetching text.

**Interface:**

```c
#include "mcidas.h"

int McArgNum(int arg_handle, int maxkey, const char* keywords[maxkey])
```

**Input:**
- `arg_handle` - Given arg-fetching handle, or 0 for McIDAS command.
- `maxkey` - Given dimension size of keywords array.
  - If <= 0, then only returns the number of keywords.

**Input and Output:**
none

**Output:**
- `keywords` - Array of keyword names as they occur in text line.

**Return values:**
- 0 - Number of keyword names found.
- -905 - Keyword dimension size exceeded
- -909 - No command line arg-fetching structure
- < -9999 - Failure status

**Remarks:**
The returned ‘keywords’ array, contains pointers to the arg-fetching structure’s malloc()ed keyword names, WHICH MUST NOT BE FREED!

If the number of keywords exceeds ‘maxkey’, then the ‘keywords’ array contains the first ‘maxkey’ occurring keyword names.

If # of keywords < ‘maxkey’, then the ‘keywords’ array is NULL filled.

If ‘maxkey’ <= 0, then the number of existing keywords is returned.

**Categories:**
USER_INTERFACE

**Filename:**
argutil.c
Mcargparse

Name:
Mcargparse - Parse the given text into arg-fetching structure.

Interface:
#include "marg.h"

int
Mcargparse(const char* txtext, const McArgSyntax* given_syntax,
int* parsed_len)

Input:
txtext - Given text string to parse.
given_syntax - Given argument syntax, or NULL to use MCIDAS default.

Input and Output:
none

Output:
parsed_len - Optional length of parsed text, including any
terminating ';' if present. (Use NULL to ignore).

Return values:
> 0 - Argument-fetching handle.
< 0 - failure status

Remarks:
Use Mcargfree() to free the argument-fetching structure.

After Mcargparse is called, arguments from the given text may be
fetched by calling Mcargstr, Mcargint, Mcargnum, Mcargman, etc.
The correct sequence of functions for text arg-fetching is:
hand = Mcargparse(txtext, NULL, &parsed_len)
atat = Mcargstr(hand, ...), Mcargint(hand, ...), etc.
atat = Mcargfree(hand)

The argument-fetching subsystem allows for parsing lines of text with
varying syntax, based on the MCIDAS command line syntax, thru
programmer given parsing definitions (use sscanf() to parse other syntax
types). Parsing syntax is defined in a programmer given array of 10 strings.
Each string may define one or more, single character, specific type of
syntactic separators, or other type of syntax, as follows.

APPLICATIONS PROGRAMMING INTERFACE

McIDAS Programmer's Manual
Preliminary Issue 10/93

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface

McIDAS Programmer's Manual
Preliminary Issue 10/93

Applications Programming Interface
[4] Quote Field Left Delimiters
   called syntax.qfld_delims_left
   MCIDAS default: "{\""
   The quote field is an argument enclosed within special delimiters,
   which may contain embedded spaces and other special characters,
   and which allows delimiter nesting, such as (prog nested field).
   The special function Mcargquo() fetches the quote field.
   Define one or more, single character, left delimiters for quote
   field arguments.
   Begins a quote field argument.
   Used to separate tokens (terminates non-quoted alphanumeric token).

[5] Quote Field Right Delimiters
   called syntax.qfld_delims_right
   MCIDAS default: ")"
   Define one or more, single character, right delimiters which
   correspond to the left delimiters defined in [4] above.
   If NULL, then these quote fields will extend to end of text line.
   Terminates a quote field argument.

[6] Quoting Char Left Delimiters
   called syntax.quoting_delims_left
   MCIDAS default: ""'
   Quoting disables a character's special meaning and allows it to be
   used literally, as itself.  A quoted argument is an argument
   enclosed within the quoting char delimiters, as defined in [6] and
   [7].  Everything between quoting_delims_left[1] and
   quoting_delims_right[1] is taken literally, except for escaping as
   described in [8] below.
   Define one or more, single character, left delimiters for quoting
   arguments containing embedded spaces, special characters, and
   escape characters, such as: 'I am a quoted argument'.
   Begins a quoted argument.

[7] Quoting Char Right Delimiters
   called syntax.quoting_delims_right
   MCIDAS default: "'
   Define right delimiters which correspond to the left delimiters
   defined in [6] above.
   Terminates a quoted argument.

[8] Quoting Char Escape Value
   called syntax.quoting_escape
   MCIDAS default: NULL
   Define a single escape character for quoted arguments as defined in
   [6] and [7] above, such as: '\"' is a single quote'
   The character following the quoting_escape is taken literally,
   and the quoting_escape character itself is removed.
   If NULL, then two quoting_delims_right delimiters in a row will
   escape itself, such as: "' is a single quote'

   called syntax.missing_argument
   MCIDAS default: "K"
   Define syntax to indicate that an argument is missing.
   Used in conjunction with the missing argument replacement
   (defined in [10] below).

    called syntax.missing_arg_value
    MCIDAS default: "*
    Define replacement for the missing argument (defined in [9] above).
    This value gives the internal representation for missing arguments.

Categories:
CONVERTER, USER_INTERFACE

Filename:
argparse.c
**Mcargquo**

**Name:** Mcargquo - Fetch the quote field string argument.

**Interface:**
```c
#include "mcidas.h"

int Mcargquo(int arg_handle, const char** value)
```

**Input:**
- `arg_handle` - Given arg-fetching handle, or 0 for McIDAS command.

**Input and Output:**
- `value` - Quote field string, without leading quotation mark, or NULL if not found.

**Output:**
- None

**Return values:**
- 10 - quote field not found
- 1010 - quote field argument returned
- -1015 - quote field exceeds fortran char*(*) size
- < -9999 - failure statuses

**Remarks:**
- None

**Categories:**
- USER_INTERFACE

**Filename:**
- argfetch.c

---

**Mcargstr**

**Name:** Mcargstr - Fetch an argument in character form.

**Interface:**
```c
#include "mcidas.h"

int Mcargstr(int arg_handle, const char* keyword, int position, const char* def, const char** value)
```

**Input:**
- `arg_handle` - Given arg-fetching handle, or 0 for McIDAS command.
- `keyword` - Given keyword name, in key,word format, or null/blank for positional arg., where key-minimum keyword name, word-maximum name.
- `position` - Given argument position within the keyword, or positional argument's place. 1 indicates 1st positional arg or 1st arg for keyword. 0 gets either the program name (for null keyword), or the entered keyword name.
- `def` - Given default value to be returned if arg not found. If not valid, will use it, but returns an error status.

**Input and Output:**
- None

**Output:**
- `value` - The desired argument's value, or NULL if not found.

**Return values:**
- 0 - given character default argument returned
- 1000 - keyed in character argument returned
- -5 - given arg def exceeds fortran char*(*) size
- -1005 - keyed in arg exceeds fortran char*(*) size
- < -9999 - failure statuses

**Remarks:**
- None

**Categories:**
- USER_INTERFACE

**Filename:**
- argfetch.c
Mcarrots

Name:
Mcarrots - Copy array of C strings to given 2-dimensional char array, left-justified, blank-padded, and non-null-terminated.

Interface:
#include "mcidas.h"

int Mcarrots(const char* const* str, int nstr,
int nele, int el_size, char* arr)

Input:
str - Given array of C strings to be copied.
(none-dimensional array of pointers to char)
nstr - Given number of C strings, str[nstr].
nele - Given dimension size of output array, arr[nele][el_size].
el_size - Given size of output char arrays, arr[nele][el_size].

Input and Output:
none

Output:
arr - Given two-dimensional char array, to copy strings to.

Return values:
0 - successful
< 0 - # of C strings that were truncated, or dropped
< -9999 - failure status

Remarks:
Used to convert a C string array to a fortran character array. Blank fills the array when given nele > nstr. Uses Mcarrots() to convert each string.

Categories:
none

Filename:
fs.c

Mcatrget

Name:
Mcatrget - get an attribute string given a key string

Interface:
#include "mcidas.h"

char * Mcatrget(char *string)

Input:
string - string containing the key to be looked up

Input and Output:

Output:

Return values:
Pointer to a string containing the value found in the file. If the key is not found then the string will be "". The string is valid until the next call to Mcatrget()!

Remarks:
none

Categories:
SYSTEM

Filename:
mcatrget.c
Mcatrnx

Name:
Mcatrnx - gets the next attribute pair from the list

Interface:
include "mcidas.h"
char * Mcatrnx(char **key)

Input:
none

Input and Output:
key - on input, key to find one greater than
on output, the next key in the collating sequence

Output:
none

Return values:
pointer to string containing value associated with output key
this string is valid until the next call

Remarks:
To use this routine start by setting key = ", repeated
calls will traverse the list in alphabetical order by key
until key is NULL, at which time, no more calls can be made
without re-initializing.

Categories:
SYSTEM

Filename:
mcatrnx.c

Mcatrput

Name:
Mcatrput - puts an attribute pair (key-value) to a file

Interface:
include "mcidas.h"
int Mcatrput(char *key, char *value)

Input:
key - string containing the key to be loaded
value - string containing the value assigned to the key

Output:
none

Return values:
0 - success
-1 - failure, file open for output failed
-2 - failure, attempt to write file failed

Remarks:
key-value pair is written into the users output file.
Missing values are represented by the string "", which
means that this key should be deleted from the table.

Categories:
SYSTEM

Filename:
mcatrput.c
Mccmd

Name:
Mccmd - Fetch the current McIDAS command line.

Interface:
#include "mcidass.h"
char* Mccmd(void)

Input:
one

Input and Output:
one

Output:
one

Return values:
Current command line, or NULL.

Remarks:
one

Categories:
USER_INTERFACE

Filename:
cmdfetch.c

Mccmddbl

Name:
Mccmddbl - Fetch a program command line argument in double type format.
Prints diagnostics to oedat device for standard errors.

Interface:
#include "mcidass.h"

int Mccmddbl(const char* keyword, int position, const char* printmsg,
double def, double min, double max, double* value)

Input:
"key.word" - Given name of the command line keyword, or
null/blank to indicate a positional argument.
"key=minimum keyword name, .word=maximum name.
"key.word" matches with "key" up thru "keyword".
"*keyword" and "keyword." match only with "keyword".
position - Given argument position within the keyword, or
positional argument's place on the command line.
1 indicates 1st positional arg or 1st arg for keyword.
printmsg - Given description of this argument (displayed if error).
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for min<value<max.
min>max defaults to no given range checking.

Input and Output:
one

Output:
value - The desired argument's double value, or 0 if invalid.

Return values:
200 - given double default argument returned
1200 - keyedin double argument returned
1210 - keyedin hexadecimal argument returned
-204 - given double default argument < given min
-205 - given double default argument > given max
-1201 - keyedin argument is illegal decimal format
-1203 - keyedin arg exceeds system double limits
-1204 - keyedin double argument < given min
-1205 - keyedin double argument > given max
-1211 - keyedin arg is illegal hexadecimal format
-1213 - keyedin hex exceeds system integer limits
-1214 - keyedin hexadecimal argument < given min
-1215 - keyedin hexadecimal argument > given max
< -9999 - failure statuses
Mccmdbl

Remarks:
Accepts numeric arguments in standard麦DAS formats:

[+-][i.E[+-]s , $x{x}[xxxxxx]

ignores leading blanks, and from any trailing blank on.

Where i = integer part
E = fractional part
x = exponent of E, e, D o d
s = exponential scalar
x = hexadecimal digit 0-9 or A-F

Acceptable Formats:
[+-]i, i., $x{xxxxxxx}
[+-]i.f
[+-].f
[+-]i.E[+-]s, iE[+-]s, .fE[+-]s, i.E[+-]s

Categories:
USER_INTERFACE

Filename:
cmdfetch.c

Mccmdhdr

Name:
Mccmdhdr - Fetch a program argument in fractional hours format hh.fffff.
Prints diagnostics to edest device for standard errors.

Interface:
#include "mcidas.h"

int
Mccmdhdr(const char* keyword, int position, const char* printmsg,
         double def, double min, double max, double* value)

Input:
keyword
- Given name of command line keyword, in keyword format, or
  null to indicate a positional argument.

position
- Given argument position within the keyword, or
  positional argument's place on the command line.
  where key-minimum keyword name, .word-maximum name.

printmsg
- Given description of this argument (displayed if error).

If not valid, will use it, but returns an error status.

def
- Given default value to be returned if arg not found.

min, max
- Given acceptable arg range, tests for min=value<max.
  min=max defaults to no given range checking.

Output and Output:
none

Output:
value
- Double time argument in format [–]hh.ffffff, or 0.

Return values:
450 - given hh.ffffff time def argument returned
1450 - keyedin time argument returned as hh.ffffff
1460 - keyedin current hh.ffffff default returned
-454 - given time default argument < given min
1455 - given time default argument > given max
-1451 - keyedin argument is illegal time format
1453 - keyedin time arg exceeds system double limits
-1454 - keyedin time argument < given min
-1455 - keyedin time argument > given max
-1464 - keyedin current hh.ffffff default < given min
-1465 - keyedin current hh.ffffff default > given max
-1471 - keyedin time has illegally formatted hours
-1473 - keyedin hours exceeds system integer limits
-1481 - keyedin time has illegal minutes format
Mccmddll

Remarks:
Accept time arguments in formats:
[-]hh:mm:ss , [ ]hh.ffffE[ ]n
Ignores leading blanks, and from any trailing blank on.
Where hh = hours (default-0)
mm = minutes <= 59 (default-0)
ss = seconds <= 59 (default=0)
ffffff = fractions of an hour.
E = exponent of E, e, d or D
n = exponential scalar
: = defaults to current time

Acceptable Formate:

: === [-] HH.FFFF (current time)
hh:mm, hh:mm, hh:mm:ss, hh:ss, === [-] hh:ffffff
hh:ffffff
hh, hh, hh: === [-] hh
: === [-] :ffffff
: === 0.0

Categories:
USER_INTERFACE, DAY/TIME

Filename:
comddfh.c

Mccmddll

Name:
Mccmddll - Fetch a program argument in fractional lat/lon form dd.fffffff.
Prints diagnostics to edest device for standard errors.

Interface:
#include "mcidas.h"

int Mccmddll(const char* keyword, int position, const char* printmsg,
double def, double min, double max, double* value)

Input:

keyword - Given name of command line keyword, in key.word format,
or null to indicate a positional argument.
where key-minimum keyword name, .word-maximum name.
position - Given argument position within the keyword, or
positional argument’s place on the command line.
1 indicates 1st positional arg or 1st arg for keyword.
printmsg - Given description of this argument (displayed if error).
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for
min<=value<=max.
min=max defaults to no given range checking.

Input and Output:
none

Output:
value - Double lat/lon argument in format [-]dd.fffffff, or 0.

Return values:
550 - given dd.fffffff lat/lon default arg returned
1550 - keyedin lat/lon arg returned as dd.fffffff
-554 - given lat/lon default argument < given min
-555 - given lat/lon default argument > given max
-1551 - keyedin argument is illegal lat/lon format
-1553 - keyedin lat/lon arg exceeds double limits
-1554 - keyedin lat/lon argument < given min
-1555 - keyedin lat/lon argument > given max
-1571 - keyedin lat/lon has illegal degrees format
-1573 - keyedin degrees exceed system integer limits
-1581 - keyedin lat/lon has illegal minutes format
-1583 - keyedin lat/lon minutes > 59
-1591 - keyedin lat/lon has illegal seconds format
-1593 - keyedin lat/lon seconds > 59
-1483 - keyedin minutes > 59
-1491 - keyedin time has illegal seconds format
-1493 - keyedin seconds > 59
< -9999 - failure status
Mccmdihr

Name:
Mccmdihr - Fetch a program argument in integer type time format hhmms.
Prints diagnostics to.stderr for standard errors.

Interface:
#include "scidas.h"

int Mccmdihr(const char* keyword, int position, const char* printmag,
int def, int min, int max, int* value)

Input:
keyword - Given name of command line keyword, in keyword format, or
null to indicate a positional argument.
where keyword is minimum keyword name, .word is maximum name.
position - Given argument position within the keyword, or
positional argument's place on the command line.
1 indicates 1st positional arg or 1st arg for keyword.
printmag - Given description of this argument (displayed if error).
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, casts for min<value<max.
min=max defaults to no given range checking.

Input and Output:
none

Output:
value - Integer time argument in format [-]hhmms, or 0.

Return values:
400 - given hhmms time default argument returned
1400 - keyedin time argument returned as hhmms
1410 - keyedin current hhmms default arg returned
-404 - given time default argument < given min
-405 - given time default argument > given max
-433 - given time default argument minutes > 59
-443 - given time default argument seconds > 59
-1400 - keyedin argument is illegal time format
-1403 - keyedin time arg exceeds system int limits
-1404 - keyedin time argument < given min
-1405 - keyedin time argument > given max
-1414 - keyedin current hhmms default arg < given min
-1415 - keyedin current hhmms default arg > given max
-1421 - keyedin time has illegally formatted hours
-1423 - keyedin hours exceeds system integer limits
-1431 - keyedin time has illegal minutes format
-1432 - keyedin minutes > 59

Categories:
USER_INTERFACE, NAVIGATION

Acceptable Formats:
dd:mm:ss, dd:mm:ss, dd:mm:ss
--- > [-] dd:ffff
1, : : --- > 0.0

Where dd = degrees (default=0)
mm = minutes <= 59 (default=0)
ss = seconds <= 59 (default=0)
ffff = fractions of an degree.
E = exponent of E, e, E or d
n = exponential scalar
1 = defaults to current lat/lon
McCmdilr

-1441 - keyedin time has illegal seconds format
-1443 - keyedin seconds > 59
< -9999 - failure statuses

Remarks:
Accepts time arguments in formats:
[+-]hh:mm:ss, [+-]hh.ffffffE[+-]n
Ignores leading blanks, and from any trailing blank on.
Where hh = hours (default=0)
mm = minutes <= 59 (default=0)
n = seconds <= 59 (default=0)
ffffff = fractions of an hour.
E = exponent of E, e, .d or d
n = exponential scalar
i = defaults to current time

Acceptable Formats: [+-]hh:mm:ss (current time)

Categories:
USER_INTERFACE, DAY/TIME

Filename:
cmdfetch.c

McCmdill

Name:
McCmdill - Fetch a program argument in integer type lat/lon form ddmmss.
Prints diagnostics to edest device for device errors.

Interface:
#include "mcides.h"

int

McCmdill(const char* keyword, int position, const char* printmsg,
int def, int min, int max, int* value)

Input:

keyword - Given name of command line keyword, in keyword format, or
null to indicate a positional argument.

position - Given argument position within the keyword, or
positional argument's place on the command line.

printmsg - Given description of this argument [displayed if error].

def - Given default value to be returned if arg not found.

min, max - Given acceptable arg range, tests for min-value<max.
min-max defaults to no given range checking.

Input and Output:

none

Output:

value - Integer lat/lon value in format [+-]ddmmss, or 0.

Return values:

500 - given ddmmss lat/lon default arg returned
1500 - keyedin lat/lon argument returned as ddmmss
-504 - given lat/lon default argument < given min
-505 - given lat/lon default argument > given max
-533 - given lat/lon default arg exceeds integer limits
-543 - given lat/lon default argument seconds > 59
-1501 - keyedin argument is illegal lat/lon format
-1503 - keyedin lat/lon has illegal degrees format
-1504 - keyedin lat/lon argument < given min
-1505 - keyedin lat/lon argument > given max
-1521 - keyedin lat/lon has illegal minutes format
-1523 - keyedin degrees exceeds integer limits
-1531 - keyedin lat/lon has illegal minutes format
-1533 - keyedin lat/lon has illegal degrees format
-1541 - keyedin lat/lon has illegal seconds format
< -9999 - failure statuses
Mccmdint

Remarks:
Accepts lat/lon arguments in formats:
[-]dd:mm:ss . [+]dd.ddddd[+-]in
Ignores leading blanks, and from any trailing blank on.
Where dd = degrees (default=0)
mm = minutes <= 59 (default=0)
ss = seconds <= 59 (default=0)
ffff = fractions of a degree.
E = exponent of E, E, D or D
n = exponential scalar

Acceptable Formats:
dd:mm, dd:mm: ===> [-] ddmm00
dd:mm:ss, dd.dddd ===> [-] ddmmss
dd, dd1, dd: ===> [-] dd0000
dd:ss ===> [-] dd00as
mm, mm: ===> [-] mm00
mm:ss, .dddd ===> [-] mmss
.. === [ ] .s

Categories:
USER_INTERFACE, NAVIGATION

Filename:
cmdfetch.c

Name:
Mccmdint - Fetch a program command line argument in integer type format.
Prints diagnostics to edest device for standard errors.

Interface:
#include "mccdas.h"

int
Mccmdint(const char* keyword, int position, const char* printmag,
int def, int min, int max, int* value)

Input:
keyword - Given name of command line keyword, in key_word format,
or null to indicate a positional argument,
where key_minimum keyword name, .word=maximum name.
position - Given argument position within the keyword, or
positional argument’s place on the command line.
printmag - Given description of this argument (displayed if error).
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for min<=value<=max.
min-max defaults to no given range checking.

Input and Output:
one

Output:
value - The desired argument’s integer value, or 0 if invalid.

Return values:
100 - given integer default argument returned
100 - keyedin integer argument returned
110 - keyedin hexadecimal argument returned
104 - given integer default argument < given min
105 - given integer default argument > given max
1101 - keyedin argument is illegal integer format
1102 - keyedin integer arg has illegal fraction
1103 - keyedin arg exceeds system integer limits
1104 - keyedin integer argument < given min
1105 - keyedin integer argument > given max
1111 - keyedin arg is illegal hexadecimal format
1113 - keyedin hex exceeds system integer limits
1114 - keyedin hexadecimal argument < given min
1115 - keyedin hexadecimal argument > given max
< -9999 - failure statusess
Mccmdiyd

Name:
Mccmdiyd - Fetch a program argument in integer type date format yyyyddd. Prints diagnostics to nearest device for standard errors.

Interface:
#include "mcdas.h"

int Mccmdiyd(const char* keyword, int position, const char* printmsg, int def, int min, int max, int* value)

Input:
keyword - Given name of command line keyword, in keyword format, or null to indicate a positional argument, where key-minimum keyword name, .word-maximum name.
position - Given argument position within the keyword, or positional argument's place on the command line. 1 indicates 1st positional arg or 1st arg for keyword.
printmsg - Given description of this argument (displayed if error).
def - Given default value to be returned if arg not found. If not valid, will use it, but returns an error status.
min, max - Given acceptable arg range, tests for min<=value<=max. min-max defaults to no given range checking.

Output:
value - Integer date argument in format [-]yyyyddd, or 0.

Return values:
300 - given date default argument returned
1300 - keyedin date argument returned as yyyyddd
1310 - keyedin current date default arg returned
-304 - given date default argument < given min
-305 - given date default argument > given max
-363 - given day default is illegal for given year
-1301 - keyedin argument is illegal date format
-1303 - keyedin date arg exceeds system int limits
-1304 - keyedin date argument < given min
-1305 - keyedin date argument > given max
-1314 - keyedin current date default arg < given min
-1315 - keyedin current date default arg > given max
-1321 - keyedin date has illegally formatted year
-1323 - keyedin year exceeds system integer limits
Mccmdiyd

-1331  - keyedin date has illegal month (mon) format
-1341  - keyedin date has illegal month (mm) format
-1343  - keyedin month (mm) not 1-12
-1351  - keyedin date is illegally formatted day
-1353  - keyedin day is illegal for given month
-1363  - keyedin day is illegal for given year
< -9999  - failure statuses

Remarks:
Accepts date arguments in formats:
    yyyy/mm/dd, mm/dd, dd-mon/yyyy, dd-mon, yyyy/ddd,
    yyyy-mm-dd, mm-dd, dd-mon-yyyy, dd-mon
    (leading - or - sign allowed)
Ignores leading blanks, and from any trailing blank on.
Where yyyy= optional year, defaults to current year,
    if yy <= 2 digits, defaults to current century
    mm = optional valid month of 1 thru 12, defaults to current month
    mon = valid calendar month (at least first 3 chars),
    defaults to current month
    dd = required valid day of the given month
    ddd = required valid day of year (1-365/366)
    / = defaults to today's date
Acceptable Formats:
    / = today's date
    / = current year
    / = current month
    / = current century
    / = given date

Categories:
    USER_INTERFACE, DAY/TIME

Filename:
    cmdfetch.c

Mccmdkey

Name:
    Mccmdkey - Validate defined and command line keywords.
    Edstat diagnostics.

Interface:
    #include "mcidas.h"
    int
    Mccmdkey(int nunkey, const char* const valid_keywords[])

Input:
    nunkey - Number of keywords in valid_keywords array.
    valid_keywords - Array of defined keywords, in key.words format, where
        key-minimum keyword name, .words-maximum keyword name.

Input and Output:
    none

Output:
    none

Return values:
    0  - successful
    -1  - defined keywords are illegal or ambiguous
    -2  - command line keywords are invalid or ambiguous
    < -9999  - failure statuses

Remarks:
The array of defined keywords, valid_keywords, is used to verify
keywords entered on the command line. A diagnostic message is printed
for each specified error, prior to returning an error status code.
Checks that all command line keywords are unambiguously within the
given array of defined keywords, and that the given defined keywords
themselves are unambiguously legal.

Categories:
    USER_INTERFACE

Filename:
    argkey.c
**Mccmdnam**

**Name:**
Mccmdnam - Fetch all keyword names occurring in the command line.

**Interface:**
```c
#include "mcidas.h"

int Mccmdnam(int maxkey, const char* keywords[maxkey])
```

**Input:**
- `maxkey` - Given dimension size of keywords array.
  - If <= 0, then only returns the number of keywords.

**Input and Output:**
none

**Output:**
- `keywords` - Array of keyword names as they occur in command line.

**Return values:**
- `>= 0` - Number of keyword names found.
- `-905` - Keyword dimension size exceeded
- `-909` - No command line arg-fetching structure
- `< -9999` - Failure status

**Remarks:**
The returned 'keywords' array contains pointers to the arg-fetching structure's malloced keyword names, WHICH MUST NOT BE FREED!

If the number of keywords exceeds 'maxkey', then the 'keywords' array contains the first 'maxkey' occuring keyword names.

If # of keywords < 'maxkey', then the 'keywords' array is NULL filled.

If 'maxkey' <= 0, then the number of existing keywords is returned.

**Categories:**
USER_INTERFACE

**Filename:**
cmdfetch.c

---

**Mccmdnum**

**Name:**
Mccmdnum - Return # values associated with given command line keyword.

**Interface:**
```c
#include "mcidas.h"

int Mccmdnum(const char* keyword)
```

**Input:**
- `keyword` - Given name of command line keyword, in keyword format, or null to indicate a positional argument, where key-minimum keyword name, .word-maximum name.

**Input and Output:**
none

**Output:**
none

**Return values:**
- `>= 0` - The number of arguments existing for given keyword.
- `< -9999` - Failure status

**Remarks:**
none

**Categories:**
USER_INTERFACE

**Filename:**
cmdfetch.c
Mccmdquo

Name:
Mccmdquo - Fetch the quote field string command line argument.
Prints diagnostics to edost device for standard errors.

Interface:
#include "mclses.h"
int Mccmdquo(const char** value)

Input:
none

Input and Output:
none

Output:
value - Quote field string, without leading quotation mark, or NULL if not found.

Return values:
10 - quote field not found
1010 - quote field argument returned
-1015 - quote field exceeds fortran char* size
< -9999 - failure statuses

Remarks:
none

Categories:
USER_INTERFACE

Filename:
cmdfetch.c

Mccmdstr

Name:
Mccmdstr - Fetch a program command line argument in character form.
Prints diagnostics to edost device for standard errors.

Interface:
#include "mclses.h"
int Mccmdstr(const char* keyword, int position, const char* def, const char** value)

Input:
keyword - Given name of command line keyword, in keyword format, or null to indicate a positional argument, where key-minimum keyword name, .word-maximum name.
position - Given argument position within the keyword, or positional argument's place on the command line. 1 indicates 1st positional arg or 1st arg for keyword. 0 gets either the program name (for null keyword), or the entered keyword name.
def - Given default value to be returned if arg not found. If not valid, will use it, but returns error status.

Input and Output:
none

Output:
value - The desired argument's value, or NULL if not found.

Return values:
0 - given character default argument returned
1000 - keyedin character argument returned
-5 - given arg def exceeds fortran char* size
-1005 - keyedin arg exceeds fortran char* size
< -9999 - failure statuses

Remarks:
none

Categories:
USER_INTERFACE

Filename:
cmdfetch.c
**Mcdev2uc**

**Name:**
Mcdev2uc - Convert given DEV- character value to its numeric uc value.

**Interface:**
```c
#include "mcidas.h"

int Mcdev2uc(char dev)
```

**Input:**
dev - Given DEV- character value.

**Input and Output:**
one

**Output:**
one

**Return values:**
- >= 0 - Numeric DEV- uc value.
- < 0 - failure

**Remarks:**
one

**Categories:**
SVC_CONFIG

**Filename:**
argglo.c

---

**Mcxhrtohms**

**Name:**
Mcxhrtohms - Convert double hours to hours, minutes and seconds.

**Interface:**
```c
#include "mcidas.h"

int Mcxhrtohms(double dhr, int* hours, int* minutes, int* seconds)
```

**Input:**
dhr - double hours (HH.FFFF)

**Input and Output:**
one

**Output:**
- hours - number of hours; set to 0 on error
- minutes - number of minutes; set to 0 on error
- seconds - number of seconds; set to 0 on error

**Return:**
- 0 - successful conversion
- -1 - on error

**Remarks:**
Convert time specified in double hours (HH.FFFF) to hours, minutes and seconds.

**Categories:**
DAY/TIME

**Filename:**
convtime.c
**Mcdhrtoihr**

**Name:**
Mcdhrtoihr - Convert double hours to integer hours.

**Interface:**
#include "mcidas.h"

int Mcdhrtoihr(double dhr, int* ihr)

**Input:**
dhr - double hours (HH.FFFFF)

**Input and Output:**
none

**Output:**
ihr - integer hours (HH.MMSS); set to 0 on error

**Return:**
0 - successful conversion
-1 - on error

**Remarks:**
Convert time specified in double hours (HH.FFFFF), to integer hours (HH.MMSS).

**Categories:**
DAY/TIME

**Filename:**
convtime.c

---

**Mcdprintf**

**Name:**
Mcdprintf - print to McIDAS debug destination

**Interface:**
#include "mcidas.h"

void Mcdprintf(const char *format, ...)

**Input:**
format - printf() format string

**Input and Output:**
none

**Output:**
none

**Return values:**
none

**Remarks:**
This is a printf()-like routine that sends the output to the McIDAS debug destination.

It is subject to the DEV- global keyword.

The output is buffered and is transmitted one line at a time.

**Categories:**
utility

**Filename:**
mprintf.c
**Mceprintf**

**Name:**
Mceprintf - print to McIDAS error destination

**Interface:**
```
#include "mcidas.h"

void mceprintf(const char *format, ...)
```

**Input:**
```
format - printf() format string
```

**Input and Output:**
```
one
```

**Output:**
```
one
```

**Return values:**
```
one
```

**Remarks:**
This is a printf()-like routine that sends the output to the McIDAS error destination.
It is subject to the #ENV global keyword.
The output is buffered and is transmitted one line at a time.

**Categories:**
utility

**Filename:**
```
mcprintf.c
```

---

**Mcfreearrr**

**Name:**
Mcfreearrr - Free the given array, including all the array's pointers.

**Interface:**
```
#include "mcidas.h"

void mcfreearrr(char** str, int nstr)
```

**Input:**
```
str - Given array of pointers to be freed.
nstr - Given number of pointers.
```

**Input and Output:**
```
one
```

**Output:**
```
one
```

**Return values:**
```
one
```

**Remarks:**
Frees all of the array's pointers, and then frees the array itself.

**Categories:**
```
one
```

**Filename:**
```
f.c
```
Mcfreestrs

Name: Mcfreestrs - Free the given array's pointers, without freeing the array.

Interface:
#include "mcidas.h"

void Mcfreestrs(char** str, int nstr)

Input: nstr - Given number of pointers.

Input and Output: str - Given array of pointers to be freed, returns array of NULL's.

Output: none

Return values: none

Remarks: Frees all of the array's pointers, and reinitializes them to NULL.

Categories: none

Filename: fs.c

Mcfstoarr

Name: Mcfstoarr - Create C string array from a given 2-dimensional array
that is left-justified, blank-padded, non-null-terminated.

Interface:
#include "mcidas.h"

char** Mcfstoarr(const char* arr, int nele, int el_size)

Input: arr - Given two-dimensional char array, to copy to C strings.
nele - Given number of array elements, arr[nele][el_size].
el_size - Given size of char arrays, arr[nele][el_size].

Input and Output: none

Output: none

Return values: Pointer to a newly malloc()ed C string array, or NULL if error.

Remarks: Used to convert a fortran character array to a C string array.
Use Mcfreextr() to free the returned C string array.

Categories: none

Filename: fs.c
Mchmstoihr

Name:
Mchmstoihr - Convert time in hours, minutes and seconds to integer.

Interface:
#include "mcidas.h"

int Mchmstoihr(int hours, int minutes, int seconds, int* ihr)

Input:
hours - number of hours
minutes - number of minutes
seconds - number of seconds

Input and Output:
none

Output:
ihr - integer hours (HMMSS); set to 0 on error

Return:
0 - successful conversion
-1 - on error

Remarks:
Convert time specified in hours, minutes and seconds to integer hours (HMMSS).

Categories:
DAY/TIME

Filename:
 convtime.c

Mcitonth

Name:
Mcitonth - Format the given number into string of 'nth'.

Interface:
#include "mcidas.h"

char* Mcitonth(int i)

Input:
i - Given number.

Input and Output:
none

Output:
none

Return values:
"nth" string (ex: 'first', '2nd', '3rd', '4th', '11th', '21st', etc.)

Remarks:
none

Categories:
none

Filename:
argerror.c
**Mlocase**

**Name:**
Mlocase - convert a string to all lower case letters

**Interface:**
```
#include "mcidas.h"

void
Mlocase(char *string)
```

**Input:**
none

**Input and Output:**

string - string to be converted

**Output:**
none

**Return values:**
none

**Remarks:**
none

**Categories:**
conversion

**Filename:**
mlocase.c

---

**Mluc**

**Name:**
Mluc - Lookup user common for given index.

**Interface:**
```
#include "mcidas.h"

Fint
Mluc(int index)
```

**Input:**
index - Given user common index.

**Input and Output:**
none

**Output:**
none

**Return values:**
Desired user common value.

**Remarks:**
Returns the value currently stored in McIDAS common storage. For positive indexes, the value is from session based common; for negative indexes, it is from a process-based common. The index 0 is treated differently between UNIX and os2.

**Categories:**
sys_config
system

**Filename:**
mlocuc.c
**Mcpathname**

**Name:**
Mcpathname - determines the system pathname of a McIDAS file

**Interface:**
```c
#include "mcidas.h"

char *
Mcpathname(char *filename)
```

**Input:**
filename - name of the file

**Input and Output:**
none

**Output:**
none

**Return values:**
- `0` - failure, could not find the file
- `>0` - success, pointer to full pathname of the file

**Remarks:**
The return pointer may point to a static data area overwritten on each call.

The return pointer may also point to the filename argument.

On OS/2, if the file is not found via REDIRECT, the return value is simply the filename.

In Fortran, use `vlnam()`.

**Categories:**
file

**Filename:**
pathname.c

**Mprintf**

**Name:**
Mprintf - print to McIDAS standard destination

**Interface:**
```c
#include "mcidas.h"

void
Mprintf(const char *format, ...)
```

**Input:**
format - printf() format string

**Input and Output:**
none

**Output:**
none

**Return values:**
none

**Remarks:**
This is a printf()-like routine that sends the output to the McIDAS standard destination.

It is subject to the DEV- global keyword.

The output is buffered and is transmitted one line at a time.

**Categories:**
utility

**Filename:**
mprintf.c
Mcpuc

Name:  Mcpuc - modify the contents of a user common location

Interface:
#include "mcidas.h"
void
Mcpuc(int value, int index)

Input:
value - new value to put in user common
index - Given user common index.

Input and Output:
none

Output:
none

Return values:
Desired user common value.

Remarks:
Changes the contents of a particular word in user common.
For positive indexes, the value is in session based common;
for negative indexes, it is in a process-based common.
The index 0 is treated differently between UNIX and os/2.

Categories:
none

Filename:
mcluc.c

---

Mcread

Name:  Mcread - read file

Interface:
#include "mcidas.h"
int
Mcread(char *file, int start, int count, void *target)

Input:
file - string containing file name
start - first byte in the file to read
count - number of bytes to read

Input and Output:
none

Output:
target - location into which bytes will be read

Return values:
0 - success, data was read from file
-1 - success, but some data was past end-of-file
-2 - failure, bad value for one or more arguments
-3 - failure opening file
-4 - failure, read failed

Remarks:
Uninitialized bytes in the file are read as if they had the value 0x80.
In Fortran, use ihi() or lhi().
A byte count less than 1 or first byte less than 0 is an error as is a null
pointer as a filename.

Categories:
file

Filename:
mcread.c
**Mcremove**

**Name:**
Mcremove - remove a file

**Interface:**
`#include "mcidas.h"
Mcremove(char *name)`

**Input:**
name - name of file to remove

**Input and Output:**
none

**Output:**
none

**Return values:**
0 - success, removed
<0 - failure, not removed

**Remarks:**
The Fortran version of this routine is lwd().

**Categories:**
file

**Filename:**
`m0cache.c`

---

**Mcsetdev**

**Name:**
Mcsetdev - Update user common with the given DEV= arguments.

**Interface:**
`#include "mcidas.h"
int
Mcsetdev(const char* dev, const char* devfile, char devdisp)`

**Input:**
dev - Given DEV=ccc value (e=N/F/C/P/S), or NULL.
devfile - Given DEV=X filename value, or NULL.
devdisp - Given DEV=X disp value, or NULL.

**Input and Output:**
none

**Output:**
none

**Return values:**
0 - success
< 0 - failure

**Remarks:**
none

**Categories:**
SYU_CONFIG

**Filename:**
`argglo.c`
Mcsleep

Name:
Mcsleep - sleep for a given number of milliseconds

Interface:
#include "mcidas.h"

void
Mcsleep(int millisecs)

Input:
millisecs - number of milliseconds to idle

Input and Output:
none

Output:
none

Return values:
none

Remarks:
For millisecs < 1, Mcsleep does not sleep.

Categories:
system, utility

Filename:
mcsleep.c

Mctrotdbl

Name:
Mctrotdbl - Convert given numeric token to double type format.

Interface:
#include "mcidas.h"

int
Mctrotdbl(const char* token, double* value)

Input:
token - Given numeric character token in standard McIDAS formats:
[-][i.eE][-][s, $][x][xxxxxx]
Ignore leading blanks, and from any trailing blank on.
where i = integer part
f = fractional part
E = exponent of E, e, D or d
s = exponential scalar
x = hexadecimal digit 0-9 or A-F

Input and Output:
none

Output:
value - Double value of token, or 0 if invalid.

Return values:
200 - successful return of given double
210 - successful return of given hexadecimal
-201 - illegally formatted decimal value (invalid char)
-203 - illegal number exceeds system double limits
-209 - no given token returns value of 0
-211 - illegally formatted hexadecimal (invalid char)
-213 - illegal hex exceeds system integer limits
-219 - no given hex returns value of 0
< -9999 - failure statuses

Remarks:
Acceptable Formats:
[-][i. eE][-][s, $][x][xxxxxx]  --->  [-] i.e
[-][i.f]  --->  [-] i.f
[-][i.fE][-][s, $][E][-][s, i.E][-][s]  --->  [-] i.f
[-][i][E][-][s]  --->  invalid

Categories:
USER_INTERFACE, CONVERTER, UTILITY

Filename:
argconv.c
Mcstrtodhr

Name:

Mcstrtodhr - Convert given time token to double fractional hours hh.ffffff

Interface:

#include "mcidas.h"

int Mcstrtodhr(const char* token, double* dhr)

Input:

token - Given character string time token in formats:

[-]hh:mm:ss , [-]hh:ffffff[-]n

Ignored leading blanks, and any trailing blank on.

Where hh  = hours

mm  = minutes <= 59 (default=0)

ss  = seconds <= 59 (default=0)

ffffff = fractions of an hour.

E  = exponent of E, e, D or d

n  = exponential scalar

;  = defaults to current time

Input and Output:

none

Output:

dhr - Double time in format [-]hh.ffffff, or 0 if invalid.

Return values:

450  = successful return of given time

460  = current time returned for given ': ' token

-451  = illegally formatted time (invalid char)

-453  = illegal time exceeds system double limits

-459  = no given time returns value of 0

-471  = illegally formatted hours within given time

-473  = illegal hours exceed system integer limits

-481  = illegally formatted minutes within given time

-483  = illegal minutes > 59

-491  = illegally formatted seconds within given time

-493  = illegal seconds > 59

< -9999  = failure status
Mctrtodll

Name:
Mctrtodll - Convert given token to double fractional lat/lon ddd....

Interface:
#include "mctdas.h"

int Mctrtodll(const char* token, double* dll)

Input:
token - Given character string lat/lon token in formats:
[+-]dddd.mmm, [+-]dddd.fffff[+-].in
Ignored leading blanks, and from any trailing blank on.
Where ddd = degrees (default=0)
mm = minutes <= 59 (default=0)
s = seconds <= 59 (default=0)
ffiff = fractions of an degree.
E = exponent of E, e, d or d
n = exponential scalar

Input and Output:
none

Output:
dll - Double lat/lon in format [+-]ddd...., or 0 if invalid.

Return values:
550 - successful return of given lat/lon
551 - illegally formatted lat/lon (invalid char)
-553 - illegal lat/lon exceeds system double limits
-559 - no given lat/lon returns value of 0
-571 - illegally formatted degrees within given lat/lon
-573 - illegal degrees exceeds system integer limits
-581 - illegally formatted minutes within given lat/lon
-583 - illegal minutes > 59
-591 - illegally formatted seconds within given lat/lon
-593 - illegal seconds > 59
< -9999 - failure statuses

Remarks:
Acceptable Formats:
ddd:mm, ddd:mm:ss, ddd:ffiff, ddd....

Odd, ddd, ddd:;:

1, 11

Categories:
USER_INTERFACE, CONVERTER, UTILITY, NAVIGATION

Filename:
argconv.c
Mcstrtos

Name:
Mcstrtos - Copy C string to fixed length, blank-padded, char block.

Interface:
#include "mcidas.h"

int Mcstrtos(char* mem, const char* str, int siz_mem)

Input:
str - Given C string to be copied, i.e. source of copy.
siz_mem - Given size of destination character array.

Input and Output:
none

Output:
mem - Given pointer to character array, destination of copy.

Return values:
0 - successful
-1 - truncation error (C string length exceeded size)
< -9999 - failure status

Remarks:
Used to convert a C string to a fortran character*(*) field.
If mem = str, then changes a null-terminated string to blank-terminated.

Categories:
none

Filename:
fs.c

Mcstrtohex

Name:
Mcstrtohex - Convert given hexadecimal token.

Interface:
#include "mcidas.h"

int Mcstrtohex(const char* token, int* value)

Input:
token - Given hexadecimal numeric char token in format:
[0-9A-F]{n}x{m}, where n = 0 thru 9 or A thru F
Ignored leading blanks, and from any trailing blank.

Input and Output:
none

Output:
value - Hex token's integer value, or 0 if invalid.

Return values:
10 - successful return of given hexadecimal
-11 - illegally formatted hexadecimal (invalid char)
-13 - illegal hex exceeds system integer limits
-19 - no given token returns value of 0
< -9999 - failure status

Remarks:
none

Categories:
USER_INTERFACE, CONVERTER, UTILITY

Filename:
argconv.c
**Mcstrtohms**

**Name:**

`Mcstrtohms` - Convert given time to integer hours, minutes and seconds.

**Interface:**

```c
#include "mcidas.h"

int Mcstrtohms(const char* token, int* hour, int* min, int* sec)
```

**Input:**

`token` - Given time token in format `[-+]hh:mm:ss`. Ignores leading blanks, and from any trailing blank on.

Where `hh` = hours (default=0)

`mm` = minutes <= 59 (default=0)

`ss` = seconds <= 59 (default=0)

`:` = defaults to current time

**Output:**

none

**Input and Output:**

none

**Output:**

`hour` - Integer hours in format `[-+]hh`, or 0 if invalid.

`min` - Integer minutes in format `[-+]mm`, or 0 if invalid.

`sec` - Integer seconds in format `[-+]ss`, or 0 if invalid.

**Return values:**

+ 400 - successful return of given time
+ 410 - current time returned for given `:` token
+ 401 - illegally formatted time (invalid char)
+ 409 - no given time returns value of 0
+ 421 - illegally formatted hours within given time
+ 423 - illegal hours exceeds system integer limits
+ 431 - illegally formatted minutes within given time
+ 433 - illegal minutes > 59
+ 441 - illegally formatted seconds within given time
+ 463 - illegal seconds > 59
+ < -9999 - failure statuses

**Remarks:**

Acceptable Formats:

- `NULL, `, `*`, `-` - `-1`
  ```c
  >>> HH, MM, SS (current time)
  >>> hh, mm, 0
  >>> hh:mm
  >>> hh:mm:ss
  >>> hh, hh, hh:
  >>> hh, hh, hh:ss
  >>> mm, mm:
  >>> mm:ss
  >>> ss
  >>> 0, 0, 0
  >>> 0, 0, ss
  >>> 0, ss
  >>> 0, ss
  >>> 0, 0
  >>> 0
  ```

**Categories:**

- `UBER_INTERFACE`, `CONVERTER`, `UTILITY`, `DAY/TIME`

**Filename:**

`argconv.c`
Mcstrtohr

Name:
Mcstrtohr - Convert given time token to integer time format hhmmss.

Interface:
#include "acidas.h"

int Mcstrtohr(const char* token, int* ihr)

Input:
token - Given character string time token in formats:
        [+/-]hhmmss, [+/-]hh:mm, hh:mm:ss, hh:mm:ffff
        ignores leading blanks, and from any trailing blank on.
        Where hh = hours (default=0)
        mm = minutes <= 59 (default=0)
        ss = seconds <= 59 (default=0)
        fffff = fractions of an hour.
        E = exponent of B, e, D or d
        n = exponential scalar
        : = defaults to current time

Input and Output:
none

Output:
  - Integer time in format [+/-]hhmmss, or 0 if invalid.

Return values:
  400 - successful return of given time
  410 - current time returned for given ': ' token
  401 - illegally formatted time (invalid char)
  402 - illegal time exceeds system integer limits
  403 - no given time returns value of 0
  421 - illegally formatted hours within given time
  422 - illegal hours exceed system limits
  431 - illegally formatted minutes within given time
  432 - illegal minutes > 59
  433 - illegally formatted seconds within given time
  441 - illegal seconds > 59
  < -9999 - failure statuses

Remarks:
Acceptable Formats:
NULL, :, +,-, -

hh:mm, hh:mm:ss

hh:mm:ss, hh:mm:ss

hh, hh, hh:

hh:as

HHMMSS (current time)
hhmm00
hh:hh00
hh00as
mm, mm:

mm:ss, .ffffff

iss

E

Categories:
USER_INTERFACE, CONVERTER, UTILITY, DAY/TIME

Filename:
argconv.c
Mcredtoii

Name:
Mcredtoii - Convert given lat/lon token to integer type format dddmss.

Interface:
#include "mcidas.h"

int Mcredtoii(const char* token, int* ill)

Input:
token - Given character string lat/lon token in formats:
[-+]+ddd:mm:ss, [+-]+ddd:.fffff[+-]+n
ignores leading blanks, and from any trailing blank on.
Where ddd = degrees
mm = minutes <= 59 (default=0)
ss = seconds <= 59 (default=0)
ffff = fractions of a degree.
E = exponent of E, e, D or d
n = exponential scalar

Input and Output:
none

Output:
ill - Integer lat/lon in format [-+]+ddd:mm:ss, or 0 if invalid.

Return values:
500 - successful return of given lat/lon
-501 - illegally formatted lat/lon (invalid char)
-503 - illegal lat/lon exceeds system integer limits
-509 - no given lat/lon returns value of 0
-521 - illegally formatted degrees within given lat/lon
-523 - illegal degrees exceeds system limits
-531 - illegally formatted minutes within given lat/lon
-533 - illegal minutes > 59
-541 - illegally formatted seconds within given lat/lon
-543 - illegal seconds > 59
< -9999 - failure statuses

Remarks:

Acceptable Formats:
ddd:mm, ddd:mm:      ===>  [-] dddmm00
ddd:mm:ss, ddd:.fffff ===>  [-] dddmss
ddd, ddd:, ddd:     ===>  [-] ddd0000
ddd:ss              ===>  [-] ddd00ss
:mm, :mm:            ===>  [-] mm00
:mm:ss, .fffff      ===>  [-] mmss
:ss                 ===>  [-] ss
:                  ===>  0

Categories:
USER_INTERFACE, CONVERTER, UTILITY, NAVIGATION

Filename:
argconv.c
Mcstrootint

Name:
Mcstrootint - Convert given numeric token to integer type format.

Interface:
#include "mcidas.h"
int
Mcstrootint(const char* token, int* value)

Input:
token - Given numeric character token in standard McIDAS formats:
-(+)[i.E[+][-]s, 6x[xxxxxxxx]]
Ignores leading blanks, and from any trailing blank on.
Where i = integer part
f = fractional part
E = exponent of E, e, D or d
s = exponential scalar
x = hexadecimal digit 0-9 or A-F

Input and Output:
none

Output:
value - Integer value of token, or 0 if invalid, or rounded integer
value (status=-102)

Return values:
100 - successful return of given integer
110 - successful return of given hexadecimal
-101 - illegally formatted integer (invalid char)
-102 - illegal fractional integer
-103 - illegal number exceeds system integer limits
-109 - no given token returns value of 0
-111 - illegally formatted hexadecimal (invalid char)
-113 - illegal hex exceeds system integer limits
-119 - no given hex returns value of 0
< -9999 - failure statuses

Remarks:
Acceptable Formats:

[+-]i, i, iE[+][-]s, i.E[+][-]s, .E[+][-]s, $x[xxxxxxxx] >>> [-] i

Categories:
USER_INTERFACE, CONVERTER, UTILITY

Filename: argconv.c

Mcstrootiyd

Name:
Mcstrootiyd - Convert given date token to integer date format yyyyddd.

Interface:
#include "mcidas.h"
int
Mcstrootiyd(const char* token, int* iyd)

Input:
token - Given character string date token in formats:

- yyyy-mm-dd , mm/dd , dd/month/yyyy , dd/month
- yyyy-mm-dd , mm-dd , dd-month-yyyy , dd-month
- (leading - or - sign allowed)
Ignores leading blanks, and from any trailing blank on.
Where yyyy- optional year, defaults to current year.
mm - optional valid month of 1 thru 12,
defaults to current month
mon - valid calendar month (at least first 3 chars),
defaults to current month
dd - required valid day of year (1-365/366)

Input and Output:
none

Output:
iyd - Integer date in format [-]yyyyddd, or 0 if invalid.

Return values:
100 - successful return of given date
310 - current date returned for given '/' token
-301 - illegally formatted date (invalid char)
-303 - illegal date exceeds system integer limits
-309 - no given date returns value of 0
-321 - illegally formatted year within given date
-323 - illegal year exceeds system limits
-331 - illegally formatted month (mon) in given date
-341 - illegally formatted month (mm) within given date
-343 - illegal month (mm) not 1-12
-351 - illegally formatted day within given date
-353 - illegal day of given month
-363 - illegal day of given year
< -9999 - failure statuses

Remarks:

Categories:
USER_INTERFACE, CONVERTER, UTILITY

Filename: argconv.c
Mcstrtoiyd

Remarks:
Acceptable Formats:

- [~] YYYYDDD (today's date)
- [~] YYYYDDD (current year)
- [~] YYYYDDD (current month)
- [~] YYYYDDD (current century)
- [~] YYYYDDD (given date)
- invalid

Categories:
USER_INTERFACE, CONVERTER, UTILITY, DAY/TIME

Filename:
argconv.c

Mcuc2dev

Name:
Mcuc2dev - Convert given DEV= numeric uc value to its character value.

Interface:

- Include "mcidas.h"
- char
- Mcuc2dev(int dev)

Input:

- dev - Given DEV= numeric uc value.

Input and Output:

- none

Output:

- none

Return values:

- nonblank - Character DEV= value.
- blank - failure

Remarks:

- none

Categories:
- SIG_CONFIG

Filename:
argglo.c
Mcocupcase

Name: Mcocupcase - convert a string to all upper case letters

Interface:
#include "mcidas.h"
void Mcocupcase(char *string)

Input: none

Input and Output: string - string to be converted

Output: none

Return values: none

Remarks: none

Categories: conversion

Filename: mcalcupcase.c

Mcowrite

Name: Mcowrite - write file

Interface:
#include "mcidas.h"
int Mcowrite(char *file, int start, int count, void *target)

Input:
file - string containing file name
start - first byte in the file to write
count - number of bytes to write
target - location from which bytes will be written

Input and Output: none

Output: none

Return values:
0 - success
<0 - failure

Remarks:
Uninitialized bytes in the file are set to the value 0x80.
In Fortran, use lbo() or lwo().

Categories: file

Filename: mcowrite.c
**abort**

Name: abort - terminates the execution of a routine

Interface: subroutine
```
abort(integer icode)
```

Input: icode - error status code

Input and Output: none

Output: none

Return values: none

Remarks: none

Categories: system

Filename: abort.for

---

**arabox**

Name: arabox - obtains a rectangular array of data from a McIDAS area file

Interface: subroutine
```
arabox(integer area, integer fslin, integer lslin, integer fsel, integer lsel, integer band, character(*) pqty, integer spac, integer snele, integer buf(*)
```

Input: area - McIDAS area number
fslin - first line of data to obtain (0 based)
lslin - last line of data to obtain
fsel - first element of data to obtain (0 based)
lsel - last element of data to obtain
band - band number (0 for non-banded data)
pqty - physical quantity ('RAW', 'RAD', 'TEMP', ...)
space - number of bytes used to store the data value (1, 2, or 4)
snele - element dimension of the array that will contain the data

Input and Output: none

Output: buf - array containing the data

Return values: none

Remarks: This routine does not initialize the output array.

Categories: image

Filename: arabox.for
**aranam**

**Name:**
aranam - create a filename from a McIDAS area number

**Interface:**
integer function
aranam(integer area, character*12 cname)

**Input:**
area - McIDAS area number
Output:
cname - filename

**Return values:**
0 - success
1 - failure, McIDAS area number not in valid range

**Remarks:**
none

**Categories:**
image, converter

**Filename:**
aranam.for
**araopt**

*Name:*  
araopt - sets the options and calibration for reading a McIDAS area

*Interface:*  
subroutine  
araopt(integer area, integer nopt, character*(*) copt(*), integer val(*))

*Input:*  
area - McIDAS area number  
nopt - number of options to be set  
copt - names of the options to be set  
val - value to set each option to

*Input and Output:*  
none

*Output:*  
none

*Return values:*  
none

*Remarks:*  
The valid options and values are:  

- **'SPAC'** spacing of output data (1, 2 or 4 bytes)  
- **'PREC'** precision of stored data (1, 2 or 4 bytes)  
- **'UNIT'** output physical quantity ('TEND', 'BRIT', 'RAW',....)  
- **'SCAL'** scale factor to apply to data  
- **'CALB'** name of package to calibrate data (1-4 character name)

Be sure that you have opened the area (OPNARA()) prior to calling this routine.

*Categories:*  
image, calibration

*Filename:*  
araopt.f
bsquez

Name:
bequez - squeezes excess blanks out of the given string

Interface:
subroutine
bequez(character*[*] cstr)

Input:
one

Input and Output:
cstr - on input contains string to squeeze; on output contains the squeezed string

Output:
one

Return values:
one

Remarks:
Removes all leading blanks and replaces multiple blanks with one blank.

Categories:
utility

Filename:
bequez.for

bttnwt

Name:
bttnt - checks if the mouse buttons are released

Interface:
subroutine
bttnwt()

Input:
one

Input and Output:
one

Output:
one

Return values:
one

Remarks:
This routine checks if either mouse button is depressed. If it is then it waits and checks again. This continues until the button has been released.

Categories:
user interface, utility

Filename:
bttnt.for
**cfd**

**Name:**
cfd - convert a double precision number, in D format, to a character*12 with leading blanks

**Interface:**
character*12 function
cfd(double precision xvalue, integer idec)

**Input:**
  - xvalue - the double precision number to convert
  - idec - number of decimal places

**Input and Output:**
none

**Output:**
none

**Return values:**
the character representation of the double precision number

**Remarks:**
Examples of the output are: +.172E3, +.95E-16, -.10000E-4.

Four places are reserved in the output for the exponent, even if they are not all used.

The number of fractional digits in the output string can be between 0 and 6.

**Categories:**
utility
converter

**Filename:**
cfd.for

---

**cff**

**Name:**
cff - convert a double precision number, in F format, to a character*12 with leading blanks

**Interface:**
character*12 function
cff(double precision dvalue, integer idec)

**Input:**
  - dvalue - the double precision number to convert
  - idec - number of decimal places

**Input and Output:**
none

**Output:**
none

**Return values:**
the character representation of the double precision number

**Remarks:**
none

**Categories:**
utility
converter

**Filename:**
cff.for
**cfi**

**Name:**
- cfi - convert a integer number, in I format, to a character*12 with leading blanks

**Interface:**
- character*12 function
  - cfi(integer i)

**Input:**
- i - the integer number to convert

**Input and Output:**
- none

**Output:**
- none

**Return values:**
- the character representation of the integer number

**Remarks:**
- This routine can only convert integers between -9999999999 and 9999999999.

**Categories:**
- utility
- converter

**Filename:**
- cfi.for

**cfj**

**Name:**
- cfj - convert a integer number, in I format, to a character*12 with leading zero

**Interface:**
- character*12 function
  - cfj(integer i)

**Input:**
- i - integer number to convert

**Input and Output:**
- none

**Output:**
- none

**Return values:**
- the character representation of the integer number

**Remarks:**
- This routine can only convert integers between -9999999999 and 9999999999.

**Categories:**
- utility
- converter

**Filename:**
- cfj.for
**cfr**

**Name:**

*cfr - convert a integer to a character*12 with trailing blanks

**Interface:**

character*12 function
cfr(integer 1)

**Input:**

1 - integer number to convert

**Input and Output:**

none

**Output:**

none

**Return values:**

the character representation of the integer number

**Remarks:**

none

**Categories:**

utility
cf

**Filename:**

cfr.for

---

**cfz**

**Name:**

*cfz - converts an integer number to a character*12 hexadecimal with leading blanks

**Interface:**

character*12 function
cfz(integer 1)

**Input:**

1 - integer number to convert

**Input and Output:**

none

**Output:**

none

**Return values:**

the character (in hexadecimal) representation of the integer number

**Remarks:**

none

**Categories:**

utility
cf

**Filename:**

cfz.for
ckwp

Name:
ckwp - Get a program keyword parameter in character string form.

Interface:
character(*) function
ckwp(character(*) keyword, integer position, character(*) def)

Input:
keyword
- The name of a keyword, or NULL/blank for positional.
position
- Specifies which argument from argument sequence to take
  (1-based), e.g., '1',1 specifies first positional param.
  '0',0 gives the program name, otherwise
  'key'.0 gives the keyedin keyword name preceded by an '='
def
- Character to use as missing value.

Input and Output:
none

Output:
none

Return values:
The desired argument's character value.

Remarks:
Obsolete routine, replaced by mcmdstr.
Returns character string representing desired parameter, or def if parameter is
missing.
If luc(-30) is set, and keyword not keyedin, then try to fetch argument from the
system string table.

Categories:
USER_INTERFACE

Filename:
argold.c

clean

Name:
clean - changes all control characters to blanks in the given string

Interface:
subroutine
clean(character(*) cline)

Input:
none

Input and Output:
cline
- on input contains string to modify; on output
  contains string where control characters are now blanks

Output:
none

Return values:
none

Remarks:
none

Categories:
utility

Filename:
cleanu.f
**clit**

**Name:**
clit - converts an integer to a character string

**Interface:**
character*4 function
clit(integer l)

**Input:**
l - integer to convert to a character string

**Input and Output:**
none

**Output:**
none

**Return values:**
none

**Remarks:**
none

**Categories:**
convert

**Filename:**
clit.for

---

**clsara**

**Name:**
clsara - closes a McIDAS area

**Interface:**
subroutine
clsara(integer area)

**Input:**
area - McIDAS area number

**Input and Output:**
none

**Output:**
none

**Return values:**
none

**Remarks:**
After calling this routine the only area access routine that will work is opnara().

**Categories:**
image, file

**Filename:**
clsara.for
**copcod**

**Name:**

copcod - copy audit trail cards from one McIDAS area to another

**Interface:**

subroutine
copcod(integer sarea, integer darea)

**Input:**
sarea - McIDAS source area number
darea - McIDAS destination area number

**Input and Output:**

none

**Output:**

none

**Return values:**

none

**Remarks:**

To get the audit trail cards in the correct order call copcod() then call stamp().

**Categories:**

image

file

**Filename:**

copcod.for

---

**cotan**

**Name:**

cotan - calculates the cotangent

**Interface:**

real function
cotan(real x)

**Input:**

x - the angle

**Input and Output:**

none

**Output:**

none

**Return values:**

the cotangent of x

**Remarks:**

This routine is needed since it is not part of standard Fortran 77.

**Categories:**

utility

**Filename:**

cotan.for
cpp

Name:

cpp - Get positional parameter in character form.

Interface:

character*() function
cpp(integer position, character* def)

Input:

position - Specifies which argument from argument sequence to take (1-based), e.g., 1 specifies first positional param.
0 gives the program name

def - Character to use as missing value.

Input and Output:

none

Output:

none

Return values:

The desired argument's character value.

Remarks:

Obsolete routine, replaced by mexpand.
Returns the 'position'th positional parameter, or def if missing.

Categories:

USER_INTERFACE

Filename:

argold.c

cqfid

Name:

cqfid - Get quote field from keyin (or return ' ' if not there).

Interface:

subroutine

cqfid(character(*) value)

Input:

none

Input and Output:

none

Output:

value - Field to extract quote from. If no quote, returns '"'.

Return values:

none

Remarks:

First char of quote string will be ' if quote field present.
Obsolete routine, replaced by mexpand.

Categories:

USER_INTERFACE

Filename:

argold.c
**czfi**

**Name:**
czfi - convert a integer number to a character*12 with leading zeros

**Interface:**
character*12 function
czfi(integer i)

**Input:**
i - the integer number to convert

**Input and Output:**
one

**Output:**
one

**Return values:**
the character representation of the integer number

**Remarks:**
This routine can only convert integers between -9999999999 and 9999999999.

**Categories:**
utility
converter

**Filename:**
czfi.for

---

**ddest**

**Name:**
ddest - Display text and an optional number on McIDAS debug output.

**Interface:**
subroutine
ddest(character*, integer num)

**Input:**
msg - Given text to display.
num - Given number to display after text if not 0.

**Input and Output:**
one

**Output:**
one

**Return values:**
one

**Remarks:**
one

**Categories:**
USER_INTERFACE

**Filename:**
ddest.for
dfline

Name:

dfline - Display ibuf to iframe at iline

Interface:

subroutine
dfline(integer iframe, integer iline, integer(*) ibuf)

Input:

iframe - frame to display the data in
iline - line number of data to display
ibuf - buffer containing data to display

Input and Output:

none

Output:

none

Return values:

none

Remarks:

Display image data in a frame. Note: there are some entry points contained at
the end, to satisfy discrepancies between OS/2 and Unix versions of McIDAS.

Categories:

image
display

Filename:

dfline.for

dkwp

Name:

dkwp - Get keyword parameter in double precision form.

Interface:

double precision function

dkwp(character*(*) keyword, integer position, double precision def)

Input:

keyword - The name of a keyword, or NULL/blank for positional.
position - Specifies which argument from argument sequence to take
(i-based), e.g., "1" specifies first positional parameter.
def - Number to use as missing value.

Input and Output:

none

Output:

none

Return values:

value - The desired argument's double value, or 0 if invalid.

Remarks:

Obsolete routine, replaced by mcargdpl.

If libc(-30) is set, and keyword not keyed in, then tries to fetch argument from
the system string table.

Categories:

USER_INTERFACE

Filename:

argold.c
**dkwpfr**

**Name:**
dkwpfr - Get double precision time parameter, and check validity.

**Interface:**
double precision function
dkwpfr(character*(*), keyword, integer position, double precision def)

**Input:**
- keyword: The name of a keyword, or NULL/blank for positional.
- position: Specifies which argument from argument sequence to take (1-based), e.g., ',1 specifies first positional param.
- def: Double precision to use as missing value.

**Input and Output:**
one

**Output:**
one

**Return values:**
value: The desired argument’s hh.ffff value, or abort.

**Remarks:**
Obsolete routine, replaced by mcmdhr.

If luc(-30) is set, and keyword not keyedin, then try to fetch argument from the system string table.

**Categories:**
USER_INTERFACE

**Filename:**
argold.c

**dkwpfl**

**Name:**
dkwpfl - Get lat/lon keyword parameter in double precision form.

**Interface:**
double precision function
dkwpfl(character*(*), keyword, integer position, double precision def)

**Input:**
- keyword: The name of a keyword, or NULL/blank for positional.
- position: Specifies which argument from argument sequence to take (1-based), e.g., ',1 specifies first positional param.
- def: Number to use as missing value.

**Input and Output:**
one

**Output:**
one

**Return values:**
The desired argument’s ddd.ffff value, or abort.

**Remarks:**
Obsolete routine, replaced by mcmdfl.

If luc(-30) is set, and keyword not keyedin, then try to fetch argument from the system string table.

**Categories:**
USER_INTERFACE

**Filename:**
argold.c
**dpp**

**Name:**
- dpp - Get positional parameter in double precision form.

**Interface:**
- double precision function
dpp(integer position, double precision def)

**Input:**
- position - Specifies which argument from argument sequence to take (1-based), e.g., 1 specifies first positional param.
- def - Number to use as missing value.

**Input and Output:**
- none

**Output:**
- none

**Return values:**
- value - The desired argument's double value, or 0 if invalid.

**Remarks:**
- Obsolete routine, replaced by mcargdbl.

**Categories:**
- USER_INTERFACE

**Filename:**
- argold.c

---

**dpdhr**

**Name:**
- dpdhr - Get double precision time parameter, and check validity.

**Interface:**
- double precision function
dpdhr(integer position, double precision def)

**Input:**
- position - Specifies which argument from argument sequence to take (1-based), e.g., 1 specifies first positional param.
- def - Double precision to use as missing value.

**Input and Output:**
- none

**Output:**
- none

**Return values:**
- value - The desired argument's hh,ffff value, or abort.

**Remarks:**
- Obsolete routine, replaced by mcnddhr.

**Categories:**
- USER_INTERFACE

**Filename:**
- argold.c
**dpill**

**Name:**

dpill - Get double precision lat/lon parameter, and check validity.

**Interface:**

double precision function
dpill(integer position, double precision def)

**Input:**

  - **position**
    - Specifies which argument from argument sequence to take
    - (1-based), e.g., 1 specifies first positional param.
  - **def**
    - Double precision to use as missing value.

**Input and Output:**

none

**Output:**

none

**Return values:**

The desired argument's ddd.fffff value, or abort.

**Remarks:**

Obsoleted routine, replaced by mcmdill.

**Categories:**

USER_INTERFACE

**Filename:**

argold.c

---

**edest**

**Name:**

edest - Display text and an optional number on McIDAS error output.

**Interface:**

subroutine
edest(character*(*) msg, integer num)

**Input:**

  - **msg** - Given text to display.
  - **num** - Given number to display after text if not 0.

**Input and Output:**

none

**Output:**

none

**Return values:**

none

**Remarks:**

none

**Categories:**

USER_INTERFACE

**Filename:**

edest.for
erest

Name:
erest - put an enhancement on a frame

Interface:
subroutine
erest(integer frame, integer ttab(816))

Input:
frame - frame number
ttab - enhancement table

Input and Output:
none

Output:
none

Return values:
none

Remarks:
This routine takes care of the different display types.
The format of ttab is documented in the Data File Structures chapter in FRAMENI.tct.

Categories:
image
display
utility

Filename:
erest.for

fixchr

Name:
fixchr - handles byte swapping for nav and cal arrays

Interface:
fixchr(integer iadir(*), integer nwn, integer inav(*), integer nwc, integer ical(*))

Input:
iadir - area directory
nwn - number of 4-byte words in inav array
nwc - number of 4-byte words in ical array

Input and Output:
inav - previously byte-swapped navigation block array to modify
ical - previously byte-swapped calibration block array to modify

Output:
fixchr - error code

Return values:
0 - success

Remarks:
This routine is called after the navigation and calibration data streams have been acquired and swapped and their length is known. It swaps all known character strings in the navigation and calibration codicils.

Previously, there was an unwritten constraint on navigation and calibration that all data in navigation and calibration blocks was either 4-byte character strings (blank filled) or scaled integers (such that the leading byte was either < 30 or the scaled integer was negative). This permitted the use of ISCHR to determine if a word was a four byte character string, and if so, the word was not byte swapped when passed from a big-endian machine to a little-endian machine using SINVAR or IMOCOPY. With the advent of GVAR (GOES-8,9), this constraint was broken, and 4-byte reals were put into the calibration block.

This routine fixes the symptom of reals not being byte flipped when they can be interpreted as four-byte character strings on little-endian machines. It will handle all cases we can identify in advance, that were previously handled by a simple one line call to ISCHR in MCOGET when the old constraint was in force.

Categories:
IMAGE, NAVIGATION, CALIBRATION

Filename:
mcoget.for
flalo

Name:
flalo - short description of purpose/use/etc

Interface:
real function
flalo(integer m)

Input:
m    - description of it

Input and Output:
none

Output:
none

Return values:
0    - success

Remarks:
none

Categories:
utility
converter
navigation

Filename:
flalo.for

fntset

Name:
fntset - set font to given font name abbreviation.

Interface:
integer function
fntset(character[*] cabfont)

Input:
cabfont - Abreviated font name to be used.

Input and Output:
none

Output:
none

Return values:
0    - successful
1    - Given font name is invalid.
2    - Given font name does not exist.

Remarks:
none

Categories:
USER_INTERFACE, SYS_CONFIG, SYSTEM, DISPLAY

Filename:
main1.for
**fsize**

**Name:**
fsize - obtains the number of lines and elements for a frame

**Interface:**
subroutine

fsize(integer item, integer iframe, integer ilins, integer ieles)

**Input:**
item - terminal number
iframe - frame number

**Input and Output:**
none

**Output:**
ilins - number of lines for the frame
ieles - number of elements for the frame

**Return values:**
none

**Remarks:**
none

**Categories:**
display, utility

**Filename:**
fsize.for

---

**ftime**

**Name:**
ftime - convert integer format time to real format

**Interface:**
real function
ftime(integer m)

**Input:**
m - integer time (format: sign HH MM SS)

**Input and Output:**
none

**Output:**
none

**Return values:**
time contained in a real

**Remarks:**
none

**Categories:**
utility
converter
day/time

**Filename:**
ftime.for
getfrm

Name:
getfrm - obtain the frame directory entry

Interface:
subroutine
getfrm(integer frame, integer array(64))

Input:
frame - frame number

Input and Output:
none

Output:
array - frame directory

Return values:
none

Remarks:
If there is not a frame directory entry then the array is filled with -1.

Categories:
display

Filename:
getfrm.for

icget

Name:
icget - obtain an audit trail card from a McIDAS area

Interface:
ninteger function
icget(integer ia, integer icard(20))

Input:
ia - McIDAS area number

Input and Output:
none

Output:
icard - array containing audit trail card

Return values:
0 - success
-1 - all audit trail cards have been returned

Remarks:
This routine does not handle checking for a valid area number correctly. This routine will return 0 if the area number is not in the valid range. Before calling this routine call ARRANM().

Categories:
image, file

Filename:
icget.for
**ifband**

**Name:**
ifband - check if the specified band exists in a McIDAS area

**Interface:**
integer function
ifband(integer area, integer band)

**Input:**
area - McIDAS area number
band - band number

**Input and Output:**
none

**Output:**
none

**Return values:**
0 - the McIDAS area does contain the specified band
-1 - the McIDAS area does not contain the specified band

**Remarks:**
If the McIDAS area is not open then ifband() calls abort().

**Categories:**
image

**Filename:**
ifband.for

---

**igcurf**

**Name:**
igcurf - sets the current grid file number

**Interface:**
integer function
igcurf(integer gnof)

**Input:**
gnof - grid file number

**Input and Output:**
none

**Output:**
none

**Return values:**
the current grid file number

**Remarks:**
If the grid file number that is passed is not in the valid range of grid file numbers then the current grid file number is not changed.

**Categories:**
grid
utility

**Filename:**
igcurf.for
**igget**

Name: igget - get a grid from a grid file

Interface:

```
integer function igget(integer gfn0, integer gno, integer maxwds, integer grid(*),
integer nr, integer nc, integer table(64))
```

Input:

- `gfno` - grid file number
- `maxwds` - maximum number of words to read from grid

Input and Output:

- `gno` - grid number, on return the actual grid number of the grid returned

Output:

- `grid` - array containing the grid
- `nr` - number of rows in the grid
- `nc` - number of columns in the grid
- `table` - array containing the grid header

Return values:

- `0` - success
- `-1` - failure, invalid grid number
- `-2` - failure, invalid grid file number or the grid file does not exist

Remarks:

If `gno` is <0 it indicates that the requested grid is maximum grid - grid number. For example if the grid number is -2 then the requested grid is maxgrid-2. On return `gno` will be set to the actual grid number that it returned.

Categories:

- grid
- file
- utility

Filename:

`igget.f`  

---

**igggmax**

Name: igggmax - returns the maximum number of grids the grid file can hold

Interface:

```
integer function igggmax(integer gfno, integer start)
```

Input:

- `gfno` - grid file number

Input and Output:

- `start` - starting location of grid address pointers

Return values:

- `0` - failure, invalid grid file number
- `>0` - the maximum number of grids the grid file can store

Remarks:

If word 10 of the grid file is >= -1 then old format is assumed which implies that the grid file can hold 159 grids.

Categories:

- grid
- utility

Filename:

`iggmax.f`
iname

Name:  
iname - create the grid filename from the grid file number

Interface:  
subroutine  
imname(integer gfno, integer filnam(2))

Input:  
gfno - grid file number

Input and Output:  
none

Output:  
filnam - grid filename

Return values:  
none

Remarks:  
none

Categories:  
grid  
file  
utility

Filename:  
imname.for

igput

Name:  
igput - write a grid to a McIDAS grid file

Interface:  
integer function  
igput(integer gfno, integer igno, integer grid(*),  
integer nr, integer nc, integer table(*),  
integer gno)

Input:  
gfno - grid file number to write grid to  
igno - grid number to write to  
grids - array containing grid values to be filed  
r - number of rows in the grid  
cn - number of columns in the grid  
table - McIDAS grid header to be filled along with grid

Input and Output:  
none

Output:  
gno - actual grid number that data was written to

Return values:  
0 - success  
-1 - failure to write grid to file because either:  
a) igno is greater than the maximum number of  
grids that can be stored in the grid file.  
b) nr or nc is less than 0 and igno is less than 0.  
c) grid file is full.  
-2 - failure because grid file doesn't exist

Remarks:  
The grid header table must be filled by the application before igput is called.

The grids stored with igput are filed in column-major format with the northwest  
corner being point (1,1). Example, for a grid of 2 rows by 3 columns the ordering  
of storage would be (1,1), (2,1), (2,2), (1,2).  

The grid points are assumed to be 4 byte integer values.

If the value of nr or nc is less than 0 and igno is greater than 0 then the grid  
in location igno is DELETED.

If igno is greater than or equal to 0 igput will file in the first empty grid  
default available that has a grid number at least as high as the value in igno.  
If igno is less than 0 then the grid will be written to the absolute value of  
the grid number specified. If a grid previously existed at that location it will  
be overwritten.
ikwp

Name:

ikwp - Get keyword parameter in integer form.

Interface:

    integer function ikwp(character*(*keyword), integer position, integer def)

Input:

    keyword - The name of a keyword, or NULL/blank for positional.
    position - Specifies which argument from argument sequence to take
               (1-based), e.g., '1'. i specifies first positional param.
    def - Number to use as default.

Input and Output:

    none

Output:

    none

Return values:

    The desired argument's integer value, or 0 if invalid.

Remarks:

    obsolete routine, replaced by scargint.
    If luc(-10) is set, and keyword not keyedin, then trys to fetch argument from
    the system string table.

Categories:

    USER_INTERFACE

Filename:

    argold.c
ikwphr

Name:
ikwphr - Get integer time parameter, check validity.

Interface:
integer function
  ikwphr(character*(*), character*(*), integer position, integer def)

Input:
  keyword - The name of a keyword, or NULL/blank for positional.
pwd - Specifies which argument from argument sequence to take
  (1-based), e.g., '1.1 specifies first positional param.
def - Integer to use as missing value.

Input and Output:
none

Output:
none

Return values:
value - The desired argument's hhmmss value, or abort.

Remarks:
Obsolete routine, replaced by mcmdhr.

If luc(-30) is set, and keyword not keyedin, then tries to fetch argument from
the system string table.

Categories:
USER_INTERFACE

Filename:
argold.c

ikwpll

Name:
ikwpll - Get lat/lon keyword parameter in integer form.

Interface:
integer function
  ikwpll(character*(*), character*(*), integer position, integer def)

Input:
  keyword - The name of a keyword, or NULL/blank for positional.
pwd - Specifies which argument from argument sequence to take
  (1-based), e.g., '1.1 specifies first positional param.
def - Number to use as missing value.

Input and Output:
none

Output:
none

Return values:
value - The desired argument's ddmmss value, or abort.

Remarks:
Obsolete routine, replaced by mcmdll.

If luc(-30) is set, and keyword not keyedin, then tries to fetch argument from
the system string table.

Categories:
USER_INTERFACE

Filename:
argold.c
**ikwpyd**

**Name:**

ikwpyd - Get date parameter, check validity.

**Interface:**

integer function
ikwpyd(character*[*]) keyword, integer position, integer def

**Input:**

keyword - The name of a keyword, or NULL/blank for positional.
position - Specifies which argument from argument sequence to take
           (1-based), e.g., ',1 specifies first positional parem.
def - Number to use as missing value.

**Input and Output:**

none

**Output:**

none

**Return values:**

value - The desired argument's yyddd value, or abort.

**Remarks:**

Obsolete routine, replaced by scondyld.
Satellite number may be a part of the input (SSYYDDD), so user may need to check
parameter if not using 85. If luc(-30) is set, and keyword not keyedin, then tries
to fetch argument from the system string table.

**Categories:**

USERINTERFACE

**Filename:**

argold.c

---

**idcday**

**Name:**

incday - increment/decrement the date

**Interface:**

integer function
incday(integer yyddd, integer inc)

**Input:**

yyddd - date
inc - number of days to increment/decrement the date by

**Input and Output:**

none

**Output:**

none

**Return values:**

returns the new date

**Remarks:**

Only increments/decrements by days.

**Categories:**

converter
day/time

**Filename:**

incday.for
**ipp**

**Name:**
ipp - Get positional parameter in integer form.

**Interface:**
integer function
ipp(integer position, integer def)

**Input:**
- position
  - Specifies which argument from argument sequence to take (1-based), e.g., 1 specifies first positional param.
- def
  - Number to use as missing value.

**Input and Output:**
none

**Output:**
none

**Return values:**
value
- The desired argument's integer value, or 0 if invalid.

**Remarks:**
Obsolete routine, replaced by mcargint.

**Categories:**
USER_INTERFACE

**Filename:**
argold.c

---

**ipphr**

**Name:**
ipphr - Get integer time parameter, check validity.

**Interface:**
integer function
ipphr(integer position, integer def)

**Input:**
- position
  - Specifies which argument from argument sequence to take (1-based), e.g., 1 specifies first positional param.
- def
  - Integer to use as missing value.

**Input and Output:**
none

**Output:**
none

**Return values:**
value
- The desired argument's hhmms value, or abort.

**Remarks:**
Obsolete routine, replaced by mcmdihr.

**Categories:**
USER_INTERFACE

**Filename:**
argold.c
_NAME:

ippl - Get integer lat/lon parameter, and check validity.

INTERFACE:

integer function
ippl(integer position, integer def)

INPUT:

position - Specifies which argument from argument sequence to take
           (1-based), e.g., 1 specifies first positional param.
def - Integer to use as missing value.

INPUT and OUTPUT:

none

OUTPUT:

none

RETURN VALUES:

value - The desired argument's ddmms value, or abort.

REMARKS:

Obsolete routine, replaced by mmcndll.

CATegories:

USER_INTERFACE

FILENAME:

argold.c

---

_NAME:

ippyd - Get date parameter, check validity.

INTERFACE:

integer function
ippyd(integer position, integer def)

INPUT:

position - Specifies which argument from argument sequence to take
           (1-based), e.g., 1 specifies first positional param.
def - Number to use as missing value.

INPUT and OUTPUT:

none

OUTPUT:

none

RETURN VALUES:

value - The desired argument's yyddd value, or abort.

REMARKS:

Obsolete routine, replaced by mmcndiyd.
Setellite number may be a part of the input (SSYYDDD), so user may need to check parameter if not using SS.

CATegories:

USER_INTERFACE

FILENAME:

argold.d

---

Applications Programming Interface
5-170

McIDAS Programmer's Manual
Preliminary Issue 10/93

McIDAS Programmer's Manual
Preliminary Issue 10/93

Application Programming Interface
5-171
**kwnams**

**Name:**
kwnams - Return all keyword names occurring in keyin line, except DEV*.

**Interface:**
subroutine
kwnams(integer maxkey, integer numkey, character(*) keywords(maxkey))

**Input:**
maxkey - Dimension size of keywords array.

**Input and Output:**
none

**Output:**
umkey - # of keyword names returned, or 0 if error.
keywords - Array to contain keyword names (except for DEV*).

**Return values:**
none

**Remarks:**
Obsolete routine, replaced by mccmdnam.
Eliminates DEV* from the array of keyword names.
If the array size is exceeded, then returns numkey-maxkey.

**Categories:**
USER_INTERFACE

**Filename:**
argold.c

---

**kwpkey**

**Name:**
kwpkey - Convert old arg-fetching keyword name to new format, and turn on system string table lookup.

**Interface:**
extern char*
kwpkey(const char* keyword, PsLen* sizkey)

**Input:**
keyword - Given keyword name, or NULL/blank for positional.

**Input and Output:**
sizkey - Given fortran character(*) size of 'keyword', updated to length of returned keyword name.

**Output:**
none

**Return values:**
Pointer to keyword name in new arg-fetching format, or NULL if error.

**Remarks:**
If keyword name doesn't end with a blank, then appends an '*' to end.
Given positional keyword of NULL/blank will return a null string, '*'.
Turns on system string table lookup.

**Categories:**
USER_INTERFACE

**Filename:**
argold.c
**Ibcchk**

**Name:**
Ibcchk - make the string table for that terminal number active

**Interface:**

```c
subroutine Ibcchk()
```

**Input:**
none

**Input and Output:**
none

**Output:**
none

**Return values:**
none

**Remarks:**
none

**Categories:**
utility

**Filename:**
ibcchk.for

---

**Ibget**

**Name:**
Ibget - obtain a string from the current string table

**Interface:**

```c
integer function Ibget(character(*) cname, character(*) cout)
```

**Input:**

cname - string name

**Input and Output:**
none

**Output:**

```c
cout - string
```

**Return values:**

```c
>0 - success, returns the length of the string
-1 - failure, unable to find string name
```

**Remarks:**
The string name must be 12 characters or less.

**Categories:**
utility

**Filename:**
ibget.for
Ibi

Name:
Ibi - read file, using byte addressing

Interface:
integer function
Ibi(character* (* file, integer start, integer count, integer buf(*))

Input:
file - string containing file name
start - first byte in the file to read
count - number of bytes to read

Input and Output:
none

Output:
buf - location into which bytes will be read

Return values:
0 - success
-1 - failure, or some data was past end of file

Remarks:
Uninitialized bytes in the file are read as if they had the value 0x80.
A byte count less than 1 or first byte less than 0 is an error.
In C, use Mread().

Categories:
file

Filename:
mcread.c

Ibo

Name:
Ibo - write file, using byte addressing

Interface:
integer function
Ibo(character* (* file, integer start, integer count, integer buf(*))

Input:
file - string containing file name
start - first byte in the file to write
count - number of bytes to write
buf - location from which bytes will be written

Input and Output:
none

Output:
none

Return values:
0 - success
<0 - failure

Remarks:
Uninitialized bytes in the file are set to the value 0x80.
In C, use Mwrite().

Categories:
file

Filename:
mfwrite.c
**Ibopen**

**Name:**
ibopen - clearing out the string table and setting it to use string table associated with the terminal number

**Interface:**
- subroutine
  - ibopen(integer item)

**Input:**
- item - terminal number

**Input and Output:**
- none

**Output:**
- none

**Return values:**
- none

**Remarks:**
If the current string table is the same as the requested table nothing is done.

**Categories:**
- utility

**Filename:**
- ibopen.for

---

**Input**

**Name:**
input - write the string out to the current string table, or delete a string from the current string table

**Interface:**
- integer function
  - input(character*(*) cnme, character*(*) cnout)

**Input:**
- cnme - string name
- cnout - string

**Input and Output:**
- none

**Output:**
- none

**Return values:**
- 0 - success
- -1 - failure, trying to overwrite system strings or trying to delete a string and the string does not exist

**Remarks:**
If cnout is greater than 1 in length, this implies that you want to add the string to the current string table.
If cnout is less than or equal to 1 in length, this implies that you want to delete the string from the current string table.

**Categories:**
- utility

**Filename:**
- input.for
**Ibtbl**

**Name:**
Ibtbl - Obtain the string names and lengths of each string in the string table.

**Interface:**
integer function
Ibtbl(integer lens(256), character*12 cnms(256))

**Input:**
none

**Input and Output:**
none

**Output:**
lens - the length of the strings in the string table
cnms - the string names in the string table

**Return values:**
>0 - success, number of strings in the table
<=0 - failure

**Remarks:**
none

**Categories:**
utility

**Filename:**
lbtbl.frd

---

**Luc**

**Name:**
Luc - Lookup user common for given index.

**Interface:**
integer function
Luc(integer index)

**Input:**
index - Given user common index.

**Input and Output:**
none

**Output:**
none

**Return values:**
Desired user common value.

**Remarks:**
Returns the value currently stored in McIDAS common storage.
For positive indexes, the value is from session based common; for negative indexes, it is from a process-based common. The index 0 is treated differently between UNIX and os2.

**Categories:**
csys_config
sysconfig

**Filename:**
Luc.c
**lwc**

**Name:**

lwc - create file

**Interface:**

integer function lwc(character*('') name)

**Input:**

name - name of the file to create

**Input and Output:**

none

**Output:**

none

**Return values:**

0 - success, file created  
<0 - failure, file not created (it may already exist)

**Remarks:**

If lwc() creates the file, it writes four missing code (0x80) bytes to the file  
so that the file does not have zero length.  
This prevents lwxop() from deleting the file.

**Categories:**

c file

**Filename:**

lwc.for

---

**lwcopy**

**Name:**

lwcopy - copy one file to another

**Interface:**

integer function lwcopy(character*('') cfile1, character*('') cfile2)

**Input:**

cfile1 - source file name  
cfile2 - destination file name

**Input and Output:**

none

**Output:**

none

**Return values:**

0 - success  
<0 - failure code from lwo()

**Remarks:**

none

**Categories:**

c file

**Filename:**

lwcopy.for
Name: lwd - remove a file

Interface:
integer function lwd(character*[*]) name

Input:
name - name of file to remove

Input and Output:
none

Output:
none

Return values:
  0  - success, removed
  <0  - failure, not removed

Remarks:
The C version of this routine is Mcrmov().

Categories:
file

Filename: mdcache.c

Name: lwend - return number of last page in file

Interface:
integer function lwend(character*[*]) name

Input:
name - file name

Input and Output:
none

Output:
none

Return values:
  >0  - success, number of last page in the file
  <0  - failure, file does not exist

Remarks:
The page number is zero-based. Pages are 4096 bytes long.

Categories:
file

Filename: lwend.for
lwendw

Name:
lwendw - return number of last word in file

Interface:
integer function
lwendw(character*[*]) name

Input:
name - file name

Input and Output:
none

Output:
none

Return values:
>=0 - success, number of last word in the file
<0 - failure, file does not exist

Remarks:
The word number is zero-based.
Words are 4 bytes long.

Categories:
file

Filename:
lwendw.for

IwI

Name:
lwI - read file, using word addressing

Interface:
integer function
dlwi(character[*] file, integer start, integer count, integer buf[*])

Input:
file - string containing file name
start - first word in the file to read
count - number of words to read

Input and Output:
none

Output:
buf - location into which words will be read

Return values:
0 - success
-1 - failure, or some data was past end of file

Remarks:
Uninitialized bytes in the file are read as if they had the value 0x80.
A word count less than 1 or a first word less than 0 is an error.
In C, use Mcread.

Categories:
file

Filename:
mcread.c
Iwo

Name:
Iwo - write file, using word addressing

Interface:
native function
Iwo(character**) file, integer start, integer count, integer buf(*)

Input:
file - string containing file name
start - first word in the file to write
count - number of words to write
buf - location from which bytes will be writ

Input and Output:
none

Output:
none

Return values:
0 - success
<0 - error

Remarks:
Uninitialized bytes in the file are set to the value 0x80.
In C, use Ncwrite().

Categories:
file

Filename:
ncwrite.c

main1

Name:
main1 - Set up state before calling users main8.

Interface:
snout
main1(integer givcmd(*), integer givlen)

Input:
givcmd - Given McIDAS command line data structure.
givlen - Given length of givcmd.

Input and Output:
none

Output:
none

Return values:
none

Remarks:
none

Categories:
USER_INTERFACE, SYSTEM, SYS_CONFIG

Filename:
main1.far
**mcacal**

Name:
mcacal - reads the calibration parameters from the current AREA

Interface:
integer function
mcacal(integer handl, integer line(*))

Input:
handl - identifier of this data stream

Input and Output:
none

Output:
line - array containign the calibration parameters

Return values:
0 - always returns 0

Remarks:
This call may be made at any time after mcaget.

You do not need to make this call if you do not need the calibration.

Categories:
IMAGE

Filename:
mcaget.for

**mcacou**

Name:
mcacou - transmit comment cards to output area

Interface:
integer function
mcacou(integer data(*))

Input:
data - array of comment card information

Input and Output:
none

Output:
none

Return values:
0 - success

Remarks:
The array contains all comment cards, 80 bytes each this must be called after
all the data is transmitted this must not be called if the area directory
entry specified that there were no comment cards.

Categories:
words from the list

Filename:
mcacou.for
mcacrd

Name:
mcacrd - reads the 'comment cards' from the AREA

Interface:
integer function
mcacrd(integer hand1, integer line(*))

Input:
hand1 - identifier of this data stream

Input and Output:
none

Output:
line - array containing the comment cards

Return values:
0 - always returns 0

Remarks:
No other calls to other handles should come between mcclin() and mcacrd() this
call should be made after all the lines have been read.

Categories:
IMAGE

Filename:
mcacrd.for

mcadel

Name:
mcadel - delete image files in a ADDE object

Interface:
integer function
mcadel(char name, int nsorts, char(*) sorts, integer msgflag )

Input:
name - description of it
nsorts - number of sort clauses, may be 0
sorts - strings which further modify the request
msgflag - if 0, means emit no messages, only return codes

Input and Output:
none

Output:
none

Return values:
0 - success
-1 - no delete privileges
-2 - bad clause in delete request
<-2 - server error code

Remarks:
This transaction opens a server connection.

Categories:
image

Filename:
mcadel.for
mcadir

Name:
mcadir - returns McIDAS AREA's directory structure from image data objects

Interface:
integer function
mcadir(character(*) name, integer nsorts, character(*) sorts(*),
integer msgflag)

Input:
name - name or nickname of data object
nsorts - number of sort clauses
sorts - strings which modify the request
msgflag - emit messages, 0 means no message

Input and Output:
none

Output:
none

Return values:

- success
-1 - failure, server does not accept your authorization
-2 - failure, server does not handle this data type
-50 - failure, server cannot find data with this name
-51 - failure, no AREA files in data object fit your search criteria
-52 - failure, too many areas would be searched, transaction cannot proceed
-53 - failure, the format of the SORT clause for TIME has a bad format
-100 - failure, communication to the server failed
other - failure, unique return codes from lower level routines

Remarks:
This transaction opens a connection. mcadir() is called one or more times to receive the output.

Categories:
IMAGE

Filename:
mcadir.for

mcadrd

Name:
mcadrd - returns one AREA directory and comment cards

Interface:
integer function
mcadrd(integer data(*), integer cards(*))

Input:
none

Input and Output:
none

Output:
data - array containing area directory
cards - array containing comment cards

Return values:

0 - success
1 - failure, unexpected end of data
-96 - communications with server timed out
-97 - connection to server has been broken

Remarks:
Word 1 is the absolute AREA number from the server.
Word 2 is the relative AREA number in the data object.
Word 3-65 are words 2-64 from the AREA directory entry.
Since only valid AREA directory entries are sent word 1 of the directory will always be 0, so it is not sent.

Call mcadrd() prior to calling this routine.

Categories:
image
system

Filename:
mcadrd.for
mcaget

Remarks:
This call opens a transaction. Other calls get data line-by-line.

Word 1 of the area directory returned in IADIR is overlaid with the relative position number of the data within the area object specified by NAME. This word should be reset to 0 if IADIR is to be written to disk as part of an actual area.

The handle returned by mcaget is used on moaln, etc. calls to identify which data stream the data is being retrieved from. This is because more than one area data stream is allowed to be open simultaneously. The handle is similar to a file descriptor, but it is not a file descriptor and cannot be used outside of this context.

Categories: IMAGE

Filename: mcaget.for

Name:
mcaget - start the transaction of receiving AREA data

Interface:
Integer function
mcaget(character name, integer nsort, character sort(*), character unit, character form, integer maxbyt, integer msgflag, integer ladir(*), integer handi)

Input:
nname - DOE object name of area data
nsort - number of clauses in sort array
sort - array of clauses which modify the request
unit - calibration units requested, or blank
form - values are 11 12 14 or blank
maxbyt - the size of data which may be returned per line
msgflag - zero means suppress error messages

Input and Output:
none

Output:
ladir - array containing the area directory
handi - handle which application must use to indicate this data stream

Return values:
1 - success
-1 - failure, server did not validate you to make the request
-2 - failure, server does not handle this data type
-30 - failure, no AREA found matching search request
-31 - failure, navigation error
-32 - failure, data request does not intersect data available
-33 - failure, size requested makes no sense
-34 - failure, requested band is not present
-35 - failure, a bad SORT clause was encountered
-36 - failure, MAIRRT is too small for lines which would be returned
-37 - failure, illegal format, not 11, 12, or 14
-39 - failure, open AREA limit exceeded
-40 - failure, calibration was needed but not found
-41 - failure, RAR table was needed but not found
-42 - failure, time ordered search over too large a range
-100 - failure, the server could not be contacted
mcname

Name:
mcname - construct image file data set name and position from
frame directory entry

Interface:
integer function
mcname(integer frame, character*(* name, character*(* pos)

Input:
frame - image frame number

Input and Output:
none

Output:
name - data set name from frame directory
pos - position of image within the data set

Return values:
0 - success
-1 - invalid frame

Remarks:
Image must be displayed using IMODISP application.

Categories:
image

Filename:
mcname_.c

mcnav

Name:
mcnav - reads the navigation from the current AREA

Interface:
integer function
mcnav(integer handl, integer line(*))

Input:
handl - identifier of this data stream

Input and Output:
none

Output:
line - array containing the navigation parameters

Return values:
0 - always returns 0

Remarks:
This call may be made at any time after mceget.
You do not need to make this call if you do not need the navigation parameters.

Categories:
IMAGE

Filename:
mceget.for
mcanum

Name:
mcanum - return number of images in a dataset

Interface:
integer function
mcanum(character*(** name))

Input:
nname - name of the dataset

Input and Output:
none

Output:
none

Return values:
<0 - error from server, probably name resolution error
>0 - number of images in the dataset

Remarks:
This will return the number of actual images, not the range of images this takes
up on the server. There may be wholes in the dataset, which would not reflect in
the range.

Categories:
image

Filename:
mcanum.for

mcaout

Name:
mcaout - send a line of data to area transaction started by mcaput

Interface:
integer function
mcaout(integer line(*) )

Input:
line - array containing preamble and data for current line

Input and Output:
none

Output:
none

Return values:
0 - success

Remarks:
Important use info, algorithm, etc.

Categories:
area, api

Filename:
mcaput.for
**mcapfx**

**Name:**
mcapfx - reads the prefix from the current line

**Interface:**
integer function
mcapfx(integer handl, integer line(*))

**Input:**
handl - identifier of this data stream

**Input and Output:**
one

**Output:**
line - the prefix for the current line

**Return values:**
0 - this routine always returns 0

**Remarks:**
No other calls to other handles should come between mcsel() and mcapfx() if you don't need the prefix, you do not have to call this routine.

**Categories:**
IMAGE

**Filename:**
mcapx.for

---

**mcaput**

**Name:**
mcaput - begin transaction of transferring an area outbound

**Interface:**
integer function
mcaput(char name, int nsort, char sort(*),
int areadir(64), int nav(*), int cal(*))

**Input:**
name - DDE trunename, nickname, or absolute area number
nsort - number of sort clauses in sort array
sort - array of strings which modify the request
areadir - area directory entry for target area
nav - navigation codicil for the target area
cal - calibration codicil for the target area

**Input and Output:**
one

**Output:**
one

**Return values:**
0 - success

**Remarks:**
Depending on the contents of areadir, nav and/or cal arrays may be empty.

**Categories:**
area, api

**Filename:**
mcaput.for
mcargcmd

Name:
mcargcmd - Build and return a McIDAS command line for the given handle.

Interface:
character(*) function
mcargcmd(integer arg_handle)

Input:
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.

Input and Output:
none

Output:
none

Return values:
McIDAS command line, or NULL.

Remarks:
Builds a McIDAS command line, from the parsed arg-fetching structure.
The returned text will be formatted in McIDAS command line syntax.

Categories:
IIDM_INTERFACE

Filename:
argutil.c

mcargdbl

Name:
mcargdbl - Fetch an argument in double type format.

Interface:
integer function
mcargdbl(integer arg_handle, character(*) keyword, integer position,
double precision def, double precision min, double precision max,
double precision value, character(*) arg)

Input:
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name in key.word format, or blank for
position - Given argument position within the keyword, or
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for min<value<max.

Output:
value - The desired argument's double value, or 0 if invalid.
arg - Pointer to global argument text string.

Return values:
200 - given double default argument returned
1200 - keyedin double argument returned
1210 - keyedin hexadecimal argument returned
-204 - given double default argument < given min
-205 - given double default argument > given max
-1201 - keyedin argument is illegal decimal format
-1203 - keyedin arg exceeds system double limits
-1204 - keyedin double argument < given min
-1205 - keyedin double argument > given max
-1211 - keyedin arg is illegal hexadecimal format
-1213 - keyedin hex exceeds system integer limits
-1214 - keyedin hexadecimal argument < given min
-1215 - keyedin hexadecimal argument > given max
< -9999 - failure statuses
mcargdhr

Name:
mcargdhr - Fetch an argument in double fractional hours format hh.ffffff.

Interface:
decl function
mcargdhr(integer arg_handle, character*(*), keyword, integer position,
double precision def, double precision min,
double precision max, double precision value, character*(*) arg)

Input:
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name, in key.word format, or blank for positional arg, where key=minimum keyword name,
position - Given argument position within the keyword, or positional argument's place.
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for min=value=min, min-max defaults to no given range checking.

Output:
value - Double time argument in format [-]hh.ffffff, or 0.
arg - Global argument text string.

Return values:
450 - given hh.ffffff time default arg returned
1450 - keyedin time argument returned as hh.ffffff
1460 - keyedin current hh.ffffff default returned
454 - given time default argument < given min
455 - given time default argument > given max
1453 - keyedin argument is illegal time format
1454 - keyedin time arg exceeds system double limits
1455 - keyedin time argument < given min
1456 - keyedin time argument > given max
1464 - keyedin current hh.ffffff default < given min
1465 - keyedin current hh.ffffff default > given max
1470 - keyedin time has illegally formatted hours
1473 - keyedin hours exceeds system integer limits
1481 - keyedin time has illegal minutes format
1483 - keyedin minutes > 59
1493 - keyedin time has illegal seconds format
< -9999 - failure statuscodes
mcargdl

Remarks:
Accepts time arguments in formats:
[-]hh:mm:ss , [-]hh.H.FFFF (-:n)
Ignores leading blanks, and any trailing blank on.
Where hh = hours (default=0)
mm = minutes <= 59 (default=0)
ss = seconds <= 59 (default=0)
fff = fractions of an hour.
E = exponent of E, e, D or d
n = exponential scalar
= defaults to current time

Acceptable Formats:

\n
Categories:
USER_INTERFACE, DAY/TIME

Filename:
argfetch.c

mcargdl

Name:
mcargdl - Fetch argument in double fractional lat/lon format ddd.ffffff.

Interface:
integer function
mcargdl(integer arg_handle, character(*) keyword, integer position,
double precision def, double precision min,
double precision max, double precision value, character(*) arg)

Input:
arg.handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name, in keyword format, or blank
for positional arg, where key-minimum keyword name,
word-maximum name.
position - Given argument position within the keyword, or
positional argument's place.
def - Given default value to be returned if arg not found.
    If not valid, will use it, but returns an error status.
min, max - Given acceptable arg range, tests for min-value=max.
    min-max defaults to no given range checking.

Input and Output:
none

Output:
value - Double lat/lon argument in format [-]ddd.ffffff, or 0.
arg - Global argument text string.

Return values:
550 - given ddd.ffffff lat/lon default arg returned
1550 - keyedIn lat/lon arg returned as ddd.ffffff
-554 - given lat/lon default argument < given min
-555 - given lat/lon default argument > given max
-1551 - keyedIn argument is Illegal lat/lon format
-1553 - keyedIn lat/lon arg exceeds double limits
-1554 - keyedIn lat/lon argument < given min
-1555 - keyedIn lat/lon argument > given max
-1571 - keyedIn lat/lon has illegal degress format
-1573 - keyedIn degress exceeds system integer limits
-1581 - keyedIn lat/lon has illegal minutes format
-1583 - keyedIn lat/lon minutes > 59
-1591 - keyedIn lat/lon has illegal seconds format
-1593 - keyedIn lat/lon seconds > 59
< -9999 - failure statuses
mcargdln

Remarks:
Accepts lat/lon arguments in formats:
[-]ddd:mm:ss, [-]ddd,fffff
ignores leading blanks, and from any trailing blank on.
Where
ddd = degrees (default=0)
m = minutes <= 59 (default=0)
s = seconds <= 59 (default=0)
ffff = fractions of an degree.
E = exponent of E, e, O or d
n = exponential scalar
Acceptable Formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddd:mm:ss</td>
<td>degrees</td>
</tr>
<tr>
<td>ddd:mm</td>
<td>minutes</td>
</tr>
<tr>
<td>ddd:ss:ee</td>
<td>seconds</td>
</tr>
<tr>
<td>ddd:ffff</td>
<td>fractions</td>
</tr>
</tbody>
</table>

Input and Output:

Categories:

USER_INTERFACE, NAVIGATION

Filename:

argfetch.c

mcargdump

Name:
mcargdump - Display parsed arg-fetching to McIDAS debug destination.

Interface:
subroutine
mcargdump(integer arg_handle)

Input:
arg_handle - Given argument-fetching handle to be displayed, or 0 to display the current McIDAS command's structure.

Output:
none

Return values:
none

Remarks:
none

Categories:

USER_INTERFACE

Filename:

argutil.c
**mcargfree**

**Name:**
mcargfree - Free parsed arg-fetching structure for the given handle.

**Interface:**
integer function
mcargfree(integer arg_handle)

**Input:**
arg_handle - Given argument-fetching handle to be freed.

**Input and Output:**
none

**Output:**
none

**Return values:**
0  - success
< 0  - failure

**Remarks:**
Frees the arg handle, the arg structure, all of the keyword structures, and all of the argument strings.

**Categories:**
USER_INTERFACE

**Filename:**
argutil.c

---

**mcargihr**

**Name:**
mcargihr - Fetch an argument in integer type time format hhmmss.

**Interface:**
integer function
mcargihr(integer arg_handle, character*(*keyword, integer position,
integer def, integer min, integer max, integer value,
character*(*arg)

**Input:**
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name, in keyword format, or blank
for positional arg, where key=minimum keyword name,
.word-maximum name.
position - Given argument position within the keyword, or
positional argument's place.
def - Given default value to be returned if arg not found.
If not valid, will use it, but returns an error status.
min, max - Given acceptable arg range, tests for min<=value<=max.
min-max defaults to no given range checking.

**Input and Output:**
none

**Output:**
value - Integer time argument in format [-]hhmmss, or 0.
arg - Global argument text string.

**Return values:**
400 - given hhmmss time default argument returned
1400 - keyedin time argument returned as hhmmss
1610 - keyedin current hhmmss default arg returned
-404 - given time default argument < given min
-405 - given time default argument > given max
-433 - given time default argument minutes > 59
-443 - given time default argument seconds > 59
-1401 - keyedin argument is illegal time format
-1403 - keyedin time arg exceeds system int limits
-1404 - keyedin time argument < given min
-1405 - keyedin time argument > given max
-1414 - keyedin current hhmmss default arg < given min
-1415 - keyedin current hhmmss default arg > given max
-1421 - keyedin time has illegally formatted hours
-1423 - keyedin hours exceeds system integer limits
-1431 - keyedin time has illegal minutes format
-1433 - keyedin minutes > 59
-1441 - keyedin time has illegal seconds format
-1443 - keyedin seconds > 59
< -9999 - failure status
mcargihr

Remarks:
Accepts time arguments in formats:

[-]',hh:mm:ss , [+-]'hh:mm:ss[.ffffff][+]',n

Ignores leading blanks, and from any trailing blank on.
Where hh = hours (default=0)
mm = minutes <= 59 (default=0)
ss = seconds <= 59 (default=0)
ffffff = fractions of an hour.
E = exponent of B, e, h or d
n = exponential scalar
    = defaults to current time

Acceptable Formats:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>{-} HHMMSS (current time)</td>
</tr>
<tr>
<td>hh:mm, hh:mm</td>
<td>=&gt;</td>
<td>[-] hhmm00</td>
</tr>
<tr>
<td>hh:mm:ss, hh:ffffff</td>
<td>=&gt;</td>
<td>[-] hhmmss</td>
</tr>
<tr>
<td>hh, hh, hh:mm</td>
<td>=&gt;</td>
<td>[-] hh0000</td>
</tr>
<tr>
<td>hh:mm:ss, hh:ffffff</td>
<td>=&gt;</td>
<td>[-] hhmmss</td>
</tr>
<tr>
<td>mm, mm</td>
<td>=&gt;</td>
<td>[-] mm00</td>
</tr>
<tr>
<td>mm:ss, .ffffff</td>
<td>=&gt;</td>
<td>[-] mmss</td>
</tr>
<tr>
<td>ss</td>
<td>=&gt;</td>
<td>[-] ss</td>
</tr>
<tr>
<td></td>
<td>=&gt;</td>
<td>0</td>
</tr>
</tbody>
</table>

Categories:
USER_INTERFACE, DAY/TIME

Filename:
argfetch.c

mcgill

Name:
mcgill - Fetch an argument in integer type lat/lon format dddmmss.

Interface:

integer function mcgill(integer arg_handle, character(*)(*) keyword, integer position, 
integer def, integer min, integer max, integer value, 
character(*)(*) arg)

Input:

arg_handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name, in key,word format, or blank 
for positional arg, where key=minimum keyword name, 
word=maximum name.
position - Given argument position within the keyword, or 
positional argument's place.
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for min<value<max.

min-max defaults to no given range checking.

Input and Output:
none

Output:

value - Integer lat/lon value in format [-]dddmmss, or 0.
arg - Global argument text string.

Return values:

500 - given dddmmss lat/lon default arg returned
1500 - keyedin lat/lon argument returned as dddmmss
-504 - given lat/lon default argument < given min
-505 - given lat/lon default argument > given max
-533 - given lat/lon default argument minutes > 59
-543 - given lat/lon default argument seconds > 59
-1501 - keyedin argument is illegal lat/lon format
-1503 - keyedin lat/lon arg exceeds integer limits
-1504 - keyedin lat/lon argument < given min
-1505 - keyedin lat/lon argument > given max
-1521 - keyedin lat/lon has illegal degrees format
-1523 - keyedin degrees exceeds integer limits
-1531 - keyedin lat/lon has illegal minutes format
-1533 - keyedin lat/lon minutes > 59
-1541 - keyedin lat/lon has illegal seconds format
-1543 - keyedin lat/lon seconds > 59
< -9999 - failure status
Remarks:
Accepts lat/lon arguments in formats:
[-][+][dd][mm][ss][,-][ddd][ff][ff][ff][-,ln]
ignores leading blanks, and from any trailing blank on.
Where dd = degrees (default=0)
   mm = minutes <= 59 (default=0)
   ss = seconds <= 59 (default=0)
   ffff = fractions of a degree.
   g = exponent of E, e, D or D
   n = exponential scalar
Acceptable Formats:
   dd:dd:dd:dd => [-] dd:mm:ss
   dd:dd:dd:dd => [-] dd:mm:ss
   dd:dd:dd => [-] dd:mm:ss
   dd:dd => [-] dd:mm:ss
   dd:dd:ss => [-] mm:ss
   dd:ss:ss => [-] mm:ss
   d:dd:ss:ss => [-] ss

Categories:
USER_INTERFACE, NAVIGATION

Filename:
argfetch.c
mcargiyd

Name:
mcargiyd - Fetch an argument in integer type date format yyyyddd.

Interface:
integer function
mcargiyd(integer arg_handle, character(*) keyword, integer position,
integer def, integer min, integer max, integer value,
character(*) arg)

Input:
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name, in keyword format, or blank
for positional arg, where key-minimum keyword name,
.word-maximum name.
position - Given argument position within the keyword, or
positional argument's place.
def - Given default value to be returned if arg not found.
If not valid, will use it, but returns an error status.
min, max - Given acceptable arg range, tests for min<=value<=max.
min=max defaults to no given range checking.

Input and Output:
none

Output:
value
arg
- Integer date argument in format [-]yyyyddd, or 0.
- Global argument text string.

Return values:
-100 - given date default argument returned
-1300 - keyedin date argument returned as yyyyddd
-1310 - keyedin current date default arg returned
-304 - given date default argument < given min
-305 - given date default argument > given max
-363 - given day default is illegal for given year
-1301 - keyedin argument is illegal date format
-1303 - keyedin date arg exceeds system int limits
-1304 - keyedin date argument < given min
-1305 - keyedin date argument > given max
-1314 - keyedin current date default arg > given min
-1315 - keyedin current date default arg > given max
-1321 - keyedin year has illegally formatted year
-1323 - keyedin year exceeds system integer limits
mcargyid
-1331 - keyed in date has illegal month (mon) format
-1341 - keyed in date has illegal month (mm) format
-1343 - keyed in month (mm) not 1-12
-1351 - keyed in date has illegally formatted day
-1353 - keyed in day is illegal for given month
-1363 - keyed in day is illegal for given year
< -9999 - failure status

Remarks:
Accept date arguments in formats:
yyyy/mm/dd, mm/dd, dd/mm/yyyy, yyyy/dd, yyyy-mm-dd, mm-dd, dd-mon-yyyy, dd-mon
(loading - or - sign allowed)
Ignores leading blanks, and from any trailing blank on.
Where yyyy = optional year, defaults to current year,
if yy <= 2 digits, defaults to current century
mm = optional valid month of 1 thru 12, defaults to current month
   = valid calendar month (at least first 3 chars),
defaults to current month
dd = required valid day of the given month
   = required valid day of year (1-365/366)
   = defaults to today’s date

Acceptable Formats:
-- [-1 YYYYDDD (today’s date)]
-1 yy/mm/dd, dd/mm, mm/dd, dd/mm/yy, yy/dd, /yy/mm/dd, /dd, //dd, //dd/
   => [-1 YYYYDDD (current year)]
   => [-1 YYYYDDD (current month)]
   => [-1 YYYYDDD (current century)]
   => [-1 YYYYDDD (given date)]

Categories:
USER_INTERFACE, DAY/TIME

Filename:
argfetch.c

mcargkey

Name:
mcargkey - Validate defined and command line keywords.

Interface:
integer function
mcargkey(integer arg_handle, integer numkey, character*(*) valid_keywords, integer printflag)

Input:
arg_handle - Handle of argument fetching structure.
numkey - Number of keywords in valid_keywords array.
valid_keywords - Array of defined keywords, in key,word format, where
   key=minimum keyword name, .words=maximum keyword name.
printflag - Print error messages if not equal to zero.

Input and Output:
none

Output:
none

Return values:
0 - successful
-1 - defined keywords are illegal or ambiguous
-2 - command line keywords are invalid or ambiguous
< -9999 - failure status

Remarks:
The array of defined keywords, valid_keywords, is used to verify keywords entered on the command line. If printflag is TRUE, a
diagnostic message is printed for each specific error, prior to
returning an error status code. Checks that all command line keywords
are unambiguously within the given array of defined keywords, and that
the given defined keywords themselves are unambiguously legal.

Categories:
USER_INTERFACE

Filename:
argkey.c
mcargnum

Name:
mcargnum - Fetch all keyword names within parsed arg-fetching text.

Interface:
integer function
mcargnum(integer arg_handle, integer maxkey,
character*(*)(*) keywords(maxkey))

Input:
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.
maxkey - Given dimension size of keywords array.
If <= 0, then only returns the number of keywords.

Input and Output:
none

Output:
keywords - Array of keyword names as they occur in text line.

Return values:
> 0 - Number of keyword names found.
< 0 - # of keyword names that were truncated
-905 - keyword dimension size exceeded
-909 - no command line arg-fetching structure
< -9999 - failure statuses

Remarks:
If the number of keywords exceeds 'maxkey', then the 'keywords' array
contains the first 'maxkey' occurring keyword names.
If # of keywords < 'maxkey', then the 'keywords' array is blank filled.
If 'maxkey' <= 0, then the number of existing keywords is returned.

Categories:
USER_INTERFACE

Filename:
argutil.c
mcargparse

Name:
mcargparse - Parse the given text into arg-fetching structure.

Interface:
Integer function
mcargparse(character** txtstr, character** given_syntax(10),
integer parsed_len)

Input:
txtstr - Given text string to parse.
given_syntax - Given argument syntax, or 0 to use McIDAS default.

Input and Output:
none

Output:
parsed_len - Optional length of parsed text, including any
terminating ';', if present. (Use NULL to ignore).

Return values:
> 0 - Argument-fetching handle.
< 0 - failure statuses

Remarks:
Use Mcargfree() to free the argument-fetching structure.

After mcargparse is called, arguments from the given text may be
fetched by calling mcargt, mcargi, mcargm, mcargn, etc.

The correct sequence of functions for text arg-fetching is:
hand = mcargparse(txtstr, 0, parsed_len)
stat = mcargt(hand, ...), mcargi(hand, ...), etc.
stat = mcargfree(hand)

The argument-fetching subsystem allows for parsing lines of text with
varying syntax, based on the McIDAS command line syntax, thru
programmer given parsing definitions (use read() to parse other syntax
types). Parsing syntax is defined in a programmer given array of 10 strings.
Each string may define one or more, single character, specific type of
syntactic separators, or other type of syntax, as follows.

character(*) syntax(10) /
 & " " " " | white space
 & " ; ; ; | keyword separators
 & " ; ; | command separators
 & " ; ; | qfield delims left
 & " ; ; | qfield delims right
 & " ; ; | quoting delims left
 & " ; ; | quoting delims right
 & 0, | quoting escape
 & 'X', | missing argument
 & '/ | missing arg value

If any syntax isn't defined, it defaults to the shown McIDAS syntax.
The value 0 leaves the syntax undefined, indicating McIDAS default.

(1) White Space Characters

McIDAS default: ' '

Define one or more white space characters. A white space
character is a special, single character (usually a blank),
used to determine how to separate tokens, and which is ignored.

Used to separate tokens (terminates non-quoted alphanumeric token).
Ignores intermixed white space and keyword separators between
tokens. Ignores white space characters between a token and a
keyword separator.

(2) Keyword Separators

McIDAS default: ' ;'

Define one or more keyword separators. A keyword separator is a
special, single character, used to identify a keyword, and which
occurs after the keyword name, possibly separated with one or more
of the white space characters defined in (1) above.

Used to separate tokens (terminates non-quoted alphanumeric token).
Ignores intermixed white space and keyword separators between
tokens.

(3) Command Separators

McIDAS default: ','

Define one or more command separators. A command separator is a
special, single character, used to separate commands.

Terminates the parsing of the text line.

Used to separate tokens (terminates non-quoted alphanumeric token).

(4) Quote Field Left Delimiters

McIDAS default: '"'

The quote field is an argument enclosed within special delimiters,
which may contain embedded spaces and other special characters,
and which allows delimiter nesting, such as (prog (nested field)).
The special routine Mcargqu)() fetches the quote field.
mcargparse

Define one or more, single character, left delimiters for quote field arguments.

Begins a quote field argument.

Used to separate tokens (terminates non-quoted alphanumeric token).

(5) Quote Field Right Delimiters

McIDAS default: ‘’

Define one or more, single character, right delimiters which correspond to the left delimiters defined in (4) above.

If NULL, then these quote fields will extend to end of text line.

Terminates a quote field argument.

(6) Quoting Char Left Delimiters

McIDAS default: ‘’

Quoting disables a character’s special meaning and allows it to be used literally, as itself. A quoted argument is an argument enclosed within the quoting char delimiters, as defined in (6) and (7). Everything between quoting_delims_left(i) and quoting_delims_right(i) is taken literally, except for escaping as described in (8) below.

Define one or more, single character, left delimiters for quoting arguments containing embedded spaces, special characters, and escape characters, such as: ‘I am a quoted argument’.

Begins a quoted argument.

(7) Quoting Char Right Delimiters

McIDAS default: ‘’

Define right delimiters which correspond to the left delimiters defined in (6) above.

Terminates a quoted argument.

(8) Quoting Char Escape Value

McIDAS default: 0

Define a single escape character for quoted arguments as defined in (6) and (7) above, such as: ‘’ is a single quote.

The character following the quoting_escape is taken literally, and the quoting_escape character itself is removed.

If 0, then two quoting_delims_right delimiters in a row will escape itself, such as: ‘’ is a single quote.

(9) Missing Argument

McIDAS default: ‘X’

Define syntax to indicate that an argument is missing.

Used in conjunction with the missing argument replacement (defined in (10) below).

(10) Missing Argument Replacement

McIDAS default: ‘’

Define replacement for the missing argument (defined in (9) above).

This value gives the internal representation for missing arguments.

Categories:

CONVERTER, USER_INTERFACE

Filename:

argparse.c
mcargquo

Name: mcargquo - Fetch the quote field string argument.

Interface: integer function
mcargquo(integer arg_handle, character(*) value)

Input: arg_handle - Given arg-fetching handle, or 0 for McIDAS command.

Input and Output: none

Output: value - Quote field string, without leading quotation mark.

Return values: 
10 - quote field not found
1010 - quote field argument returned
-1015 - quote field exceeds fortran char(*) size
< -9999 - failure statuses

Remarks: none

Categories: USER_INTERFACE

Filename: argfetch.c

mcargstr

Name: mcargstr - Fetch an argument in character form.

Interface: integer function
mcargstr(integer arg_handle, character(*) keyword, integer position, character(*) def, character(*) value)

Input: 
arg_handle - Given arg-fetching handle, or 0 for McIDAS command.
keyword - Given keyword name, in keyword format, or blank for positional arg, where key-minimum keyword name, word-maximum name.
position - Given argument position within the keyword, or positional argument's place.
1 indicates 1st positional arg or 1st arg for keyword.
0 gets either the program name (for null keyword), or
the entered keyword name.
def - Given default value to be returned if arg not found.
If not valid, will use it, but returns an error status.

Input and Output: none

Output: value - The desired argument's value.

Return values: 
0 - given character default argument returned
1000 - keyedin character argument returned
-5 - given arg def exceeds fortran char(*) size
-1005 - keyedin arg exceeds fortran char(*) size
< -9999 - failure statuses

Remarks: none

Categories: USER_INTERFACE

Filename: argfetch.c
mcasort

Name:
mcasort - get area sort parameters from command line and add to sort array for future mcatget call

Interface:
integer function 
mcasort(integer naorts, character*(*) sort(*), integer parmpos)

Input:
parmpos - position of parameter within the keyword; used when information for more than one area is specified.

Input and Output:
aorts - number of sorts in sort array passed in sort - array containing sort conditions

Output:
none

Return values:
0 - all's well
<0 - error occurred

Remarks:
aorts and sort may contain data when mcasort is called, any sort conditions picked up from the command line will be appended.

Categories:
area

Filename:
mcasort.for

mcatget

Name:
mcatget - get an attribute string given a key string

Interface:
integer function 
mcatget(character*(*) key, character*(*) value)

Input:
key - string containing key to be looked up

Input and Output:
none

Output:
value - contents of value found associated with key

Return values:
0 - success
-1 - failure, insufficient room in value string to receive whole value

Remarks:
Searches a list of files looking for key-value. Missing values are represented by a blank value string. In case of bad return code, the truncated value is returned.

Categories:
SYSTEM

Filename:
mcatget.c
**mcattrnxt**

**Name:**
mcattrnxt - gets the next attribute pair from the list

**Interface:**
integer function
mcattrnxt(character*(* key, character*(* value)

**Input:**
none

**Input and Output:**
key - on input, key to find one greater than
on output, the next key in the collating sequence

**Output:**
value - contents of value found associated with new key

**Return values:**
0 - success
-1 - failure, insufficient room in value string to receive whole value
-2 - failure, insufficient room in key string to receive whole new key

**Remarks:**
To use this routine start by setting key = ***, repeated calls will traverse the list in alphabetical order by key until key is ***, at which time, no more calls can be made without re-initializing.

**Categories:**
SYSTEM

**Filename:**
mcattrget.c

---

**mcattrput**

**Name:**
mcattrput - puts an attribute pair (key=value) to a file

**Interface:**
integer function
mcattrput(character*(* key, character*(* value)

**Input:**
key - string containing key to be loaded
value - string containing value to be associated with key

**Input and Output:**
none

**Output:**
none

**Return values:**
0 - success
-1 - failure, could not open output file
-2 - failure, error writing in output file

**Remarks:**
key=value pair is written into the users output file.

Missing values are represented by the string "**", which means that this key should be deleted from the table.

Uses output file pointed to by environment.

**Categories:**
SYSTEM

**Filename:**
mcattrget.c
**mccmd**

**Name:**
mccmd - Fetch the current McIDAS command line.

**Interface:**
- `character(*)` function `mccmd()`

**Input:**
- `none`

**Input and Output:**
- `none`

**Output:**
- `none`

**Return values:**
- Current command line, or blank.

**Remarks:**
- `none`

**Categories:**
- USER_INTERFACE

**Filename:**
- cmdfetch.c

---

**mccmddbl**

**Name:**
mccmddbl - Fetch a program command line argument in double type format.

**Prints diagnostics to edeas device for standard errors.**

**Interface:**
- `integer function mccmddbl(character(*) keyword, integer position, character(*) printmsg, double precision def, double precision min, double precision max, double precision value)`

**Input:**
- `key_word` - Given name of the command line keyword, or blank to indicate a positional argument.
- `keyword` - maximum keyword name.
- `keyword` - minimum keyword name.
- `keyword` - matches with `key` up thru `keyword`
- `keyword` - and `keyword` match only with `keyword`

**position** - Given argument position within the keyword, or positional argument's place on the command line.
- `1` indicates 1st positional arg or 1st arg for keyword.

**printmsg** - Given description of this argument (displayed if error).

**def** - Given default value to be returned if arg not found.

**min, max** - Given acceptable arg range, tests for min<value<max.
- `min>max` defaults to no given range checking.

**Input and Output:**
- `none`

**Output:**
- `value` - The desired argument's double value, or 0 if invalid.

**Return values:**
- 200 - given double default argument returned
- 1200 - keyedin double argument returned
- 1210 - keyedin hexadecimal argument returned
- -204 - given double default argument < given min
- -205 - given double default argument > given max
- -1201 - keyedin argument is illegal decimal format
- -1203 - keyedin arg exceeds system double limits
- -1204 - keyedin double argument < given min
- -1205 - keyedin double argument > given max
- -1211 - keyedin arg is illegal hexadecimal format
- -1213 - keyedin hex exceeds system integer limits
- -1214 - keyedin hexadecimal argument < given min
- -1215 - keyedin hexadecimal argument > given max
- < -9999 - failure status
mccmddbl

Remarks:
Accepts numeric arguments in standard MeDAS formats:
[-]i.ES[-]E.s, $x$(xxxxxxx)
Ignores leading blanks, and from any trailing blank on.
Where i = integer part
E = fractional part
s = exponent of E, e, D or d
x = hexadecimal digit 0-9 or A-F
Acceptable Formats:
[-]i, l, $x$(xxxxxxx) >>> [-] i
[-]i.f >>> [-] i.f
[-]l >>> [-] l
[-]l.f >>> [-] l.f
[-]i,E[-]E, iE[-]E.s, .E[-]E.s, i.E[-]E.s >>> [-] I.l.f

Categories:
USER_INTERFACE

Filename:
cmdfetch.c

mccmddhr

Name:
mccmddhr - Fetch a program argument in fractional hours format hh.fffff.
Prints diagnostics to edest device for standard errors.

Interface:
integer function
mccmddhr(character*($) keyword, integer position,
character*($) printmsg, double precision def,
double precision min, double precision max,
double precision value)

Input:
keyword - Given name of command line keyword, in keyword format, or
blank to indicate a positional argument,
where key-minimum keyword name, .word-maximum name.
position - Given argument position within the keyword, or
positional argument's place on the command line.
printmsg - Given description of this argument (displayed if error).
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range. tests for min<=value<=max.
min-max defaults to no given range checking.

Input and Output:
none

Output:
value - Double time argument in format [-]hh.fffff, or 0.

Return values:
450 - given hh.fffff time def argument returned
1450 - keyedin time argument returned as hh.fffff
1460 - keyedin current hh.fffff default returned
-454 - given time default argument < given min
-455 - given time default argument > given max
-1451 - keyedin argument is illegal time format
-1453 - keyedin time arg exceeds system double limits
-1454 - keyedin time argument < given min
-1455 - keyedin time argument > given max
-1464 - keyedin current hh.fffff default < given min
-1465 - keyedin current hh.fffff default > given max
-1471 - keyedin time has illegally formatted hours
-1473 - keyedin hours exceeds system integer limits
mccmdhdr

-1481 - keyedin time has illegal minutes format
-1483 - keyedin minutes > 59
-1491 - keyedin time has illegal seconds format
-1493 - keyedin seconds > 59
< -9999 - failure statuses

Remarks:
Accepts time arguments in formats:
[-+]hh:mm:ss, [-+]hh, fffffff
Ignores leading blanks, and from any trailing blank on.
Where hh = hours (default=0)
mm = minutes <= 59 (default=0)
ss = seconds <= 59 (default=0)
ffffff = fractions of an hour.
E = exponent of E, e, D or d
n = exponential scalar
I = defaults to current time

Acceptable Formats:

[-+]hh:mm:ss, hh:mm:ss, hh:mm, hh::ss, hh:ss, hh:::ffffff
hh:::ffffff hh::ffffff hh::ffffff hh::ffffff i:i::ffffff

Categories:
USERINTERFACE, DAY/TIME

Filename:
cmdfetch.c

mccmddll

Name:
mccmddll - Fetch a program argument in fractional lat/lon form dd.ddddfs.
Prints diagnostics to error device for standard errors.

Interface:
integer function
mccmddll(character**), keyword, integer position,
character(**) printmsg, double precision def,
double precision min, double precision max,
double precision value)

Input:
keyword - Given name of command line keyword, in key.word format,
or blank to indicate a positional argument,
where key:minimum keyword name, .word:maximum name.
position - Given argument position within the keyword, or
positional argument's place on the command line.
1 indicates 1st positional arg or 1st arg for keyword.
printmsg - Given description of this argument (displayed if error).
def - Given default value to be returned if arg not found.
If not valid, will use it, but returns an error status.
min, max - Given acceptable arg range, tests for min<value<max.
min=max defaults to no given range checking.

Input and Output:
none

Output:
value - Double lat/lon argument in format [-]dd.dddf, or 0.

Return values:
550 - given dd.fffff lat/lon default arg returned
1550 - keyedin lat/lon arg returned as dd.fffff
554 - given lat/lon default argument < given min
-555 - given lat/lon default argument > given max
-1595 - keyedin argument is illegal lat/lon format
1593 - keyedin lat/lon arg exceeds double limits
-1554 - keyedin lat/lon argument < given min
-1555 - keyedin lat/lon argument > given max
1571 - keyedin lat/lon has illegal degrees format
-1573 - keyedin degrees exceeds system integer limits
1581 - keyedin lat/lon has illegal minutes format
1583 - keyedin lat/lon minutes > 59
-1591 - keyedin lat/lon has illegal seconds format
-1593 - keyedin lat/lon seconds > 59
< -9999 - failure statuses
mcmdihr

Name:
mcmdihr - Fetch a program argument in integer type time format hmms.
Prints diagnostics to edest device for standard errors.

Interface:
integer function
mcmdihr(character(*) keyword, integer position,
character(*) printmsg, integer def, integer min, integer max,
integer value)

Input:
keyword - Given name of command line keyword, in keyword format,
or blank to indicate a positional argument.
position - Given argument position within the keyword, or
positional argument's place on the command line.
printmsg - Given description of this argument (displayed if error).
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for min<value<max.
min-max defaults to no given range checking.

Output:
value - Integer time argument in format [-]hmms, or 0.

Return values:
400 - given hmms time default argument returned
1400 - keyedin time argument returned as hmms
1410 - keyedin current hmms default arg returned
-404 - given time default argument < given min
-405 - given time default argument > given max
-433 - given time default argument minutes > 59
-443 - given time default argument seconds > 59
-1401 - keyedin argument is illegal time format
-1403 - keyedin time arg exceeds system int limits
-1404 - keyedin time argument < given min
-1405 - keyedin time argument > given max
-1414 - keyedin current hmms default arg < given min
-1415 - keyedin current hmms default arg > given max
-1421 - keyedin time has illegally formatted hours

Applications Programming Interface
5-240

McIDAS Programmer’s Manual
Preliminary Issue 10/95

McIDAS Programmer’s Manual
Preliminary Issue 10/95

Applications Programming Interface
5-241
mccmdlhr

-1423 - keyed in hours exceeds system integer limits
-1431 - keyed in time has illegal minutes format
-1433 - keyed in minutes > 59
-1441 - keyed in time has illegal seconds format
-1443 - keyed in seconds > 59
< -9999 - failure statuses

Remarks:
Accepts time arguments in formats:
[+-]hh:mm:ss , [+-]hh:mm:ff
Ignores leading blanks, and from any trailing blank on.
Where hh = hours (default=0)
mm = minutes <= 59 (default=0)
ss = seconds <= 59 (default=0)
ff = fractions of an hour.
E = exponent of E, e, D or d
n = exponential scalar

Acceptable Formats:

| hh:mm, hh:mm: | [+-] HH:MM:SS (current time) |
| hh:mm:ss, hh:ffff | [+-] hh:mm:ss |
| hh, hh1, hh: | [+-] hh000 |
| hh:mm | [+-] hh:mm |
| mm, mm: | [+-] mm00 |
| mm:ss, .ffff | [+-] mm:ss |
| mm: | [+-] mm |
| ss | [+-] ss |
| : | [+-] 0 |

Categories:
USER_INTERFACE, DAY/TIME

Filename:
cmdfetch.c

mccdll

Name:
mccdll - Fetch a program argument in integer type lat/lon form dd:mm:ss.
Prints diagnostics to device for standard errors.

Interface:
integer function
mccdll(character*[*] keyword, integer position,
character*[*] printmsg, integer def, integer min, integer max,
integer value)

Input:
keyword - Given name of command line keyword, in key_word format, or blank to indicate a positional argument.
position - Given argument position within the keyword, or positional argument's place on the command line.
printmsg - Given description of this argument (displayed if error).
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for min<value<max.
min>max defaults to no given range checking.

Input and Output:
none

Output:
value - Integer lat/lon value in format [-]dd:mm:ss, or 0.

Return values:
500 - given dd:mm:ss lat/lon default arg returned
1500 - keyed in lat/lon argument returned as dd:mm:ss
504 - given lat/lon default argument < given min
505 - given lat/lon default argument > given max
533 - given lat/lon default argument minutes > 59
543 - given lat/lon default argument seconds > 59
1501 - keyed in argument is illegal lat/lon format
1503 - keyed in lat/lon arg exceeds integer limits
1504 - keyed in lat/lon argument < given min
1505 - keyed in lat/lon argument > given max
1521 - keyed in lat/lon has illegal degrees format
1523 - keyed in degrees exceeds integer limits
1531 - keyed in lat/lon has illegal minutes format
1533 - keyed in lat/lon minutes > 59
1541 - keyed in lat/lon has illegal seconds format
1543 - keyed in lat/lon seconds > 59
< -9999 - failure statuses

Applications Programming Interface
McIDAS Programmer's Manual
Preliminary Issue 10/95

McIDAS Programmer's Manual
Preliminary Issue 10/95

Applications Programming Interface
S-242
**mcmdint**

**Name:**
mcmdint - Fetch a program command line argument in integer type format.
Prints diagnostics to edest device for standard errors.

**Interface:**
integer function
mcmdint(character**[*] keyword, integer position.
character**[*] printmsg, integer def, integer min, integer max. integer value)

**Input:**
keyword - Given name of command line keyword, in key,word format, or blank to indicate a positional argument,
where key-minimum keyword name, word-maximum name.
position - Given argument position within the keyword, or positional argument's place on the command line.
printmsg - Given description of this argument (displayed if error).
def - Given default value to be returned if arg not found.
If not valid, will use it, but return an error status.
min, max - Given acceptable arg range, tests for min<value<max.
min>max defaults to no given range checking.

**Output:**
value - The desired argument's integer value, or 0 if invalid.

**Return values:**
100 - given integer default argument returned
1100 - keyedin integer argument returned
1110 - keyedin hexadecimal argument returned
-104 - given integer default argument < given min
-105 - given integer default argument > given max
-1101 - keyedin argument is illegal integer format
-1102 - keyedin integer arg has illegal fraction
-1103 - keyedin arg exceeds system integer limits
-1104 - keyedin integer argument < given min
-1105 - keyedin integer argument > given max
-1111 - keyedin arg is illegal hexadecimal format
-1113 - keyedin hex exceeds system integer limits
-1114 - keyedin hexadecimal argument < given min
-1115 - keyedin hexadecimal argument > given max
< -9999 - failure status

**Remarks:**
Accepts lat/lon arguments in formats:
[+-]d:d:m:ss , [+-]d.d:fff:tttt
Ignores leading blanks, and from any trailing blank on.
Where dd = degrees (default=0)
mm = minutes <= 59 (default=0)
ss = seconds <= 59 (default=0)
fff = fractions of a degree.
E = exponent of E, E, D or d
Acceptable Forms:

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd:mm</td>
<td>[-] ddm00</td>
</tr>
<tr>
<td>dd:mm:ss, dd:fff</td>
<td>[-] ddm0es</td>
</tr>
<tr>
<td>dd, dd, dd:d</td>
<td>[-] dd000</td>
</tr>
<tr>
<td>dd:ss</td>
<td>[-] dd0ses</td>
</tr>
<tr>
<td>mm, mm</td>
<td>[-] mm00</td>
</tr>
<tr>
<td>mm:ss, .fff</td>
<td>[-] mmes</td>
</tr>
<tr>
<td>s.s</td>
<td>[-] ss</td>
</tr>
</tbody>
</table>

**Categories:**
USER_INTERFACE, NAVIGATION

**Filename:**
cmdfetch.c
Remarks:
Accepts integer arguments in standard McIDAS formats:
[-]i, sE[+]s, $x[xxxxxxx]
Ignores leading blanks, and from any trailing blank on.
Where i = integer part
f = fractional part
E = exponent of E, e, D or d
x = hexadecimal digit 0-9 or A-F
Acceptable Formats:
[-]i, i, E[+]s, i.E[+]s, i.E[+]s.s, $x[xxxxxxx] => [-] i

Categories:
USER_INTERFACE

Filename:
cmdfetch.c

**mccmdiyd**

**Name:**
mccmdiyd - Fetch a program argument in integer type date format yyyyddd.
Prints diagnostics to stderr device for standard errors.

**Interface:**
integer function
mccmdiyd(character[strlen(keyword), integer position,
character[strlen(printmsg), integer def, integer min, integer max,
integer value)

**Input:**

keyword - Given name of command line keyword, in keyword format,
or blank to indicate a positional argument.
position - Given argument position within the keyword, or
where keyword name, min maximum name.
printmsg - Given description of this argument (displayed if error).
def - Given default value to be returned if arg not found.
min, max - Given acceptable arg range, tests for min<value<max.
min-max defaults to no given range checking.

**Input and Output:**

none

**Output:**

value - Integer date argument in format [-]yyyyddd, or 0.

**Return values:**

300 - given date default argument returned
1300 - keyedin date argument returned as yyyyddd
1310 - keyedin current date default arg returned
-304 - given date default argument < given min
-305 - given date default argument > given max
-363 - given day default is illegal for year
-1301 - keyedin argument is illegal date format
-1303 - keyedin date arg exceeds system int limits
-1304 - keyedin date argument < given min
-1305 - keyedin date argument > given max
-1314 - keyedin current date default arg < given min
-1315 - keyedin current date default arg > given max
-1321 - keyedin date has illegally formatted year
-1323 - keyedin year exceeds system integer limits
mccmdiyd

-1331 - keyed in date has illegal month (mon) format
-1343 - keyed in month (mm) not 1-12
-1351 - keyed in date has illegally formatted day
-1353 - keyed in day is illegal for given month
-1363 - keyed in day is illegal for given year
< - 9999 - failure status

Remarks:
Accepts date arguments in formats:
    yyyy/mm/dd, mm/dd, dd/month/yyyy, dd/month, yyyy/dd, yyyy-mm-dd, mm-dd, dd-mon-yyyy, dd-mon
(leading + or - sign allowed)
Ignores leading blanks, and any trailing blank on.
Where yyyy = optional year, defaults to current year.
    if yy <= 2 digits, defaults to current century
    mm = optional valid month of 1 thru 12, defaults to current month
    mon = valid calendar month (at least first 3 chars),
    defaults to current month
    dd = required valid day of the given month
    ddd = required valid day of year (1-365/366)
/ = defaults to today's date

Acceptable Formats:

  mm/dd, dd/month, dd/mon, mm/dd, dd/mon/
  /dd, /dd, //dd, //dd/    ==> [-] YYYYddd (current year)
  yyyy/mm/dd, dd/month, yyyy/dd, yyyy-mm-dd, dd/mon/yyyy, yyyyddd
  ==> [-] YYYYdd (current month)
  ==> [-] YYYYddd (current century)
  ==> [-] YYYYyddd (given date)

Categories:
    USER_INTERFACE, DAY/TIME

Filename:
cmdfetch.c

mccmdkey

Name:
mccmdkey - Validate defined and command line keywords.
    Edest diagnostics.

Interface:
    integer function
    mccmdkey(integer numkey, character*(* val_key)

Input:
    numkey     - Number of keywords in valid.keywords array.
    valid_keywords - Array of defined keywords, in keyword format, where
                     key=minimum keyword name, .words=maximum keyword name.

Input and Output:
    none

Output:
    none

Return values:
    0           - successful
    -1          - defined keywords are illegal or ambiguous
    -2          - command line keywords are invalid or ambiguous
    < - 9999    - failure status

Remarks:
The array of defined keywords, valid.keywords, is used to verify keywords entered on the command line. A diagnostic message is printed for each specific error, prior to returning an error status code.
Checks that all command line keywords are unambiguously within the given array of defined keywords, and that the given defined keywords themselves are unambiguously legal.

Categories:
    USER_INTERFACE

Filename:
    errkey.c
**mccmdnam**

**Name:**
Mccmdnam - Fetch all keyword names occurring in the command line.

**Interface:**
Integer function
mccmdnam(integer maxkey, character*(*) keywords(maxkey))

**Input:**
maxkey - Given dimension size of keywords array.
If <= 0, then only returns the number of keywords.

**Input and Output:**
none

**Output:**
keywords - Array of keyword names as they occur in command line.

**Return values:**
>= 0 - Number of keyword names found.
< 0 - # of keyword names that were truncated
-905 - keyword dimension size exceeded
-909 - no command line arg-fetching structure
< -9999 - failure statuses

**Remarks:**
If the number of keywords exceeds 'maxkey', then the 'keywords' array contains the first 'maxkey' occurring keyword names.
If # of keywords < 'maxkey', then the 'keywords' array is blank filled.
If 'maxkey' <= 0, then the number of existing keywords is returned.

**Categories:**
USER_INTERFACE

**Filename:**
cmdfetch.c

---

**mccmdnum**

**Name:**
Mccmdnum - Return # values associated with given command line keyword.

**Interface:**
Integer function
mccmdnum(character*(*) keyword)

**Input:**
keyword - Given name of command line keyword, in keyword format,
or blank to indicate a positional argument,
where key-minimum keyword name, .word-maximum name.

**Input and Output:**
none

**Output:**
none

**Return values:**
> 0 - The number of arguments existing for given keyword.
< -9999 - failure statuses

**Remarks:**
none

**Categories:**
USER_INTERFACE

**Filename:**
cmdfetch.c
**mccmdquo**

**Name:**
mccmdquo - Fetch the quote field string command line argument. Prints diagnostics to edest device for standard errors.

**Interface:**
integer function mccmdquo(character**(*) value)

**Input:**
one

**Input and Output:**
one

**Output:**
value - Quote field string, without leading quotation mark.

**Return values:**
10 - quote field not found
1010 - quote field argument returned
-1015 - quote field exceeds fortran char**(*) size
< -9999 - failure statuses

**Remarks:**
one

**Categories:**
USER_INTERFACE

**Filename:**
emdfetch.c

**mccmdstr**

**Name:**
mccmdstr - Fetch a program command line argument in character form. Prints diagnostics to edest device for standard errors.

**Interface:**
integer function mccmdstr(character**(*) keyword, integer position, character**(*) def, character**(*) value)

**Input:**
keyword - Given name of command line keyword, in keyword format, or blank to indicate a positional argument.

position - Given argument position within the keyword, or positional argument's place on the command line. 1 indicates 1st positional arg or 1st arg for keyword.

def - Given default value to be returned if arg not found. If not valid, will use it, but returns error status.

**Input and Output:**
one

**Output:**
value - The desired argument's value.

**Return values:**
0 - given character default argument returned
1000 - keyedin character argument returned
-5 - given arg def exceeds fortran char**(*) size
-1005 - keyedin arg exceeds fortran char**(*) size
< -9999 - failure statuses

**Remarks:**
one

**Categories:**
USER_INTERFACE

**Filename:**
emdfetch.c
**mcdev2uc**

**Name:**
mcdev2uc - Convert given DEV= character value to its numeric uc value.

**Interface:**
integer function
mcdev2uc(character*1 dev)

**Input:**
dev - Given DEV= character value.

**Input and Output:**
none

**Output:**
none

**Return values:**
> 0 - Numeric DEV= uc value,
< 0 - failure

**Remarks:**
none

**Categories:**
SFE_CONFIG

**Filename:**
argglo.c

---

**mcdsnum**

**Name:**
mcdsnum - return number of positions in a dataset name

**Interface:**
integer function
mcdsnum(character*(*)(*) name, type)

**Input:**
name - name of the dataset
type - type of data (AREA, MD, GRID)

**Input and Output:**
none

**Output:**
none

**Return values:**
-2 - unable to resolve dataset name
-1 - error from server, probably name resolution error
>0 - number of images in the dataset

**Remarks:**
Determines the number of possible positions in the dataset.
The number is determined based on the range of data files in the
dataset. The value may differ from the actual number of files
in the dataset, because there may be gaps.

**Categories:**
image

**Filename:**
mcdsnum.for
mcfndgrd

Name:
mcfndgrd - returns the first grid number that matches the sorting conditions

Interface:
integer function mcfndgrd(integer gridf, integer day(*), integer nunday,
integer time(*), integer numtm, integer vt(*),
integer numvt, integer level(*), integer numlev,
character(*) parm(*), integer numparm,
character(*) origin(*), integer numorg,
integer head(*), integer flag)

Input:
gridf - grid file number to search
day - array containing list of days to search.
nunday - number of elements to search in 'day' array
time - array containing list of times to search.
the format is [hmmm].
umtm - number of elements to search in 'time' array
the format is [hmmm].
numvt - number of elements to search in 'vt' array
level - array containing list of valid levels to search.
umlev - number of elements to search in 'level' array
parm - parameter name(s) to search for.
umparm - number of parameters used in parm. if you want
to ignore the parm argument set numparm to 0.
onig - source(s) of the grid.
umorg - number of sources used in origin. if you want
to ignore the origin argument set numorg to 0.

Output:
flag - flag is set to the first grid in the grid file that you want mcfndgrd() to search from if subsequent calls to mcfndgrd() are made, the search begins at the first grid after the last match was found. if you want the search to begin at the top of the grid file again, reset flag to 1.

head - array containing the McIDAS grid header

Return values:
n - the number of the grid that matched the search conditions
0 - no grid was found that matched the search conditions
-1 - grid file does not exist
-2 - invalid gridfile number is specified

Remarks:
If you specify values for 'day', 'time', 'vt', 'level' in ascending order mcfndgrd() assumes you want everything within the range. if you specify them in descending order, then it will only return those grids that match the condition exactly. For example, if vt(1)=120000 and vt(2) = 480000 then all forecast times between 12 and 48 hours inclusive will be returned.
if vt(1)=480000 vt(2) = 120000 then only the 12 and 48 hour forecasts will be returned.

There is a list of 3 values for the level parameter with special meaning:

1013 - mean sea level (msl)
1001 - surface (sfc)
0 - tropopause (tro)

Categories:
grid

Filename:
mcfndgrd.for
mcgget

Name:
mcgget - request a single grid from a grid server

Interface:

integer function
mcgget(string name, int nsort, string sort(*),
string unit, string form, int maxbyt, int msgflag)

Input:
name - DDE truename, nickname, or GRIDfile number
nsort - number of sort clauses
sort - array of clauses defining the request
form - currently 'F4' or 'F4'
maxbyt - the largest grid size which the user can handle, in bytes
msgflag - 0 means do not output error messages

Input and Output:
none

Output:
none

Return values:
0 - success

Remarks:
Important use info, algorithm, etc.

Categories:
grid, api

Filename:
mcgget.f90

mglast

Name:
mcglas - returns the number of the last grid in a grid file

Interface:

integer
mcglas(character(*) grid_file)

Input:
#include 'mcidas.h'
grid_file - grid file name

Input and Output:
none

Output:
none

Return values:
-5 - invalid file name (no length)
-4 - file does not exist
-3 - can not determine byte order
-2 - can not read header
-1 - can not read grid table
0 - no grids in grid file
>0 - number of the last grid in the grid file

Remarks:
none

Categories:
GRID

Filename:
mcglas.c
**mcgrc2**

**Name:**
mcgrc2 - receive grid & grid header in c-ordering

**Interface:**
integer function
mcname(int grid(*), int head(64))

**Input:**
one

**Input and Output:**
one

**Output:**
grid - target array for grid
head - target for grid header

**Return values:**
0 - success

**Remarks:**
This is a FORTRAN program, but uses c-ordering on the array for the convenience of C callers

**Categories:**
grid, api

**Filename:**
mogget.for

---

**mcgrcv**

**Name:**
mogrcv - receive grid which was requested by mogget

**Interface:**
integer function
mcgrcv(integer grid(*), integer header(64))

**Input:**
one

**Input and Output:**
one

**Output:**
grid - array which will receive the grid
header - array which will receive the grid header

**Return values:**
0 - success

**Remarks:**
This returns a 2-d array in the FORTRAN ordering

Missing values in fixed point are returned as 'bex80'
Missing values in floating point are returned as greater than 1.0e30

**Categories:**
words from the list

**Filename:**
mogget.for
**mcisipad**

Name: mcisipad - validate an ip address

Interface: Integer function
mcisipad(character(*) ip_address)

Input: ip_address - ip address to validate

Input and Output: none

Output: none

Return values: 0 - success
-1 - failed

Remarks: none

Categories: system

Filename: mcisipad.f90

---

**mclocase**

Name: mclocase - convert a string to all lower case letters

Interface: subroutine
mclocase(character(*) string)

Input: none

Input and Output: string - string to be converted

Output: none

Return values: none

Remarks: none

Categories: conversion

Filename: mclocase.f90
mcmdks

Name:
mcmdks - call 'md-by-keys' server, to set up for mcmdrd calls

Interface:
integer function
mcmdks(char name, int nsort, char sort[9],
     int nkeys, char keys[9], char units[9], char form[9],
     int scales[9], int size, int had[9], int msgflag )

Input:
name - DDE trename, nickname, or MD file number
nsort - number of search condition clauses in the sort array
sort - array of search condition clauses
size - maximum number of bytes which mcmdrd call should return
msgflag - zero means suppress text messages

Input and Output:
nkeys - number of keys requested
keys - keys which will be returned
if blank on input, changed to source keys
units - units in which the keys will be returned
if blank, filled in with source units
form - currently may be C4, C4, F4, or blank
       if blank, filled in with source format

Output:
had - array of metadata (future expansion)
scales - for integer units, a scaling factor. unused otherwise

Return values:
0    - success

Remarks:
Sort clauses are <key> <loval> <hival> <units>
or <key> LIST <element1> <element2> ... <element n>
If <units> is omitted, values must be in the scale of the schema
otherwise they are converted. If <units> are omitted, then <hival>
can also be omitted, and will be set equal to <loval>

Categories:
MD, api

Filename:
mcmdks.for

mcmdrd

Name:
mcmdrd - read a record from 'md-data-by-keys' service

Interface:
integer function
mcmdrd(integer data[*] )

Input:
none

Input and Output:
none

Output:
data - array into which a record of data is returned

Return values:
0    - success
1    - end-of-data
neg  - error

Remarks:
Size of data array in bytes is input to mcmdks, to avoid overflow

Categories:
MD api

Filename:
mcmdks.for
**mcpnav**

**Name:**
mcpnav - parse a list of comment cards for valid navigation information  

**Interface:**
integer function
mcpnav(int cards(*), int ncards, real lat, real lon,  
      real latres, real lonres)

**Input:**
cards - integer array of comment cards from a MOAREA  
ncards - number of cards contained in the cards array  

**Input and Output:**
card - integer array of comment cards from a MOAREA  
ncards - number of cards contained in the cards array

**Output:**
lat - center latitude of image  
lon - center longitude of image  
latres - resolution in the latitude direction (km)  
lonres - resolution in the longitude direction (km)

**Return values:**
0 - found lat, found lon  
-1 - failed on at least one item

**Remarks:**
This routine assumes that you have already read the list of comment cards for a mcidas area.

**Categories:**
IMAGE, CONVERTER

**Filename:**
mcpcrd.for
Name: mcraob - collect a complete sounding for one station/time

Interface:
subroutine mcraob(int option, string md1, string md2, int day, int time, int station, int cleve, int stdat(4), int lev(100), real pres(100), real temp(100), real td(100), real dir(100), real spd(100), real z(100), int status)

Input:
option - desired option, 0 or any sum of:
1: RE-COMPUTE HEIGHTS FOR ALL LEVELS WITH P/T/TD DATA;
   IF THIS OPTION IS NOT TAKEN, NO INTERPOLATION OF ANY KIND
   IS PERFORMED; THE RAW DATA ONLY IS RETURNED TO THE CALLER.
2: INTERPOLATE PRESSURES FOR THE SIG. WINDS
4: INTERPOLATE MISSING TEMPS & DBZ POINTS
8: RETURN VALUES EVEN IF Z OR P ARE MISSING
md1 - DDE true name, nickname, or mDfile number for manditory data
md2 - DDE true name, nickname, or mDfile number for sig. data
day - day of report requested as YYYYDD
time - time of report requested as HH0000
station - wmo station id

Input and Output:
none

Output:
nlev - number of levels returned, bounded at 100
fndat - four word array containing lat, lon, elevation, and state code
lev - array containing the level
for mandatories, 'SFC' the integer pressure(emb)
or 'TR01', 'TR02' or 'MAXM' for significants, either 'SIGT' or 'SIOW'
pres - pressure in millibars
temp - temperature in deg K
td - dewpoint in deg K
dir - wind direction in degrees
spd - wind speed in meters
z - level height in meters
status - 0 good, -1 stuff missing, -2 manditory file missing
**mcstrtodbl**

**Name:**
mcstrtodbl - Convert given numeric token to double type format.

**Interface:**
integer function
mcstrtodbl(character(*) token, double precision value)

**Input:**
token - Given numeric character token in standard McIDAS formats:
[-]i.E[-]s, $x$[xxxxxxx]
Where $i$ = integer part
$E$ = exponent of $E$, $e$, $d$ or $d$
$s$ = exponential scalar
$x$ = hexadecimal digit 0-9 or A-F

**Input and Output:** none

**Output:**
value - Double value of token, or 0 if invalid.

**Return values:**
200 - successful return of given double
210 - successful return of given hexadecimal
-201 - illegally formatted decimal value (invalid char)
-203 - illegal number exceeds system double limits
-209 - no given token returns value of 0
-211 - illegally formatted hexadecimal (invalid char)
-213 - illegal hex exceeds system integer limits
-219 - no given hex returns value of 0
< -9999 - failure statuses

**Remarks:**
Acceptable Formats:
[-]i.E[-]s, $x$[xxxxxxx]  =>  [-]i
[-]i.f  =>  [-]i.f
[-]E  =>  [-]E
[-]i.E[-]s, $x$[xxxxxxx]  =>  [-]i.f
[-]i.E[-]s  =>  invalid

**Categories:**
USER_INTERFACE, CONVERTER, UTILITY

**Filename:**
argconv.c

---

**mcstrtodhr**

**Name:**
mcstrtodhr - Convert given time token to double fractional hours hh:fffff

**Interface:**
integer function
mcstrtodhr(character(*) token, double precision dhr)

**Input:**
token - Given character string time token in formats:
[-]hh:mm:ss ; [-]hh:fffff[-]n
Where $hh$ = hours (default=0)
$mm$ = minutes = 59 (default=0)
$ss$ = seconds = 59 (default=0)
$ffff$ = fractions of an hour
$E$ = exponent of $E$, $e$, $d$ or $d$
$n$ = exponential scalar
$s$ = defaults to current time

**Input and Output:**
none

**Output:**
dhr - Double in format [-]hh:fffff, or 0 if invalid.

**Return values:**
450 - successful return of given time
460 - current time returned for given ':' token
451 - illegally formatted time (invalid char)
453 - illegal time exceeds system double limits
459 - no given time returns value of 0
471 - illegally formatted hours within given time
473 - illegal hours exceeds system integer limits
481 - illegally formatted minutes within given time
483 - illegal minutes = 59
491 - illegally formatted seconds within given time
493 - illegal seconds > 59
< -9999 - failure statuses

**Remarks:**
Acceptable Formats:
NULL, $i$, $i$, $i$  =>  [-] HH:PPPP (current time)
$hh:mm$, $hh:mm:ss$, $hh:ss$,  =>  [-] hh:fffff
$hh:fffff$  =>  [-] hh
$mm$, $mm:mm$, $mm:mm:ss$, $ss$, $fffff$  =>  [-] .fffff
$ii$  =>  0.0
mcstrtodll

Name:
mcstrtodll - Convert given token to double fractional lat/lon ddd.ffffff.

Interface:
integer function
mcstrtodll(character*(*) token, double precision dll)

Input:
token - Given character string lat/lon token in formats:
(+-)ddd:mm:ss, [+/-]ddd.ffffff([+-]in
Ignore leading blank, and from any trailing blank on.
Where ddd = degrees (default=0)
mm = minutes <= 59 (default=0)
ss = seconds <= 59 (default=0)
ffffff = fractions of an degree.
E = exponent of E, e, D or d
n = exponential scalar

Input and Output:
none

Output:
dll - Double lat/lon in format [-]ddd.ffffff, or 0 if invalid.

Return values:
-550 - successful return of given lat/lon
-551 - illegally formatted lat/lon (invalid char)
-553 - illegal lat/lon exceeds system double limits
-559 - no given lat/lon returns value of 0
-571 - illegally formatted degrees within given lat/lon
-573 - illegal degrees exceeds system integer limits
-581 - illegally formatted minutes within given lat/lon
-583 - illegal minutes > 59
-591 - illegally formatted seconds within given lat/lon
-593 - illegal seconds > 59
< -9999 - failure status

Remarks:
Acceptable Formats:
ddd:mm, ddd:mm:mm, ddd:mm:ss, ddd:ss, ddd.ffffff
>>> [-] ddd.ffffff
ddd, ddd, ddd:
>>> [-] ddd
mm, mm:mm, mm:ss, mm:ss, .ffffff
>>> [-] .ffffff
; , :, 

Applications Programming Interface
McIDAS Programmer's Manual
Preliminary Issue 1095
mctrtodll

Categories: USER_INTERFACE, CONVERTER, UTILITY, NAVIGATION

Filename: argconv.c

mctrtodll

Name: mctrtodll

Convert given hexadecimal token to integer type format.

Interface:

integer function
mctrtodll(character*(*), token, integer value)

Input:

- token: Given hexadecimal numeric char token in format: [0-9][abcdef], where x = 0 thru 9 or A thru F
- Ignore leading blanks, and from any trailing blank on.

Input and Output:

- None

Output:

- value: New token's integer value, or 0 if invalid.

Return values:

- 10: Successful return of given hexadecimal
- 11: Illegally formatted hexadecimal (invalid char)
- 13: Illegal hex exceeds system integer limits
- 19: No given token returns value of 0
- 19999: Failure status

Remarks:

- None

Categories:

USER_INTERFACE, CONVERTER, UTILITY

Filename:

argconv.c
**mcstrtohms**

**Name:**
mcstrtohms - Convert given time to integer hours, minutes and seconds.

**Interface:**
int mcstrtohms(char *token, int hour, int min, int sec);

**Input:**
token - Given time token in format [-+]hh:mm:ss.
        Ignores leading blanks, and from any trailing blank on.
        Where hh = hours  (default=0)
        mm = minutes <= 59 (default=0)
        ss = seconds <= 59 (default=0)
        ; = defaults to current time

**Input and Output:**
none

**Output:**
hour - Integer hours in format [-+]hh, or 0 if invalid.

min - Integer minutes in format [-+]mm, or 0 if invalid.

sec - Integer seconds in format [-+]ss, or 0 if invalid.

**Return values:**
400 - Successful return of given time
410 - Current time returned for given ':' token
-401 - Illegally formatted time (invalid char)
-409 - No given time returns value of 0
-421 - Illegally formatted hours within given time
-423 - Illegal hours exceeds system integer limits
-431 - Illegally formatted minutes within given time
-433 - Illegal minutes > 59
-441 - Illegally formatted seconds within given time
-443 - Illegal seconds > 59
< -9999 - Failure status

**Remarks:**
Acceptable Formats:
- NULL, ';', ' ', ':' => HH, MM, SS (current time)
- hh:mm, hh:mm:ss => hh, mm, 0
- hh:mm:ss => hh, mm, ss
- hh, hh:mm, hh:mm:ss => hh, 0, 0
- mm, mm:ss => 0, mm, ss
- ss => 0, ss
- s => 0, 0, s

**Categories:**
USER_INTERFACE, CONVERTER, UTILITY, DAY/TIME

**Filename:**
argconv.c
**mcstrtoihr**

**Name:**
mcstrtoihr - Convert given time token to integer time format hh:mm:ss.

**Interface:**
integer function
mcstrtoihr(character*) token, integer ihr

**Input:**
token - Given character string time token in formats:
[-]hh:mm:ss, [+-]hh:ff[ff][+-]m
Ignore leading blanks, and from any trailing blank on.
Where hh = hours (default=0)
mm = minutes <= 59 (default=0)
ss = seconds <= 59 (default=0)
fff = fractions of an hour.
E = exponent of E, e, D or d
n = exponential scalar
i = defaults to current time

**Input and Output:**
none

**Output:**
ihr - Integer time in format [-]hh:mm:ss, or 0 if invalid.

**Return values:**
400 - Successful return of given time
410 - Current time returned for given ':' token
-401 - Illegally formatted time (invalid char)
-403 - Illegal time exceeds system integer limits
-409 - No given time returns value of 0
-421 - Illegally formatted hours within given time
-423 - Illegal hours exceeds system limits
-431 - Illegally formatted minutes within given time
-433 - Illegal minutes > 59
-441 - Illegally formatted seconds within given time
-443 - Illegal seconds > 59
< -9999 - Failure status

**Remarks:**
Acceptable Formats:
NULL, i, +1, -:
[-] hh:mm:ss
[-] hh:mm
[-] hh:mm:ss, hh:ff
[-] hh:mm:ss, hh:ff
[-] hh:ss

**Categories:**
USER_INTERFACE, CONVERTER, UTILITY, DAY/TIME

**Filename:**
argconv.c
**Name:**
mcsrtoill - Convert given lat/lon token to integer type format ddd:mm:ss.

**Interface:**
Integer function
mcsrtoill(character*(* token, integer ill)

**Input:**
token - Given character string lat/lon token in formats:
[-]dddm:ss , [-]dd.d,fffff[-.]-jn

Ignores leading blanks, and from any trailing blank on.

Where

ddd = degrees (default=0)
m = minutes <= 59 (default=0)
s = seconds <= 59 (default=0)
ffff = fractions of a degree

E = exponent of E, e, D or d
n = exponential scalar

**Input and Output:**
none

**Output:**
ill - Integer lat/lon in format [-]dddm:ss, or 0 if invalid.

**Return values:**
- 500 - successful return of given lat/lon
- 501 - illegally formatted lat/lon (invalid char)
- 503 - illegal lat/lon exceeds system integer limits
- 509 - no given lat/lon returns value of 0
- 521 - illegally formatted degrees within given lat/lon
- 523 - illegal degrees exceeds system limits
- 531 - illegally formatted minutes within given lat/lon
- 533 - illegal minutes > 59
- 541 - illegally formatted seconds within given lat/lon
- 543 - illegal seconds > 59
< -9999 - failure status

**Remarks:**
Acceptable Formats:

<table>
<thead>
<tr>
<th>d</th>
<th>ddm:ss, ddd:mm:</th>
<th>ddm:</th>
<th>ddd:mm:ss, ddd,fffff</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>d</td>
<td>ddm:</td>
<td>ddm:mm:ss</td>
</tr>
<tr>
<td>d</td>
<td>d</td>
<td>ddm:</td>
<td>ddm:mm:ss</td>
</tr>
<tr>
<td>m</td>
<td>m</td>
<td>mm:</td>
<td>mm:mm:ss</td>
</tr>
<tr>
<td>s</td>
<td>s</td>
<td>ss</td>
<td>ss:ss</td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>ffff</td>
<td>ffff:ss</td>
</tr>
</tbody>
</table>

**Categories:**
USER_INTERFACE, CONVERTER, UTILITY, NAVIGATION

**Filename:**
argconv.c
mcstrtoint

Name:
mcstrtoint - Convert given numeric token to integer type format.

Interface:
integer function
mcstrtoint(character*(*) token, integer value)

Input:
token - Given numeric character token in standard MoDAS formats:
[-]i.E[±]s , $x[xxxxxxxx]
  Ignores leading blanks, and from any trailing blank on.
  Where i = integer part
  f = fractional part
  E = exponent of E, e, D or d
  s = exponential scalar
  x = hexadecimal digit 0-9 or A-F

Input and Output:
one

Output:
value - Integer value of token, or 0 if invalid,
or rounded integer value (status=-102)

Return values:
-100 - Successful return of given integer
-110 - Successful return of given hexadecimal
-101 - illegally formatted integer (invalid char)
-102 - illegal fractional integer
-103 - illegal number exceeds system integer limits
-109 - no given token returns value of 0
-111 - illegally formatted hexadecimal (invalid char)
-113 - illegal hex exceeds system integer limits
-119 - no given hex returns value of 0
< -9999 - failure statuses

Remarks:
Acceptable Formats:
[-]i, i,E[±]s, i.E[±]s, .E[±]s, $x[xxxxxxxx] ==> [-] i

Categories:
USER_INTERFACE, CONVERTER, UTILITY

mcstrtoiyd

Name:
mcstrtoiyd - Convert given date token to integer date format yyyyddd.

Interface:
integer function
mcstrtoiyd(character*(*) token, integer iyd)

Input:
token - Given character string date token in formats:
  yyyy/mm/dd , mm/dd , dd/m/m/yyyy , dd/mon/yyyy , yyyyddd
  yyyy-mm-dd , m-d-d , dd-mon-yyyy , dd-mon
  (leading + or - sign allowed)
  Ignores leading blanks, and from any trailing blank on.
  Where yyyy, optional year, defaults to current year
  if yy <= 2 digits, defaults to current century
  mm = optional valid month of 1 thru 12,
  defaults to current month
  mon = valid calendar month (at least first 3 chars),
  defaults to current month
  dd = required valid day of the given month
  ddd = required valid day of year (1-365/366)
  / = defaults to today's date

Input and Output:
one

Output:
iyd - Integer date in format [-]yyyyddd, or 0 if invalid.

Return values:
300 - Successful return of given date
310 - current date returned for given '/' token
301 - illegally formatted date (invalid char)
302 - illegal date exceeds system integer limits
309 - no given date returns value of 0
321 - illegally formatted year within given date
322 - illegal year exceeds system limits
331 - illegally formatted month (mon) in given date
332 - illegally formatted month (mm) within given date
333 - illegal month (mm) not 1-12
331 - illegally formatted day within given date
333 - illegal day of month
< -9999 - failure statuses

Categories:
USER_INTERFACE, CONVERTER, UTILITY

Filename:
argconv.c

Applications Programming Interface
5-284

McIDAS Programmer's Manual
Preliminary Issue 10/95

McIDAS Programmer's Manual
Preliminary Issue 10/95

Applications Programming Interface
5-285
mcstrtoiyd

Remarks:

Acceptable Formats:

/                      === [-] YYYYYDD (today's date)
mm/dd, dd/mm, dd, /mm/dd, dd/month/ === [-] YYYYddd (current year)
/dd, dd/, //dd, dd/     === [-] YYYYddd (current month)
yy/mm/dd, dd/month/yy, yy/dd   === [-] Yyyrddd (current century)
yyy/mm/dd, dd/month/yyyy, yyyyddd === [-] yyyyddd (given date)
invalid

categories:

USER_INTERFACE, CONVERTER, UTILITY, DAY/TIME

Filename:

argconv.c

mctvanot

Name:

mctvanot - place documentation into an image frame

Interface:

subroutine

mctvanot(integer frame, integer line, integer numline,
integer numele, integer numchar, integer message(*),
integer on, integer off)

Input:

frame - frame to place the annotation on
line - first line to use
numline - number of lines annotation should use
numele - number of elements in the frame
numchar - number of words in message array
message - each word contains a single ascii character to be output
on - foreground of annotation line
off - background of annotation line

Input and Output:

none

Output:

none

Remarks:

numline is typically 12. The actual text takes 8 lines.

Categories:

frame api

Filename:

cvanot.f90
**mcuc2dev**

**Name:**
muc2dev - Convert given DEV= numeric uc value to its character value.

**Interface:**
character*1 function
mcuc2dev(integer dev)

**Input:**
dev - Given DEV= numeric uc value.

**Input and Output:**
none

**Output:**
none

**Return values:**
nonblank - Character DEV= value.
blank - failure

**Remarks:**
none

**Categories:**
SVP_CONFIG

**Filename:**
argprog.c

---

**mcucvtld**

**Name:**
mucvtld - convert an array of double precision values from one unit to another

**Interface:**
integer function
mcucvtld(integer num, character*(* uniti, double precision bufin(*), character*(* unito, double precision bufo(*), integer idif)

**Input:**
um - number of conversions to make
uniti - units of the input data
bufin - array of input values
unito - units of output data
idif - difference flag, see grid directory word 13
0: values are not differences

**Input and Output:**
bufo - array of output values; may be same array as bufin

**Output:**
none

**Return values:**
0 - success
-1 - failed, no conversion possible

**Remarks:**
none

**Categories:**
conversion

**Filename:**
mucvtld.for
**mcucvtr**

Name: mcucvtr - convert an array of real values from one unit to another

Interface:

integer function mcucvtr(integer num, character(*) uniti, real bufin(*), character(*) unito, real bufo(*), integer idif);

Input:
- num - number of conversions to make
- uniti - units of the input data
- bufin - array of input values
- unito - units of output data
- idif - difference flag, see grid directory word 13
  0= values are not differences

Input and Output:
- bufo - array of output values; may be same array as bufin

Output:
- none

Return values:
- 0 - success
- -1 - failed

Remarks:
- none

Categories:
- conversion

Filename:
- mcucvtr.for

**mcupcase**

Name: mcupcase - convert a string to all upper case letters

Interface:

subroutine mcupcase(character(*) string)

Input:
- none

Input and Output:
- string - string to be converted

Output:
- none

Return values:
- none

Remarks:
- none

Categories:
- conversion

Filename:
- mcupcase.for
**mdclos**

Name:  
mdclos - closes an MD file

Interface:  
subroutine  
mdclos(integer mdno)

Input:  
mdno - MD file number

Input and Output:  
none

Output:  
none

Return values:  
none

Remarks:  
To close all MD files pass 0 as the mdno.

Categories:  
pt_src

Filename:  
mdclos.for

---

**mdcode**

Name:  
mdcode - returns the missing data code for the MD file

Interface:  
integer function  
mdcode(integer mdno)

Input:  
mdno - MD file number

Input and Output:  
none

Output:  
none

Return values:  
< 0 - success, value for the missing code  
0 - failure, MD file not open

Remarks:  
The MD file must be open (see mdoen()) before you can obtain its missing data code.

Categories:  
pt_src

Filename:  
mdcode.for
mdcrnt

Name:
mdcrnt - sets the current MD file

Interface:
integer function
mdcrnt(integer mdno)

Input:
mdno - MD file number

Input and Output:
none

Output:
none

Return values:
the MD file number that is current

Remarks:
If the MD file number you pass in is not in the valid range
then the current MD file is not changed.

Categories:
pt_src

Filename:
mdcrnt.for

mdinfo

Name:
mdinfo - returns the MD header

Interface:
integer function
mdinfo(integer mdno, integer mhd[64])

Input:
mdno - MD file number

Input and Output:
none

Output:
mhd - array to hold MD header

Return values:
0 - success
<0 - failure

Remarks:
none

Categories:
pt_src

Filename:
mdinfo.for
### mdname

**Name:** mdname - create a filename from a MD file number

**Interface:**

```plaintext
subroutine mdname(integer mndo, integer filenam(2))
```

**Input:**

- `mndo` - MD file number

**Input and Output:**

- `filenam` - filename

**Output:**

- none

**Return values:**

- none

**Remarks:**

- Check that the MD file number is in the valid range before calling this routine.

**Categories:**

- converter
- pt_src

**Filename:**

`mdname.for`

---

### mdopen

**Name:** mdopen - opens a MD file

**Interface:**

```plaintext
integer function mdopen(integer mndo, integer access)
```

**Input:**

- `mndo` - MD file number
- `access` - unused parameter

**Input and Output:**

- none

**Output:**

- none

**Return values:**

- 0 - success, MD file opened
- <0 - failure, unable to open MD file

**Remarks:**

- none

**Categories:**

- pt_src

**Filename:**

`mdopen.for`
mdsvc

Name:
mdsvc - returns the real-time MD file number for a specified data type and date

Interface:
integer function
mdsvc(character*(*) ctyp, integer yyydd)

Input:
ctyp - name of data type
yyydd - year and Julian day

Input and Output:
none

Output:
none

Return values:
>0 - success, contains the current MD file number
< 0 - failure, invalid data type

Remarks:
Currently known data types are: SVCA, MAOB, REID, SHIP, FOGS.
To use the current date pass a negative number for the date.

Categories:
pt_src
converter

Filename:
mdsvc.for

mdtab

Name:
mdtab - Make a table of all values for a given key from the row or column headers of an MD file.

Interface:
integer function
mdtab(integer mdf, character*(*) ckey, integer max,
integer itab(*))

Input:
mdf - md file number
ckey - key name
max - maximum number of values in itab

Input and Output:
none

Output:
itab - array containing the table of values

Return values:
-1 - failed, key name not found
>0 - number of values returned in itab

Remarks:
none

Categories:
md

Filename:
mdtab.for
**mkwp**

**Name:**
mkwp - Retrieve keyword value in the (babylonian) minutes/seconds form.

**Interface:**
integer function
mkwp(character**(*) keyword, integer position, integer def)

**Input:**
keyword - The name of a keyword, or NULL/blank for positional.
position - Sequence # of the desired value.
def - Returned as function value if not found.

**Input and Output:**
none

**Output:**
none

**Return values:**
The desired argument's (-)HDOBMS or (-)DDOBDMS value.
Whole part of time is constrained to (approx) +/- 10000.
If out of range, function value will be 0.

**Remarks:**
Obsolete routine, replaced by mcmdhr.
If lue(-30) is set, and keyword not keyedin,
then try to fetch argument from the system string table.

**Categories:**
USER_INTERFACE

**Filename:**
argold.c

---

**movb**

**Name:**
movb - moves bytes from one array to another array with a destination byte offset

**Interface:**
subroutine
movb(integer num, integer inbuffer(*), integer outbuffer(*), integer offset)

**Input:**
num - number of bytes to move
inbuffer - input array
offset - byte offset in output buffer; zero based

**Input and Output:**
none

**Output:**
outbuffer - output array

**Return values:**
none

**Remarks:**
If the number of bytes to move is less than 1 then nothing happens.

**Categories:**
utility

**Filename:**
movb.c
movblk

Name:
movblk - moves blocks of bytes from one array to another array with both source and destination byte offsets

Interface:
subroutine
movblk(integer n, integer ssiz, integer source(*), integer soff, integer sinc, integer destination(*), integer doff, integer dincc)

Input:
- n - number of blocks to move
- ssiz - number of bytes to a block
- source - input array
- soff - byte offset in input array
- sinc - input increment in bytes
- doff - byte offset in output array
- dincc - output increment in bytes

Input and Output:
none

Output:
destination - output array

Return values:
none

Remarks:
The input/output increment can be positive, negative or zero.
If positive an equal to ssiz, then successive blocks will be moved.
If ssiz is zero, that block pointed to beginning at soff will be
replicated in the destination. If sinc is negative, the source is
read through in reverse order.

Categories:
image
utility

Filename:
movblk.c

movc

Name:
movc - moves bytes from one array to another array with source and destination offsets

Interface:
subroutine
movc(integer num, integer inbuf(*), integer soff, integer outbuf(*), integer doff)

Input:
um - number of bytes to move
inbuf - input array
soff - byte offset in input buffer; zero based
doff - byte offset in output buffer; zero based

Input and Output:
none

Output:
outbuf - output array

Return values:
0 - success

Remarks:
If the number of bytes to move is less than 1 then nothing happens.

Categories:
utility

Filename:
movc.c
movpix

Name:
movpix - samples/moves bytes from one array to another array
        with both source and destination byte offsets

Interface:
        subroutine
        movpix(integer n, integer source(*), integer soff,
               integer destination(*), integer doff, integer dinc)

Input:
        n       - number of bytes to move
        source - input array
        soff   - byte offset in input array
        inc    - input increment in bytes
        doff   - byte offset in output array
        dinc   - output increment in bytes

Input and Output:
        none

Output:
        destination   - output array

Return values:
        none

Remarks:
The input/output increment can be positive, negative or zero.
If positive, then successive bytes will be moved. If inc is zero,
that byte pointed to beginning at soff will be replicated in the
destination. If inc is negative, the source is read through in
reverse order.

Categories:
        image
        utility

Filename:
        movpix_.c

movw

Name:
movw - moves words from one array to another array

Interface:
        subroutine
        movw(integer num, integer inbuf(*), integer outbuf(*))

Input:
        num       - number of words to move
        inbuf     - input array

Input and Output:
        none

Output:
        outbuf   - output array

Return values:
        none

Remarks:
If the number of bytes to move is less than 1 then nothing happens.

Categories:
        utility

Filename:
        movw_.c
**mpixel**

**Name:**
mpixel - expands/packs data values

**Interface:**
Integer function
mpixel(integer n, integer isou, integer ides, integer buffer(*))

**Input:**
n - number of data values to move
isou - source data length in bytes
ides - destination data length in bytes

**Input and Output:**
buffer - array of data values

**Output:**
none

**Return values:**
0 - source and destination data lengths are the same; array not changed

**Remarks:**
The data length in bytes can be 1, 2, or 4 bytes.
If the data length is going to increase from the source to the destination, make sure that the array is large enough to hold the data values.

**Categories:**
image calibration

**Filename:**
mpixel_.c

---

**mpixtb**

**Name:**
mpixtb - expands/packs data values with a lookup table

**Interface:**
Subroutine
mpixtb(integer n, integer isou, integer ides, integer buffer(*),
integer itab(*))

**Input:**
n - number of data values to move
isou - source data length in bytes
ides - destination data length in bytes
itab - lookup table array

**Input and Output:**
buffer - array of data values

**Output:**
none

**Return values:**
none

**Remarks:**
The data length in bytes can be 1, 2, or 4 bytes.
If the data length is going to increase from the source to the destination, make sure that the array is large enough to hold the data values.

**Categories:**
calibration

**Filename:**
mpixtb_.c
**mpp**

**Name:**
mpp - Get positional parameter (SUNMGRS form).

**Interface:**
integer function
mpp(integer position, integer def)

**Input:**
position - Sequence # of the desired value.
def - Default value, returned as function value.

**Input and Output:**
none

**Output:**
none

**Return values:**
The desired argument number value, or 0 if error.

**Remarks:**
Obsolete routine, replaced by mcmgrs.

**Categories:**
USER_INTERFACE

**Filename:**
argold.c

---

**nkwp**

**Name:**
nkwp - Return the number of values associated with a keyword in the invoking command.

**Interface:**
integer function
nkwp(character*[*] keyword)

**Input:**
keyword - The name of a keyword, or NULL/blank for positional.

**Input and Output:**
none

**Output:**
none

**Return values:**
> 0 - The number of values associated with given keyword.
  0 - failure - either keyword not found, or error

**Remarks:**
Obsolete routine, replaced by mcmgrnum.

If luc(-30) is set, and keyword not keyedin, then returns the number of values associated with the system string table keyword.

**Categories:**
USER_INTERFACE

**Filename:**
argold.c
**nvtiro**

**Name:**

nvtiro - creates nav codicil for TIRO navigation

**Interface:**

integer function

**Input:**

ctyp  - sensor type (HIRS, MSU, AVHRR, ERB)
navfil - number of system nav file to read
iedy  - satellite date (SSYTDOD)
itim  - time of 1st scan in image (JEDMSS)

**Input and Output:**

none

**Output:**

iarr  - 512 byte TIRO nav codicil

**Return values:**

0  - success
-1  - failure

**Remarks:**

none

**Categories:**

none

**Filename:**

nvtiro.for

---

**prescn**

**Name:**

prescn - Pre-scanning of command line, doing what can be done.

**Interface:**

subroutine

prescn(character*) cmd

**Input:**

cmd  - Given McIDAS command line text.

**Input and Output:**

none

**Output:**

none

**Return values:**

none

**Remarks:**

none

**Categories:**

USER-INTERFACE, SYSTEM

**Filename:**

prescn.for
**putfrm**

**Name:**
putfrm - writes the frame directory to a file

**Interface:**
subroutine
putfrm(integer frame, integer entry(64))

**Input:**
frame - frame number
entry - array containing frame directory information

**Input and Output:**
none

**Output:**
none

**Return values:**
none

**Remarks:**
Before calling this routine make sure the frame number is valid.

**Categories:**
file
display
utility

**Filename:**
putfrm.for

---

**roablk**

**Name:**
roablk - reads nav information from a system nav file

**Interface:**
integer function

**Input:**
filnum  - master navigation file number
satyd   - satellite-year-day of data
hhmmss  - time of data

**Input and Output:**
none

**Output:**
oanda - data

**Return values:**
-1  - no nav for that day in file
  1  - exact match of satyd and hhmmss found
  2  - data returned is not that exactly asked for

**Remarks:**
none

**Categories:**
none

**Filename:**
roablk.for
**rpixel**

**Name:**
rpixel - replicate elements of an array in place

**Interface:**
integer function
rpixel(integer num, integer rep, integer siz, integer buf(*) )

**Input:**
num - number of array elements
rep - repeat factor
siz - byte size of each element in the array (1, 2 or 4)

**Input and Output:**
buf - array of elements

**Output:**
none

**Return values:**
-3 - invalid value for num
-2 - invalid value for rep
-1 - invalid value for siz
0 - success

**Remarks:**
Internal functions rep1, rep2, and rep4 replicate array elements by casting the incoming array to the size indicated by the siz parameter. Only element sizes of 1, 2 and 4 bytes can be replicated.

**Categories:**
utility

**Filename:**
rpixel_.c

---

**sleep**

**Name:**
sleep - sleep for a given number of milliseconds

**Interface:**
subroutine
sleep(integer millisecs)

**Input:**
millisecs - idle time in milliseconds

**Input and Output:**
none

**Output:**
none

**Return values:**
none

**Remarks:**
for millisecs < 1, sleep does not sleep.

**Categories:**
system, utility

**Filename:**
sleep_.c
**stamp**

**Name:**
stamp - Stamps the day/time and audit information into AREA codicil.

**Interface:**
subroutine
        stamp(integer anum)

**Input:**
anum - Given area number.

**Input and Output:**
none

**Output:**
none

**Return values:**
none

**Remarks:**
none

**Categories:**
DATE, AREA, CODICIL

**Filename:**
stamp.for

---

**tvlims**

**Name:**
tvlims - recommended line and element limits for a frame when drawing graphics

**Interface:**
subroutine
tvlims(integer n, integer bl, integer el, integer be, integer ee)

**Input:**
n - the panel number

**Input and Output:**
none

**Output:**
bl - beginning line number
el - ending line number
be - beginning element number
ee - ending element number

**Return values:**
none

**Remarks:**
If the panel number (n) is 0 then it returns the line and element for the whole image frame. If the panel number is between 1 and 4 (NW, NE, SW, SE) then it returns the line and element for the desired quarter panel.

**Categories:**
display
utility

**Filename:**
tvlims.for
**tvsat**

**Name:**
tvsat - converts a tv line and element to a image line and element

**Interface:**
subroutine
tvsat(integer fram, integer itvlin, integer itvele, integer ilin, integer ile, integer ss, integer jday, integer time)

**Input:**
fram - frame number
itvlin - tv line
itvele - tv element

**Input and Output:**
none

**Output:**
ilin - image line
ile - image element
ss - satellite number
jday - year and Julian date
time - time of the image

**Return values:**
none

**Remarks:**
This routine assumes that the upper left corner of the frame is 1,1.

Before calling this routine make sure that the frame number is in the valid range.

If the returned line and element are 0 there is no information about the frame.

**Categories:**
image
utility
converter
navigation

**Filename:**
tvsat.for

---

**volnam**

**Name:**
volnam - determines the system pathname of a McIDAS file

**Interface:**
integer function
volnam(character*(*) filename, character*(*) pathname)

**Input:**
filename - name of the file

**Input and Output:**
none

**Output:**
pathname - full pathname to use for the file

**Return values:**
<0 - error
>0 - success, determined a pathname to use for the file

**Remarks:**
volnam() fails if the pathname buffer is not large enough.

On OS/2, if the file is not found via REDIRECT, the pathname is simply the filename.

In C, use Mcpathname().

**Categories:**
file

**Filename:**
pathname.c
Data File Structures

This chapter describes the file formats for the data file structures developed for applications running under the McIDAS-X and McIDAS-OS2 environments. The data file structures are listed alphabetically, with the following information provided for each:

- file type (ASCII, binary, ASCII/binary)
- remarks
- word allocation
- Applications Programming Interface (API) routines
### ALLOC.FLG

Frame allocation status.

<table>
<thead>
<tr>
<th>Type</th>
<th>binary</th>
</tr>
</thead>
</table>

**Remarks**

This fixed-length file is used as a status to determine if frame allocation information is appended when logging on to a host workstation.

This file is initialized by setup.fp in McIDAS-OS2.

### API Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>savsiz.for</td>
<td>writes status information to ALLOC.FLG in McIDAS-X</td>
</tr>
</tbody>
</table>
**ALLOC.SDA**

Frame allocation table for SDA workstations.

**Type**

ASCII

**Remarks**

This file consists of a fixed-length, 80-character record. One record describes the frame allocation. Each frame is 480 lines by 640 elements.

This file is initialized by setup.fp in McIDAS-OS2.

**Word Allocation**

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SDA model number</td>
</tr>
<tr>
<td>2</td>
<td>blank</td>
</tr>
<tr>
<td>3 - 5</td>
<td>number of image frames allocated</td>
</tr>
<tr>
<td>6</td>
<td>blank</td>
</tr>
<tr>
<td>7 - 9</td>
<td>number of graphics frames allocated</td>
</tr>
<tr>
<td>10 - 12</td>
<td>blank</td>
</tr>
<tr>
<td>13 - 15</td>
<td>number of lines (480) for the SDA image display</td>
</tr>
<tr>
<td>16 - 19</td>
<td>blank</td>
</tr>
<tr>
<td>20 - 22</td>
<td>number of elements (640) for the SDA image display</td>
</tr>
<tr>
<td>23 - 80</td>
<td>blank</td>
</tr>
</tbody>
</table>

**API Routines**

Currently, no API routines exist for reading and writing this file.

---

**ALLOC.WWW**

Frame allocation table for WIDE WORD and McIDAS-X workstations.

**Type**

binary

**Remarks**

This file is fixed at 2002 words. McIDAS-X uses words 1004 through 1502 only. This file is initialized by setup.fp in McIDAS-OS2.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>total frame allocation (number of memory boards * 512 * 65536)</td>
</tr>
<tr>
<td>1</td>
<td>number of image frames allocated</td>
</tr>
<tr>
<td>2</td>
<td>number of graphics frames allocated</td>
</tr>
<tr>
<td>3</td>
<td>not used</td>
</tr>
<tr>
<td>4</td>
<td>WWW memory address of the first image frame</td>
</tr>
<tr>
<td>5</td>
<td>WWW memory address of the second image frame</td>
</tr>
<tr>
<td>502</td>
<td>WWW memory address of the four-hundred and ninety-ninth image frame</td>
</tr>
<tr>
<td>503</td>
<td>not used</td>
</tr>
<tr>
<td>504</td>
<td>WWW memory address of the four-hundred and ninety-ninth graphics frame</td>
</tr>
<tr>
<td>1001</td>
<td>usage for the second graphics frame</td>
</tr>
<tr>
<td>1002</td>
<td>usage for the first graphics frame</td>
</tr>
<tr>
<td>1003</td>
<td>not used</td>
</tr>
<tr>
<td>1004</td>
<td>lines/elements as halfwords for the first image frame</td>
</tr>
<tr>
<td>1005</td>
<td>lines/elements as halfwords for the second image frame</td>
</tr>
<tr>
<td>1502</td>
<td>lines/elements as halfwords for the four-hundred and ninety-ninth image frame</td>
</tr>
<tr>
<td>1503</td>
<td>not used</td>
</tr>
<tr>
<td>1504</td>
<td>lines/elements as halfwords for the four-hundred and ninety-ninth graphics frame</td>
</tr>
<tr>
<td>2001</td>
<td>lines/elements as halfwords for the second graphics frame</td>
</tr>
<tr>
<td>2002</td>
<td>lines/elements as halfwords for the first graphics frame</td>
</tr>
</tbody>
</table>

---

Data File Structures

McIDAS Programmer's Manual
Preliminary issue 10/95
**AREAannnn**

*Area files, where nnnn is a user-defined number.*

**Type**
ASCII/binary

**Remarks**
The first 64 words of every area file are the area directory block for the image. The area directory describes the kind of data in the area and contains pointers used for locating the remaining blocks and other subsets of data in the file. It also contains information about the image from which the area was created.

The data in the area directory is stored as 32-bit (4-byte) twos complement binary integers or as ASCII character data. Each of the directory's 64 words is described below. Some words are satellite specific. All byte offsets and pointers are zero-based.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>contains zeros if the record is valid</td>
</tr>
<tr>
<td>1</td>
<td>area format: always 4</td>
</tr>
<tr>
<td>2</td>
<td>sensor source number</td>
</tr>
<tr>
<td>3</td>
<td>date the image data was collected, YYDDD</td>
</tr>
<tr>
<td>4</td>
<td>time the image data was collected, HHMMSS</td>
</tr>
<tr>
<td>5</td>
<td>image line coordinate of area line 0, element 0</td>
</tr>
<tr>
<td>6</td>
<td>image element coordinate of area line 0, element 0</td>
</tr>
<tr>
<td>7</td>
<td>not used</td>
</tr>
<tr>
<td>8</td>
<td>number of lines in the area</td>
</tr>
<tr>
<td>9</td>
<td>number of elements in each line</td>
</tr>
<tr>
<td>10</td>
<td>number of bytes per element: 1, 2 or 4</td>
</tr>
<tr>
<td>11</td>
<td>line resolution; number of image lines between consecutive area lines</td>
</tr>
<tr>
<td>12</td>
<td>element resolution; number of image elements between consecutive area elements</td>
</tr>
<tr>
<td>13</td>
<td>maximum number of bands per line of the area</td>
</tr>
<tr>
<td>14</td>
<td>length of the DATA block line prefix, in bytes; sum of W48, W49, W50 (+ 4 if W35 validity code is present)</td>
</tr>
<tr>
<td>15</td>
<td>McIDAS user project number under which the area was created</td>
</tr>
<tr>
<td>16</td>
<td>date the area was created, YYDDD</td>
</tr>
<tr>
<td>17</td>
<td>time the area was created, HHMMSS</td>
</tr>
</tbody>
</table>
| 18   | 32-bit filter band map for multichannel images; if a bit is set, data exists for the band; band 1 is the least significant byte, i.e., rightmost
### DNKEYS

List of all known MD file keys, units and maximum field widths.

#### Type

ASCII

#### Remarks

Lines beginning with an asterisk (*) or slash (/) are comments. Comment lines are followed by two lines of header information. Succeeding lines contain keys, possibly followed by comment lines, in the format below.

This file is supplied with McIDAS-X and -OS2.

#### Word Allocation

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 4</td>
<td>key name, left-adjusted</td>
</tr>
<tr>
<td>5 - 9</td>
<td>blank</td>
</tr>
<tr>
<td>10 - 11</td>
<td>maximum field width</td>
</tr>
<tr>
<td>12</td>
<td>blank</td>
</tr>
<tr>
<td>13 - 16</td>
<td>units, left-adjusted</td>
</tr>
<tr>
<td>17 - 23</td>
<td>blank</td>
</tr>
<tr>
<td>24 - 72</td>
<td>description</td>
</tr>
<tr>
<td>73 - 80</td>
<td>blank</td>
</tr>
</tbody>
</table>

#### API Routines

Currently, no API routines exist for reading and writing this file.
**ENHANCES**

Color enhancement storage file for tower workstations.

**Type**

binary

**Remarks**

ENHANCES is a variable-length file.

To conserve space, each workstation is allocated its own block when the enhancement feature is used for the first time.

The starting word of the next available data block is computed as: old starting word + (256*66 + 132), where there are 256 tables per workstation, 66 words per table, plus two 66-word tables for the current enhancement data. Each table has 64 words of enhancement data and two words for a name.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>word location of the next available data block</td>
</tr>
<tr>
<td>1 - 200</td>
<td>starting word locations of the data for each of the 200 possible workstations</td>
</tr>
<tr>
<td>1024 - n</td>
<td>enhancement tables</td>
</tr>
</tbody>
</table>

**API Routines**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>enhio.for</td>
<td>I/O enhancement tables to and from ENHANCES</td>
</tr>
</tbody>
</table>

---

***.ET**

Enhancement files, where * is a user-defined file name.

**Type**

binary

**Remarks**

Each file contains 817 words. The red, green and blue color lookup tables in each file contain the red, green and blue values for a given image brightness value on a scale of 0 to 255. The remaining 48 words contain the red, green and blue values for the 16 graphics color levels on a VGA display on a scale of 0 to 255.

IMAGE.ET and RADAR.ET are supplied with McIDAS-X. IMAGE.ET, RADAR.ET and GRAPHIC.ET are supplied with McIDAS-OS2.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>not used</td>
</tr>
<tr>
<td>1</td>
<td>red lookup value for an image brightness of 0</td>
</tr>
<tr>
<td>256</td>
<td>red lookup value for an image brightness of 255</td>
</tr>
<tr>
<td>257</td>
<td>green lookup value for an image brightness of 0</td>
</tr>
<tr>
<td>512</td>
<td>green lookup value for an image brightness of 255</td>
</tr>
<tr>
<td>513</td>
<td>blue lookup value for an image brightness of 0</td>
</tr>
<tr>
<td>768</td>
<td>blue lookup value for an image brightness of 255</td>
</tr>
<tr>
<td>769</td>
<td>red graphics color value for VGA level 0</td>
</tr>
<tr>
<td>784</td>
<td>red graphics color value for VGA level 15</td>
</tr>
<tr>
<td>785</td>
<td>green graphics color value for VGA level 0</td>
</tr>
<tr>
<td>800</td>
<td>green graphics color value for VGA level 15</td>
</tr>
<tr>
<td>801</td>
<td>blue graphics color value for VGA level 0</td>
</tr>
<tr>
<td>816</td>
<td>blue graphics color value for VGA level 15</td>
</tr>
</tbody>
</table>
**FLDWIDTH**

*Maximum field widths of all known MD file keys.*

<table>
<thead>
<tr>
<th>Type</th>
<th>binary</th>
</tr>
</thead>
</table>

**Remarks**

- FLDWIDTH is a variable-length LW file used by the MDL command for formatting.
- This file is supplied with McIDAS-X and -OS2.

**Word Allocation**

- Two words are allocated for each key. The first word contains the key name, which is 4-character and left-adjusted; the second word contains the maximum field width.
- The last word in the file contains the stop code 999999.

**API Routines**

- Currently, no API routines exist for reading and writing this file.
**FRAMED.ttt**

Frame directory table, where ttt is the workstation's terminal number.

**Type**

binary

**Remarks**

Each frame directory contains 64 words. Each frame allocated for the workstation has its own 64-word directory.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 63</td>
<td>unused</td>
</tr>
<tr>
<td>64</td>
<td>sensor source (SS) number; (-1) means no image is currently loaded on that frame and the remaining words can be ignored; see the SATANNOT file for a description of the sensor source numbers</td>
</tr>
<tr>
<td>65</td>
<td>year and Julian day of the image, YYDDD</td>
</tr>
<tr>
<td>66</td>
<td>time of the image, HHMMSS</td>
</tr>
<tr>
<td>67</td>
<td>spectral band</td>
</tr>
<tr>
<td>68</td>
<td>upper-left image line in satellite coordinates; corresponds to word 5 in the area directory</td>
</tr>
<tr>
<td>69</td>
<td>upper-left image element in satellite coordinates; corresponds to word 6 in the area directory</td>
</tr>
<tr>
<td>70</td>
<td>upper-left image Z coordinate; not currently used</td>
</tr>
<tr>
<td>71</td>
<td>TV line containing the image line in word 68</td>
</tr>
<tr>
<td>72</td>
<td>TV element containing the image element in word 69</td>
</tr>
<tr>
<td>73</td>
<td>line blowup; also see word 96</td>
</tr>
<tr>
<td>74</td>
<td>line blowdown</td>
</tr>
<tr>
<td>75</td>
<td>element blowdown; also see word 96</td>
</tr>
<tr>
<td>76</td>
<td>creation date of the area, YYDDD</td>
</tr>
<tr>
<td>77</td>
<td>creation time of the area, HHMMSS</td>
</tr>
<tr>
<td>78</td>
<td>creation date of the frame, YYDDD</td>
</tr>
<tr>
<td>79</td>
<td>creation time of the frame, HHMMSS</td>
</tr>
<tr>
<td>80</td>
<td>number of the area from which the frame was loaded</td>
</tr>
<tr>
<td>81 - 95</td>
<td>identification block (optional); CHARACTER if positive, element blowup, if negative, same as word 73</td>
</tr>
<tr>
<td>96</td>
<td>digital data byte offset in the associated area</td>
</tr>
<tr>
<td>98</td>
<td>navigation block byte offset in the associated area</td>
</tr>
<tr>
<td>99</td>
<td>number of bytes per data element</td>
</tr>
<tr>
<td>100</td>
<td>original source type if changed by AA, AAMAP, etc.; CHARACTER</td>
</tr>
<tr>
<td>101 - 102</td>
<td>table file name used by program SU; CHARACTER</td>
</tr>
<tr>
<td>103 - 118</td>
<td>breakpoints for VGA; gny levels 0 through 15; (-1) indicates graphics level usage</td>
</tr>
</tbody>
</table>

**API Routines**

- erafrm.for
- getfrm.for
- putfrm.for

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>erafrm.for</td>
<td>flags a frame directory entry as unused (erase)</td>
</tr>
<tr>
<td>getfrm.for</td>
<td>reads frame directory entries</td>
</tr>
<tr>
<td>putfrm.for</td>
<td>writes frame directory entries</td>
</tr>
</tbody>
</table>

ASCII DDE dataset name

binary position number within the named object

reserved for system use
**FRAMENH.ttt**

Frame enhancement table, where ttt is the workstation's terminal number.

<table>
<thead>
<tr>
<th>Type</th>
<th>binary</th>
</tr>
</thead>
</table>

**Remarks**
The size of these files depends on the number of frames allocated on the workstation. Each file contains 816-word blocks of information. The first block is not used. Individual file size can be calculated using: $816 \times (\text{number of frames} + 1)$.

The first 768 words in each block contain image enhancement values. The red, green and blue color lookup tables in each file contain the values for a given image brightness value on a scale of 0 to 255. The remaining 48 words contain the red, green and blue values for the 16 graphics color levels on a VGA display on a scale of 0 to 15.

**Word Allocation**
The structure of each block is shown below, one block per frame.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>red lookup value for an image brightness of 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>255</td>
<td>red lookup value for an image brightness of 255</td>
</tr>
<tr>
<td>256</td>
<td>green lookup value for an image brightness of 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>511</td>
<td>green lookup value for an image brightness of 255</td>
</tr>
<tr>
<td>512</td>
<td>blue lookup value for an image brightness of 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>767</td>
<td>blue lookup value for an image brightness of 255</td>
</tr>
<tr>
<td>768</td>
<td>red graphics color value for VGA level 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>783</td>
<td>red graphics color value for VGA level 15</td>
</tr>
<tr>
<td>784</td>
<td>green graphics color value for VGA level 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>799</td>
<td>green graphics color value for VGA level 15</td>
</tr>
<tr>
<td>800</td>
<td>blue graphics color value for VGA level 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>815</td>
<td>blue graphics color value for VGA level 15</td>
</tr>
</tbody>
</table>
**FRNV.ttt**

Frame navigation table, where ttt is the workstation's terminal number.

**Type**

binary

**Remarks**

The size of these files depends on the number of frames allocated on the workstation. To calculate the length of a file, in words, use: 128 * number of frames. To calculate the first word of a 128-word entry, use: (frame number - 1) * 128.

**Word Allocation**

The contents of each 128-word table entry vary depending on the type of navigation.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>navigation type, A4 format; a zero means the frame is not navigated</td>
</tr>
<tr>
<td>1 - 127</td>
<td>varies, depending on navigation type</td>
</tr>
</tbody>
</table>

**API Routines**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>erafrm.for</td>
<td>flags a navigation entry as unused (erase)</td>
</tr>
<tr>
<td>frronv.for</td>
<td>reads a navigation table</td>
</tr>
<tr>
<td>nvtofr.for</td>
<td>writes to a navigation table</td>
</tr>
</tbody>
</table>

---

**FRNVDIR.ttt**

Frame navigation directory table, where ttt is the terminal number.

**Type**

binary

**Remarks**

Each file has 2000 words. Words not used contain null values (HEX 000000).

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>number of frame entries in this file</td>
</tr>
<tr>
<td>1 - 1999</td>
<td>frame number containing the expanded navigation table; see FRNVEXP.ttt</td>
</tr>
</tbody>
</table>

**API Routines**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>erafrm.for</td>
<td>writes null values on erase</td>
</tr>
<tr>
<td>frronv.for</td>
<td>reads a directory table</td>
</tr>
<tr>
<td>nvtofr.for</td>
<td>reads/writes to a directory table</td>
</tr>
</tbody>
</table>
FRNVEXP.ttt

Expanded frame navigation entry, where ttt is the terminal number.

**Type**

binary

**Remarks**

The length of this LW file depends on the number of frames containing expanded navigation entries. These navigation table entries are larger than those in the FRNV.ttt tables.

**Word Allocation**

Each frame entry contains a 640-word navigation table; the contents of this table vary, depending on the type of navigation used.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>navigation type, A4 format</td>
</tr>
<tr>
<td>1 - 639</td>
<td>varies, depending on navigation type</td>
</tr>
</tbody>
</table>

**API Routines**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>frnrvs.for</td>
<td>reads an expanded navigation table</td>
</tr>
<tr>
<td>nvtols.for</td>
<td>writes to an expanded navigation table</td>
</tr>
</tbody>
</table>

GMSCAL

GMS calibration file.

**Type**

binary

**Remarks**

GMSCAL contains calibration data for GMS VIS and IR sensor data transmitted in the GMS Stretched-VISSR real-time signal. VIS data is 6-bit. Calibration is achieved with a 64-value VIS level-albedo lookup table. This calibration table is interpolated to make a 256-value table.

IR data is 8-bit. Calibration is achieved with a 256-value IR level-temperature lookup table which may change with the spacecraft.

This file is supplied with McIDAS-X and -OS2.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>number of IR calibration tables in the file</td>
</tr>
<tr>
<td>1</td>
<td>identification number of the first IR table in the file</td>
</tr>
<tr>
<td>2</td>
<td>identification number of the second IR table in the file</td>
</tr>
<tr>
<td>511</td>
<td>identification number of the 511th IR table, if present</td>
</tr>
<tr>
<td>512 - 767</td>
<td>interpolated VIS calibration table; albedos are multiplied by 10^<strong>6</strong> and stored as integers</td>
</tr>
<tr>
<td>1024 - 1279</td>
<td>first IR calibration table in the file; it corresponds to the identifier stored in word 1; temperatures are multiplied by 10^<strong>3</strong> and stored as integers</td>
</tr>
<tr>
<td>1280 - 1535</td>
<td>second IR calibration table in the file; it corresponds to the identifier stored in word 2 and is only present if the calibration table changes with the spacecraft</td>
</tr>
</tbody>
</table>

**API Routines**

Currently, no API routines exist for reading and writing this file.
**.GRA for SDA Workstations**

*Graphics save tables, where * is a user-defined file name.*

<table>
<thead>
<tr>
<th>Type</th>
<th>binary</th>
</tr>
</thead>
</table>

**Remarks**

These files contain the red, green and blue color values for the four graphics color levels on an SDA display. Each file has 193 words. Note that each color value is a byte and values are repeated for graphics levels greater than one.

GRAPHIC.GRA is supplied with McIDAS-OS2.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0 - 63 | red color values table; the bytes listed below contain the red color value for the designated graphics level  
byte 1 = graphics level 1  
bytes 2 and 3 = graphics level 2  
bytes 4 through 7 = graphics level 3  
bytes 8 through 15 = graphics level 4  
bytes 16 through 256 are not used |
| 64 - 127 | green color values table; the bytes listed contain the green color value for the designated graphics level  
byte 1 = graphics level 1  
bytes 2 and 3 = graphics level 2  
bytes 4 through 7 = graphics level 3  
bytes 8 through 15 = graphics level 4  
bytes 16 through 256 are not used |
| 128 - 191 | blue color values table; the bytes listed contain the blue color value for the designated graphics level  
byte 1 = graphics level 1  
bytes 2 and 3 = graphics level 2  
bytes 4 through 7 = graphics level 3  
bytes 8 through 15 = graphics level 4  
bytes 16 through 256 are not used |
| 192 | graphics level bit mask; 0=on, 1=off; the least significant bit corresponds to graphics level 1 |

**API Routines**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>delgra.for</td>
<td>deletes a saved graphics table</td>
</tr>
<tr>
<td>resgra.for</td>
<td>restores a saved graphics table</td>
</tr>
<tr>
<td>savgra.for</td>
<td>saves the current graphics table</td>
</tr>
</tbody>
</table>
*.GRA for WIDE WORD Workstations

Graphics save tables, where * is a user-defined file name.

**Type**
binary

**Remarks**
Each file contains the red, green and blue color values for the eight graphics color levels on an WWW. Each file has 193 words. Each color value is a byte; values are repeated for graphics levels greater than one.

GRAPHIC.GRA is supplied with McIDAS-OS2.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 63</td>
<td>red color values table; the bytes listed below contain the red color value for the designated graphics level byte 1 = graphics level 1 bytes 2 and 3 = graphics level 2 bytes 4 through 7 = graphics level 3 bytes 8 through 15 = graphics level 4 bytes 16 through 31 = graphics level 5 bytes 32 through 63 = graphics level 6 bytes 64 through 127 = graphics level 7 bytes 128 through 255 = graphics level 8</td>
</tr>
<tr>
<td>64 - 127</td>
<td>green color values table; the bytes listed contain the green color value for the designated graphics level byte 1 = graphics level 1 bytes 2 and 3 = graphics level 2 bytes 4 through 7 = graphics level 3 bytes 8 through 15 = graphics level 4 bytes 16 through 31 = graphics level 5 bytes 32 through 63 = graphics level 6 bytes 64 through 127 = graphics level 7 bytes 128 through 255 = graphics level 8</td>
</tr>
</tbody>
</table>

**API Routines**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>delgra.for</td>
<td>deletes a saved graphics table</td>
</tr>
<tr>
<td>resgra.for</td>
<td>restores a saved graphics table</td>
</tr>
<tr>
<td>savgra.for</td>
<td>saves the current graphics table</td>
</tr>
</tbody>
</table>
GRIDn

Grid files, where nnnn is a user-defined number.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>starting word of the next available block of data</td>
</tr>
<tr>
<td>1 - 200</td>
<td>starting word location of the data for each of the 200 possible terminals</td>
</tr>
<tr>
<td>201 - 1023</td>
<td>not used</td>
</tr>
<tr>
<td>1024 - n</td>
<td>terminal data starts</td>
</tr>
</tbody>
</table>

Each terminal data block holds 128 save tables plus the current table. Each table consists of 256 graphics words plus two words to hold the save table name. The current graphics table is the first table of each terminal block. The 288-word graphics table is described below.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>red color value for graphics level 0; the scale is 0-63</td>
</tr>
<tr>
<td>1</td>
<td>green color value for graphics level 0; the scale is 0-63</td>
</tr>
<tr>
<td>2</td>
<td>blue color value for graphics level 0; the scale is 0-63</td>
</tr>
<tr>
<td>3</td>
<td>graphics level flag for level 0; 0=on, 1=off</td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>red color value for graphics level 7; the scale is 0-63</td>
</tr>
<tr>
<td>29</td>
<td>green color value for graphics level 7; the scale is 0-63</td>
</tr>
<tr>
<td>30</td>
<td>blue color value for graphics level 7; the scale is 0-63</td>
</tr>
<tr>
<td>31</td>
<td>graphics level flag for level 7; 0=on, 1=off</td>
</tr>
<tr>
<td>32 - 255</td>
<td>not used</td>
</tr>
<tr>
<td>256 - 257</td>
<td>8-character table name</td>
</tr>
</tbody>
</table>

API Routines

- delgra.for: deletes a saved graphics table
- resgra.for: restores a saved graphics table
- savgra.for: saves the current graphics table

A grid file can contain several thousand grids. By default, a grid file is created with 159 grids unless otherwise specified.

Grid file numbers can be between 1 and 999999. If a grid file number is five or six digits, the file name begins with only GRI or GR. For example, grid file number 12345 has the file name GRI12345, but grid file number 123456 has the file name GR123456.

Word Allocation

The word allocation of each grid file and grid header are shown below.

<table>
<thead>
<tr>
<th>Grid File Directory</th>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 7</td>
<td>32 characters of label information</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>project number under which the grid file was created file creation date, YYDDD</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>maximum number of grids (n) in the grid file</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>word offset, from the beginning of the grid file, where grid 1 starts; if the offset is -1, no grids exist</td>
<td></td>
</tr>
<tr>
<td>11+(n-1)</td>
<td>word offset for grid n</td>
<td></td>
</tr>
<tr>
<td>11+n</td>
<td>next available address to start writing the next grid</td>
<td></td>
</tr>
</tbody>
</table>
**Grid Header Information**

Each grid header contains 64 words. The first word in the header is defined by the word offset in Words 11 through 11+(n-1).

<table>
<thead>
<tr>
<th>Header Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>total size; rows * columns (not to exceed 65525 or contouring routines will not work)</td>
</tr>
<tr>
<td>1</td>
<td>number of rows</td>
</tr>
<tr>
<td>2</td>
<td>number of columns</td>
</tr>
<tr>
<td>3</td>
<td>Julian date of the data, YYDDD</td>
</tr>
<tr>
<td>4</td>
<td>time of the data, HHMMSS</td>
</tr>
<tr>
<td>5</td>
<td>valid time for the grid, if applicable</td>
</tr>
<tr>
<td>6</td>
<td>name of the gridded variable, MD file terms</td>
</tr>
<tr>
<td>7</td>
<td>scale of the gridded variable, MD file terms</td>
</tr>
<tr>
<td>8</td>
<td>units of the gridded variable, MD file terms</td>
</tr>
<tr>
<td>9</td>
<td>value of the vertical level; can be: 1013 = MSL 999 = ' 0 = TRO 1001 = SFC (Otherwise, it is displayed as entered.)</td>
</tr>
<tr>
<td>10</td>
<td>scale of the vertical level</td>
</tr>
<tr>
<td>11</td>
<td>unit of the vertical level</td>
</tr>
<tr>
<td>12</td>
<td>gridded variable type: 1 = time difference 2 = time average 4 = level difference 8 = level average (or any sum of 1, 2, 4 and 8)</td>
</tr>
<tr>
<td>13</td>
<td>used if the grid parameter is a time difference or time average, HHMMSS</td>
</tr>
<tr>
<td>14</td>
<td>used if the grid parameter is a time difference or time average, HHMMSS</td>
</tr>
<tr>
<td>15</td>
<td>not used</td>
</tr>
<tr>
<td>16-31</td>
<td>reserved</td>
</tr>
<tr>
<td>32</td>
<td>grid origin; identifies the type of program that generated the grid data</td>
</tr>
<tr>
<td>33</td>
<td>grid type: 1 = pseudo-Mercator 2 = polar stereographic or Lambert Conformal 3 = equidistant 4 = pseudo-Mercator (more general) 5 = no navigation</td>
</tr>
</tbody>
</table>

-Allocation of the remaining header words if the grid type is **pseudo-Mercator:**

<table>
<thead>
<tr>
<th>Header Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>maximum latitude of the grid, degrees*10000</td>
</tr>
<tr>
<td>35</td>
<td>maximum longitude of the grid, degrees*10000</td>
</tr>
<tr>
<td>36</td>
<td>minimum latitude of the grid, degrees*10000</td>
</tr>
<tr>
<td>37</td>
<td>minimum longitude of the grid, degrees*10000</td>
</tr>
<tr>
<td>If TYPE=1:</td>
<td>increment between the grid points; same in x and y directions</td>
</tr>
<tr>
<td>38</td>
<td>reserved</td>
</tr>
<tr>
<td>If TYPE=4:</td>
<td>increment between the grid points (latitude)</td>
</tr>
<tr>
<td>38</td>
<td>increment between the grid points (longitude)</td>
</tr>
<tr>
<td>40-51</td>
<td>reserved</td>
</tr>
<tr>
<td>52-63*</td>
<td>grid directory</td>
</tr>
</tbody>
</table>

-Allocation of the remaining header words if the grid type is **polar stereographic** or **Lambert Conformal:**

<table>
<thead>
<tr>
<th>Header Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>row number of the North Pole*10000</td>
</tr>
<tr>
<td>35</td>
<td>column number of the North Pole*10000</td>
</tr>
<tr>
<td>36</td>
<td>column spacing at standard latitude, meters</td>
</tr>
<tr>
<td>37</td>
<td>longitude parallel to columns, degrees*10000</td>
</tr>
<tr>
<td>38-39</td>
<td>standard latitudes, degrees*10000; set these two equal for polar stereographic</td>
</tr>
<tr>
<td>40-51</td>
<td>reserved</td>
</tr>
<tr>
<td>52-63</td>
<td>grid directory</td>
</tr>
</tbody>
</table>

-Allocation of the remaining header words if the grid type is **equidistant:**

<table>
<thead>
<tr>
<th>Header Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>latitude of (1,1), degrees*10000</td>
</tr>
<tr>
<td>35</td>
<td>longitude of (1,1), degrees*10000</td>
</tr>
<tr>
<td>36</td>
<td>clockwise rotation of column 1 relative to north, degrees*10000</td>
</tr>
<tr>
<td>37</td>
<td>column spacing, meters</td>
</tr>
<tr>
<td>38</td>
<td>row spacing, meters</td>
</tr>
<tr>
<td>39-51</td>
<td>reserved</td>
</tr>
<tr>
<td>52-63</td>
<td>grid directory</td>
</tr>
</tbody>
</table>
*.GRX for McIDAS-X Workstations

Graphics save tables, where * is a user-defined file name.

**Type**

binary

**Remarks**

These files contain the red, green and blue values (0 to 255) for the graphics color levels on a McIDAS-X workstation. They are variable length, depending on the number of graphics levels allocated per session. To compute the file size, use: 3 * number of levels + 1.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>not used</td>
</tr>
<tr>
<td>1</td>
<td>red color value for graphics level 1</td>
</tr>
<tr>
<td>2</td>
<td>green color value for graphics level 1</td>
</tr>
<tr>
<td>3</td>
<td>blue color value for graphics level 1</td>
</tr>
<tr>
<td>3*(n-1)+1, 3*(n-1)+2, 3*(n-1)+3</td>
<td>red color value for graphics level n, green color value for graphics level n, blue color value for graphics level n</td>
</tr>
</tbody>
</table>

**API Routines**

- igmakx.for creates a grid file
- igqat.for deletes a grid file
- igget.for reads a grid from a grid file
- igput.for writes a grid to a grid file

- delgra.for deletes a saved graphics table
- resgra.for restores a saved graphics table
- savgra.for writes the current graphics table
HIRSCRPF

HIRS calibration reference parameters.

**Type**

binary

**Remarks**

HIRS calibration reference parameters are used to compute HIRS brightness temperatures. The file is organized chronologically by satellite with the last record being the most recent. Each record contains 48 words.

The first record is a header record. The following records contain data. Words 0 through 39 in the data records are the coefficients of fourth degree polynomials used to convert platinum resistance thermistor count values to temperatures. The first four words in each group of eight are for the warm blackbody target, and the fifth through eighth words in each group are for the cool blackbody which is not routinely used in the calibration process.

This file is supplied with McIDAS-X and -OS2.

**Word Allocation**

The first record is a header record. Word 1 of this record contains the total number of records in the file. The remaining 47 words are unused. The other records in the file are organized as follows.

<table>
<thead>
<tr>
<th>Record 1</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>total number of records in the file</td>
</tr>
<tr>
<td>1 - 47</td>
<td>unused</td>
</tr>
</tbody>
</table>

Records 2 through n

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 7</td>
<td>array of thermistor counts-to-temperature coefficients, zeroth order</td>
</tr>
<tr>
<td>8 - 15</td>
<td>array of thermistor counts-to-temperature coefficients, first order</td>
</tr>
<tr>
<td>16 - 23</td>
<td>array of thermistor counts-to-temperature coefficients, second order</td>
</tr>
<tr>
<td>24 - 31</td>
<td>array of thermistor counts-to-temperature coefficients, third order</td>
</tr>
<tr>
<td>32 - 39</td>
<td>array of thermistor counts-to-temperature coefficients, fourth order</td>
</tr>
</tbody>
</table>

**API Routines**

Currently, no API routines exist for reading and writing this file.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 - 43</td>
<td>array of weights for blackbody target temperature</td>
</tr>
<tr>
<td>46</td>
<td>coefficient for calibrating the HIRS visible channel</td>
</tr>
<tr>
<td>45</td>
<td>reserved</td>
</tr>
<tr>
<td>46</td>
<td>NOAA satellite number</td>
</tr>
<tr>
<td>47</td>
<td>date, YYDDD</td>
</tr>
</tbody>
</table>
HIRSTAUL
HIRS transmittance coefficients.

Type  binary

Remarks
HIRSTAUL contains 506-word records, keyed by the sensor source. Each sensor source has 20 records. Records 1 through 19 contain transmittance coefficients for channels 1 through 19 of the HIRS instrument. Record 20 contains other parameters used for radiative transfer calculations.

This file is supplied with McIDAS-X and -OS2.

Word Allocation

<table>
<thead>
<tr>
<th>Records 1-19</th>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 - 199</td>
<td>carbon dioxide transmittance coefficients</td>
</tr>
<tr>
<td></td>
<td>200 - 319</td>
<td>carbon dioxide angular correction coefficients</td>
</tr>
<tr>
<td></td>
<td>320 - 325</td>
<td>continuum transmittance coefficients</td>
</tr>
<tr>
<td></td>
<td>326 - 339</td>
<td>water vapor transmittance coefficients</td>
</tr>
<tr>
<td></td>
<td>340 - 419</td>
<td>ozone transmittance coefficients</td>
</tr>
<tr>
<td></td>
<td>420 - 499</td>
<td>ozone angular correction coefficients</td>
</tr>
<tr>
<td></td>
<td>500 - 505</td>
<td>flags indicating whether to use coefficients</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record 20</th>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 - 19</td>
<td>central wave numbers of the channels</td>
</tr>
<tr>
<td></td>
<td>20 - 59</td>
<td>coefficients for the Planck function</td>
</tr>
<tr>
<td></td>
<td>60 - 99</td>
<td>band correction coefficients</td>
</tr>
<tr>
<td></td>
<td>100 - 139</td>
<td>reserved</td>
</tr>
<tr>
<td></td>
<td>140 - 144</td>
<td>solar coefficients</td>
</tr>
<tr>
<td></td>
<td>145 - 157</td>
<td>flux coefficients</td>
</tr>
<tr>
<td></td>
<td>158 - 179</td>
<td>reserved</td>
</tr>
<tr>
<td></td>
<td>180 - 187</td>
<td>synthetic coefficients for the surface temperature</td>
</tr>
<tr>
<td></td>
<td>188 - 189</td>
<td>reserved</td>
</tr>
<tr>
<td></td>
<td>190 - 197</td>
<td>empirical coefficients for the surface temperature</td>
</tr>
<tr>
<td></td>
<td>198 - 505</td>
<td>reserved</td>
</tr>
</tbody>
</table>

API Routines
Currently, no API routines exist for reading and writing this file.

HOST.PTR
IP address of the host file.

Type  ASCII

Remarks
HOST.PTR contains the IP address of the host for your workstations, if applicable, using TCP/IP. The IP address immediately follows the word CONNECT, for example: CONNECT 144 92 116 1. The periods found in an address are replaced with blanks.

This file is initialized by setup.fp in McIDAS-OS2 and supplied with McIDAS-X.

API Routines
Currently, no API routines exist for reading and writing this file.
**IDMSL**

Reporting stations list.

**Type**
- binary

**Remarks**
IDMSL lists all reporting stations and location information; its length is variable. This file is supplied with McIDAS-X and -OS2.

<table>
<thead>
<tr>
<th>Word Allocation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( n ), number of stations in the list</td>
</tr>
<tr>
<td>1 to ( 2^n )</td>
<td>8-character station call letters, left justified</td>
</tr>
<tr>
<td>2( n+1 ) to 3( n )</td>
<td>6-digit station identifier</td>
</tr>
<tr>
<td>3( n+1 ) to 4( n )</td>
<td>latitude, degrees*10000</td>
</tr>
<tr>
<td>4( n+1 ) to 5( n )</td>
<td>longitude, degrees*10000</td>
</tr>
<tr>
<td>5( n+1 ) to 6( n )</td>
<td>elevation, meters</td>
</tr>
<tr>
<td>6( n+1 ) to 7( n )</td>
<td>2-character state code</td>
</tr>
<tr>
<td>7( n+1 ) to 8( n )</td>
<td>2-character country code</td>
</tr>
<tr>
<td>8( n+1 ) to 9( n )</td>
<td>reported data types; packed integer representation</td>
</tr>
</tbody>
</table>

**Bit Reporting type**
- 0: 3-hourly synoptic
- 1: 6-hourly synoptic
- 2: off-hourly synoptic
- 3: AERO
- 4: airways
- 5: METAR
- 6: RAOB
- 7: Pibal
- 8: TAF forecast
- 9: PLATF forecast
- 10: radar
- 11: coastal/SMARS
- 12: short range terminal forecast
- 13: hourly (SA)
- 14: forecast (FT)
- 15: radar (SD)

9\( n+1 \) to 9\( n+5 \): 20-character station name

14\( n+1 \) to 15\( n \): surface station reporting characteristics

15\( n+1 \) to 16\( n \): upper air reporting characteristics

16\( n+1 \) to 17\( n \): AFGWC station indicator

17\( n+1 \) to 18\( n \): runway direction, degrees

18\( n+1 \) to 19\( n \): runway latitude, degrees*10000

19\( n+1 \) to 20\( n \): runway longitude, degrees*10000

20\( n+1 \) to 21\( n \): runway elevation, meters

Currently, no API routines exist for reading and writing this file.
**MCRGB.TXT**

Table of color names for McIDAS-X workstations.

<table>
<thead>
<tr>
<th>Type</th>
<th>ASCII</th>
</tr>
</thead>
</table>

**Remarks**

Each line in this text file contains the selected red, green and blue color value and the color name, separated by spaces. For example:

127 35 35 BROWN

**API Routines**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>kolors.for</td>
<td>reads the information in file MCRGB.TXT</td>
</tr>
</tbody>
</table>

**MDXXnnnn**

MD files, where and nnnn is a user-defined number.

<table>
<thead>
<tr>
<th>Type</th>
<th>ASCII/Binary</th>
</tr>
</thead>
</table>

**Remarks**

MD files are large data records addressed by row and column coordinates. All records in a row share a label called a row header. Similarly, all records in a column share a column header. A complete record thus consists of the row header, column header, and data.

The headers hold parameters common to a large number of records. Row headers are located in column 0; column headers are in row 0. Fields in a record are identified by 4-character names called keys. Data values are stored as integers. Each record can be up to 400 words long; individual fields in a record are one word (4 bytes) long.

The MD file structure is self-contained. All information needed to access a file exists in the 4096-word header at the beginning of the file. This header contains the schema, which specifies the default number of rows and columns in an MD file; the composition of the row headers, column headers and data records; and the names, scale factors, and units of the keys.

A copy of the schema resides in the MD file itself as well as in the LW file SCHEMA, which contains all schemas recognized by the system. The copy of the schema held in SCHEMA serves as a blueprint for all MD files of a particular kind of data. When an MD file is created, the schema is copied from SCHEMA to the MD file header block with all appropriate modifications.

MD file numbers can be between 1 and 999999. If an MD file number is five or six digits, the file name begins with only MDX or MD. For example, MD file number 12345 has the file name MDX12345, but MD file number 123456 has the file name MD123456.
<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>schema name</td>
</tr>
<tr>
<td>1</td>
<td>schema version number</td>
</tr>
<tr>
<td>2</td>
<td>schema registration date</td>
</tr>
<tr>
<td>3</td>
<td>default number of rows</td>
</tr>
<tr>
<td>4</td>
<td>default number of columns</td>
</tr>
<tr>
<td>5</td>
<td>total number of keys in the record</td>
</tr>
<tr>
<td>6</td>
<td>number of keys in the row header</td>
</tr>
<tr>
<td>7</td>
<td>number of keys in the column header</td>
</tr>
<tr>
<td>8</td>
<td>number of keys in the data portion</td>
</tr>
<tr>
<td>9</td>
<td>1-based position of the column header</td>
</tr>
<tr>
<td>10</td>
<td>1-based position of the data portion</td>
</tr>
<tr>
<td>11</td>
<td>number of repeat groups</td>
</tr>
<tr>
<td>12</td>
<td>size of the repeat group</td>
</tr>
<tr>
<td>13</td>
<td>starting position of the repeat group</td>
</tr>
<tr>
<td>14</td>
<td>missing data code</td>
</tr>
<tr>
<td>15</td>
<td>integer ID of the file</td>
</tr>
<tr>
<td>16-23</td>
<td>text ID of the file</td>
</tr>
<tr>
<td>24</td>
<td>creator's project number</td>
</tr>
<tr>
<td>25</td>
<td>creation date</td>
</tr>
<tr>
<td>26</td>
<td>creator's ID</td>
</tr>
<tr>
<td>27</td>
<td>zero-based offset to the row header</td>
</tr>
<tr>
<td>28</td>
<td>zero-based offset to the column header</td>
</tr>
<tr>
<td>29</td>
<td>zero-based offset to the data portion</td>
</tr>
<tr>
<td>30</td>
<td>first unused word in the file</td>
</tr>
<tr>
<td>31</td>
<td>start of the user record</td>
</tr>
<tr>
<td>32</td>
<td>start of the key names</td>
</tr>
<tr>
<td>33</td>
<td>start of the scale factors</td>
</tr>
<tr>
<td>34</td>
<td>start of the units</td>
</tr>
<tr>
<td>35-60</td>
<td>reserved</td>
</tr>
<tr>
<td>61-62</td>
<td>file name</td>
</tr>
<tr>
<td>63</td>
<td>MD file number</td>
</tr>
<tr>
<td>64 - 463</td>
<td>user record, MD coordinates (0,0); not described by the schema; use for storing arbitrary information</td>
</tr>
<tr>
<td>464 - 863</td>
<td>names of the file keys</td>
</tr>
<tr>
<td>864 - 1264</td>
<td>scale factors for the keys</td>
</tr>
<tr>
<td>1264 - 1663</td>
<td>units of the keys</td>
</tr>
<tr>
<td>1664 - 4095</td>
<td>reserved</td>
</tr>
</tbody>
</table>

The row header begins at word 4096. The row header's length is rows * number of keys in the row header.

Following the row header is the column header. The column header's length is columns * number of keys in the column header.

The MD data follows the column header. The data's length is rows * columns * number of keys in the data portion. Any remaining words are available for user purposes.

### API Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdi.for</td>
<td>writes to an MD file</td>
</tr>
<tr>
<td>mdo.for</td>
<td>reads an MD file</td>
</tr>
<tr>
<td>mdquit.for</td>
<td>deletes an MD file</td>
</tr>
</tbody>
</table>
**MSUSCRPF**

*MSU calibration reference parameters.*

**Type**

binary

**Remarks**

The parameters in MSUSCRPF compute MSU (Microwave Sounding Unit) brightness temperatures and check the quality of MSU data. The file is organized chronologically by date; the last record is the most recent. Each record contains 48 words. There may be more than one record for a particular satellite.

The first record is a header record. The following records contain data. Words 0 through 11 in the data records are the coefficients of second-degree polynomials used to convert platinum resistance thermometer resistance measurements to temperatures for the internal warm targets.

This file is supplied with McIDAS-X and -OS2.

**Word Allocation**

The first record is a header record. Word 1 of this record contains the total number of records in the file; the remaining 47 words are unused. The other records in the file are formatted as shown below.

<table>
<thead>
<tr>
<th>Record 1</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>total number of records in the file</td>
</tr>
<tr>
<td>1 - 47</td>
<td>unused</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Records 2 through n</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 3</td>
<td>resistance-to-temperature coefficients, zeroth order; if Word 0 is the missing value, the record was deleted</td>
</tr>
<tr>
<td>4 - 7</td>
<td>resistance-to-temperature coefficients, first order</td>
</tr>
<tr>
<td>8 - 11</td>
<td>resistance-to-temperature coefficients, second order</td>
</tr>
</tbody>
</table>

**API Routines**

Currently, no API routines exist for reading and writing this file.
**NAM**

File path redirection entries, where * is a user-defined file name.

**Type**
ASCII

**Remarks**
A *.NAM file is written when a redirect file is saved. Its length depends on the number of entries in the file. Each entry contains 100 characters which include the file name and/or extension, followed by a space and the specified path.

**API Routines**
Currently, no API routines exist for reading and writing this file.

---

**OUTL* **

Base map files, where * is the map file name.

**Type**
binary

**Remarks**
These variable length files contain the base map data for drawing graphical map outlines. The available map files are:

- OUTLSUPU—high resolution USA
- OUTLSUPW—world coastal
- OUTLUSAM—medium resolution USA
- OUTLHPOL—world political boundaries
- OUTLHRES—high resolution world coastal outline
- OUTLUSAL—low resolution USA and North America
- OUTLWRLD—world coastal

**Word Allocation**
The following word allocation applies to the OUTLSUPU, OUTLSUPW, OUTLUSAM, OUTLHPOL and OUTLHRES data structures.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>number of blocks (line segments)</td>
</tr>
<tr>
<td>1 - 6000</td>
<td>directory for line segments; each directory block contains six words of information</td>
</tr>
<tr>
<td>1</td>
<td>minimum latitude, degrees * 10000</td>
</tr>
<tr>
<td>2</td>
<td>maximum latitude, degrees * 10000</td>
</tr>
<tr>
<td>3</td>
<td>minimum longitude, degrees * 10000; west is positive</td>
</tr>
<tr>
<td>4</td>
<td>maximum longitude, degrees * 10000; west is positive</td>
</tr>
<tr>
<td>5</td>
<td>beginning word of data start for segment 1</td>
</tr>
<tr>
<td>6</td>
<td>number of words to read</td>
</tr>
</tbody>
</table>
Word Description
7 - 12 directory for line segment 2
5995-6000 directory for line segment 1000
6001 latitude degrees * 1000
6002 longitude degrees * 1000
6003 latitude degrees * 1000
6004 longitude degrees * 1000

The maximum number of line segments is 1000. The number of points in a segment is limited to 3000 by the arrays in the MAP program. All segments cannot be maximum size or the map file will exceed its maximum size.

The word allocation below is for the OUTLUSAL and OUTLWRLD data structures.

Word Description
0 ... n-1 n number of latitude/longitude pairs as halfwords; longitude, degrees * 10, 0 to 360 west is positive; latitude, degrees * 20, 0 to 180 north is positive; odd number means pen down; even number means pen up end of file marker; HEX FFF89000
n+1

API Routines Currently, no API routines exist for reading and writing this file.

McIDAS-X virtual frame files, where * is a user-defined file name.

Type binary

Remarks These variable-length files save McIDAS-X image/graphics, frame directory, navigation and enhancement information. A file may contain information for one or more image/graphics frames. Subsequent frame entries are appended sequentially.

Word Allocation Word Description
0 number of saved frames
1 frame size, in bytes
2 frame dimension, lines and elements as halfwords
3 number of graphics color levels
4 number of image gray levels
5 version data stamp
6 1 = flag for independent graphics; saves pixmap with and without graphics
7 - 9 not used
10 - n frame entry information blocks for one or more frames

Below is a description of one frame entry in the file.

Word Description
0 - 63 frame directory; see FRAMEDH.tr in this chapter
64 - n variable length frame navigation block; see FRNVDIR.tr and FRNVEXP.tr in this chapter
n+1 - nn image frame data block; (frame size/4) words if no independent graphics; (2*frame size/4) words if independent graphics
nn+1 - nn+817 816-word frame enhancement table; see FRAMENH.tr
nn+818 - mnn red, green and blue graphics color levels; (number of graphics color levels*3) words

API Routines Currently, no API routines exist for reading and writing this file.
SATANNOT

Image satellite annotation description file.

Type
binary

Remarks
The satellite description text information in this file is displayed as

text annotation when a satellite image is loaded onto a display device.

Each 80-character record contains a satellite name, band specifier code
(if applicable) and sensor source number.

File size depends on the number of satellite description entries and is

computed by (20*number of entries) words. For example, one entry

may look like this: GOES-7 IR D 33.

This file is supplied with McIDAS-X and -OS2.

API Routines
Currently, no API routines exist for reading and writing this file.

SAVESTR

String table save file.

Type
binary

Remarks
File length depends on the number of string tables saved in the file.

Word Allocation
SAVESTR consists of two regions. The first region is the directory.
It contains the first 4096 words of the file.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 511</td>
<td>author code; ID of the user who created the string table; four characters per word ID</td>
</tr>
<tr>
<td>512 - 2047</td>
<td>12-character name assigned to a saved table</td>
</tr>
<tr>
<td>2048 - 2559</td>
<td>accounting information, project number</td>
</tr>
<tr>
<td>2560 - 3071</td>
<td>accounting information, creation date</td>
</tr>
<tr>
<td>3072 - 3583</td>
<td>accounting information, user terminal number</td>
</tr>
<tr>
<td>3584 - 4095</td>
<td>accounting information, date of last restore</td>
</tr>
</tbody>
</table>

The second region contains the text of each saved table arranged in
sets of 15 pages each. The position of a table is determined by its
directory entry location. Directory entry 1 is the first 15 pages of the
saved table space. The save space can hold up to 512 string tables. The
word organization is identical to the form described in the STRTABLE
file documented in this chapter.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096 - 19456</td>
<td>saved string table 1</td>
</tr>
<tr>
<td>19457 - 34817</td>
<td>saved string table 2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>saved string table n, where n is a maximum of 512</td>
</tr>
</tbody>
</table>

API Routines
Currently, no API routines exist for reading and writing this file.
**SCHEMA**

Blueprints of all registered schemas.

**Type**
ASCII/binary

**Remarks**
File length depends on the number of schemas registered in the file using the SCHE command.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 31999</td>
<td>up to 500 MD file headers, 64 words each</td>
</tr>
<tr>
<td>32000 - 32767</td>
<td>reserved</td>
</tr>
<tr>
<td>32768...</td>
<td>4096 words per schema</td>
</tr>
</tbody>
</table>

The 64-word file headers are organized as follows.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>schema name</td>
</tr>
<tr>
<td>1</td>
<td>schema version number</td>
</tr>
<tr>
<td>2</td>
<td>registration date</td>
</tr>
<tr>
<td>3</td>
<td>number of rows</td>
</tr>
<tr>
<td>4</td>
<td>number of columns</td>
</tr>
<tr>
<td>5</td>
<td>total number of keys</td>
</tr>
<tr>
<td>6</td>
<td>number of keys in the row header</td>
</tr>
<tr>
<td>7</td>
<td>number of keys in the column header</td>
</tr>
<tr>
<td>8</td>
<td>number of keys in the data record</td>
</tr>
<tr>
<td>9</td>
<td>1-based position in the column header record</td>
</tr>
<tr>
<td>10</td>
<td>1-based position in the data record</td>
</tr>
<tr>
<td>11</td>
<td>number of repetitions of the repeat group</td>
</tr>
<tr>
<td>12</td>
<td>size of the repeat group</td>
</tr>
<tr>
<td>13</td>
<td>starting position of the repeat group</td>
</tr>
<tr>
<td>14</td>
<td>missing data code</td>
</tr>
<tr>
<td>15</td>
<td>integer ID of the file</td>
</tr>
<tr>
<td>16 - 23</td>
<td>text ID of the file</td>
</tr>
<tr>
<td>24</td>
<td>creator's project number</td>
</tr>
<tr>
<td>25</td>
<td>creation date</td>
</tr>
<tr>
<td>26</td>
<td>creator's initials (Logon ID)</td>
</tr>
<tr>
<td>27</td>
<td>zero-based offset to the row header</td>
</tr>
<tr>
<td>28</td>
<td>zero-based offset to the column header</td>
</tr>
<tr>
<td>29</td>
<td>zero-based offset to the data portion</td>
</tr>
<tr>
<td>30</td>
<td>position of the first unused word in the file</td>
</tr>
<tr>
<td>31</td>
<td>starting word location of the user record</td>
</tr>
<tr>
<td>32</td>
<td>starting word location of the key names</td>
</tr>
</tbody>
</table>

**API Routines**
Currently, no API routines exist for reading and writing this file.
SKEDFILE

Scheduled McIDAS commands.

Type

ASCII/binary

Remarks

SKEDFILE has 7172 words and is divided into two sections. The first section contains basic directory information for the scheduler entries. The scheduler (SKED) uses this section to quickly scan all entries to decide which are due to run. The second section contains additional directory information and the text of the McIDAS command for each entry. This section is read when listings are made or a command is ready to run. The file division is invisible to all parts of the schedule system accessing the file via the ID number.

Word Allocation

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>unused</td>
</tr>
<tr>
<td>8-807</td>
<td>first section of scheduler entry information</td>
</tr>
<tr>
<td>808-871</td>
<td>unused</td>
</tr>
<tr>
<td>872-7171</td>
<td>second section of scheduler entry information</td>
</tr>
</tbody>
</table>

The first section of scheduler entry information contains eight words for each entry (maximum of 100). The allocation is shown below.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>time of the next scheduled run; all times are kept internally, in seconds, since 1 January 1972</td>
</tr>
<tr>
<td>1</td>
<td>number of runs remaining</td>
</tr>
<tr>
<td>2</td>
<td>interval between runs</td>
</tr>
<tr>
<td>3</td>
<td>late tolerance if the scheduler is delayed</td>
</tr>
<tr>
<td>4</td>
<td>terminal at which the command runs</td>
</tr>
<tr>
<td>5</td>
<td>ID number or name assigned when the command is entered in the schedule</td>
</tr>
<tr>
<td>6</td>
<td>initials of the person who entered the command in the schedule</td>
</tr>
<tr>
<td>7</td>
<td>project number under which the command runs</td>
</tr>
</tbody>
</table>

The second section of scheduler entry information contains 63 words for each entry. The allocation is shown below.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>time that the command first runs</td>
</tr>
<tr>
<td>1</td>
<td>total number of times the command runs</td>
</tr>
<tr>
<td>2</td>
<td>unused</td>
</tr>
<tr>
<td>3-63</td>
<td>text of the McIDAS command</td>
</tr>
</tbody>
</table>

API Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>skio.for</td>
<td>schedule file I/O routines</td>
</tr>
</tbody>
</table>
Image data stretch tables, where * is a user-defined file name.

Type
binary

Remarks
These 401-word files contain image calibration and input/output breakpoint information for performing image data stretching. Each file contains up to 64 input/output breakpoint pairs.

Word Allocation

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>not used</td>
</tr>
<tr>
<td>1</td>
<td>calibration type, C*4</td>
</tr>
<tr>
<td>2</td>
<td>calibration units, C*4</td>
</tr>
<tr>
<td>3</td>
<td>interpretation type, C*4</td>
</tr>
<tr>
<td>4</td>
<td>number of breakpoints, C*4</td>
</tr>
<tr>
<td>5</td>
<td>band number, C*4</td>
</tr>
<tr>
<td>7</td>
<td>low input range endpoint, C*12</td>
</tr>
<tr>
<td>10</td>
<td>high input range endpoint, C*12</td>
</tr>
<tr>
<td>13</td>
<td>number of VGA levels, integer</td>
</tr>
<tr>
<td>17</td>
<td>first input breakpoint value entered in specified calibration units, C*12</td>
</tr>
<tr>
<td>20</td>
<td>first output brightness value, the range is 0 to 255, C*12</td>
</tr>
<tr>
<td>(n-1) * 6+17</td>
<td>nth input breakpoint value, C*12</td>
</tr>
<tr>
<td>(n-1) * 6+20</td>
<td>nth output brightness value, C*12</td>
</tr>
</tbody>
</table>

Currently, no API routines exist for reading and writing this file.
**STRTABLE**

String table entries file.

**Type**

binary

**Remarks**

STRTABLE is a variable-length file that holds all string tables currently active on the system. All string tables not currently active reside in the string table save file SAVESTR. STRTABLE is partitioned into blocks of 15360 words arranged by increasing terminal number. Valid terminal numbers for this structure are zero to 99. To calculate the beginning word position of a block, use: terminal number*15360.

**Word Allocation**

Each block contains the following information.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 255</td>
<td>length of each string, in bytes</td>
</tr>
<tr>
<td>256 - 1023</td>
<td>string names, C*12</td>
</tr>
<tr>
<td>1024 - 3071</td>
<td>reserved</td>
</tr>
<tr>
<td>3072 - 13311</td>
<td>256 strings, 160 bytes per string</td>
</tr>
<tr>
<td>13312 - 15359</td>
<td>reserved</td>
</tr>
</tbody>
</table>

**API Routines**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbget.for</td>
<td>writes to a table</td>
</tr>
<tr>
<td>lbput.for</td>
<td>reads from a table</td>
</tr>
</tbody>
</table>

**SYSKEY.TAB**

System options table.

**Type**

binary

**Remarks**

This variable-length file contains system option values unique to a system and is different for each system. SSEC uses the word allocation below.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>date the file was last written</td>
</tr>
<tr>
<td>1</td>
<td>header label, C*4</td>
</tr>
<tr>
<td>5</td>
<td>largest foreground terminal number</td>
</tr>
<tr>
<td>6</td>
<td>maximum area number in use, 32767 maximum</td>
</tr>
<tr>
<td>7</td>
<td>maximum grid file number in use, 99999 maximum</td>
</tr>
<tr>
<td>8</td>
<td>reserved for mainframe</td>
</tr>
<tr>
<td>9</td>
<td>maximum MD file number in use, 99999 maximum</td>
</tr>
<tr>
<td>10</td>
<td>default system printer number</td>
</tr>
<tr>
<td>14</td>
<td>password flag used for LOGONs -1 = off</td>
</tr>
<tr>
<td>0 = on, allows NULL</td>
<td></td>
</tr>
<tr>
<td>1 = on full force</td>
<td></td>
</tr>
<tr>
<td>898</td>
<td>default for string table tracking for programs UP, SP, SC and UC (Yes or No); C*4</td>
</tr>
<tr>
<td>899</td>
<td>default projection for programs UP, SP, SC and UC (PS, MERC); C*4</td>
</tr>
<tr>
<td>900</td>
<td>default units for programs SL, SP and SC (A for American or M for metric)</td>
</tr>
<tr>
<td>901</td>
<td>number of kbytes allowed in one inter system transfer</td>
</tr>
<tr>
<td>910</td>
<td>maximum number of image frames per terminal on the system</td>
</tr>
<tr>
<td>911</td>
<td>maximum number of graphics frames per terminal on the system</td>
</tr>
<tr>
<td>1000 - 3599</td>
<td>reserved for applications</td>
</tr>
<tr>
<td>3604</td>
<td>end of table</td>
</tr>
<tr>
<td>5000</td>
<td>start of the user or site defined SYSKEY words</td>
</tr>
<tr>
<td>5999</td>
<td>end of the user or site defined SYSKEY words</td>
</tr>
</tbody>
</table>

**API Routines**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>keys.for</td>
<td>reads a SYSKEY.TAB word</td>
</tr>
<tr>
<td>sysin.for</td>
<td>writes a SYSKEY.TAB word</td>
</tr>
</tbody>
</table>
TERMCHAR.ttt

Terminal characteristics file, where ttt is the terminal number.

Type

binary

Remarks

This 10266-word file contains terminal characteristic information specific to the workstation display type.

Word Allocation

The first 1024-word block contains the terminal characteristic specification strings.

<table>
<thead>
<tr>
<th>Word</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 2</td>
<td><code>TERMINAL_TYP</code></td>
<td>terminal type</td>
</tr>
<tr>
<td>3 - 5</td>
<td><code>DSP_LINES</code></td>
<td>maximum number of display lines</td>
</tr>
<tr>
<td>6 - 8</td>
<td><code>DSP_ELEMENTS</code></td>
<td>maximum number of display elements</td>
</tr>
<tr>
<td>9 - 11</td>
<td><code>MAX_COLORS</code></td>
<td>maximum available colors</td>
</tr>
<tr>
<td>12 - 14</td>
<td><code>MX_CUR_SIZ_H</code></td>
<td>maximum cursor height</td>
</tr>
<tr>
<td>15 - 17</td>
<td><code>MX_CUR_SIZ_W</code></td>
<td>maximum cursor width</td>
</tr>
<tr>
<td>18 - 20</td>
<td><code>IND_GRAPHIC</code></td>
<td>does the display have independent graphics?</td>
</tr>
<tr>
<td>21 - 23</td>
<td><code>VAR_FR_SIZE?</code></td>
<td>does the display have variable frame size?</td>
</tr>
<tr>
<td>24 - 26</td>
<td><code>DISP_TOGGLE?</code></td>
<td>does the display type toggle to display image frames?</td>
</tr>
<tr>
<td>27 - 29</td>
<td><code>NORTLE?</code></td>
<td>does the display type have nortel capability?</td>
</tr>
<tr>
<td>30 - 32</td>
<td><code>STRETCHING?</code></td>
<td>does the display type have interactive contrast stretching?</td>
</tr>
<tr>
<td>33 - 35</td>
<td><code>BITS_PER_PIX</code></td>
<td>number of bits per pixel</td>
</tr>
<tr>
<td>36 - 38</td>
<td><code>GRAY_LEVELS</code></td>
<td>number of gray levels supported</td>
</tr>
<tr>
<td>39 - 41</td>
<td><code>ANNOTAT_SIZE</code></td>
<td>annotation size</td>
</tr>
<tr>
<td>42 - 44</td>
<td><code>DWELL_RATE_K</code></td>
<td>dwell rate constant</td>
</tr>
<tr>
<td>45 - 47</td>
<td><code>CUR_BOX?</code></td>
<td>does the display support the box cursor type?</td>
</tr>
<tr>
<td>48 - 50</td>
<td><code>CUR_XHAIR?</code></td>
<td>does the display support the crosshair cursor type?</td>
</tr>
<tr>
<td>51 - 53</td>
<td><code>CUR_XBOX?</code></td>
<td>does the display support the crosshair with center box cursor type?</td>
</tr>
<tr>
<td>54 - 56</td>
<td><code>CUR_SOLID?</code></td>
<td>does the display support the solid box cursor type?</td>
</tr>
</tbody>
</table>

The following 1024-word blocks contain the terminal characteristics for the various display types.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 - 1049</td>
<td>terminal characteristics for tower</td>
</tr>
<tr>
<td>1050 - 2047</td>
<td>not used</td>
</tr>
<tr>
<td>2048 - 2073</td>
<td>terminal characteristics for WIDE WORD</td>
</tr>
<tr>
<td>2074 - 4095</td>
<td>not used</td>
</tr>
<tr>
<td>4096 - 4121</td>
<td>terminal characteristics for VGA</td>
</tr>
<tr>
<td>4122 - 5119</td>
<td>not used</td>
</tr>
<tr>
<td>5120 - 5145</td>
<td>terminal characteristics for MODE-13</td>
</tr>
<tr>
<td>5146 - 6143</td>
<td>not used</td>
</tr>
<tr>
<td>6144 - 6169</td>
<td>terminal characteristics for the SDA</td>
</tr>
<tr>
<td>6170 - 7167</td>
<td>not used</td>
</tr>
<tr>
<td>7168 - 7193</td>
<td>terminal characteristics for Unix</td>
</tr>
<tr>
<td>7194 - 9215</td>
<td>not used</td>
</tr>
<tr>
<td>9216 - 9241</td>
<td>terminal characteristics for Presentation Manager</td>
</tr>
<tr>
<td>9242 - 10266</td>
<td>not used</td>
</tr>
</tbody>
</table>

API Routines

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>initrm.for</td>
<td>creates TERMCHAR.ttt, called by SETUP.FP</td>
</tr>
<tr>
<td>itrmch.for</td>
<td>reads information from file TERMCHAR.ttt</td>
</tr>
</tbody>
</table>
## UC

**McIDAS User Common area.**

### Type
- ASCII/binary

### Remarks
User Common variables reside in memory. They are lost if McIDAS stops, but are not lost if your process terminates. UC variables with negative subscripts are carried along from the calling process through all spawned processes. UC variables with positive subscripts and UC(0) pertain only to individual tasks that are running whether spawned or not.

### Word Allocation

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-120</td>
<td>font set flag; McIDAS-X only</td>
</tr>
<tr>
<td>0</td>
<td>font not set</td>
</tr>
<tr>
<td>1</td>
<td>font set</td>
</tr>
<tr>
<td>-46</td>
<td>graphics width</td>
</tr>
<tr>
<td>-39</td>
<td>number of the remote terminal set by the scanner or TOKANL.FOR</td>
</tr>
<tr>
<td>-1</td>
<td>local terminal</td>
</tr>
<tr>
<td>-2</td>
<td>local txw</td>
</tr>
<tr>
<td>-38</td>
<td>next record pointer for the file created when using DEV=F</td>
</tr>
<tr>
<td>-37 to -35</td>
<td>LW file name for the DEV=f file</td>
</tr>
<tr>
<td>-34</td>
<td>virtual graphics number to plot</td>
</tr>
<tr>
<td>&gt;0</td>
<td>virtual graphics number</td>
</tr>
<tr>
<td>-33</td>
<td>current diagnostic message (DDEST) device</td>
</tr>
<tr>
<td>0</td>
<td>black hole</td>
</tr>
<tr>
<td>1</td>
<td>CRT screen</td>
</tr>
<tr>
<td>2</td>
<td>local printer</td>
</tr>
<tr>
<td>3</td>
<td>system printer</td>
</tr>
<tr>
<td>-32</td>
<td>current error message (EDEST) device</td>
</tr>
<tr>
<td>0</td>
<td>black hole</td>
</tr>
<tr>
<td>1</td>
<td>CRT screen</td>
</tr>
<tr>
<td>2</td>
<td>local printer</td>
</tr>
<tr>
<td>3</td>
<td>system printer</td>
</tr>
<tr>
<td>-31</td>
<td>current device used when calling the LTQ.FOR, SDEST or SPOUT subroutine</td>
</tr>
<tr>
<td>0</td>
<td>black hole</td>
</tr>
<tr>
<td>1</td>
<td>CRT screen</td>
</tr>
<tr>
<td>2</td>
<td>local printer</td>
</tr>
<tr>
<td>3</td>
<td>system printer</td>
</tr>
<tr>
<td>-29</td>
<td>reserved</td>
</tr>
<tr>
<td>-28</td>
<td>name (characters 5 through 8) of the program currently running, C*4</td>
</tr>
<tr>
<td>-27</td>
<td>name (first four characters) of the program currently running, C*4</td>
</tr>
<tr>
<td>-26</td>
<td>program type state</td>
</tr>
<tr>
<td>0</td>
<td>program currently running is not a macro</td>
</tr>
<tr>
<td>1</td>
<td>program currently running is a macro</td>
</tr>
<tr>
<td>-25</td>
<td>program environment state</td>
</tr>
<tr>
<td>0</td>
<td>program currently running is background</td>
</tr>
<tr>
<td>1</td>
<td>program currently running is foreground</td>
</tr>
<tr>
<td>-1</td>
<td>program currently running is background with FORTRAN main; can do FORTRAN I/O</td>
</tr>
<tr>
<td>-24</td>
<td>2 = program currently running is a console started task, i.e., tape job</td>
</tr>
<tr>
<td>-23</td>
<td>current level when starting another process</td>
</tr>
<tr>
<td>-22</td>
<td>0 = scanner</td>
</tr>
<tr>
<td>-21</td>
<td>1 = next program, etc.</td>
</tr>
<tr>
<td>-17</td>
<td>number of this initiator</td>
</tr>
<tr>
<td>-20 to -18</td>
<td>command initialization status</td>
</tr>
<tr>
<td>-16</td>
<td>reserved</td>
</tr>
<tr>
<td>-15</td>
<td>user initials under which this command runs; may be different from the logged on initials in UC(2) if the command is scheduled, C*4</td>
</tr>
<tr>
<td>-10</td>
<td>project number under which this command runs; may be different from the logged on project number in UC(1)</td>
</tr>
</tbody>
</table>

---

**Data File Structures**

6-40  

**McIDAS Programmer's Manual**  
Preliminary Issue 10/93  

---

**Data File Structures**

6-41
Words -12 to -1 contain a snapshot of the terminal state taken just before the command begins. The format is similar to the terminal protocol.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-12</td>
<td>cursor element position; see UC word 64</td>
</tr>
<tr>
<td>-11</td>
<td>cursor line position; see UC word 63</td>
</tr>
<tr>
<td>-10</td>
<td>cursor size, horizontal; see UC word 62</td>
</tr>
<tr>
<td>-9</td>
<td>cursor size, vertical; see UC word 61</td>
</tr>
<tr>
<td>-8</td>
<td>graphics lower bound; see UC word 58</td>
</tr>
<tr>
<td>-7</td>
<td>graphics upper bound; see UC word 57</td>
</tr>
<tr>
<td>-6</td>
<td>current graphics frame; see UC word 56</td>
</tr>
<tr>
<td>-4</td>
<td>frame state control word; if bit 0=1, the frame is connected to loop control (Y key), if bit 1=1, the frame is looping (L key), if bit 2=0, the frame is blanked (K key)</td>
</tr>
<tr>
<td>-3</td>
<td>image frame lower bound; see UC word 53</td>
</tr>
<tr>
<td>-2</td>
<td>image frame upper bound; see UC word 52</td>
</tr>
<tr>
<td>-1</td>
<td>current image frame; see UC word 51</td>
</tr>
<tr>
<td>0</td>
<td>user's terminal number</td>
</tr>
<tr>
<td>1</td>
<td>project number under which the current user is logged on; may be different from UC(-16)</td>
</tr>
<tr>
<td>2</td>
<td>user's initials, C*4</td>
</tr>
<tr>
<td>4</td>
<td>current navigation file number</td>
</tr>
<tr>
<td>5</td>
<td>current MD file number</td>
</tr>
<tr>
<td>6</td>
<td>current grid file number</td>
</tr>
<tr>
<td>7</td>
<td>current 3-D grid file number</td>
</tr>
<tr>
<td>9</td>
<td>second printer number for this user</td>
</tr>
<tr>
<td>10</td>
<td>first printer number; local printer for this user</td>
</tr>
<tr>
<td>11</td>
<td>number of lines on the TV screen</td>
</tr>
<tr>
<td>12</td>
<td>number of elements on the TV screen</td>
</tr>
<tr>
<td>13</td>
<td>number of image frames</td>
</tr>
<tr>
<td>14</td>
<td>number of graphics frames</td>
</tr>
<tr>
<td>15</td>
<td>terminal communications type state</td>
</tr>
<tr>
<td>0</td>
<td>local ProNET</td>
</tr>
<tr>
<td>1</td>
<td>terminal is remote synchronous</td>
</tr>
<tr>
<td>2</td>
<td>asynchronous</td>
</tr>
<tr>
<td>16</td>
<td>terminal type state</td>
</tr>
<tr>
<td>0</td>
<td>terminal is nonvideo</td>
</tr>
<tr>
<td>1</td>
<td>terminal is video</td>
</tr>
<tr>
<td>2</td>
<td>host</td>
</tr>
<tr>
<td>17</td>
<td>flag for the E key</td>
</tr>
<tr>
<td>0</td>
<td>lat/lon are displayed in DDDMMSS</td>
</tr>
<tr>
<td>1</td>
<td>lat/lon are displayed in decimal</td>
</tr>
</tbody>
</table>

Word 18 - 19: reserved
Word 20: set to 1 by the G key, otherwise 0
Word 21: set to 1 by the Q key, otherwise 0
Word 22 - 29: reserved
Word 37: image display hardware
Word 0: tower
Word 1: WIDE WORD
Word 3: VQA
Word 4: MODE-13
Word 5: SDA (SSSEC Display Adapter)
Word 6: McIDAS-X
Word 38: graphics state flag
Word 0: does not draw graphics
Word 1: draws graphics
Word 39: virtual graphics state flag
Word 0: does not write virtual graphics nonzero = virtual frame to write
Word 40 - 45: default graphics line width
Word 46: graphics dash pattern, dash length in pixels
Word 47: graphics dash pattern, gap length in pixels
Word 48: graphics dash pattern, gap color
Word 49: the loop control system consists of the LS command and the A, B, J, Y, O and L keys; images and graphics are independently connected to and disconnected from the loop system via UC 54 and 59
Word 50: loop control system is looping
Word 2: loop control system is not looping
Word 51: current image frame
Word 52: image frame loop, upper bound
Word 53: image frame loop, lower bound
Word 54: image frame loop connection state flag
Word 1: image frames are connected to the loop control
Word 0: image frames are not connected
Word 55: image frame on/off state flag
Word 1: image frames are visible
Word 0: image frames are blanked
Word 56: current graphics frame
Word 57: graphics upper bound
Word 58: graphics lower bound
Word 59: graphics frame loop connection state flag
Word 1: graphics frames are connected to the loop control
Word 0: graphics frames are not connected
Word | Description
--- | ---
60 | graphics frame on/off state flag 1 = graphics frames are visible 0 = graphics frames are blanked
61 | cursor size, vertical
62 | cursor size, horizontal
63 | cursor position, line number
64 | cursor position, element number
65 | cursor type state flag 1 = box 2 = crosshair 3 = inside box 4 = solid box 5 = external crosshair with bullseye
66 | cursor color, McIDAS-OS2
67 | mouse control state flag 0 = disconnected; program controls cursor position 1 = mouse controls the cursor position reserved
69 - 82 | terminal type for the PC at logon; CHARACTER, 4 bytes; zero if not a PC
83 | reserved
84 - 100 | copy of the raw text the current or last command entered from the keyboard
101 - 120 | used by the scheduler function DOSKED.FOR to indicate the last time through all the scheduled entries flag
122 | reserved
123 - 124 | user or site defined UC words
125 - 129 | TCP/IP address of session partner
158 - 159 | TCP rate measurement word
160 | TCP connection status code
163 | TCP reserved
164 | velocity cursor toggle used by the PCMW program 0 = velocity cursor off 1 = velocity cursor on
165 | sampling flag used by the PCMW program
166 | mode is active flag used by the PCMW program
167 | mode switch for VGA; set by VGA.PGM, SCRCPTL.PGM and KBDCTL.PGM
168 | asynchronous communications state flag 0 = async comm is OK 1 = async comm is suspended; port available
169 | reserved

Word | Description
--- | ---
170 | image loop direction control flag -1 = loop backwards 1 = loop in both directions
171 | VGA frame save flag 0 = save the frames 1 = don't save them
172 - 176 | communication connection state flag 0 = no carrier 1 = communications connected 2 = connection in progress indicates application intercepting mouse button press 0 = button press goes to mouse (McIDAS-OS2) or mcwindow (McIDAS-X) 1 = button press goes to program scheduler update flag used by DOSKED.FOR 0 = check update flags, otherwise don't
179 | depth counter for started processes on the system; used by ABORT.FOR and COMM.PGM; integer value greater or equal to zero pacing strategy flag; 2 = turn pacing ON in XONOFF.FOR, otherwise don't serialization mode used by PFKEY.FOR 0 = F Key menu system not active 1 = F Key menu system active reserved
183 | counter used by string table I/O utilities when creating new strings; integer value range is 0 to 256 mouse button status flag used by MOUSE.PGM (McIDAS-OS2) or mcwindow (McIDAS-X); two 2-byte integer (1 * 2) values that correspond to mouse buttons 1 and 2 respectively mouse movement values (mickies); two 2-byte integer (1 * 2) values that correspond to movement up/down and left/right respectively; McIDAS-OS2 only
185 | switch to tell MOUSE.PGM to set mickies into UC (186); McIDAS-OS2 only 0 = switch off 1 = switch on, just set mickies
187 | image nortle toggle 0 = nortle off 1 = nortle on
188 | reserved
189 | Data File Structures 6-64 McIDAS Programmer's Manual Preliminary Issue 1095
<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>189</td>
<td>command routing switch 0 = command runs locally 1 = command goes to the host 0 = spool not in use 1 = spool in use</td>
</tr>
<tr>
<td>190</td>
<td>outgoing spool-in-use flag used by SENOPN.FOR 0 = spool not in use 1 = spool in use</td>
</tr>
<tr>
<td>191</td>
<td>reserved</td>
</tr>
<tr>
<td>192</td>
<td>outgoing message request pointer used by TCPIP.PGM; integer value greater than or equal to 0</td>
</tr>
<tr>
<td>193</td>
<td>outgoing message completed pointer used by TCPIP.PGM; integer value greater than or equal to 0 system shutdown request status; integer value not equal to 0 indicates a request for system shutdown</td>
</tr>
<tr>
<td>195</td>
<td>window number of the last text window displayed</td>
</tr>
<tr>
<td>196</td>
<td>counter of number of lines of text waiting to be displayed; incremented by SPOUTFOR</td>
</tr>
<tr>
<td>197</td>
<td>pointer from the decoder, set by IMPORT.PGM and used by TCPIP.PGM; integer value greater than or equal to 0</td>
</tr>
<tr>
<td>198</td>
<td>communications port name, literal; binary 0 for ProNET</td>
</tr>
<tr>
<td>199</td>
<td>last written pointer from the decoder; set by TCPIP.PGM; integer value greater than or equal to 0 text window to display; integer value; VGA only -1 = display video frame</td>
</tr>
<tr>
<td>200</td>
<td>first line in the text window to display; integer value; the range is 1 to 57; VGA only; value is ignored on windows 5 through 9 same as UC(63)</td>
</tr>
<tr>
<td>202</td>
<td>same as UC(64)</td>
</tr>
<tr>
<td>203</td>
<td>last frame displayed; integer value; VGA only 0 = forces redisplay of the current frame</td>
</tr>
<tr>
<td>205</td>
<td>keystroke routing switch used by KBDC1TLS.PGM 0 = pass keystrokes to the command line -1 = pass keystrokes to the ASK process place where KBDC1TLS.PGM puts characters when UC(205) is nonzero mode switch used by KBDC1TLS.PGM 0 = not in the insert mode; otherwise in insert mode PF key mode restriction switch 0 = no restrictions; otherwise the restrictions are that command line text is not allowed and the user cannot change the text window; the programs must change the window</td>
</tr>
</tbody>
</table>

- **209**: address of a memory segment used for displaying text on windows 5 through 9; each text window uses 4000 bytes; each character is represented in a 2-byte format where the least significant byte is the character displayed and the most significant byte is the color attribute corresponding to that character; McIDAS-OS2 only
- **210**: number of memory segments supporting frames for a VGA display; four segments per frame
- **211+**: addresses of memory segments supporting frames for a VGA display; four segments per frame
- **215+**: WIDE WORD addresses of memory segments for enhancements and graphics (16 segments) or SDA display addresses of memory segments for image and graphics enhancements; the graphics enhancements immediately follow the image enhancements for each frame; for SDA, the graphics enhancement tables occupy 48 bytes at the end of the image enhancements image zoom factor; integer value; WIDE WORD and SDA only
- **300**: secondary display identifier; integer value; WIDE WORD and SDA only 0 = opposite frame 0 = image frame 0 = graphics frame 0 = image frame; SDA handler lock out request used by TVSYNC.ASM 0 = not locked; otherwise locked 0 = not locked; otherwise locked 0 = WIDE WORD briefing frame threshold (WWW.PGM); an integer value greater than or equal to zero |
| 303   | last character from the WIDE WORD briefing frame control port; literal value; used by BRIEF.PGM cursor control identifier; integer value; WIDE WORD only 0 = cursor on center 1 = cursor dragged to edge SDA display image memory model mode 0 = minimum memory model 1 = extended memory model SDA display graphics memory model mode 0 = minimum memory model 1 = extended memory model SDA memory model mode 0 = minimum memory model 1 = medium memory model 2 = extended memory model |
### Word Description

309  | SDA display graphics channel mask
310 - 311 | reserved
312  | SDA display frame load switch
1 = no load pending
0 = SDA frame load pending
313  | number of image frames configured on the SDA adapter
314  | number of graphics frames configured on the SDA adapter
364  | last image frame displayed; SDA only
0 = forces redisplay of the current image frame
last graphics frame displayed; SDA only
366  | enhancement table load switch
0 = no load enhancement table from PC memory to SDA memory; SDA only
371  | number of shared segments supporting SDA images, graphics and enhancement tables; SDA only
400  | number of text screens in use; McIDAS-X only
401 - 450 | name of the pipe for text screens; McIDAS-X only
451 - 453 | cursor color (red, green, blue); the range is 0 to 255; McIDAS-X only
454  | maximum number of frames allowed; McIDAS-X only
455  | abort signal handling; see UC(-21); McIDAS-X only
0 = no trap
1 = traps abort signals
1 = image enhancements can be modified
1 = graphics enhancements can be modified
461  | enhancement modification switch; McIDAS-X only
0 = image enhancements cannot be modified
1 = image enhancements can be modified
462  | graphics modification switch; McIDAS-X only
0 = graphics colors cannot be modified
1 = graphics colors can be modified
463  | loop frame dwell rate switch; McIDAS-X only
0 = dwell rates are applied to the loop
1 = dwell rates are applied to the frame number corresponding to the shared memory block; McIDAS-X only
464  | command window identification; literal; McIDAS-X only
465  | looping toggle; McIDAS-X only
0 = forward looping
1 = backward looping

### Word API Routines

470  | font set flag; McIDAS-X only
0 = font not set
1 = font set
471 - 480 | reserved for McIDAS-X expansion
490  | current text window; McIDAS-X only
500  | number of graphics levels; McIDAS-X only
501+ | block containing the McIDAS-X graphics levels
500  | number of gray levels; McIDAS-X only
601+ | block containing the McIDAS-X gray levels
700  | address into shared memory for this McIDAS User Common segment; McIDAS-X only
2000  | image window identification; literal; McIDAS-X only
2001+ | frame allocation pointers for WIDE WORD and SDA displays; image frames start at 2001 and increase; graphics frames start at 2999 and decrease; pixmap addresses for McIDAS-X workstations
3001+ | line and element sizes for frames (halfwords); McIDAS-X, WIDE WORD, and SDA workstations; number of lines in the most significant 2 bytes; number of elements in the least significant 2 bytes relative pointer used for random looping
4000  | image frame number listing for random image looping where a zero is the end of the loop
5001+ | list of frame dwell rates for random looping where a zero is the end of the list
6000  | relative pointer for random graphics looping where a zero is the end of the loop
6001+ | graphics frame number listing for random graphics looping where a zero is the end of the loop
7001+ | opposite frame number listing for random looping where a zero is the end of the list
16383 | end of the UC segment

#### Data File Structures

- luc.for: reads a word from User Common
- puc.for: writes a word to User Common
**UCTERM.ttt**

*Pointer to shared memory files, where ttt is the terminal number.*

**Type**

binary

**Remarks**

This file points to the shared memory block for storing McIDAS User Common (UC), allowing several McIDAS sessions to run simultaneously on one McIDAS-X workstation.

**Word Allocation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>not used</td>
</tr>
<tr>
<td>1</td>
<td>index into which of the shared memory blocks is attached to this terminal</td>
</tr>
</tbody>
</table>

**API Routines**

Currently, no API routines exist for reading and writing this file.

---

**VASTBLS**

*VAS calibration tables.*

**Type**

binary

**Remarks**

VASTBLS holds the current calibration values of the radiance, temperature and brightness for all 12 channels of GOES AA satellite data. Its fixed size is 1622016 words.

**Word Allocation**

The primary blocks for VASTBLS are shown below. Each block represents a table of calibrated values for each raw value.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 540671</td>
<td>radiance calibration block (watts/meter**2/steradian*1000)</td>
</tr>
<tr>
<td>540672 - 1081343</td>
<td>temperature calibration block, in Kelvin</td>
</tr>
<tr>
<td>1081344 - 1622015</td>
<td>brightness calibration block; the range is 0 to 255</td>
</tr>
</tbody>
</table>

Each primary segment contains secondary segments allocated by band number; 12 are possible. All word addresses are relative to the start of each primary segment.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 45055</td>
<td>band 1 block</td>
</tr>
<tr>
<td>45056 - 90111</td>
<td>band 2 block</td>
</tr>
<tr>
<td>90112 - 135167</td>
<td>band 3 block</td>
</tr>
<tr>
<td>135168 - 180223</td>
<td>band 4 block</td>
</tr>
<tr>
<td>180224 - 225279</td>
<td>band 5 block</td>
</tr>
<tr>
<td>225280 - 270335</td>
<td>band 6 block</td>
</tr>
<tr>
<td>270336 - 315391</td>
<td>band 7 block</td>
</tr>
<tr>
<td>315392 - 360447</td>
<td>band 8 block</td>
</tr>
<tr>
<td>360448 - 405503</td>
<td>band 9 block</td>
</tr>
<tr>
<td>405504 - 450559</td>
<td>band 10 block</td>
</tr>
<tr>
<td>450560 - 495615</td>
<td>band 11 block</td>
</tr>
<tr>
<td>495616 - 540671</td>
<td>band 12 block</td>
</tr>
</tbody>
</table>
Each secondary segment contains base segments allocated by the Delta-F value. This value ranges from -5 to +5, providing 11 base segments per secondary segment. Each of the word addresses below is relative to the start of the secondary segment. The values are the actual numbers used to calibrate the data.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 4095</td>
<td>Delta-F = -5</td>
</tr>
<tr>
<td>4096 - 8191</td>
<td>Delta-F = -4</td>
</tr>
<tr>
<td>8192 - 12287</td>
<td>Delta-F = -3</td>
</tr>
<tr>
<td>12288 - 16383</td>
<td>Delta-F = -2</td>
</tr>
<tr>
<td>16384 - 20479</td>
<td>Delta-F = -1</td>
</tr>
<tr>
<td>20480 - 24575</td>
<td>Delta-F = 0</td>
</tr>
<tr>
<td>24576 - 28671</td>
<td>Delta-F = +1</td>
</tr>
<tr>
<td>28672 - 32767</td>
<td>Delta-F = +2</td>
</tr>
<tr>
<td>32768 - 36863</td>
<td>Delta-F = +3</td>
</tr>
<tr>
<td>36864 - 40959</td>
<td>Delta-F = +4</td>
</tr>
<tr>
<td>40960 - 45055</td>
<td>Delta-F = +5</td>
</tr>
</tbody>
</table>

**API Routines**

Currently, no API routines exist for reading and writing this file.

**Remarks**

This file identifies the current version of McIDAS software on your workstation and the date of its release.

This file is supplied with McIDAS-X and -OS2.

**API Routines**

Currently, no API routines exist for reading and writing this file.
*.VIF

McIDAS-OS2 virtual frame files, where * is a user-defined file name.

Type

binary

Remarks

These variable-length files store image, graphics, frame directory, navigation and enhancement information for VGA workstations. A file may contain information for one or more image and graphics frames. Subsequent frames are appended sequentially.

Word Allocation

Below is a description of one frame entry in the file.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>version date stamp</td>
</tr>
<tr>
<td>1 - 28160</td>
<td>4 VGA bit planes for the frame; each bit plane has 7040 words</td>
</tr>
<tr>
<td>28161 - 28225</td>
<td>64-word frame directory; see FRAMED Italics</td>
</tr>
<tr>
<td>28226 - n</td>
<td>variable length frame navigation block; see FRNVDIR Italics and FRNVEXP Italics</td>
</tr>
<tr>
<td>n+1 - n+816</td>
<td>816-word frame enhancement table; see FRAMENH Italics</td>
</tr>
</tbody>
</table>

API Routines

Currently, no API routines exist for reading and writing this file.

VIRTnnnn

Virtual graphics files, where nnnn is the virtual graphics number.

Type

binary

Remarks

These variable-length files contain one or more virtual graphics scenes. A scene contains graphics attribute information, such as page boundary, line width, pen position and color information.

Word Allocation

Below is a description of one scene in a virtual graphics file.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>top of the page boundary and virtual graphics version as halfwords</td>
</tr>
<tr>
<td>1</td>
<td>bottom of the page boundary and line width as halfwords</td>
</tr>
<tr>
<td>2</td>
<td>left edge of the page boundary</td>
</tr>
<tr>
<td>3</td>
<td>right edge of the page boundary</td>
</tr>
</tbody>
</table>
| 4    | beginning pen position/color triples where (in an x-y plane):
word n = pen y position
word n+1 = pen x position
word n+2 = pen color
Decimal 256 marks the end of a scene. |

API Routines

Name | Function
-----|------------
vput.ft | writes to a virtual graphics file